Microsoft

OFFICIAL MICROSOFT LEARNING PRODUCT

20463C

Implementing a Data Warehouse with Microsoft® SQL Server®

Product Number: 20463C

Part Number (if applicable): X19-32474

Released: 08/2014

MICROSOFT LICENSE TERMS
MICROSOFT INSTRUCTOR-LED COURSEWARE

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to your use of the content accompanying this agreement which includes the media on which you received it, if any. These license terms also apply to Trainer Content and any updates and supplements for the Licensed Content unless other terms accompany those items. If so, those terms apply.

BY ACCESSING, DOWNLOADING OR USING THE LICENSED CONTENT, YOU ACCEPT THESE TERMS. IF YOU DO NOT ACCEPT THEM, DO NOT ACCESS, DOWNLOAD OR USE THE LICENSED CONTENT.

If you comply with these license terms, you have the rights below for each license you acquire.

1. DEFINITIONS.

a. "Authorized Learning Center" means a Microsoft IT Academy Program Member, Microsoft Learning Competency Member, or such other entity as Microsoft may designate from time to time.

b. "Authorized Training Session" means the instructor-led training class using Microsoft Instructor-Led Courseware conducted by a Trainer at or through an Authorized Learning Center.

c. "Classroom Device" means one (1) dedicated, secure computer that an Authorized Learning Center owns or controls that is located at an Authorized Learning Center's training facilities that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.

d. "End User" means an individual who is (i) duly enrolled in and attending an Authorized Training Session or Private Training Session, (ii) an employee of a MPN Member, or (iii) a Microsoft full-time employee.

e. "Licensed Content" means the content accompanying this agreement which may include the Microsoft Instructor-Led Courseware or Trainer Content.

f. "Microsoft Certified Trainer" or "MCT" means an individual who is (i) engaged to teach a training session to End Users on behalf of an Authorized Learning Center or MPN Member, and (ii) currently certified as a Microsoft Certified Trainer under the Microsoft Certification Program.

g. "Microsoft Instructor-Led Courseware" means the Microsoft-branded instructor-led training course that educates IT professionals and developers on Microsoft technologies. A Microsoft Instructor-Led Courseware title may be branded as MOC, Microsoft Dynamics or Microsoft Business Group courseware.

h. "Microsoft IT Academy Program Member" means an active member of the Microsoft IT Academy Program.

i. "Microsoft Learning Competency Member" means an active member of the Microsoft Partner Network program in good standing that currently holds the Learning Competency status.

j. "MOC" means the "Official Microsoft Learning Product" instructor-led courseware known as Microsoft Official Course that educates IT professionals and developers on Microsoft technologies.

k. "MPN Member" means an active Microsoft Partner Network program member in good standing.

l. "Personal Device" means one (1) personal computer, device, workstation or other digital electronic device that you personally own or control that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.

m. "Private Training Session" means the instructor-led training classes provided by MPN Members for corporate customers to teach a predefined learning objective using Microsoft Instructor-Led Courseware. These classes are not advertised or promoted to the general public and class attendance is restricted to individuals employed by or contracted by the corporate customer.

n. "Trainer" means (i) an academically accredited educator engaged by a Microsoft IT Academy Program Member to teach an Authorized Training Session, and/or (ii) a MCT.

o. "Trainer Content" means the trainer version of the Microsoft Instructor-Led Courseware and additional supplemental content designated solely for Trainers' use to teach a training session using the Microsoft Instructor-Led Courseware. Trainer Content may include Microsoft PowerPoint presentations, trainer preparation guide, train the trainer materials, Microsoft One Note packs, classroom setup guide and Pre-release course feedback form. To clarify, Trainer Content does not include any software, virtual hard disks or virtual machines.

2. USE RIGHTS. The Licensed Content is licensed not sold. The Licensed Content is licensed on a ***one copy per user basis***, such that you must acquire a license for each individual that accesses or uses the Licensed Content.

2.1 Below are five separate sets of use rights. Only one set of rights apply to you.

a. **If you are a Microsoft IT Academy Program Member:**
   i. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
   ii. For each license you acquire on behalf of an End User or Trainer, you may either:
      1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User who is enrolled in the Authorized Training Session, and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
      2. provide one (1) End User with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
      3. provide one (1) Trainer with the unique redemption code and instructions on how they can access one (1) Trainer Content,

   provided you comply with the following:
   iii. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
   iv. you will ensure each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
   v. you will ensure that each End User provided with the hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
   vi. you will ensure that each Trainer teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,

vii. you will only use qualified Trainers who have in-depth knowledge of and experience with the Microsoft technology that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Authorized Training Sessions,
viii. you will only deliver a maximum of 15 hours of training per week for each Authorized Training Session that uses a MOC title, and
ix. you acknowledge that Trainers that are not MCTs will not have access to all of the trainer resources for the Microsoft Instructor-Led Courseware.

b. **If you are a Microsoft Learning Competency Member**:
i. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
ii. For each license you acquire on behalf of an End User or Trainer, you may either:
    1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Authorized Training Session and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware provided, **or**
    2. provide one (1) End User attending the Authorized Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
    3. you will provide one (1) Trainer with the unique redemption code and instructions on how they can access one (1) Trainer Content,

    provided you comply with the following:
iii. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
iv. you will ensure that each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
v. you will ensure that each End User provided with a hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
vi. you will ensure that each Trainer teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,
vii. you will only use qualified Trainers who hold the applicable Microsoft Certification credential that is the subject of the Microsoft Instructor-Led Courseware being taught for your Authorized Training Sessions,
viii. you will only use qualified MCTs who also hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Authorized Training Sessions using MOC,
ix. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
x. you will only provide access to the Trainer Content to Trainers.

c. **If you are a MPN Member**:

i. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.

ii. For each license you acquire on behalf of an End User or Trainer, you may either:

1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Private Training Session, and only immediately prior to the commencement of the Private Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
2. provide one (1) End User who is attending the Private Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
3. you will provide one (1) Trainer who is teaching the Private Training Session with the unique redemption code and instructions on how they can access one (1) Trainer Content,

provided you comply with the following:

iii. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,

iv. you will ensure that each End User attending an Private Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Private Training Session,

v. you will ensure that each End User provided with a hard copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,

vi. you will ensure that each Trainer teaching an Private Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Private Training Session,

vii. you will only use qualified Trainers who hold the applicable Microsoft Certification credential that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Private Training Sessions,

viii. you will only use qualified MCTs who hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Private Training Sessions using MOC,

ix. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and

x. you will only provide access to the Trainer Content to Trainers.

d. **If you are an End User:**

For each license you acquire, you may use the Microsoft Instructor-Led Courseware solely for your personal training use. If the Microsoft Instructor-Led Courseware is in digital format, you may access the Microsoft Instructor-Led Courseware online using the unique redemption code provided to you by the training provider and install and use one (1) copy of the Microsoft Instructor-Led Courseware on up to three (3) Personal Devices. You may also print one (1) copy of the Microsoft Instructor-Led Courseware. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.

e. **If you are a Trainer.**

i. For each license you acquire, you may install and use one (1) copy of the Trainer Content in the form provided to you on one (1) Personal Device solely to prepare and deliver an Authorized Training Session or Private Training Session, and install one (1) additional copy on another Personal Device as a backup copy, which may be used only to reinstall the Trainer Content. You may not install or use a copy of the Trainer Content on a device you do not own or control. You may also print one (1) copy of the Trainer Content solely to prepare for and deliver an Authorized Training Session or Private Training Session.

ii. You may customize the written portions of the Trainer Content that are logically associated with instruction of a training session in accordance with the most recent version of the MCT agreement. If you elect to exercise the foregoing rights, you agree to comply with the following: (i) customizations may only be used for teaching Authorized Training Sessions and Private Training Sessions, and (ii) all customizations will comply with this agreement. For clarity, any use of "*customize*" refers only to changing the order of slides and content, and/or not using all the slides or content, it does not mean changing or modifying any slide or content.

2.2 **Separation of Components.** The Licensed Content is licensed as a single unit and you may not separate their components and install them on different devices.

2.3 **Redistribution of Licensed Content**. Except as expressly provided in the use rights above, you may not distribute any Licensed Content or any portion thereof (including any permitted modifications) to any third parties without the express written permission of Microsoft.

2.4 **Third Party Notices**. The Licensed Content may include third party code tent that Microsoft, not the third party, licenses to you under this agreement. Notices, if any, for the third party code ntent are included for your information only.

2.5 **Additional Terms**. Some Licensed Content may contain components with additional terms, conditions, and licenses regarding its use. Any non-conflicting terms in those conditions and licenses also apply to your use of that respective component and supplements the terms described in this agreement.

3. LICENSED CONTENT BASED ON PRE-RELEASE TECHNOLOGY. If the Licensed Content's subject matter is based on a pre-release version of Microsoft technology ("**Pre-release**"), then in addition to the other provisions in this agreement, these terms also apply:

a. **Pre-Release Licensed Content.** This Licensed Content subject matter is on the Pre-release version of the Microsoft technology. The technology may not work the way a final version of the technology will and we may change the technology for the final version. We also may not release a final version. Licensed Content based on the final version of the technology may not contain the same information as the Licensed Content based on the Pre-release version. Microsoft is under no obligation to provide you with any further content, including any Licensed Content based on the final version of the technology.

b. **Feedback.** If you agree to give feedback about the Licensed Content to Microsoft, either directly or through its third party designee, you give to Microsoft without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft technology, Microsoft product, or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its technology, technologies, or products to third parties because we include your feedback in them. These rights survive this agreement.

c. **Pre-release Term**. If you are an Microsoft IT Academy Program Member, Microsoft Learning Competency Member, MPN Member or Trainer, you will cease using all copies of the Licensed Content on the Pre-release technology upon (i) the date which Microsoft informs you is the end date for using the Licensed Content on the Pre-release technology, or (ii) sixty (60) days after the commercial release of the technology that is the subject of the Licensed Content, whichever is earliest ("**Pre-release term**"). Upon expiration or termination of the Pre-release term, you will irretrievably delete and destroy all copies of the Licensed Content in your possession or under your control.

4. **SCOPE OF LICENSE**. The Licensed Content is licensed, not sold. This agreement only gives you some rights to use the Licensed Content. Microsoft reserves all other rights. Unless applicable law gives you more rights despite this limitation, you may use the Licensed Content only as expressly permitted in this agreement. In doing so, you must comply with any technical limitations in the Licensed Content that only allows you to use it in certain ways. Except as expressly permitted in this agreement, you may not:
   - access or allow any individual to access the Licensed Content if they have not acquired a valid license for the Licensed Content,
   - alter, remove or obscure any copyright or other protective notices (including watermarks), branding or identifications contained in the Licensed Content,
   - modify or create a derivative work of any Licensed Content,
   - publicly display, or make the Licensed Content available for others to access or use,
   - copy, print, install, sell, publish, transmit, lend, adapt, reuse, link to or post, make available or distribute the Licensed Content to any third party,
   - work around any technical limitations in the Licensed Content, or
   - reverse engineer, decompile, remove or otherwise thwart any protections or disassemble the Licensed Content except and only to the extent that applicable law expressly permits, despite this limitation.

5. **RESERVATION OF RIGHTS AND OWNERSHIP**. Microsoft reserves all rights not expressly granted to you in this agreement. The Licensed Content is protected by copyright and other intellectual property laws and treaties. Microsoft or its suppliers own the title, copyright, and other intellectual property rights in the Licensed Content.

6. **EXPORT RESTRICTIONS**. The Licensed Content is subject to United States export laws and regulations. You must comply with all domestic and international export laws and regulations that apply to the Licensed Content. These laws include restrictions on destinations, end users and end use. For additional information, see www.microsoft.com/exporting.

7. **SUPPORT SERVICES**. Because the Licensed Content is "as is", we may not provide support services for it.

8. **TERMINATION.** Without prejudice to any other rights, Microsoft may terminate this agreement if you fail to comply with the terms and conditions of this agreement. Upon termination of this agreement for any reason, you will immediately stop all use of and delete and destroy all copies of the Licensed Content in your possession or under your control.

9. **LINKS TO THIRD PARTY SITES**. You may link to third party sites through the use of the Licensed Content. The third party sites are not under the control of Microsoft, and Microsoft is not responsible for the contents of any third party sites, any links contained in third party sites, or any changes or updates to third party sites. Microsoft is not responsible for webcasting or any other form of transmission received from any third party sites. Microsoft is providing these links to third party sites to you only as a convenience, and the inclusion of any link does not imply an endorsement by Microsoft of the third party site.

10. **ENTIRE AGREEMENT.** This agreement, and any additional terms for the Trainer Content, updates and supplements are the entire agreement for the Licensed Content, updates and supplements.

11. **APPLICABLE LAW.**
    a. United States. If you acquired the Licensed Content in the United States, Washington state law governs the interpretation of this agreement and applies to claims for breach of it, regardless of conflict of laws principles. The laws of the state where you live govern all other claims, including claims under state consumer protection laws, unfair competition laws, and in tort.

b. Outside the United States. If you acquired the Licensed Content in any other country, the laws of that country apply.

12. LEGAL EFFECT. This agreement describes certain legal rights. You may have other rights under the laws of your country. You may also have rights with respect to the party from whom you acquired the Licensed Content. This agreement does not change your rights under the laws of your country if the laws of your country do not permit it to do so.

13. DISCLAIMER OF WARRANTY. THE LICENSED CONTENT IS LICENSED "AS-IS" AND "AS AVAILABLE." YOU BEAR THE RISK OF USING IT. MICROSOFT AND ITS RESPECTIVE AFFILIATES GIVES NO EXPRESS WARRANTIES, GUARANTEES, OR CONDITIONS. YOU MAY HAVE ADDITIONAL CONSUMER RIGHTS UNDER YOUR LOCAL LAWS WHICH THIS AGREEMENT CANNOT CHANGE. TO THE EXTENT PERMITTED UNDER YOUR LOCAL LAWS, MICROSOFT AND ITS RESPECTIVE AFFILIATES EXCLUDES ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.

14. LIMITATION ON AND EXCLUSION OF REMEDIES AND DAMAGES. YOU CAN RECOVER FROM MICROSOFT, ITS RESPECTIVE AFFILIATES AND ITS SUPPLIERS ONLY DIRECT DAMAGES UP TO US$5.00. YOU CANNOT RECOVER ANY OTHER DAMAGES, INCLUDING CONSEQUENTIAL, LOST PROFITS, SPECIAL, INDIRECT OR INCIDENTAL DAMAGES.

This limitation applies to

- anything related to the Licensed Content, services, content (including code) on third party Internet sites or third-party programs; and
- claims for breach of contract, breach of warranty, guarantee or condition, strict liability, negligence, or other tort to the extent permitted by applicable law.

It also applies even if Microsoft knew or should have known about the possibility of the damages. The above limitation or exclusion may not apply to you because your country may not allow the exclusion or limitation of incidental, consequential or other damages.

Please note: As this Licensed Content is distributed in Quebec, Canada, some of the clauses in this agreement are provided below in French.

Remarque : Ce le contenu sous licence étant distribué au Québec, Canada, certaines des clauses dans ce contrat sont fournies ci-dessous en français.

EXONÉRATION DE GARANTIE. Le contenu sous licence visé par une licence est offert « tel quel ». Toute utilisation de ce contenu sous licence est à votre seule risque et péril. Microsoft n'accorde aucune autre garantie expresse. Vous pouvez bénéficier de droits additionnels en vertu du droit local sur la protection dues consommateurs, que ce contrat ne peut modifier. La ou elles sont permises par le droit locale, les garanties implicites de qualité marchande, d'adéquation à un usage particulier et d'absence de contrefaçon sont exclues.

LIMITATION DES DOMMAGES-INTÉRÊTS ET EXCLUSION DE RESPONSABILITÉ POUR LES DOMMAGES. Vous pouvez obtenir de Microsoft et de ses fournisseurs une indemnisation en cas de dommages directs uniquement à hauteur de 5,00 $ US. Vous ne pouvez prétendre à aucune indemnisation pour les autres dommages, y compris les dommages spéciaux, indirects ou accessoires et pertes de bénéfices.
Cette limitation concerne:

- tout ce qui est relié au le contenu sous licence, aux services ou au contenu (y compris le code) figurant sur des sites Internet tiers ou dans des programmes tiers; et.
- les réclamations au titre de violation de contrat ou de garantie, ou au titre de responsabilité stricte, de négligence ou d'une autre faute dans la limite autorisée par la loi en vigueur.

Elle s'applique également, même si Microsoft connaissait ou devrait connaître l'éventualité d'un tel dommage. Si votre pays n'autorise pas l'exclusion ou la limitation de responsabilité pour les dommages indirects, accessoires ou de quelque nature que ce soit, il se peut que la limitation ou l'exclusion ci-dessus ne s'appliquera pas à votre égard.

EFFET JURIDIQUE. Le présent contrat décrit certains droits juridiques. Vous pourriez avoir d'autres droits prévus par les lois de votre pays. Le présent contrat ne modifie pas les droits que vous confèrent les lois de votre pays si celles-ci ne le permettent pas.

Revised July 2013

Welcome!

Thank you for taking our training! We've worked together with our Microsoft Certified Partners for Learning Solutions and our Microsoft IT Academies to bring you a world-class learning experience—whether you're a professional looking to advance your skills or a student preparing for a career in IT.

- **Microsoft Certified Trainers and Instructors**—Your instructor is a technical and instructional expert who meets ongoing certification requirements. And, if instructors are delivering training at one of our Certified Partners for Learning Solutions, they are also evaluated throughout the year by students and by Microsoft.

- **Certification Exam Benefits**—After training, consider taking a Microsoft Certification exam. Microsoft Certifications validate your skills on Microsoft technologies and can help differentiate you when finding a job or boosting your career. In fact, independent research by IDC concluded that 75% of managers believe certifications are important to team performance[1]. Ask your instructor about Microsoft Certification exam promotions and discounts that may be available to you.

- **Customer Satisfaction Guarantee**—Our Certified Partners for Learning Solutions offer a satisfaction guarantee and we hold them accountable for it. At the end of class, please complete an evaluation of today's experience. We value your feedback!

We wish you a great learning experience and ongoing success in your career!

Sincerely,

Microsoft Learning
www.microsoft.com/learning

Microsoft | Learning

[1] *IDC,* Value of Certification: Team Certification and Organizational Performance, *November 2006*

Acknowledgments

Microsoft Learning would like to acknowledge and thank the following for their contribution towards developing this title. Their effort at various stages in the development has ensured that you have a good classroom experience.

Graeme Malcolm – Lead Content Developer

Graeme Malcolm is a Microsoft SQL Server subject matter expert and professional content developer at Content Master—a division of CM Group Ltd. As a Microsoft Certified Trainer, Graeme has delivered training courses on SQL Server since version 4.2; as an author, Graeme has written numerous books, articles, and training courses on SQL Server; and as a consultant, Graeme has designed and implemented business solutions based on SQL Server for customers all over the world.

Chris Testa-O'Neill – Technical Reviewer

Chris Testa-O'Neill is a SQL Server Microsoft Most Valuable Professional (MVP), Microsoft Certified Trainer and independent DBA and SQL Server Business Intelligence consultant at Claribi. He is a regular speaker on the international circuit and runs the Manchester (UK) SQL Server User Group, SQLBits and co-founder of SQLRelay. Chris is a Microsoft Certified Trainer (MCT), MCDBA, MCTS, MCITP, MCSA and MCSE in SQL Server. He can be contacted at chris@claribi.com or @ctesta_oneill.

Contents

About This Course

This section provides you with a brief description of the course, audience, suggested prerequisites, and course objectives.

Course Description

This course describes how to implement a data warehouse platform to support a BI solution. Students will learn how to create a data warehouse with Microsoft® SQL Server® 2014, implement ETL with SQL Server Integration Services, and validate and cleanse data with SQL Server Data Quality Services and SQL Server Master Data Services.

Audience

This course is intended for database professionals who need to create and support a data warehousing solution. Primary responsibilities include:

- Implementing a data warehouse.
- Developing SSIS packages for data extraction, transformation, and loading.
- Enforcing data integrity by using Master Data Services.
- Cleansing data by using Data Quality Services.

Student Prerequisites

This course requires that you meet the following prerequisites:

- At least 2 years' experience of working with relational databases, including:
  - Designing a normalized database.
  - Creating tables and relationships.
  - Querying with Transact-SQL.
- Some exposure to basic programming constructs (such as looping and branching).
- An awareness of key business priorities such as revenue, profitability, and financial accounting is desirable.

Course Objectives

After completing this course, students will be able to:

- Describe data warehouse concepts and architecture considerations.
- Select an appropriate hardware platform for a data warehouse.
- Design and implement a data warehouse.
- Implement Data Flow in an SSIS Package.
- Implement Control Flow in an SSIS Package.
- Debug and Troubleshoot SSIS packages.
- Implement an ETL solution that supports incremental data extraction.
- Implement an ETL solution that supports incremental data loading.

- Implement data cleansing by using Microsoft Data Quality Services.
- Implement Master Data Services to enforce data integrity.
- Extend SSIS with custom scripts and components.
- Deploy and Configure SSIS packages.
- Describe how BI solutions can consume data from the data warehouse.

Course Outline

This section provides an outline of the course:

Module 1, "Introduction to Data Warehousing"

Module 2, "Planning Data Warehouse Infrastructure"

Module 3, "Designing and Implementing a Data Warehouse"

Module 4, "Creating an ETL Solution with SSIS"

Module 5, "Implementing Control Flow in an SSIS Package"

Module 6, "Debugging and Troubleshooting SSIS Packages"

Module 7, "Implementing a Data Extraction Solution"

Module 8, "Loading Data into a Data Warehouse"

Module 9, "Enforcing Data Quality"

Module 10, "Master Data Services"

Module 11, "Extending SQL Server Integration Services"

Module 12, "Deploying and Configuring SSIS Packages"

Module 13, "Consuming Data in a Data Warehouse"

Course Materials

The following materials are included with your kit:

- ***Course Handbook*** A succinct classroom learning guide that provides all the critical technical information in a crisp, tightly-focused format, which is just right for an effective in-class learning experience.
  - **Lessons**: Guide you through the learning objectives and provide the key points that are critical to the success of the in-class learning experience.
  - **Labs**: Provide a real-world, hands-on platform for you to apply the knowledge and skills learned in the module.
  - **Module Reviews and Takeaways**: Provide improved on-the-job reference material to boost knowledge and skills retention.
  - **Lab Answer Keys**: Provide step-by-step lab solution guidance at your fingertips when it's needed.

Course Companion Content on the *http://www.microsoft.com/learning/companionmoc/* Site*:* *Searchable, easy-to-navigate digital content with integrated premium on-line resources designed to supplement the Course Handbook.*

- **Modules**: Include companion content, such as questions and answers, detailed demo steps and additional reading links, for each lesson. Additionally, they include Lab Review questions and answers and Module Reviews and Takeaways sections, which contain the review questions and answers, best practices, common issues and troubleshooting tips with answers, and real-world issues and scenarios with answers.
- **Resources**: Include well-categorized additional resources that give you immediate access to the most up-to-date premium content on TechNet, MSDN®, Microsoft Press®.

Student Course files on the *http://www.microsoft.com/learning/companionmoc/* Site: Includes the Allfiles.exe, a self-extracting executable file that contains all the files required for the labs and demonstrations.

- ***Course evaluation*** At the end of the course, you will have the opportunity to complete an online evaluation to provide feedback on the course, training facility, and instructor.
  - To provide additional comments or feedback on the course, send e-mail to support@mscourseware.com. To inquire about the Microsoft Certification Program, send e-mail to mcphelp@microsoft.com.

Virtual Machine Environment

This section provides the information for setting up the classroom environment to support the business scenario of the course.

Virtual Machine Configuration

In this course, you will use Microsoft Hyper-V to perform the labs.

The following table shows the role of each virtual machine used in this course:

Virtual machine	Role
20463C-MIA-SQL	Database Server
20463C -MIA-DC	Domain Controller

Software Configuration

The following software is installed on each VM:

- Windows Server® 2012
- Microsoft SQL Server 2014
- Microsoft SharePoint Server 2013
- Microsoft Office 2013
- Microsoft Visual Studio 2012

Course Files

There are files associated with the labs in this course. The lab files are located in the folder D:\Labfiles\LabXX on the 20463C-MIA-SQL virtual machine.

Classroom Setup

Each classroom computer will have the same virtual machine configured in the same way.

To ensure a satisfactory student experience, Microsoft Learning requires a minimum equipment configuration for trainer and student computers in all Microsoft Certified Partner for Learning Solutions (CPLS) classrooms in which Official Microsoft Learning Product courseware are taught.

Course Hardware Level 6+

- Processor: Intel Virtualization Technology (Intel VT) or AMD Virtualization (AMD-V)
- Hard Disk: Dual 120 GB hard disks 7200 RM SATA or better (Striped)
- RAM: 12GB or higher. 16 GB or more is recommended for this course.
- DVD/CD: DVD drive
- Network adapter with Internet connectivity
- Video Adapter/Monitor: 17-inch Super VGA (SVGA)
- Microsoft Mouse or compatible pointing device
- Sound card with amplified speakers

In addition, the instructor computer must be connected to a projection display device that supports SVGA 1024 x 768 pixels, 16 bit colors.

Note: For the best classroom experience, a computer with solid state disks (SSDs) is recommended. For optimal performance, adapt the instructions below to install the 20463C-MIA-SQL virtual machine on a different physical disk than the other virtual machines to reduce disk contention.

Module 1

Introduction to Data Warehousing

Contents:

Module Overview

Data warehousing is a solution that organizations can use to centralize business data for reporting and analysis. Implementing a data warehouse solution can provide a business or other organization with significant benefits, including:

- Comprehensive and accurate reporting of key business information.
- A centralized source of business data for analysis and decision-making.
- The foundation for an enterprise Business Intelligence (BI) solution.

This module provides an introduction to the key components of a data warehousing solution and the high-level considerations you must take into account when you embark on a data warehousing project.

Objectives

After completing this module, you will be able to:

- Describe the key elements of a data warehousing solution.
- Describe the key considerations for a data warehousing project.

Lesson 1
Overview of Data Warehousing

Data warehousing is a well-established technique for centralizing business data for reporting and analysis. Although the specific details of individual solutions can vary, there are some common elements in most data warehousing implementations. Familiarity with these elements will enable you to better plan and build an effective data warehousing solution.

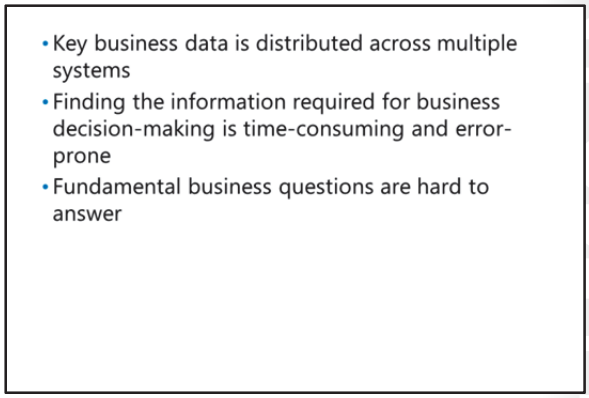
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the business problem addressed by data warehouses.
- Define a data warehouse.
- Describe the commonly-used data warehouse architectures.
- Identify the components of a data warehousing solution.
- Describe a high-level approach to implementing a data warehousing project.
- Identify the roles involved in a data warehousing project.
- Describe the components and features of Microsoft® SQL Server® and other Microsoft products you can use in a data warehousing solution.

The Business Problem

Running a business effectively can present a significant challenge, particularly as it grows or is affected by trends in its target market or the global economy. To be successful, a business must adapt to changing conditions, which requires individuals within the organization to make good strategic and tactical decisions. However, the following problems can often make effective business decision-making difficult:

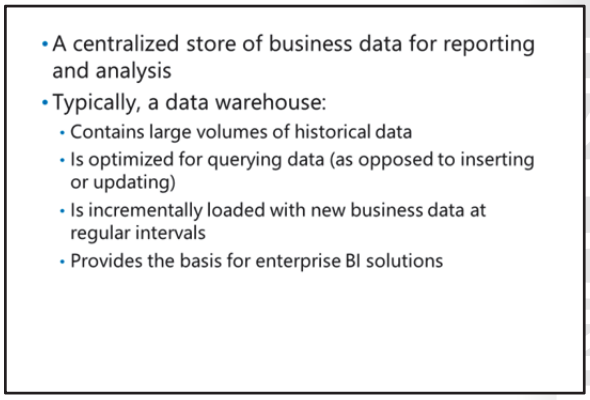


- Key business data is distributed across multiple systems. This makes it hard to collate all the information necessary for a particular business decision.
- Finding the information required for business decision-making is time-consuming and error-prone. The need to gather and reconcile data from multiple sources results in slow, inefficient decision-making processes that can be further undermined by inconsistencies between duplicate, contradictory sources of the same information.
- Fundamental business questions are hard to answer. Most business decisions require a knowledge of fundamental facts, such as "How many customers do we have?" or "Which products do we sell most often?" Although these may seem like simple questions, the distribution of data throughout multiple systems in a typical organization can make them difficult, or even impossible, to answer.

By resolving these problems, it is possible to make effective decisions that will help the business to be more successful, both at the strategic, executive level and during day-to-day operations.

What Is a Data Warehouse?

A data warehouse provides a solution to the problem of distributed data that prevents effective business decision-making. There are many definitions for the term "data warehouse," and disagreements over specific implementation details. It is generally agreed, however, that a data warehouse is a centralized store of business data that can be used for reporting and analysis to inform key decisions.



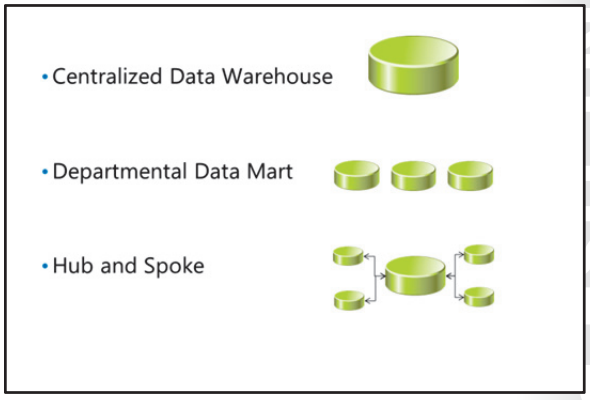
Typically, a data warehouse:

- Contains a large volume of data that relates to historical business transactions.
- Is optimized for read operations that support querying the data. This is in contrast to a typical Online Transaction Processing (OLTP) database that is designed to support data insert, as well as update and delete operations.
- Is loaded with new or updated data at regular intervals.
- Provides the basis for enterprise BI applications.

Data Warehouse Architectures

There are many ways that you can implement a data warehouse solution in an organization. Some common approaches include:

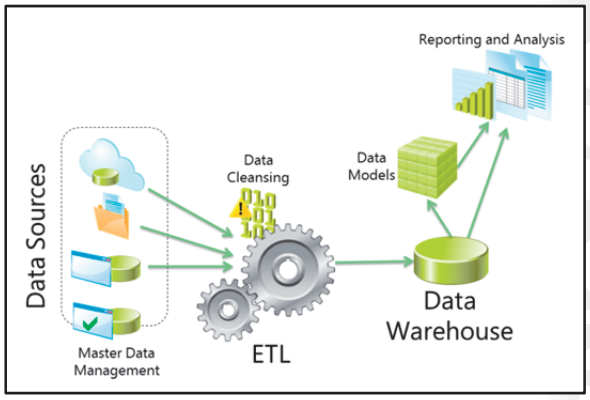
- Creating a single, central enterprise data warehouse for all business units.
- Creating small, departmental data marts for individual business units.
- Creating a hub-and-spoke architecture that synchronizes a central enterprise data warehouse with departmental data marts containing a subset of the data warehouse data.



The right architecture for a given business might be one of these, or a combination of various elements from all three approaches.

Components of a Data Warehousing Solution

A data warehousing solution usually consists of the following elements:



- **Data sources**. Sources of business data for the data warehouse, often including OLTP application databases and data exported from proprietary systems, such as accounting applications.
- **An Extract, Transform, and Load (ETL) process**. A workflow for accessing data in the data sources, modifying it to conform to the data model for the data warehouse, and loading it into the data warehouse.
- **Data staging areas**. Intermediary locations where the data to be transferred to the data warehouse is stored. It is prepared here for import and to synchronize loading into the data warehouse.
- **A data warehouse**. A relational database designed to provide high-performance querying of historical business data for reporting and analysis.

Many data warehousing solutions also include:

- **Data cleansing and deduplication**. A solution for resolving data quality issues before it is loaded into the data warehouse.
- **Master Data Management (MDM)**. A solution that provides an authoritative data definition for business entities used by multiple systems across the organization.

Data Warehousing Projects

A data warehousing project has much in common with any other IT implementation, so it is possible to apply most commonly-used methodologies, such as Agile or Microsoft Solutions Framework (MSF). However, a data warehousing project often requires a deeper understanding of the key business objectives and metrics that are used to drive decision-making than other software development or infrastructure.

1. Start by identifying the business questions that the data warehousing solution must answer
2. Determine the data that is required to answer these questions
3. Identify data sources for the required data
4. Assess the value of each question to key business objectives versus the feasibility of answering it from the available data

- For large enterprise-level projects, an incremental approach can be effective:
  - Break the project down into multiple sub-projects
  - Each sub-project deals with a particular subject area in the data warehouse

A high-level approach to implementing a data warehousing project usually includes the following steps:

1. Work with business stakeholders and information workers to determine the questions to which the data warehouse must provide answers. They may include questions such as:
   - What was the total sales revenue for each geographic territory in a given month?
   - What are our most profitable products or services?
   - Are business costs growing or reducing over time?
   - Which sales employees are meeting their sales targets?

2. Determine the data required to answer these questions. It is normal to think of this data in terms of "dimensions" and "facts." Facts contain the numerical measures required to aggregate data so you can answer the business questions identified in step 1. For example, to determine sales revenue, you may need the sales amount for each individual transaction. Dimensions represent the different aspects of the business by which you want to aggregate the measures. For example, to determine sales revenue for each territory in a given month, you may need a geographic dimension, so you can aggregate sales by territory, and a time dimension enabling you to aggregate sales by month. Fact and dimensional modeling is covered in more detail in Module 3, *Designing and Implementing a Data Warehouse*.
3. Identify data sources containing the data required to answer the business questions. These are commonly relational databases used by existing line-of-business applications, but they can also include:
   - Flat files or XML documents that have been extracted from proprietary systems.
   - Data in Microsoft SharePoint® lists.
   - Commercially available data that has been purchased from a data supplier such as the Microsoft Windows Azure™ Marketplace.
4. Determine the priority of each business question based on:
   - The importance of answering the question in relation to driving key business objectives.
   - The feasibility of answering the question from the data available.

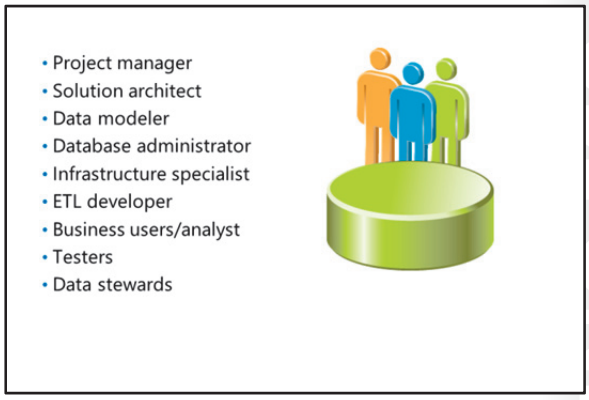
A common approach to prioritizing the business questions you will address in the data warehousing solution, is to work with key business stakeholders and plot each question on a quadrant-based matrix like the one shown below. The position of the questions in the matrix helps you to agree the scope of the data warehousing project.

Business importance of the question	**High importance, low feasibility**	**High importance, high feasibility**
	Low importance, low feasibility	Low importance, high feasibility
	Feasibility of answering the question	

If a large number of questions fall into the high importance, high feasibility category, you may want to consider taking an incremental approach to the project in which you break down the challenge into a number of sub-projects. Each sub-project tackles the problem of implementing the data warehouse schema, ETL solution, and data quality procedures for a specific area of the business, starting with the highest-priority questions. If you adopt this incremental approach, take care to create an overall design for dimension and fact tables in early iterations of the solution so that subsequent additions can reuse them.

Data Warehousing Project Roles

A data warehousing project typically involves several roles, including:

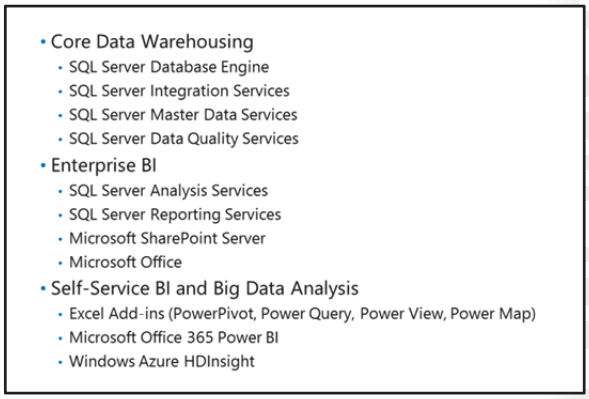


- **A project manager**. Coordinates project tasks and schedules, ensuring that the project is completed on time and within budget.
- **A solution architect**. Has overall responsibility for the technical design of the data warehousing solution.
- **A data modeler**. Designs the data warehouse schema.
- **A database administrator**. Designs the physical architecture and configuration of the data warehouse database. In addition, database administrators with responsibility for data sources used in the data warehousing solution, must be involved in the project to provide access to the data sources required by the ETL process.
- **An infrastructure specialist**. Implements the server and network infrastructure for the data warehousing solution.
- **An ETL developer**. Builds the ETL workflow for the data warehousing solution.
- **Business users**. Provide requirements and help to prioritize the questions that the data warehousing solution will answer. Often, the team includes a business analyst as a full-time member, helping to interpret the questions and ensuring that the solution design meets the users' needs.
- **Testers**. Verify the business and operational functionality of the solution as it is developed.
- **Data stewards for each key subject area in the data warehousing solution**. Determine data quality rules and validate data before it enters the data warehouse. Data stewards are sometimes referred to as data governors.

In addition to ensuring the appropriate assignment of these roles, you should also consider the importance of executive-level sponsorship of the data warehousing project. It is significantly more likely to succeed if a high-profile executive sponsor is seen to actively support the creation of the data warehousing solution.

SQL Server as a Data Warehousing Platform

SQL Server includes components and features that you can use to implement various architectural elements of a data warehousing solution, including:



- **The SQL Server database engine**. A highly scalable Relational Database Management System (RDBMS) on which you can implement a data warehouse.

  SQL Server Enterprise includes features that

make it particularly appropriate for data warehousing solutions. One feature is optimization of star join queries, which significantly enhances the performance of queries in a typical data warehouse schema. Also, column store indexes can significantly enhance the performance of data warehouse workloads.

- **SQL Server Integration Services**. A comprehensive and extensible platform for creating ETL solutions, including support for a wide range of data sources and numerous built-in data flow transformations and control flow tasks for common ETL requirements.
- **SQL Server Master Data Services**. A master data management solution that enables organizations to create authoritative data definitions for key business entities, and ensure data consistency across multiple applications and systems.
- **SQL Server Data Quality Services**. A knowledge-based solution for validating, cleansing, and deduplicating data.

These components of the Microsoft data platform are the focus of this course.

In addition, you can use some SQL Server components and other Microsoft products to build an enterprise BI solution that significantly extends the value of your data warehouse. These components and products include:

- **SQL Server Analysis Services**. A service for creating multidimensional and tabular analytical data models for so-called "slice and dice" analysis, and for implementing data mining models you can use to identify trends and patterns in your data.
- **SQL Server Reporting Services**. A solution for creating and distributing reports in a variety of formats for online viewing or printing.
- **Microsoft® SharePoint® Server**. A web-based portal through which information workers can consume reports and other BI deliverables.
- **Microsoft Office®**. In particular Microsoft Excel®.

To learn more about using data platform components to implement enterprise BI solutions, you should attend course 20466C: *Implementing Data Models and Reports with Microsoft SQL Server*.

If you need to move beyond a traditional managed enterprise BI solution, you can empower business users to perform self-service analysis and reporting, and engage in Big Data analysis with the following data platform components:

- **Excel Add-Ins**. Add-ins such as PowerPivot, Power Query, Power View, and Power Map help extend Excel to make it a powerful tool for self-service data modeling, analysis, and reporting.
- **Microsoft Office 365® Power BI**. Power BI is a cloud-based BI collaboration platform that enables business users to share queries, datasets, and reports and to access data visualizations from virtually any device.
- **Microsoft Windows Azure™ HDInsight™**. As analytical data grows in volume and complexity, new techniques for processing it have evolved. HDInsight is a cloud-based implementation of Hadoop from Microsoft, and provides a platform for Map/Reduce processing of Big Data.

To learn more about these components of the Microsoft data platform, you should attend course 20467C: *Designing Self-Service Business Intelligence and Big Data Solutions*.

Lesson 2
Considerations for a Data Warehouse Solution

Before starting a data warehousing project, you need to understand the considerations that will help you create a data warehousing solution that addresses your specific needs and constraints.

This lesson describes some of the key considerations for planning a data warehousing solution.

Lesson Objectives

After completing this lesson, you will be able to describe considerations for:

- Designing a data warehouse database.
- Data sources.
- Designing an ETL process.
- Implementing data quality and master data management.

Data Warehouse Database and Storage

A data warehouse is a relational database that is optimized for reading data for analysis and reporting. When you are planning a data warehouse, you should take the following considerations into account:

- Database Schema
- Hardware
- High Availability and Disaster Recovery
- Security

Database Schema

The logical schema of a data warehouse is typically designed to denormalize data into a structure that minimizes the number of join operations required in the queries used to retrieve and aggregate data. A common approach is to design a star schema in which numerical measures are stored in fact tables that have foreign keys to multiple dimension tables containing the business entities by which the measures can be aggregated. Before designing your data warehouse, you must know which dimensions your business users employ when aggregating data, which measures need to be analyzed and at what granularity, and which facts include those measures. You must also carefully plan the keys that will be used to link facts to dimensions, and consider whether your data warehouse must support the use of dimensions that change over time. For example, handling dimension records for customers who change their address.

You must also consider the physical implementation of the database, because this will affect the performance and manageability of the data warehouse. It is common to use table partitioning to distribute large fact data across multiple file groups, each on a different physical disk. This can increase query performance and enables you to implement a file group-based backup strategy that can help reduce downtime in the event of a single-disk failure. You should also consider the appropriate indexing strategy for your data, and whether to use data compression when storing it.

Note: Designing a data warehouse schema is covered in more detail in Module 3, *Designing and Implementing a Data Warehouse*.

Hardware

Your choice of hardware for your data warehouse solution can make a significant difference to the performance, manageability, and cost of your data warehouse. The hardware considerations include:

- Query processing requirements, including anticipated peak memory and CPU utilization.
- Storage volume and disk input/output requirements.
- Network connectivity and bandwidth.
- Component redundancy for high availability.

You can build your own data warehouse solution by purchasing and assembling individual components, using a pre-tested reference architecture, or purchasing a hardware appliance that includes preconfigured components in a ready-to-use package. Factors that influence your choice of hardware include:

- Budget.
- Existing enterprise agreements with hardware vendors.
- Time to construct the solution.
- Hardware assembly and configuration expertise.

 **Note:** Hardware for data warehousing solutions is discussed in more detail in Module 2, *Data Warehouse Hardware*.

High Availability and Disaster Recovery

A data warehouse can very quickly become a business-critical part of your overall application infrastructure, so it is essential to consider how you will ensure its availability. SQL Server includes support for several high-availability techniques, including database mirroring and server clustering. You must assess these technologies and choose the best one for your individual solution, based on:

- Failover time requirements.
- Hardware requirements and cost.
- Configuration and management complexity.

In addition to a server-level high-availability solution, you must also consider redundancy at the individual component level for network interfaces and storage arrays.

The most robust high-availability solution cannot protect your data warehouse from every eventuality, so you must also plan a suitable disaster recovery solution that includes a comprehensive backup strategy. This should take into account:

- The volume of data in the data warehouse.
- The frequency of changes to data in the data warehouse.
- The effect of the backup process on data warehouse performance.
- The time to recover the database in the event of a failure.

Security

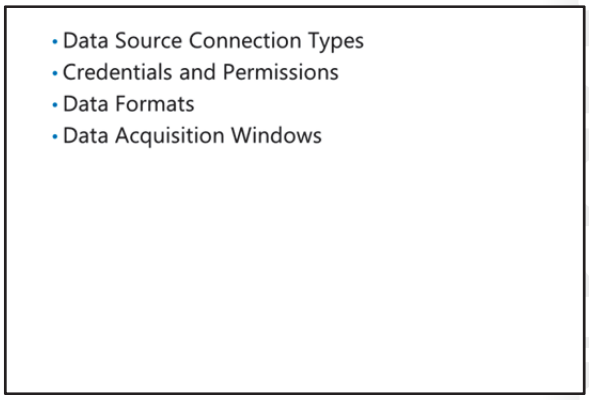
Your data warehouse contains a huge volume of data that is typically commercially sensitive. You may also want to provide access to some data by all users, but restrict access to other data for a subset of users.

Considerations for securing your data warehouse include:

- The authentication mechanisms that you must support to provide access to the data warehouse.
- The permissions that the various users who access the data warehouse will require.
- The connections over which data is accessed.
- The physical security of the database and backup media.

Data Sources

You must identify the sources that provide the data for your data warehouse, and consider the following factors when planning your solution:



Data Source Connection Types

Your data warehouse may require data from a variety of sources. For each source, you must consider how the ETL process can connect and extract the required data. In many cases, your data sources will be relational databases for which you can use an OLEDB or ODBC provider. However, some data sources may use proprietary storage that requires a bespoke provider or for which no provider exists. In this case, you must develop a custom provider or determine whether it is possible to export data from the data source in a format that the ETL process can easily consume, such as XML or comma-delimited text.

Credentials and Permissions

Most data sources require secure access with user authentication and, potentially, individual permissions on the data. You must work with the owners of the data sources used in your data warehousing solution to establish:

- Credentials that your ETL process can use to access the data source.
- The required permissions to access the data used by the data warehouse.

Data Formats

A data source may store data in a different format. Your solution must take into account issues arising from this, including:

- Conversion of data from one data type to another, for example, extracting numeric values from a text file.
- Truncation of data when copying to a destination with a limited data length.
- Date and time formats used in data sources.
- Numeric formats, scales, and precisions.
- Support for Unicode characters.

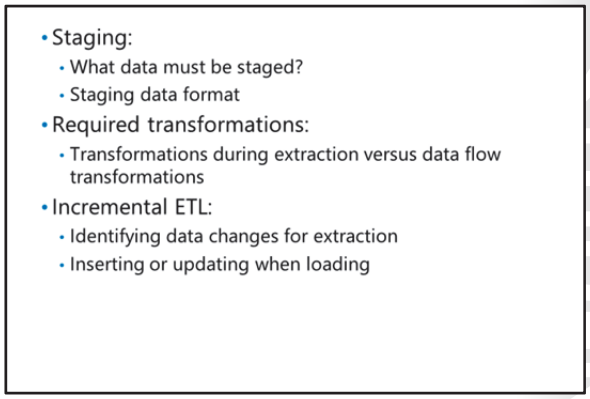
Data Acquisition Windows

Depending on the workload patterns of the business, there may be times when individual data sources are not available or the usage level is such that the additional overhead of a data extraction is undesirable. When planning a data warehousing solution, you must work with each data source owner to determine appropriate data acquisition windows based on:

- The workload pattern of the data source, and its resource utilization and capacity levels.
- The volume of data to be extracted, and the time that process takes.
- The frequency with which you need to update the data warehouse with fresh data.
- If applicable, the time zones in which business users are accessing the data.

Extract, Transform, and Load Processes

A significant part in the creation of a data warehouse solution is the implementation of an ETL process. When you design an ETL process for a data warehousing solution, you must consider the following factors:



Staging

In some solutions, you can transfer data directly from data sources to the data warehouse without any intermediary staging. In many cases, however, you should consider staging data to:

- Synchronize a data warehouse refresh that includes source data extracted during multiple data acquisition windows.
- Perform data validation, cleansing, and deduplication operations on the data before it is loaded into the data warehouse.
- Perform transformations on the data that cannot be performed during the data extraction or data flow processes.

If a staging area is required in your solution, you must decide on a format for the staged data. Possible formats include:

- A relational database.
- Text or XML files.
- Raw files (binary files in a proprietary format of the ETL platform being used).

The decision on format is based on several factors including:

- The need to access and modify the staged data.
- The time taken to store and read the staging data.

Finally, if a relational database is used as the staging area, you must decide where this database will reside. Possible choices include:

- A dedicated staging server.
- A dedicated SQL Server instance on the data warehouse server.
- A dedicated staging database in the same instance of SQL Server as the data warehouse.
- A collection of staging tables (perhaps in a dedicated schema) in the data warehouse database.

Factors you should consider when deciding the location of the staging database include:

- Server hardware requirements and cost.
- The time taken to transfer data across network connections.
- The use of Transact-SQL loading techniques that perform better when the staging data and data warehouse are co-located on the same SQL Server instance.
- The server resource overheads associated with the staging and data warehouse load processes.

Required Transformations

Most ETL processes require that the data being extracted from data sources is modified to match the data warehouse schema. When planning an ETL process for a data warehousing solution, you must examine the source data and destination schema, and identify what transformations are required. You must then determine the optimal place within the ETL process to perform these transformations. Choices for implementing data transformations include:

- **During the data extraction**. For example, by concatenating two fields in a SQL Server data source into a single field in the Transact-SQL query used to extract the data.
- **In the data flow**. For example, by using a Derived Column data transformation task in a SQL Server Integration Services data flow.
- **In the staging area**. For example, by using a Transact-SQL query to apply default values to null fields in a staging table.

Factors affecting the choice of data transformation technique include:

- **The performance overhead of the transformation**. Typically, it is best to use the approach with the least performance overhead. Set-based operations that are performed in Transact-SQL queries usually function better than row-based transformations being applied in a data flow.
- **The level of support for querying and updating in the data source or staging area**. In cases where you are extracting data from a comma-delimited file and staging it in a raw file, your options are limited to performing row-by-row transformations in the data flow.
- **Dependencies on data required for the transformation**. For example, you might need to look up a value in one data source to obtain additional data from another. In this case, you must perform the data transformation in a location where both data sources are accessible.
- **The complexity of the logic involved in the transformation**. In some cases, a transformation may require multiple steps and branches depending on the presence or value of specific data fields. In this case, it is often easier to apply the transformation by combining several steps in a data flow than it is to create a Transact-SQL statement to perform the transformation.

Incremental ETL

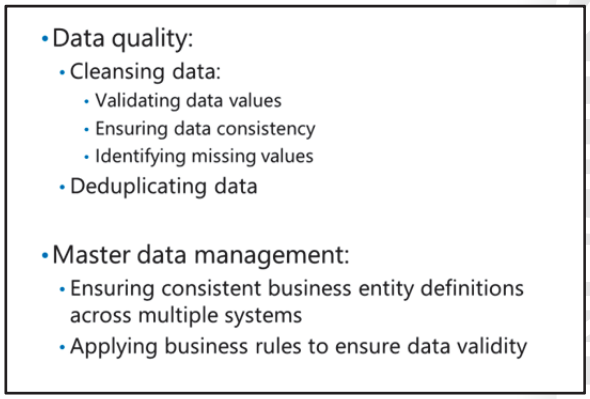
After the initial load of the data warehouse, you will usually need to incrementally add new or updated source data. When you plan your data warehousing solution, you must consider the following factors that relate to incremental ETL:

- How will you identify new or modified records in the data sources?
- Do you need to delete records in the data warehouse when corresponding records in the data sources are removed? If so, will you physically delete the records, or simply mark them as inactive (often referred to as a logical delete)?
- How will you determine whether a record that is to be loaded into the data warehouse should be new or an update to an existing one?
- Are there records in the data warehouse for which historical values must be preserved by creating a new version instead of updating the existing one?

Note: Managing data changes in an incremental ETL process is discussed in more detail in Module 7, *Implementing an Incremental ETL Process*.

Data Quality and Master Data Management

The value of a data warehouse is largely determined by the quality of the data it contains. When planning a data warehousing project, therefore, you should determine how you will ensure data quality. You should consider using a master data management solution.



Data Quality

To validate and enforce the quality of data in your data warehouse, it is recommended that business users with knowledge of each subject area addressed by the data warehouse become data stewards. A data steward is responsible for:

- Building and maintaining a knowledge base that identifies common data errors and corrections.
- Validating data against the knowledge base.
- Ensuring that consistent values are used for data attributes where multiple forms of the value may be considered valid. For example, ensuring that a Country field always uses the value "United States" when referring to America, even though "USA," "The U.S." and "America" are also valid values.
- Identifying and correcting missing data values.
- Identifying and consolidating duplicate data entities, such as records for "Robert Smith" and "Bob Smith" that both refer to the same physical customer.

You can use SQL Server Data Quality Services to provide a data quality solution that helps the data steward to perform these tasks.

Note: SQL Server Data Quality Services is discussed in more detail in Module 9, *Enforcing Data Quality*.

Master Data Management

It is common for large organizations to have multiple business applications, and in many cases, these systems perform tasks that are related to the same business entities. For example, an organization may have an e-commerce application enabling customers to purchase products, and a separate inventory

management system that also stores product data. A record representing a particular product may exist in both systems. In this scenario, it can be useful to implement a master data management system that provides an authoritative definition of each business entity (in this example, a particular product) that you can use across multiple applications to ensure consistency.

Master data management is especially important in a data warehousing scenario because it ensures that data conforms to the agreed definition for the business entities to be included in any analysis and reporting solutions that it must support.

You can use SQL Server Master Data Services to implement a master data management solution.

Note: SQL Server Master Data Services is discussed in more detail in Module 10, *Using Master Data Services*.

Lab: Exploring a Data Warehousing Solution

Scenario

The labs in this course are based on Adventure Works Cycles, a fictional company that manufactures and sells bicycles and accessories to customers worldwide. Adventure Works sells direct through an e-commerce website, as well as an international reseller network. Throughout this course, you will develop a data warehousing solution for Adventure Works Cycles, including a data warehouse, an ETL process to extract data from source systems and populate the data warehouse, a data quality solution, and a master data management solution.

The lab for this module provides a high-level overview of the solution you will create later. Use this lab to become familiar with the various elements of the data warehousing solution that you will learn to build in subsequent modules. Don't worry if you do not understand the specific details of how each component of the solution has been built, as you will explore these in greater depth later in the course.

Objectives

After completing this lab, you will be able to:

- Describe the data sources in the Adventure Works data warehousing scenario.
- Describe the ETL process in the Adventure Works data warehousing scenario.
- Describe the data warehouse in the Adventure Works data warehousing scenario.

Estimated Time: 30 minutes

Virtual Machine: 20463C-MIA-SQL

User name: ADVENTUREWORKS\Student

Password: Pa$$w0rd

Exercise 1: Exploring Data Sources

Scenario

Adventure Works uses various software applications to manage different aspects of the business, and each has its own data store. Specifically these are:

- Internet sales processed through an e-commerce web application.
- Reseller sales processed by sales representatives, who use a specific application. Sales employee details are stored in a separate human resources system.
- Reseller payments are processed by an accounting application.
- Products are managed in a product catalog and inventory system.

This distribution of data has made it difficult for users to answer key questions about the overall performance of the business.

In this exercise, you will examine some of the Adventure Works data sources that will be used in the data warehousing solution.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. View the Solution Architecture
3. View the Internet Sales Data Source
4. View the Reseller Sales Data Source
5. View the Products Data Source
6. View the Human Resources Data Source
7. View the Accounts Data Source
8. View the Staging Database

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.
2. Run Setup.cmd in the D:\Labfiles\Lab01\Starter folder as Administrator.

▶ Task 2: View the Solution Architecture

1. Use Paint to view the Adventure Works DW Solution.jpg JPEG image in the D:\Labfiles\Lab01\Starter folder.
2. Note the data sources in the solution architecture.

Note: In addition to the data sources that you will examine in this lab, the diagram includes a Microsoft® SQL Server® Master Data Services model for product data and a SQL Server Data Quality Services task to cleanse data as it is staged. These elements form part of the complete solution for the lab scenario in this course, but they are not present in this lab.

▶ Task 3: View the Internet Sales Data Source

1. Use Microsoft® SQL Server® Management Studio to open the **View Internet Sales.sql** query file in the D:\Labfiles\Lab01\Starter folder. Use Windows authentication to connect to the **MIA-SQL** instance of SQL Server.
2. Execute the query and examine the results. Note that this data source contains data about customers and the orders they have placed through the e-commerce web application.

▶ Task 4: View the Reseller Sales Data Source

1. Use SQL Server Management Studio to open the **View Reseller Sales.sql** query file in the D:\Labfiles\Lab01\Starter folder.
2. Execute the query and examine the results. Note that this data source contains data about resellers and the orders they have placed through Adventure Works reseller account managers.

▶ Task 5: View the Products Data Source

1. Use SQL Server Management Studio to open the **View Products.sql** query file in the D:\Labfiles\Lab01\Starter folder.
2. Execute the query and examine the results. Note that this database contains data about products that Adventure Works sells, organized into categories and subcategories.

▶ Task 6: View the Human Resources Data Source

1. Use SQL Server Management Studio to open the **View Employees.sql** query file in the D:\Labfiles\Lab01\Starter folder.
2. Execute the query and examine the results. Note that this database contains data about employees, including sales representatives.

▶ Task 7: View the Accounts Data Source

1. Examine the comma-delimited text files in the D:\Accounts folder by opening them in Microsoft Excel®. Note that they contain details of payments made by resellers.
2. Close all files when you have finished reviewing them, without saving any changes.

▶ Task 8: View the Staging Database

1. In SQL Server Management Studio, in the Object Explorer pane, examine the tables in the **Staging** database in the MIA-SQL instance of SQL Server. Ensure you examine the **Staging** database, not the DQS_STAGING_DATA database.
2. Note that all tables other than **dbo.ExtractLog** in this database are empty.

Results: After this exercise, you should have viewed data in the InternetSales, ResellerSales, Human Resources, and Products SQL Server databases, viewed payments data in comma-delimited files, and viewed an empty staging database.

Exercise 2: Exploring an ETL Process

Scenario

Now that you are familiar with the data sources in the Adventure Works data warehousing solution, you will examine the ETL process used to stage the data, and then load it into the data warehouse.

Adventure Works uses a solution based on SQL Server Integration Services to perform this ETL process.

The main tasks for this exercise are as follows:

1. View the Solution Architecture
2. Run the ETL Staging Process
3. View the Staged Data
4. Run the ETL Data Warehouse Load Process

▶ Task 1: View the Solution Architecture

1. Use Paint to view **Adventure Works DW Solution.jpg** in the D:\Labfiles\Lab01\Starter folder.
2. Note that the ETL solution consists of two main phases: a process to extract data from the various data sources and load it into a staging database, and another to load the data in the staging database into the data warehouse. In this exercise, you will observe these ETL processes as they run.

▶ Task 2: Run the ETL Staging Process

1. Use Visual Studio to open the **AdventureWorksETL.sln** solution file in the D:\Labfiles\Lab01\Starter folder.
2. In Solution Explorer, view the SSIS packages that this solution contains, and then double-click **Stage Data.dtsx** to open it in the designer.
3. View the control flow of the **Stage Data.dtsx** package, and then run the package by clicking **Start Debugging** on the **Debug** menu. The package will run other packages to perform the tasks in the control flow. This may take several minutes.
4. When the package has finished running, a message box will be displayed, although this may be hidden by the Visual Studio window. Look for a new icon on the taskbar, and then click it to bring the message box to the front. After viewing this message box, stop the package by clicking **Stop Debugging** on the **Debug** menu.

▶ Task 3: View the Staged Data

1. Use SQL Server Management Studio to view the **Staging** database in the MIA-SQL instance of SQL Server. Take care to view the **Staging** database, not the DQS_STAGING_DATA database.
2. Note the following tables now contain data:
   - dbo.Customers
   - dbo.EmployeeInserts
   - dbo.InternetSales
   - dbo.Payments
   - dbo.Resellers
   - dbo.ResellerSales

▶ Task 4: Run the ETL Data Warehouse Load Process

1. In Visual Studio, in Solution Explorer, view the SSIS packages that the AdventureWorksETL.sln solution contains, and then double-click **Load DW.dtsx** to open it in the designer.
2. View the control flow of the Load DW.dtsx package, and then run the package by clicking **Start Debugging** on the **Debug** menu. The package will run other packages to perform the tasks in the control flow. This may take several minutes.
3. When the package has finished running, a message box will be displayed, although it may be hidden by the Visual Studio window. Look for a new icon on the taskbar, and then click it to bring the message box to the front. After viewing this message box, stop the package by clicking **Stop Debugging** on the **Debug** menu.
4. Close Visual Studio when you have finished.

Results: After this exercise, you should have viewed and run the SQL Server Integration Services packages that perform the ETL process for the Adventure Works data warehousing solution.

Exercise 3: Exploring a Data Warehouse

Scenario

Now that you have explored the ETL process that is used to populate the Adventure Works data warehouse, you can explore the data warehouse to see how it enables business users to view key information.

The main tasks for this exercise are as follows:

1. View the Solution Architecture

2. Query the Data Warehouse

▶ Task 1: View the Solution Architecture

1. Use Paint to view **Adventure Works DW Solution.jpg** in the D:\Labfiles\Lab01\Starter folder.
2. Note that the data warehouse provides a central data source for business reporting and analysis. Then close Paint.

▶ Task 2: Query the Data Warehouse

1. Use SQL Server Management Studio to open the **Query DW.sql** query file in the D:\Labfiles\Lab01\Starter folder.
2. Execute the query and examine the results. Note that the data warehouse contains the data necessary to view key metrics across multiple aspects of the business.

Results: After this exercise, you should have successfully retrieved business information from the data warehouse.

Module Review and Takeaways

In this module, you have learned about the key elements in a data warehousing solution.

Review Question(s)

Question: Why might you consider including a staging area in your ETL solution?

Question: What options might you consider for performing data transformations in an ETL solution?

Question: Why would you assign the data steward role to a business user rather than a database technology specialist?

Module 2

Planning Data Warehouse Infrastructure

Contents:

Module Overview

The server and hardware infrastructure for a Business Intelligence (BI) solution is a key consideration in any BI project. You must balance the performance and scalability gains you can achieve by maximizing hardware specifications and distributing the elements of your BI solution across multiple servers against hardware and software licensing costs, and implementation complexity.

This module discusses considerations for selecting hardware and distributing SQL Server facilities across servers.

Objectives

After completing this module, you will be able to:

- Describe key considerations for BI infrastructure.
- Plan data warehouse infrastructure.

Lesson 1
Considerations for Data Warehouse Infrastructure

Planning the infrastructure for a SQL Server-based data warehousing solution requires an understanding of how the various SQL Server components work together, and how their typical workloads use hardware resources. This lesson considers all the components in a BI solution, including the data warehouse, and servers for Extract, Transform and Load (ETL) workloads, Analysis Services, and Reporting Services.

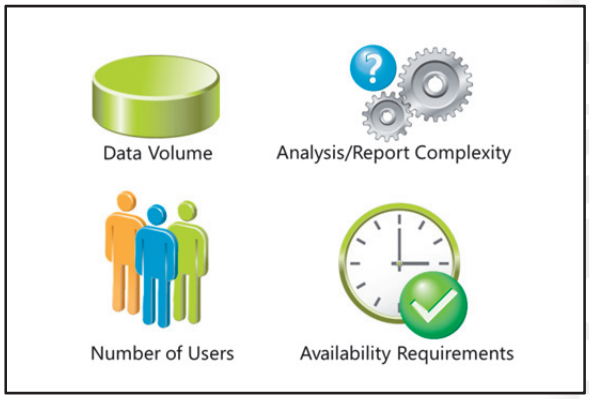
Lesson Objectives

After completing this lesson, you will be able to describe:

- Considerations for characterizing the size of a BI solution.
- Key features of the workloads associated with SQL Server components.
- High-level topology designs for BI solutions.
- Scale-out options for services in a BI solution.
- High-availability options for services in a BI solution.

System Sizing Considerations

Each data warehouse solution has unique characteristics that depend on the business requirements it is designed to address. There is no such thing as a "standard" data warehouse solution. However, when planning the infrastructure for a data warehouse, it can be useful to start by classifying the type of solution to be implemented into one of three generic sizes – small, medium, or large. The factors that determine the size of a BI solution are:



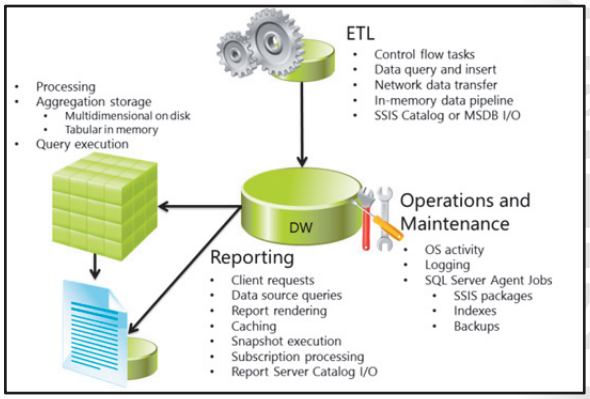
- **Data volume**. This is the amount of data that the data warehouse must store, and the size and frequency of incremental loads of new data. The primary consideration is the number of rows in fact tables, but you must also allow for dimension data, indexes, and data models that are stored on disk.
- **Analysis and Reporting Complexity**. This includes the number, complexity, and predictability of the queries that will be used to analyze the data or produce reports. Typically, BI solutions must support a mix of the following query types:
  - *Simple*. Relatively straightforward SELECT statements.
  - *Medium*. Repeatedly executed queries that include aggregations or many joins.
  - *Complex*. Unpredictable queries with complex aggregations, joins, and calculations.
- **Number of Users**. This is the total number of information workers who will access the system, and how many of them will do so concurrently.
- **Availability Requirements**. These include when the system will need to be used, and what planned or unplanned downtime the business can tolerate.

Although it is difficult to be precise when categorizing a solution, the following table suggests some typical examples of the characteristics of small, medium, and large BI systems.

	Small	Medium	Large
Data Volume	100s of GBs to 1 TB	1 to 10 TB	10 TB to 100s of TBs
Analysis and Reporting Complexity	• Over 50% simple • 30% medium • Less than 10% complex	• 50% simple • 30-35% medium • 10-15% complex	• 30-35% simple • 40% medium • 20-25% complex
Number of Users	100 total 10 to 20 concurrent	1,000 total 100 to 200 concurrent	1,000s of concurrent users
Availability Requirements	Business hours	1 hour of downtime per night	24/7 operations

Data Warehouse Workloads

In addition to the size categorization of the solution your infrastructure needs to support, it is useful to understand the workload types of that might occur in a BI solution. It is important to assess the total impact of all workloads on hardware resource utilization and identify any potential for contention between workloads with similar requirements.



Extract, Transform, and Load (ETL) Workloads

The ETL subsystem of the BI solution performs the following workloads:

- **Control flow tasks**. SQL Server Integration Services (SSIS) packages often include control flow tasks that require CPU processing time, memory, disk I/O, and network I/O. The specific resource requirements for control flow tasks can vary significantly so, if your ETL solution includes a substantial number of them, you should monitor resource utilization on a test system to better understand the workload profile.
- **Data query and insert**. Fundamentally, ETL involves querying data sources and inserting and updating data into staging and data warehouse tables. This incurs I/O and query processing on data sources, the staging databases, and the data warehouse, especially if data loads require rebuilding indexes or partition management.
- **Network data transfer**. Typically, ETL processes transfer large volumes of data from one server to another. This incurs network I/O and can require significant network bandwidth.

- **In-memory data pipeline**. Data flow tasks in an SSIS package use an in-memory pipeline architecture to process transformations, and that places a demand on system memory resources. On systems where there is contention between the SQL Server database engine and SSIS for memory resources, data flow buffers might need to spill to disk, which reduces data flow performance and incurs disk I/O.
- **SSIS catalog or MSDB database activity**. SSIS packages deployed in project mode are stored in an SSIS Catalog database. Alternatively, packages deployed in package mode are stored either in the MSDB database or on the file system. Whichever deployment model is used, the ETL process must access the package storage to load packages and their configuration. If the SSIS catalog or MSDB database is used, and it is located on a SQL Server instance hosting other databases used in the BI solution (such as the data warehouse, staging database, or Report Server catalog), there may be contention for database server resources.

Data Model Workloads

The SQL Server Analysis Services (SSAS) data models in a BI system require the following workloads:

- **Processing**. Data models contain aggregated values for the measures in the data warehouse. They must be processed to load the required data from the data warehouse tables into the model and perform the necessary aggregation calculations. When data in the data warehouse is refreshed, the data models must be partially or fully processed again to reflect the new and updated data.
- **Aggregation storage**. Data models must store the aggregated data in a structure that is optimized for analytical queries. Typically, multidimensional models are stored on disk with some data cached in memory for performance reasons. Tabular models are usually stored completely in memory, so sufficient resources must be available.
- **Query execution**. When users perform analytical queries against a data model, it must process the query and generate results. This requires CPU processing time, memory, and potentially disk I/O.

Reporting Workloads

Although reporting is often performed by client applications directly against the data warehouse or data models, many BI solutions include SQL Server Reporting Services (SSRS). An SSRS reporting solution typically involves the following workloads:

- **Handling client requests**. The report server must listen for client requests submitted to Reporting Services over HTTP and process them.
- **Data source queries**. Reports are based on datasets that must be retrieved by querying data sources. Typically, data sources for the reports in a BI solution are SSAS data models or the data warehouse, so report execution incurs query processing overheads.
- **Report rendering**. After the report data has been retrieved, SSRS must render it into the required format using the report definition language (RDL) and the specific rendering extension. Depending on the report's size, format, and complexity, rendering can incur substantial CPU and memory resources.
- **Caching**. To reduce query processing and rendering overheads, SSRS can cache datasets and reports in the report server temporary database. Datasets and reports can be cached on first use or you can use scheduled jobs to pre-cache objects at a regular interval.
- **Snapshot execution**. In addition to caching reports, you can create persisted report snapshots at a scheduled interval and store them in the report server database.
- **Subscription processing**. You can configure SSRS to execute and deliver reports to file shares or email addresses on a scheduled basis.

- **Report Server catalog I/O**. Report definitions and resources, such as images, are stored in the report server catalog, and database I/O tasks are required to retrieve these when needed. Database I/O is required to retrieve cached reports and datasets from the temporary database, and to recover report snapshots from the catalog database. This database access may compete for resources with other databases hosted in the same SQL Server instance.

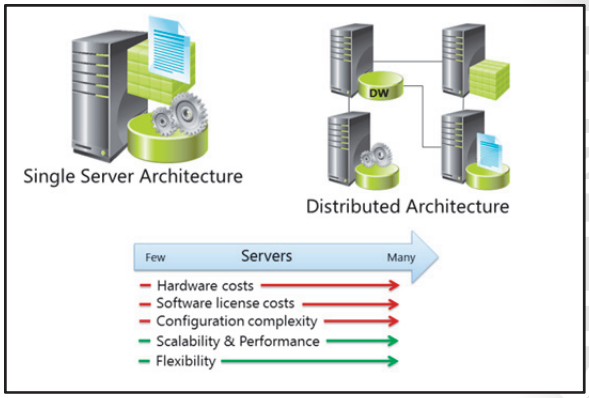
Operations and Maintenance

In addition to the fundamental BI activity, a BI solution must support ongoing system operations and maintenance activity, such as:

- Operating system activity.
- Logging activity.
- SQL Server Agent jobs, including:
  - Execution of SSIS packages for ETL processes.
  - Index and statistics maintenance in databases, including the data warehouse.
  - Database backup tasks.

Typical Server Topologies for a BI Solution

SQL Server provides a versatile service architecture enabling you to choose a topology that best fits your needs. Selecting the right topology for your solution usually involves balancing cost and complexity against the need for high scalability and flexibility.



Single Server BI Architecture

The simplest architecture for a SQL Server-based BI solution is to install all its elements on a single server. Depending on the business requirements of the BI solution, the components on the server typically include:

- **SQL Server Database Engine**. Used to store the data warehouse, staging database, Reporting Services databases, and SSIS Catalog database. Additionally, the SQL Server Agent may be used to automate SSIS package execution and other operations by creating jobs, and schedules stored in the MSDB.
- **SQL Server Integration Services**. Used to execute packages that encapsulate ETL tasks and data flows to extract data from source systems into the staging database, and then load it into the data warehouse.
- **SQL Server Analysis Services**. Used to provide analytical data models and data mining functionality. Depending on business requirements, two instances on Analysis Services may be required – one for multidimensional models and data mining, the other for tabular models.
- **SQL Server Reporting Services**. Used to provide report publishing and execution services. Reporting Services in native mode consists of a web service application, a web-based management user interface, and two SQL Server databases.

Depending on business requirements, you may also choose to install SQL Server Data Quality Services on the server to support data cleansing and matching capabilities for staged data before it is loaded into the data warehouse.

Note: If SharePoint Server is required, you can deploy the SharePoint farm and SQL Server integration components for Reporting Services and PowerPivot on the BI server. This architecture is not recommended for BI solutions that require significant scalability or performance.

A single server architecture is suitable for test and development environments, and can be used in production locations with minimal data volumes and scalability requirements.

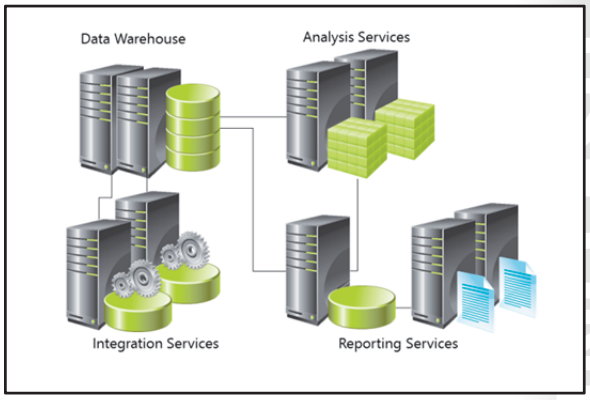
Distributed BI Architecture

If your BI solution requires even moderate levels of scalability, it will benefit from expanding beyond a single server architecture and distributing the BI workload across multiple servers. Typical approaches to distributing SQL Server components in a BI solution include:

- **Creating a dedicated report server**. In many BI solutions, Analysis Services provides a data model that contains most (or even all) the data in the data warehouse, and all reporting is performed against the data model. In these scenarios, there is little database activity in the data warehouse other than during ETL loading and data model processing (loading data from the data warehouse tables into the model and aggregating it). The server workloads that compete for resources most of the time are Analysis Services and Reporting Services, so you can increase scalability and performance by moving the reporting workload to a separate server. You can install the Reporting Services database on either server. Leaving it on the data warehouse server keeps all the data engine elements on a single server but can result in I/O workloads competing for disk resources. To install the reporting services database on the report server, the database engine must be installed on both servers but the result means a cleaner separation of workloads. In extremely large enterprise solutions, with intensive reporting requirements, you can add a third server as a dedicated host for the Reporting Services database.
- **Separating data warehouse and ETL workloads from analytical and reporting workloads**. If the data warehouse will be refreshed with new data frequently, or if it must support direct query access in addition to processing data models, the database I/O activity might compete with Analysis Services for disk, CPU, and memory resources. To prevent this, you can deploy Analysis Services on a separate server. Depending on analytical and reporting workloads, you might choose to co-locate Analysis Services and Reporting Services on the same server, or use a dedicated one for each. If you need to support tabular and multidimensional or data mining models, and a single Analysis Services server is not adequate to support both workloads, you could consider using separate servers for the different types of Analysis Services model.
- **Using a dedicated ETL server**. If your ETL process requires frequent data extractions and loads, or involves particularly resource-intensive transformations, overall performance might benefit from moving Integration Services and the SSIS Catalog database to a dedicated server. Depending on the specific transformation and load operations that your ETL process requires, you can choose to locate the staging database on the ETL server or the data warehouse server. Alternatively, you could use a two-phase staging approach where extracted data is staged on the ETL server for transformation and cleansing, and then loaded into staging tables on the data warehouse server before being inserted into the data warehouse tables.

When designing a distributed architecture, the key goal is to eliminate contention for hardware resources. You then gain the greatest benefits by identifying workloads with similar hardware utilization profiles and separating them.

Scaling-out a BI Solution



A distributed architecture increases scalability by separating the discrete workloads in a BI solution across multiple servers. To support the highest level of scalability, you can use a scale-out architecture to share the same workload across multiple servers.

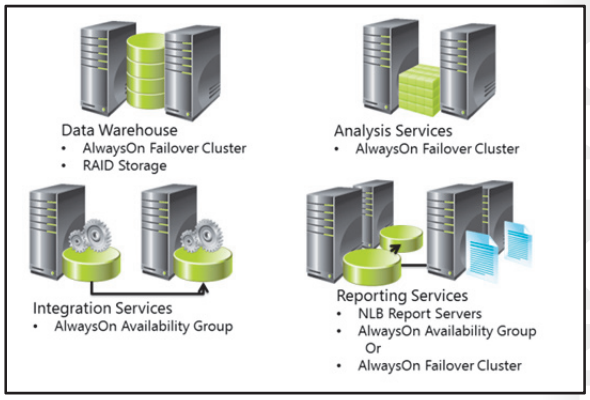
You can use a scale-out architecture for the following components:

- **SQL Server Reporting Services**. Install the Reporting Services database on a single database server, and then install the Reporting Services report server service on multiple servers that all connect to the same Reporting Services database. This approach separates the report execution and rendering workloads from the database workloads required to store and retrieve report definitions, cached datasets and reports, and snapshots.
- **SQL Server Analysis Services**. Create a read-only copy of a multidimensional database and connect to it from multiple Analysis Services query servers. To accomplish this, an SSAS server processes the cubes in the database. The database is then detached, copied to a standby location, and reattached so it can be used by applications requiring write-back capabilities. The copied database is then attached in read-only mode to multiple SSAS instances, providing query services to clients. Client requests can be distributed across query servers using a load-balancing middle tier, or you can assign specific subsets of the client population to dedicated query servers.
- **The data warehouse**. You can scale out an extremely large data warehouse in several ways. Typically, this is done by partitioning the data across multiple database servers and using middle-tier logic to direct queries to the appropriate server instance. SQL Server Parallel Data Warehouse edition, which is provided in pre-configured enterprise data warehouse appliances, uses a massively parallel processing (MPP) shared-nothing architecture to scale out data warehouse queries across multiple independent compute and storage nodes.
- **SQL Server Integration Services**. Although it is not a pure scale-out technique, you can use multiple SSIS servers to perform a subset of the ETL processes in parallel. This approach requires extensive custom logic to ensure that all tasks are completed, and should be considered only if your ETL requirements cannot be met through a scale-up approach in which you add hardware resources to a single SSIS server.

Additional Reading: For more information about designing a scale-out solution for Reporting Services, review the content and links in the SQLCAT *Reporting Services Scale-Out Architecture* technical notes at http://sqlcat.com/sqlcat/b/technicalnotes/archive/2008/06/05/reporting-services-scale-out-architecture.aspx. For more information about using read-only databases to implement a scale-out solution for Analysis Services, go to *Scale-Out Querying for Analysis Services with Read-Only Databases* at http://msdn.microsoft.com/en-us/library/ff795582(v=SQL.100).aspx.

Planning for High Availability

If one or more aspects of the BI solution requires high availability, you can design a solution that uses redundant servers with failover capabilities. SQL Server 2014 includes a number of high availability technologies that you can use to protect critical services. These include:



- **AlwaysOn Failover Clustering**. Server-level protection that uses Windows Server cluster services to create a virtual server with multiple redundant failover nodes.
- **AlwaysOn Availability Groups**. Database-level protection that combines Windows Server cluster services with synchronization of database transactions across multiple SQL Server instances.
- **Log Shipping**. Database-level protection that copies transaction logs from a primary server to a secondary server, where they can be applied to a secondary copy of the database.

Planning High Availability for the Data Warehouse

The data warehouse is fundamentally a SQL Server database and, theoretically, it can be protected by using any of the high-availability technologies provided by SQL Server. However, considering the large volume of data typical of data warehouses and the fact that most activity consists of data reads or non-logged bulk load operations, AlwaysOn Failover Clustering is the most appropriate technology to consider.

Additionally, considering the importance of the data warehouse data, the database files should be stored on a Redundant Array of Independent Disks (RAID) that provides protection in the case of a disk failure.

Planning High Availability for Analysis Services

Analysis Services supports AlwaysOn Failover Clustering, so you should use this technology if you require high availability for an Analysis Services instance. If Analysis Services is to be installed on the same server as the database engine instance for the data warehouse, both components can be installed in a single cluster operation.

Planning High Availability for Integration Services

Integration Services is not cluster-aware, and does not support failover from one cluster node to another. When SSIS packages are deployed in project mode, you can use an AlwaysOn Availability Group to protect the SSIS Catalog database. However, there are some additional considerations for using AlwaysOn Availability groups with the SSIS catalog:

- On failover, you must re-encrypt the master database key in the new primary server before any packages that include encrypted sensitive data can be executed.
- Your SSIS packages must be able to recover the ETL process as a whole to a consistent state if unplanned failover occurs. This means that you must include cleanup and failover logic in your packages.
- If you need to apply an update that modifies the SSIS catalog schema, you should remove the SSIS Catalog database from the availability group, update it, and then re-establish the availability group.

 **Additional Reading:** For more information about using AlwaysOn Availability Groups with the SSIS catalog, go to *SSIS with AlwaysOn* at http://blogs.msdn.com/b/mattm/archive/2012/09/19/ssis-with-alwayson.aspx.

Planning High Availability for Reporting Services

As described earlier in this lesson, Reporting Services can be distributed across a multi-tier architecture in which a report server processes requests and executes reports. A separate database tier hosts the report server catalog and temporary database. In this configuration, you can use Network Load Balancing (NLB) to distribute requests across report servers, and remove failed report servers from the NLB cluster to maintain a highly available solution for managing report requests. To protect the database tier, you can use either AlwaysOn Failover Clustering or an AlwaysOn Availability Group.

Lesson 2
Planning Data Warehouse Hardware

The data warehouse is the foundation for a BI solution. You should consider a number of recommendations from Microsoft and its hardware partners when planning a data warehouse system. Data warehousing is substantially different from other database workloads, and the conventional database design approach for optimizing hardware to support the highest possible number of I/O Operations Per Second (IOPS) is not always appropriate.

This lesson introduces Microsoft SQL Server Fast Track Data Warehouse reference architectures, and explains how some of the Fast Track design principles can be applied when planning data warehouse hardware.

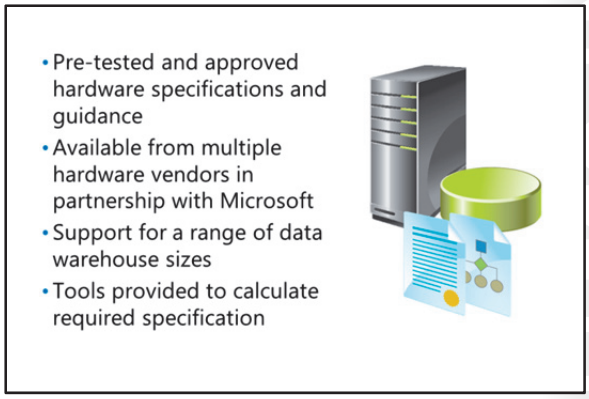
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the key features of SQL Server Fast Track Data Warehouse reference architectures.
- Explain how a core-balanced system architecture supports data warehousing workloads.
- Determine processor and memory requirements for a data warehouse.
- Determine storage requirements for a data warehouse.
- Describe considerations for choosing a storage solution.

SQL Server Fast Track Reference Architectures

Data warehouse design is a specialist skill that many organizations may not have in-house, so the BI team must work with consultants and hardware vendors to determine the required hardware and its configuration. This adds cost and time to the project, and does not always guarantee that the resulting infrastructure is appropriate for the data warehousing workload.



To help organizations overcome these challenges, Microsoft has partnered with multiple hardware vendors to create Fast Track Data Warehouse reference architectures that reduce the time and effort it takes to specify a data warehouse system. These reference architectures, together with hardware appliances that can be purchased as out-of-the-box solutions, make it easier and quicker to implement an effective data warehousing solution.

Pre-Tested Specifications and Guidance

Fast Track Data Warehouse reference architectures are designed by Microsoft SQL Server specialists and consultants from hardware partners. The reference architectures include specific hardware components that can be combined in a prescribed configuration to create a SQL Server-based data warehouse system. The architectures have been extensively tested with real-world data warehouse workloads based on thousands of customer case studies. They are certified to deliver specific performance levels based on the size of your data warehouse solution.

Multi-Vendor Support

Microsoft has partnered with multiple vendors to create pre-tested system designs that use commodity hardware. If your organization has an existing supplier relationship with one of the Fast Track hardware vendors, you can easily specify a system based on components from that supplier. You can use the support and consulting services offered by the hardware vendor to create a data warehouse system based on a proven design.

Support for a Range of Data Warehouse Sizes

The Fast Track program is not a "one size fits all" solution. It includes multiple reference architectures for small to large data warehouse solutions, with flexibility within each design to support your specific requirements. You can evaluate the size of your required solution by using the principles described in the previous lesson, and then identify the reference architecture that best suits your needs.

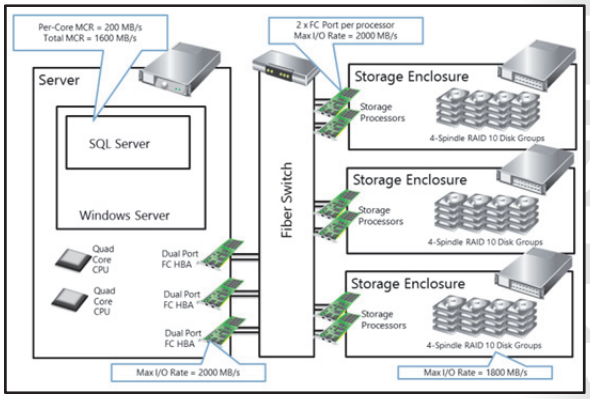
System Specification Tools

Microsoft provides a generic Excel workbook that you can use to determine the Fast Track configuration you require. Additionally, many Fast Track hardware vendors offer tools that make it easy to create a system specification that meets your needs and includes the right mix of components for your solution.

Additional Reading: For more information about Fast Track Data Warehouse reference architectures, go to *An Introduction to Fast Track Data Warehouse Architectures* at http://msdn.microsoft.com/en-us/library/dd459146(v=SQL.100).aspx and *Fast Track Data Warehouse Reference Guide for SQL Server 2012* at http://msdn.microsoft.com/en-us/library/hh918452.aspx.

Core-Balanced System Architecture

SQL Server Fast Track Data Warehouse reference architectures are based on a core-balanced system design that reflects the performance characteristics of a typical data warehouse workload. Even if you don't plan on using one of the published Fast Track Data Warehouse reference architectures, you will benefit from understanding the principles on which they are designed and applying them to your own system specification.



The core-balanced system architecture is based on the fact that most data warehouse workloads need to transfer large amounts of data, usually accessed by sequential read operations, across multiple system components, from where it is stored to the applications that request it. Each component through which the data is transferred is a potential bottleneck that will limit the overall performance of the system. The data can only flow to the requesting application at the rate of the slowest component. Any components that can operate at a higher rate are underutilized, which unbalances the system and can represent significant wasted cost.

The diagram on the slide shows a balanced system in which the I/O rates of each component are reasonably similar. This diagram represents a large-scale data warehousing system in which data is kept in a storage area network with fiber channel connectivity and multiple storage enclosures, each containing multiple disk arrays. The same principles apply to a smaller architecture.

The I/O rate of hardware components, such as hard disks, array storage processors, and fiber channel host bus adapters (HBAs) can be established through careful testing and monitoring by using tools like SQLIO. Many manufacturers, particularly those that participate in the Fast Track program, publish the maximum rates. However, the initial figure that you need to start designing a data warehouse system is the maximum consumption rate (MCR) of a single processor core, combined with the SQL Server database engine. After the MCR for the CPU core architecture you intend to use has been established, you can determine the number of processors required to support your workload and the storage architecture needed to balance the system.

Note that MCR is specific to a combination of a CPU and motherboard, and SQL Server. It is not a measure of pure processing speed or an indication of the performance you can expect for all solution queries. Instead, MCR is a system-specific benchmark measure of maximum throughput per core for a data warehouse query workload. Calculating MCR requires executing a query that can be satisfied from cache while limiting execution to a single core, and reviewing the execution statistics to determine the number of megabytes of data processed per second.

Demonstration: Calculating Maximum Consumption Rate

This demonstration shows how to use a benchmark query to retrieve I/O statistics that can be used to calculate a system's MCR.

Demonstration Steps

Create tables for benchmark queries

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Demofiles\Mod02 folder, run **Setup.cmd** as Administrator.
3. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine using Windows authentication.
4. In SQL Server Management Studio, open the **Create BenchmarkDB.sql** query file from the D:\Demofiles\Mod02 folder.
5. Click **Execute**, and wait for query execution to complete. This query creates a database containing two tables, one with a clustered index and one without. Both tables contain a substantial number of rows.

Execute a query to retrieve I/O statistics

1. In SQL Server Management Studio, open the **Measure MCR.sql** query file from the D:\Demofiles\Mod02 folder.
2. Click **Execute**, and wait for query execution to complete. The queries retrieve an aggregated value from each table, and are performed twice. This ensures that on the second execution (for which statistics are shown), the data is in cache so the I/O statistics do not include disk reads. Note that the MAXDOP=1 clause ensures that only a single core is used to process the query.

Calculate MCR from the I/O statistics

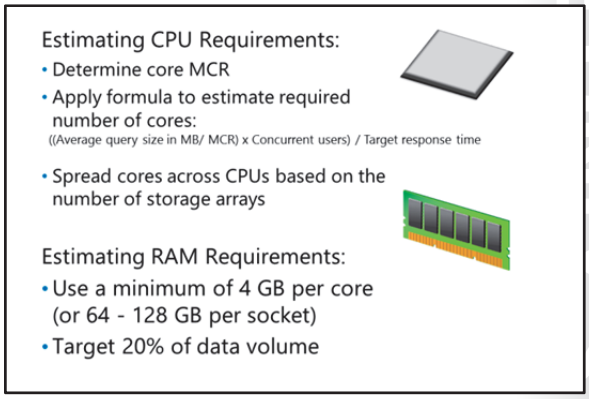
1. In the results pane, click the **Messages** tab. This shows the statistics for the queries.
2. Add the **logical reads** value for the two queries together, and then divide the result by two to find the average.
3. Add the **CPU time** value for the two queries together, and then divide the result by two to find the average. Divide the result by 100 to convert it to seconds.

4. Calculate MCR by using the following formula:

```
(average logical reads / average CPU time) * 8 / 1024
```

Determining Processor and Memory Requirements

After determining the MCR for the CPU cores you intend to use, you can start to estimate the number of cores that will be required to support your anticipated query workload. You can also make an initial assessment of memory requirements.



Estimating CPU Requirements

MCR indicates the amount of data that can be processed by a single core in a second. To determine the number of cores required, you must apply this rate to the following factors:

- The amount of data returned by an average query.
- The number of concurrent users you need to support.
- The target response time for a query.

The specific formula to apply is:

```
((Average query size in MB ÷ MCR) x Concurrent users) ÷ Target response time
```

For example, suppose the MCR of the CPU core you intend to use is 200 MBps. If an average query is expected to return 18,000 MB, the anticipated number of concurrent users is 10, and each query must respond within 60 seconds, the calculation to find the number of cores required is:

```
((18000 ÷ 200) x 10) ÷ 60
```

This results in a requirement for 15 cores, which should be rounded up to 16 because no CPU architecture includes exactly 15 cores.

Now you know the number of cores required, you can make an initial determination of the number of processors. For example, to provide 16 cores using quad-core processors, you would need four processors. Alternatively, if dual-core processors are used, eight CPUs would be required. Remember that you need to balance the number of CPUs to closely match the number of storage arrays that will be used, which in turn may depend on the volume of data your data warehouse must support.

Estimating RAM Requirements

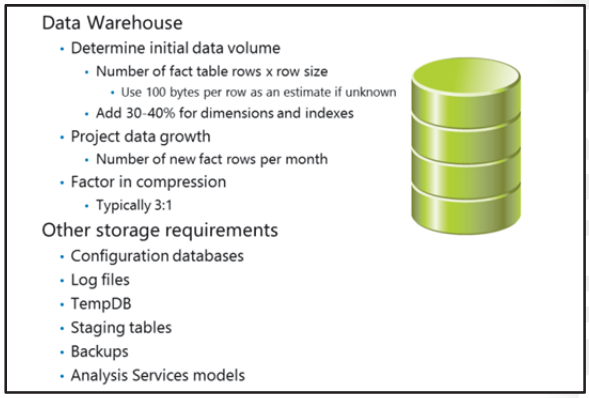
Calculating the amount of RAM required is difficult because memory can be utilized by many workloads to increase overall performance. You should generally consider a minimum figure for a small to medium sized data warehouse system to be 4 GB per core, or 64 to 128 GB per CPU socket. If you intend to use column store indexes or support tabular data models on the data warehouse server, you should favor the higher end of these estimates.

Another way to estimate memory requirements is to consider that, in an average data warehouse workload, users regularly need to access approximately 20 percent of the data stored. For example, in a data warehouse that stores five years of sales records, users most frequently query the data for the most

recent year. Having enough memory to maintain approximately 20 percent of the data in cache will significantly enhance performance.

Determining Storage Requirements

Before you can fully determine CPU, memory, and storage hardware requirements, you must assess the volume of data that the system must support.



Estimating Data Volumes for the Data Warehouse

Most data warehouses consist predominantly of fact data. Determining the volume of fact data the data warehouse must store is the most significant factor in assessing overall storage requirements.

1. Estimate Initial Fact Data

To start estimating data warehouse data volumes, determine the number of fact rows that will be initially loaded into the data warehouse and multiply that by the average size of a fact row. If you don't know the average fact row size at this stage, use a conservative estimate such as 100 bytes per row. For example, a data warehouse that will contain 200,000,000 fact rows, each 100 bytes in length, will have an initial fact data volume of approximately 20 GB.

2. Allow for Indexes and Dimensions

After estimating the initial fact data, add approximately 30 to 40 percent to allow for indexes and dimensions. So, to continue the example with 20 GB of fact data, you would add approximately 8 GB (40 percent of 20 GB), giving an initial data volume of approximately 28 GB.

3. Project Fact Data Growth

Most data warehouses are refreshed with new data on a regular basis. To be sure that your storage solution will support the data warehouse in the future (say, three years from now), you must factor in the anticipated incremental data that will be loaded. For example, suppose the fact data in our data warehouse represents individual items that have been ordered in sales transactions, and the company typically sells 5,000,000 items a month, you can expect to load 5,000,000 rows (each containing 100 bytes of data), or approximately 500 MB each month. That equates to a data growth rate of 6 GB per year, so in three years, the example data warehouse would need to support the initial 28 GB of data plus another 18 GB (6 GB per year multiplied by three years), giving a total of 46 GB.

4. Factor In Compression

Finally, you should plan to compress the data in your data warehouse. Typically, SQL Server provides a compression factor of approximately 3:1, so the 46 GB of data should compress to approximately 15.5 GB on disk.

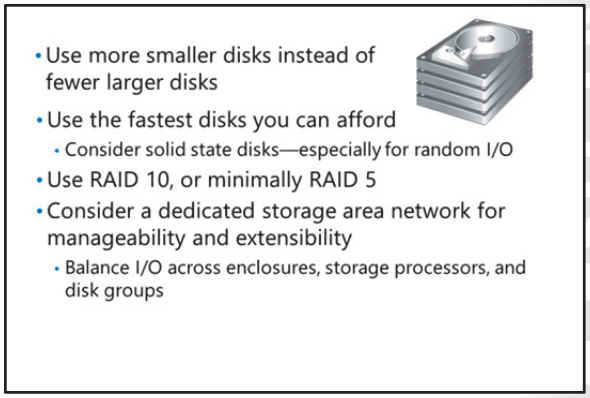
Other Storage Requirements

In addition to the data warehouse, you must include other data in your storage estimation. Additional storage is required for:

- **Configuration databases**. If databases used by other BI services, including the SSIS Catalog and Reporting Services databases, are to be installed on the data warehouse server, you should include them in your storage estimate. Additionally, the SQL Server instance includes system databases, though in practice, these are usually stored separately from the data warehouse data files.
- **Transaction log files**. Each database requires a transaction log. Typically, data warehouses are configured to use the simple recovery model and few transactions are actually logged.
- **TempDB**. Many data warehouse queries require temporary storage space. It is generally recommended to locate TempDB on a suitable storage column and assign an appropriate initial size to avoid the system having it grow automatically as needed.
- **Staging tables**. Whether or not data is staged in a dedicated staging database, in tables within the data warehouse database itself, or in a combination of both, you must allocate enough space to allow for data staging during ETL processes.
- **Backups**. If you intend to back up the data warehouse and other databases to disk, you must ensure that the storage design provides space for backup files.
- **Analysis Services models**. If you intend to host multidimensional Analysis Services data models on the data warehouse server, you must allocate sufficient disk space for them.

Considerations for Storage Hardware

The optimal storage hardware solution for a data warehouse depends on several factors, including the volume of data and the system MCR that data throughput from the storage system must support. When planning a storage solution, consider the following guidelines:

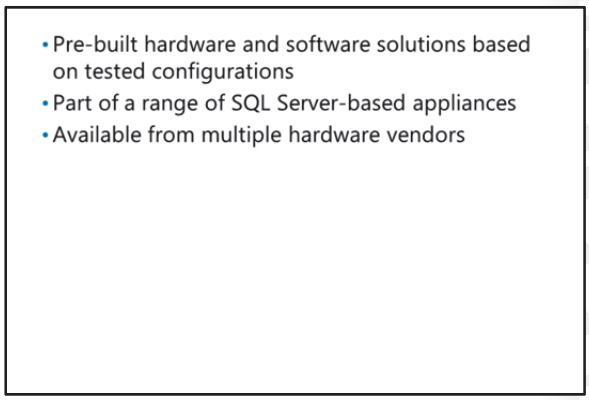


- **Use extra smaller disks instead of fewer larger disks**. Although it's possible to create a data warehouse that stores all its data on a single, large hard disk, a better balance of throughput (and therefore overall system performance) can usually be achieved by distributing the data across multiple small disks. This enables multiple disk reads to be performed in parallel and reduces wait times for I/O operations.
- **Use the fastest disks you can afford**. Disk technologies have advanced dramatically in recent years, with the speed of mechanical disks increasing and the advent of solid state disks with no moving parts. Most data warehouse read operations are sequential scans instead of the random I/O patterns of OLTP systems, so seek times are minimized. Regardless of this advantage, however, a faster disk means greater throughput when reading data. Solid state disks are typically more expensive than mechanical disks, but if disk performance is critical, you may decide that the additional cost is worth paying. The lack of moving parts makes them particularly effective for random I/O data access, typical of queries against multidimensional data models.
- **Use RAID 10, or minimally RAID 5**. RAID 10 (in which data is both mirrored and striped) provides the best balance of read performance and protection from disk failure, and this should usually be the

first choice for a data warehouse. However, the requirement for a complete set of redundant disks per array can make this an expensive option. As an alternative, you can use RAID 5, which provides striping for high read performance and parity-based data redundancy to protect against disk failure.

- **Consider a dedicated storage area network**. Although you can build a data warehouse with direct attached storage (DAS), using a storage area network (SAN) generally makes it easier to manage disk array configuration and to add future storage as the data warehouse grows. If you decide to use a SAN, it is best to have one dedicated to the BI solution and not shared with other business applications. Additionally, try to balance the number of enclosures, storage processors per enclosure, and disk groups to achieve a consistent I/O rate that takes advantage of parallel core processing and matches the MCR of the system.

SQL Server Data Warehouse Appliances



While Fast Track Data Warehouse reference architectures can reduce the time and effort taken to implement a data warehouse, organizations still require technical expertise to assemble the solution. To reduce the technical burden on organizations, and reduce the time it takes to implement a solution, Microsoft has partnered with hardware vendors to create pre-configured data warehouse appliances you can procure with a single purchase.

The data warehouse appliances that are available from Microsoft and its hardware partners are based on tested configurations, including Fast Track reference architectures, and can significantly reduce the time it takes to design, install, and optimize a data warehouse system.

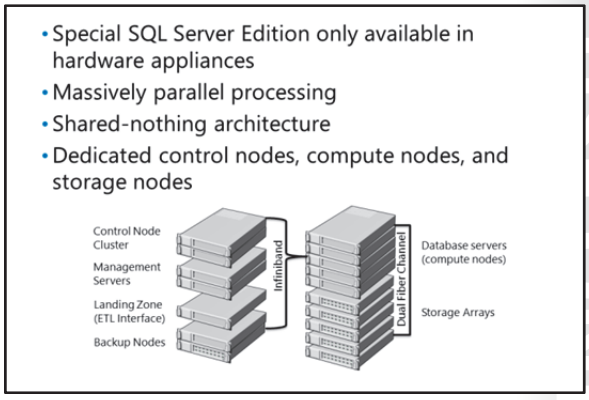
Data warehouse appliances based on Fast Track Data Warehouse reference architectures are available for organizations or departments that need to deploy a solution quickly and with minimal installation and configuration effort. Additionally, large organizations requiring an enterprise data warehouse can purchase an appliance based on SQL Server Parallel Data Warehouse for extreme scalability and performance.

Data warehouse appliances form part of a range of SQL Server-based appliances that Microsoft and its hardware partners have developed for common database workloads. Other types include business decision appliances that provide self-service BI capabilities, and database server consolidation appliances that use virtualization technologies to create a private cloud infrastructure for database servers. SQL Server-based appliances are available from multiple hardware vendors, and include technical support for the entire appliance, including software and hardware.

Additional Reading: For more information about SQL Server 2008 R2 Data Warehouse and Business Intelligence Appliances, go to http://go.microsoft.com/fwlink/?LinkID=246721.

SQL Server Parallel Data Warehouse

Fast Track Data Warehouse systems and appliances based on them use a symmetric multiprocessing (SMP) architecture. With SMP, the system bus is the limiting component that prevents scaling up beyond a certain level. As the number of processors and the data load increases, the bus can become overloaded and a bottleneck occurs. For data warehouses requiring greater scalability than a SMP system can provide, you can use an enterprise data warehouse appliance based on Microsoft® SQL Server® Parallel Data Warehouse.



Microsoft® SQL Server® Parallel Data Warehouse is an edition of SQL Server only available as a preinstalled and configured solution in enterprise data warehouse appliances from Microsoft and its hardware partners. Parallel Data Warehouse is designed specifically for extremely large-scale data warehouses that need to store and query hundreds of terabytes of data.

Massively Parallel Processing

Parallel Data Warehouse uses a shared-nothing, massively parallel processing (MPP) architecture, which delivers improved scalability and performance over SMP systems. MPP systems deliver much better performance than SMP servers for large data loads. MPP systems use multiple servers, called nodes, which process queries independently in parallel. Parallel processing involves distributing queries across the nodes so that each processes only a part of the query. The results of the partial queries are combined after processing completes to create a single result set.

Shared-nothing Architecture

Systems that use shared components, such as memory or disk storage, can suffer from performance issues because of contention for those shared components. Contention occurs when multiple nodes attempt to access a component at the same time, and usually results in degraded performance as nodes queue to access resources. Shared-nothing architectures eliminate contention because each node has its own dedicated set of hardware, which is not used by the others. Removing contention from a system results in improved performance, and enables it to handle larger workloads.

Control Nodes, Compute Nodes, and Storage Nodes

A Parallel Data Warehouse appliance consists of a server acting as the control node, and multiple servers representing compute and storage nodes. Each compute node has its own dedicated processors and memory, and is associated with a dedicated storage node. A dual InfiniBand network connects the nodes, and dual fiber channels link the compute nodes to the storage nodes. The control node intercepts incoming queries, divides each query into multiple smaller operations, which it passes on to the compute nodes to process. Each compute node returns the results of its processing back to the control node. The control node integrates the data to create a result set, which it then returns to the client.

Control nodes are housed in a control rack. There are three other types of node that share this rack:

- Management nodes, through which administrators manage the appliance.
- Landing Zone nodes, which act as staging areas for data that you load into the data warehouse by using an ETL tool.
- Backup nodes, which back up the data warehouse.

Compute nodes and storage nodes are housed separately in a data rack. To scale the application, you can add more racks as required. Hardware components are duplicated, including control and compute nodes, to provide redundancy.

You can use a Parallel Data Warehouse appliance as the hub in a hub-and-spoke configuration, and populate data marts directly from the data warehouse. Using a hub-and-spoke configuration enables you to integrate the appliance with existing data marts or create local data marts. If you use Fast Track Data Warehouse systems to build the data marts, you can achieve very fast transfers of data between the hub and the spokes.

 **Additional Reading:** For more information about the Parallel Data Warehouse for Microsoft SQL Server 2008 R2, go to http://go.microsoft.com/fwlink/?LinkID=246722.

Lab: Planning Data Warehouse Infrastructure

Scenario

You are planning a data warehouse solution for Adventure Works Cycles, and have been asked to specify the hardware required. You must design a SQL Server-based solution that provides the right balance of functionality, performance, and cost.

Objectives

After completing this lab, you will be able to:

- Plan data warehouse hardware for a SQL Server-based BI solution.

Estimated Time: 30 Minutes

Virtual Machine: 20463C-MIA-SQL

Use Name: ADVENTUREWORKS\Student

Password: Pa$$w0rd

Exercise 1: Planning Data Warehouse Hardware

Scenario

Now you have planned the server infrastructure, you must create a hardware specification for the data warehouse server. You will begin by calculating the MCR of the system you are currently using, then complete a planning worksheet for a new system with a published MCR figure.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Measure Maximum Consumption Rate

3. Estimate Server Hardware Requirements

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. Run Setup.cmd in the D:\Labfiles\Lab02\Starter folder as Administrator.

▶ Task 2: Measure Maximum Consumption Rate

1. Use the **Create BenchmarkDB.sql** script in the D:\Labfiles\Lab02\Starter folder to create a benchmark database on the MIA-SQL instance of SQL Server.
2. Use the **Measure MCR.sql** script in the D:\Labfiles\Lab02\Starter folder to generate query performance statistics.
3. Use the statistics to calculate MCR for the database server.
4. Use the calculated MCR figure to estimate the number of cores required to support the following workload:
    - Average data per query: 500 MB
    - Concurrent users: 10
    - Target response time: 20 sec

▶ Task 3: Estimate Server Hardware Requirements

1. Use Microsoft Excel® to open the **DW Hardware Spec.xlsx** workbook in the D:\Labfiles\Lab02\Starter folder.
2. Use the information in the workbook to:
   - Calculate the number of cores required in the data warehouse server.
   - Recommend the number and type (dual core or quad core) of processors to include in the data warehouse server.
   - Calculate the estimated volume of data in the data warehouse.
   - Suggest a suitable amount of memory for the data warehouse server.
   - Calculate the storage requirements for the data warehouse server, assuming a compression ratio of 3:1.
3. Use Microsoft Word® to open the **Storage options.docx** document in the D:\Labfiles\Lab02\Starter folder and review the available storage options.
4. Based on the storage requirements you have identified, select a suitable storage option and record your selection in the DW Hardware Spec.xlsx workbook.

Results: After this exercise, you should have a completed worksheet that specifies the required hardware for your data warehouse server.

Question: Review **DW Hardware Spec.xlsx** in the D:\Labfiles\Lab02\Solution folder. How does the hardware specification in this workbook compare to the one you created in the lab?

Module Review and Takeaways

This module has described some key considerations for planning the hardware infrastructure for a SQL Server-based BI solution. You should use this as a starting point, and use the "more information" references to learn more about the supported, distributed and scale-out architectures for SQL Server components, and about the design principles used in Fast Track Data Warehouse reference architectures.

Review Question(s)

Question: In a growing number of organizations, virtualization has become a core platform for infrastructure. Hyper-V in Windows Server® 2012, together with enterprise operations and management software such as Microsoft® System Center® 2012, has enabled IT departments to benefit from more simple provisioning, management, mobility, and recoverability of services.

What components of a BI infrastructure would you consider virtualizing, and why?

Module 3

Designing and Implementing a Data Warehouse

Contents:

Module Overview

The data warehouse is at the heart of most business intelligence (BI) solutions. Designing the logical and physical implementation of the data warehouse is crucial to the success of the BI project. Although a data warehouse is fundamentally a database, there are some significant differences between the design process and best practices for an online transaction processing (OLTP) database and a data warehouse that will support online analytical processing (OLAP) and reporting workloads.

This module describes key considerations for the logical design of a data warehouse, and then discusses best practices for its physical implementation.

Objectives

After completing this module, you will be able to:

- Describe a process for designing a dimensional model for a data warehouse.
- Design dimension tables for a data warehouse.
- Design fact tables for a data warehouse.
- Design and implement effective physical data structures for a data warehouse.

Lesson 1
Data Warehouse Design Overview

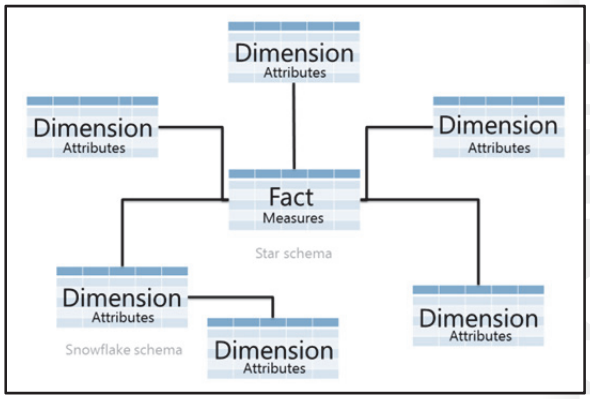
Before designing individual database tables and relationships, it is important to understand the key concepts and design principles for a data warehouse. This lesson describes the dimensional model used in most data warehouse designs and the process used to translate business requirements into a data warehouse schema.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe where data warehouse design fits into an overall BI project.
- Describe the dimensional model used for most data warehouses.
- Apply an effective process for data warehouse design.
- Use a business process-based approach to dimensional modeling.
- Document dimensional model designs.

The Dimensional Model



Although data warehouses can be implemented as normalized, relational database schemas, most designs are based on the dimensional model advocated by Ralph Kimball. In the dimensional model, the numeric business measures that are analyzed and reported are stored in fact tables, which are related to multiple dimension tables, in which the attributes by which the measures can be aggregated are stored. For example, a fact table might store sales order measures, such as revenue and profit, and be related to dimension tables representing business entities such as product and customer. These relationships make it possible to aggregate the sales order measures by the attributes of a product (for example, to find the total profit for a particular product model) or a customer (for example, to find the total sales revenue for customers who live in a particular country).

Ideally, a dimensional model can be implemented in a database as a "star" schema, in which each fact table is directly related to its relevant dimension tables. However, in some cases, one or more dimensions may be normalized into a collection of related tables to form a "snowflake" schema. Generally, you should avoid creating snowflake dimensions because, in a typical data warehouse workload, the performance benefits of a single join between fact and dimension tables outweigh the data redundancy reduction benefits of normalizing the dimension data.

The query optimizer in the Enterprise edition of SQL Server 2012 includes logic that detects star schema joins in queries and optimizes the way these joins are processed accordingly. Based on the selectivity of the query (that is, the proportion of rows from the fact table that the query is likely to return), the query optimizer uses bitmap filters to quickly eliminate non-qualifying rows from the fact table (which generally accounts for the largest cost in a data warehouse query).

Additional Reading: For more detailed information about star join query optimization, go to *Introduction to New Data Warehouse Scalability Features in SQL Server 2008* at http://msdn.microsoft.com/en-us/library/cc278097(v=SQL.100).aspx and *Data Warehouse Query Performance* at http://technet.microsoft.com/en-us/magazine/2008.04.dwperformance.aspx.

The Data Warehouse Design Process

Although every project has its unique considerations, there is a commonly-used process for designing a dimensional data warehouse that many BI professionals have found effective. The method is largely based on the data warehouse design patterns identified and documented by Ralph Kimball and the Kimball Group, though some BI professionals may adopt a varied approach to each task.

1. Determine analytical and reporting requirements
2. Identify the business processes that generate the required data
3. Examine the source data for those business processes
4. Conform dimensions across business processes
5. Prioritize processes and create a dimensional model for each
6. Document and refine the models to determine the database logical schema
7. Design the physical data structures for the database

Additional Reading: For a detailed exploration of how to apply the Kimball dimensional modeling methodology to a SQL Server-based data warehouse design, read *The Microsoft Data Warehouse Toolkit* (Wiley, 2011).

1. Determine analytical and reporting requirements

After gathering the business requirements for the BI solution, you must interpret them regarding the analytical and reporting capabilities that the solution must provide. Typically, analytical and reporting requirements support business requirements, so you will probably need to spend time with the stakeholders to understand the information that they need and to discuss how to achieve the best results. For example, a sales executive might express a business requirement as: "We want to improve the performance of sales representatives in the most poorly performing sales territories." To meet this requirement, you need to understand how "sales performance" is measured (for example, revenue, profitability, number of orders, or a combination of all three) and against what aspects of the business it should be aggregated.

Typically, asking questions such as: "How will you be able to tell if the business requirement is being met?" leads the discussion toward the analytical and reporting requirements. For example, to determine if sales performance is improving, the sales executive might need to be able to see "order volume by territory" or "sales revenue by salesperson." Requirements expressed like this make it easier to determine the measures and dimensions the solution must include, because the requirement often takes the form "measure by dimension".

Additionally, most analytical and reporting requirements include a time-based aggregation. For example, the sales executive might want to compare sales revenue by month or quarter.

2. Identify the business processes that generate the required data

Typically, the data required in the dimensional model is generated by an existing business process. After determining the data you need to support analysis and reports, you must identify the business processes that generate the source data.

For example, a business might include the following processes:

- Order processing
- Stock management
- Order fulfillment
- Manufacturing
- Marketing and promotions
- Financial accounting
- Human Resources management

Each process generates data that includes numeric values and events that can be counted (these can be sources for measures in a dimensional model) and information about key business entities (these can be sources for dimension attributes).

3. Examine the source data for those business processes

In most organizations, each business process captures data in at least one system. For example, order processing might store details of orders, customers, and products. A financial accounting process typically stores details of accounts and balances.

A significant part of the data warehouse solution design process involves exploring the data in these source systems and interviewing the users, system administrators, and application developers who understand it best. Initial exploration might simply involve running Transact-SQL queries to determine distinct value counts, average numerical values, and row counts. You can use the basic information gathered from these initial queries and discussions with data specialists as a foundation for deeper data profiling using tools such as the Data Profiling task in SQL Server Integration Services. This can determine minimum and maximum field lengths, data sparseness and null counts, and the reliability of relational dependencies.

At this stage, you do not need to perform a full data audit and start planning the extract, transform, and load (ETL) solution. You do, however, need to identify if and where the measures and dimension attributes you need to meet the reporting requirements are stored, what range of values exist for each required data field, what data is missing or unknown, and at what granularity the data is available.

4. Conform dimensions across business processes

Identifying the business processes and exploring the data generated by each one will help you identify some key business entities that are common across the various processes. For example, the order processing, manufacturing, and stock management business processes might both deal with a "product" entity. Similarly, the order processing and order fulfillment processes might both deal with a "customer" entity. Identifying dimensions that can be shared across multiple business processes is an important part of data warehouse design because it enables you to define "conformed" dimensions. These ensure that the same definition of a business entity can be used to aggregate multiple facts and produce meaningful comparisons. For example, by using a conformed product dimension, a data warehouse based on the order processing, manufacturing, and stock management business processes will enable you to analyze a specific product and compare the number ordered, the number manufactured, and the number held in stock. If the product is perishable and has a limited shelf-life, this kind of comparison could provide significant information for production planning and help reduce losses from spoiled, unsold products.

5. Prioritize business processes and define a dimensional model for each

Based on the business value of the identified reporting and analytical requirements, prioritize the business processes and create a dimensional model for each one that is required to provide the necessary analytical and reporting data. To do this, you must perform the following steps:

1. Identify the grain.
2. Select the required dimensions.
3. Identify the facts.

The details are discussed in the next topic.

6. Document and refine the models to determine the database logical schema

After you create initial dimensional models for each required business process, you can document the models to show:

- The measures in the fact tables.
- The related dimension tables.
- The attributes and hierarchies required in the dimension tables.

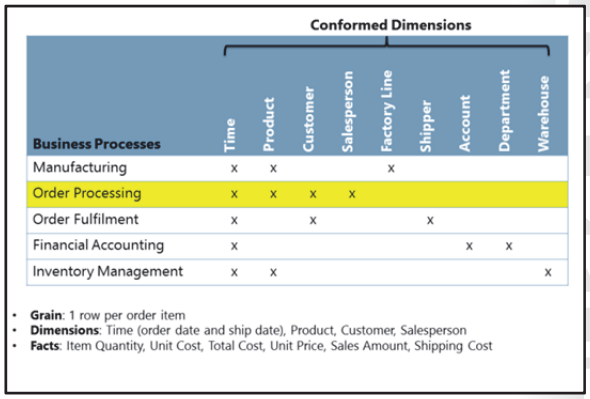
You can then iteratively refine the model to design the fact and dimension tables that will be required in the data warehouse database. Considerations for fact and dimension tables are discussed later in this module.

7. Design the physical data structures for the database

After you complete the logical database design, you can consider the physical implementation of the database, including data file placement, indexes, table partitions, and data compression. These topics are discussed in more depth later in this module.

Dimensional Modeling

After you identify the business processes and conformed dimensions, you can document them in a matrix, as shown on the slide. This approach is based on the bus matrix design technique promoted by the Kimball Group.



Additional Reading: For more information about using a bus matrix as part of a data warehouse design project, read *The Microsoft Data Warehouse Toolkit* (Wiley, 2011).

You can then use the matrix to select each business process based on priority, and design a dimensional model by performing the following steps:

1. **Identify the grain**. The grain of a dimensional model is the lowest level of detail at which you can aggregate the measures. It is important to choose the level of grain that will support the most granular of reporting and analytical requirements, so typically the lowest level possible from the source data is the best option. For example, an order processing system might record data at two levels. There may be order-level data, such as the order date, salesperson, customer, and shipping cost, as well as line item-level data like the products included in the order and their individual quantities, unit costs, and selling prices. To support the most granular analysis and reporting, the grain should be declared at the line item level, so the fact table will contain one row per line item.
2. **Select the required dimensions**. Next, determine which of the dimensions related to the business process should be included in the model. The selection of dimensions depends on the reporting and analytical requirements, specifically on the business entities by which users need to aggregate the measures. Almost all dimensional models include a time-based dimension, and the others generally become obvious as you review the requirements. At this stage, you might also begin to identify specific attributes of the dimensions that will be needed, such as the country, state, and city of a customer or the color and size of a product.

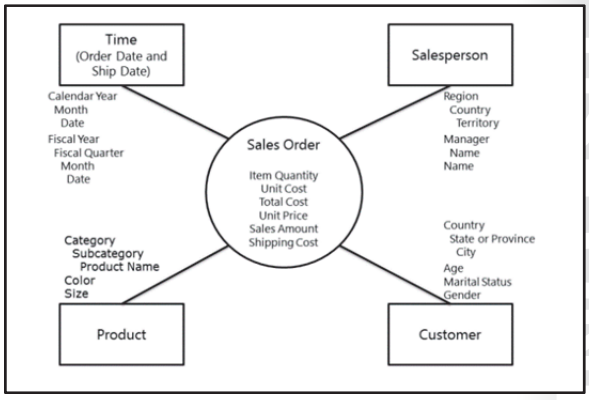
In the example on the slide, the **Time**, **Customer**, **Product**, and **Salesperson** dimensions are selected.

Note: In this example, the **Time** dimension is used for both order and ship date. Although it would be possible to define an individual dimension for each date type, it is more common to create a single time dimension and use it for multiple roles. In an analytical model, these multiple usages of the same dimension table are known as "role-playing dimensions". This technique is most commonly used for timetables, but it can be applied to any dimension that is used in multiple ways. For example, a dimensional model for an airline flight-scheduling business process might use a single **Airport** dimension to support **Origin** and **Destination** role-playing dimensions.

3. **Identify the facts**. Finally, identify the facts that you want to include as measures. These are numeric values that can be expressed at the level of the grain chosen earlier and aggregated across the selected dimensions. Some facts will be taken directly from source systems, and others might be derived from the base facts. For example, you might choose **Quantity** and **Unit Price** facts from an order processing source system, and then calculate a total **Sales Amount**. Additionally, depending on the grain you choose for the dimensional model and the grain of the source data, you might need to allocate measures from a higher level of grain across multiple fact rows. For example, if the source system for the order processing business process includes a **Tax** measure at the order level, but the facts are to be stored at the line item level, you must decide how to allocate the tax amount across the line items. Typically, tax is calculated as a percentage of selling price, so it should be straightforward to apply the appropriate tax rate to each line item based on the sales amount.

In the example on the slide, the **Item Quantity**, **Unit Cost**, and **Unit Price** measures are taken from the source system at the line item level. From these, the **Total Cost** and **Sales Amount** measures for each line item can be calculated. Additionally, the **Shipping Cost** measure is defined at the order level in the source system, so it must be allocated across the line items. You do this by dividing it equally across each row or applying a calculation that distributes the shared cost based on the quantity of each item ordered, total line item weight, or some other appropriate formula.

Documenting Dimensional Models



After you design the initial dimensional models for each business process, you can document them in a simple diagram. A common format for this documentation is a *sun* diagram, in which a fact table is shown at the center of the dimensions to which it is related.

As you refine the dimensional model, you can add more detail to the sun diagram, including the measures in the fact table and the attributes in the dimension tables. In most cases, some or all of the dimension attributes can be used to form a hierarchy for drill-down analysis, for example, enabling users to view aggregations of sales amount by year, month, and date or by country, state, and city. You can add these hierarchies to the sun diagram to help you communicate and validate model design with business stakeholders.

Eventually, the simple diagram will be refined to the point where it can be easily translated into a schema design for database tables. At this stage, you can use a diagramming tool such as Microsoft Visio® or a specialist database modeling tool to start designing the logical schema of your data warehouse.

Lesson 2
Designing Dimension Tables

After designing the dimensional models for the data warehouse, you can translate the design into a logical schema for the database. However, before you design dimension tables, it is important to consider some common design patterns and apply them to your table specifications.

This lesson discusses some of the key considerations for designing dimension tables.

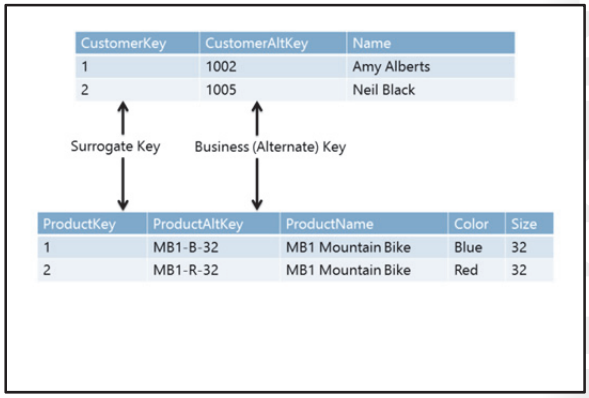
Lesson Objectives

After completing this lesson, you will be able to:

- Describe considerations for dimension keys.
- Describe considerations for dimension attributes and hierarchies.
- Design dimensions that support values for "none" or "unknown".
- Design appropriate slowly-changing dimensions for your business requirements.
- Design time dimension tables.
- Design self-referencing dimension tables.
- Include junk dimensions in a data warehouse design where appropriate.

Considerations for Dimension Keys

Each row in a dimension table represents an instance of a business entity by which the measures in the fact table can be aggregated. Like other tables in a database, a key column uniquely identifies each row in the dimension table. In many scenarios, the dimension data is obtained from a source system in which a key is already assigned (sometimes referred to as the "business" key). When designing a data warehouse, however, it is standard practice to define a new "surrogate" key that uses an integer value to identify each row. A surrogate key is recommended for the following reasons:

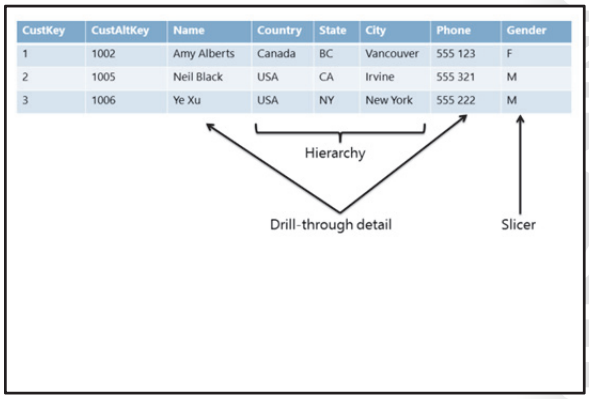


- The data warehouse might use dimension data from multiple source systems, so it is possible that business keys are not unique.
- Some source systems use non-numeric keys, such as a globally unique identifier (GUID), or natural keys, such as an email address, to uniquely identify data entities. Integer keys are smaller and more efficient to use in joins from fact tables.
- Each row in a dimension table represents a specific version of a business entity instance. If the dimension table supports "Type 2" slowly-changing dimensions, the table might need to contain multiple rows that represent different versions of the same entity. These rows will have the same business key and won't be uniquely identifiable without a surrogate key.

Typically, the business key is retained in the dimension table as an "alternate" key. Business keys that are based on natural keys can be familiar to users analyzing the data. For example, a **ProductCode** business key that users will recognize might be used as an alternate key in the **Product** dimension table. However, the main reason for retaining a business key is to make it easier to manage slowly-changing dimensions when loading new data into the dimension table. The ETL process can use the alternate key as a lookup column to determine whether an instance of a business entity already exists in the dimension table.

Dimension Attributes and Hierarchies

In addition to the surrogate and alternate key columns, a dimension table includes a column for each attribute of the business entity that is needed to support reporting and analytical requirements. When designing a dimension table, you need to identify and include attributes that will be used in reports and analysis. Typically, dimension attributes are used in one of the following three ways:



- **Hierarchies**. Multiple attributes can be combined to form hierarchies that enable users to drill down into deeper levels of detail. For example, the **Customer** table in the slide includes **Country**, **State**, and **City** attributes that can be combined to form a natural geographical hierarchy. Business users can view aggregated fact data at each level, for example, to see sales order revenue by country. They can then access a specific country to see a breakdown by state, before drilling further into a specific state to see sales revenue by city.
- **Slicers**. Attributes do not need to form hierarchies to be useful in analysis and reporting. Business users can group or filter data based on single-level hierarchies to create analytical sub-groupings of data. For example, the **Gender** attribute in the **Customer** table can be used to compare sales revenue for male and female customers.
- **Drill-through detail**. Some attributes have little value as slicers or members of a hierarchy. For example, it may be unlikely that a business user will need to analyze sales revenue by customer phone number. However, it can be useful to include entity-specific attributes to facilitate drill-through functionality in reports or analytical applications. For example, in a sales order report that enables users to drill down to the individual order level, users might want to double-click an order and drill through to see the name and phone number of the customer who placed it.

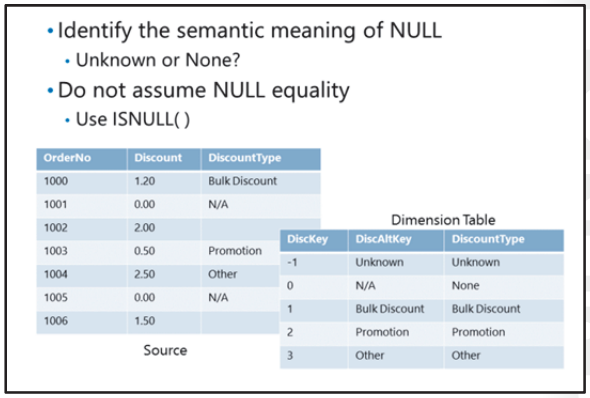
Note: Terminology for interacting with report data can be confusing, and is sometimes used inconsistently. For clarity in this course, the term "drill down" means expanding a hierarchy to see the next level of aggregation, and "drill through" means viewing details outside the current hierarchy for a selected row. For example, while considering sales revenue by customer geography, you might view total revenue for a specific country in the hierarchy. You might then "drill down" to see a subtotal for each state within that country (the next level in the hierarchy), or "drill through" to see the country's demographic details.

In the example on the slide, note that the **Name** column contains the full name of the customer. In a data warehouse table schema, it is not usually necessary to normalize the data to its most atomic level as is common in OLTP systems. In this example, it is unlikely that users will want to group or filter data by the customer's first or last name, and the data only has drill-through value at the full name level of detail.

Therefore, the **FirstName**, **MiddleName**, and **LastName** columns in the source system have been combined into a single **Name** field in the data warehouse.

Unknown and None

As a general rule, try to design your data warehouse in a way that eliminates, or at least minimizes, NULL values, particularly in fact table key columns that reference dimension tables. NULL values make it easy to accidentally eliminate rows from reports and produce misleading totals.



Identifying the semantic meaning of NULL

When you explore the source data for your BI solution, pay particular attention to how NULL values are used. The semantic meaning of NULL might be "None" or "Unknown," depending on the context, and only by examining the data and consulting the users, administrators, and developers who are familiar with the source system, will you be able to confidently determine which is relevant. In the example on the slide, the source data includes a column named **DiscountType**, in which two rows have a missing, or NULL, value. The fact that these rows include a non-zero **Discount** value indicates that NULL does not necessarily always mean "None", and is more likely to denote "Unknown." Additionally, on the rows where the **Discount** value is zero, the **DiscountType** value is consistently "N/A," implying that "N/A" is used in this system to mean "None."

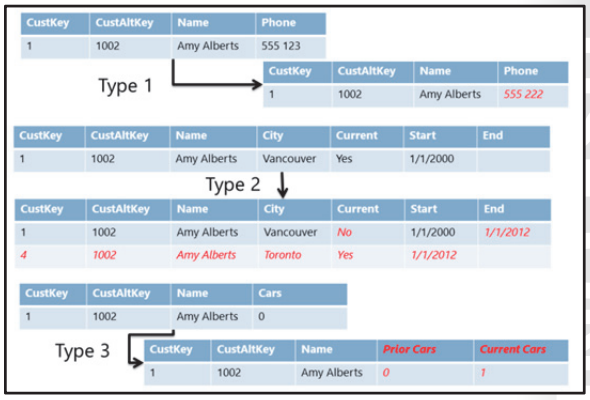
To support these two cases, a row for each is added to the dimension table with appropriate surrogate keys, such as -1 for "Unknown" and 0 for "None". If the source systems were more ambiguous, you could add a single row to the dimension table to represent "None or Unknown."

NULL equality

Depending on the settings in a SQL Server database, you might not be able to compare NULL values for equality. In its strictest definition, NULL means unknown, so a "NULL = NULL" comparison is actually asking if one unknown value is the same as another, and because both values are unknown, the answer is also unknown (and therefore NULL). You should not use NULL as the alternate key for the "Unknown" dimension row, because lookup queries during the ETL load process must compare this key to the data being loaded to determine whether a dimension row already exists. Instead, use an appropriate key value that is unlikely to be the same as an existing business key, and use the Transact-SQL ISNULL function to compare source rows with dimension rows, as shown in the following code sample:

```
SELECT d.DiscKey, s.DiscountType
FROM DimDiscount AS d
JOIN SourceData AS s ON ISNULL(s.DiscountType, 'Unknown') = d.DiscAltKey
```

Designing Slowly Changing Dimensions



Slowly changing dimensions (SCDs) are a significant consideration in the design of dimension tables. You should try to identify requirements for maintaining historic dimension attribute values as early as possible in the design process.

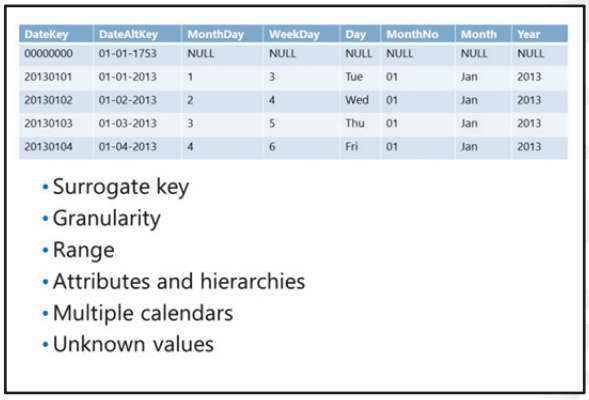
There are three common techniques used to handle attribute value changes in SCDs:

- **Type 1**. These changes are the simplest type of SCD to implement. Attribute values are updated directly in the existing dimension table row and no history is maintained. This makes Type 1 changes suitable for attributes that are used to provide drill-through details, but unsuitable for analytical slicers or hierarchy members where historic comparisons must reflect the attribute values as they were at the time of the fact event.
- **Type 2**. These changes involve the creation of a fresh version of the dimension entity in the form of a new row. Typically, a bit column in the dimension table is used as a flag to indicate which version of the dimension row is the current one. Additionally, datetime columns are often used to indicate the start and end of the period for which a version of the row was (or is) current. Maintaining start and end dates makes it easier to assign the appropriate foreign key value to fact rows as they are loaded so they are related to the version of the dimension entity that was current at the time the fact occurred.
- **Type 3**. These changes are rarely used. In a Type 3 change, the previous value (or sometimes a complete history of previous values) is maintained in the dimension table row. This requires modifying the dimension table schema to accommodate new values for each tracked attribute, and can result in a complex dimension table that is difficult to manage.

After you define the dimensional model for the data warehouse and are evolving your design from a sun diagram to a database schema, it can be useful to annotate dimension attributes to indicate what kind of SCD changes they must support. This will help you plan the metadata columns required for each dimension table.

Time Dimension Tables

Most analysis and reporting requirements include a need to aggregate values over time periods, so almost every data warehouse includes a time dimension table. When you design a time dimension table, you must take into account the following considerations:



- **Surrogate key**. Although best practice for surrogate keys in dimension tables is normally to use a simple integer value with no semantic meaning, time dimension tables can benefit from an integer representation of the date or time that the row represents. Ideally, the values should be in ascending order relative to the dates they represent, so the best approach is to concatenate the integer values for each date part in descending order of scope. For example, using the YYYYMMDD pattern to represent dates, the value for January 31, 2013, would be 20130131. This ensures that the value used for the next sequential date of February 1, 2013, is a higher value of 20130201. Ascending values are recommended because data warehouse queries typically filter on a range of date or time values. The ascending numeric key enables you to use indexes and partitions that store the fact data in chronological order, and enables the query optimizer to use sequential scans to read the data. Additionally, the actual datetime value for the row is generally used as the alternate key to support datetime functions or client applications that can apply datetime-specific logic.
- **Granularity**. The level of granularity used for a time dimension table depends on business requirements. For many reporting and analysis scenarios, such as viewing details about sales orders, the lowest level of granularity likely to be required is a day. However, in some scenarios, users might need to aggregate facts by hours, minutes, or seconds, or even smaller increments. The lower the level of granularity used, the more rows will exist in the dimension table, and storing a row for increments of less than a day can result in extremely large tables. An alternative approach is to create a "date" dimension table that contains a row for each day, and a "time" dimension table that stores a row for each required time increment in a 24-hour period. Fact tables that are used for analysis of measures at the day level or higher can only be related to the date dimension table. Facts measured at smaller time increments can be related to both date and time dimension tables.
- **Range**. Typically, a time dimension table stores a row for each increment between a start point and an end point with no gaps. For example, a data warehouse time dimension used to analyze sales orders might have a row for each day between the first ever and most recent orders, even if no orders were placed during the intervening days. In reality, the start and end dates are typically based on key calendar dates. For example, the start date might be January 1 of the year the company started trading, or when its first fiscal year began. The end date is usually some future point, such as the end of the current year. To maintain a buffer of future dates, more rows are added automatically as the end date gets closer. If the data warehouse will be used to create and store projections or budget figures for future operations, you will need to choose an end date that is far enough into the future.

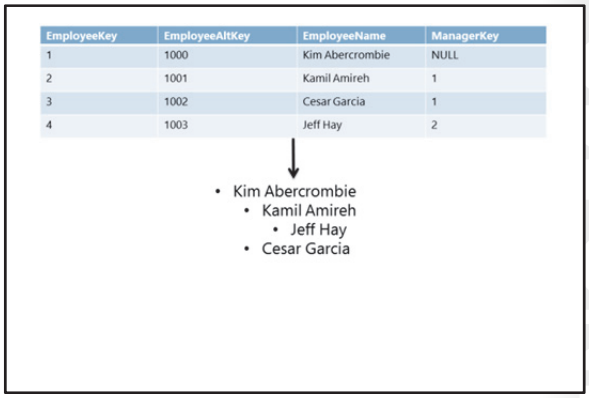
- **Attributes and hierarchies**. You need to include attributes for each time period by which data will be aggregated, for example, year, quarter, month, week, and day. These attributes tend to form natural hierarchies. You can also add attributes to be used as slicers, such as "weekday" which, for example, would enable users to compare typical sales volumes for each day of the week. In addition to numeric values, you might want to include attributes for date element names, such as "month" and "day". This enables more accessible reports where, for example, users could compare sales in March and April instead of month 3 and 4. You should also include the numeric equivalents so that client applications can use them to sort data into the correct chronological order, such as sorting months by month number instead of by month name.
- **Multiple calendars**. Many organizations need to support multiple calendars, for example a normal year that runs from January to December, and a fiscal calendar, which might run from April to March. If this is the case in your data warehouse, you can either create a separate time dimension table for each calendar or, more preferably, include attributes for all alternative calendar values in a single time dimension table. For example, a time dimension table might include a **Calendar Year** and a **Fiscal Year** attribute.
- **Unknown values**. In common with other dimension tables, you might need to support facts for which a date or time value is unknown. Instead of requiring a NULL value in the fact table, consider creating a row for unknown values in the time dimension table. You can use an obvious surrogate key value, such as 00000000, for this row. However, because the alternate key must be a valid date, you should choose one outside the normal range of business operations, such as January 1, 1753, or December 31, 9999. These are the minimum and maximum values supported by the datetime data type.

Populating a Time Dimension Table

Unlike most other tables in a data warehouse, time dimension tables are not usually populated with data that has been extracted from a source system. Generally, the data warehouse developer populates the time dimension table with rows at the appropriate granularity. These rows usually consist of a numeric primary key that is derived from the temporal value (such as 20110101 for January 1, 2011) and a column for each dimension attribute (such as the date, day of year, day name, month of year, month name, year, and so on). To generate the rows for the time dimension table, you can use one of the following techniques:

- **Create a Transact-SQL script**. Transact-SQL includes many date and time functions that you can use in a loop construct to generate the required attribute values for a sequence of time intervals. The following Transact-SQL functions are commonly used to calculate date and time values:
  - DATEPART (datepart, date) returns the numerical part of a date, such as the weekday number, day of month, month of year, and so on.
  - DATENAME (datepart, date) returns the string name of a part of the date, such as the weekday name or month name.
  - MONTH (date) returns the month number of the year for a given date.
  - YEAR (date) returns the year for a given date.
- **Use Microsoft® Excel®**. Excel includes several functions that you can use to create formulas for date and time values. You can then use the auto-fill functionality in Excel to quickly create a large table of values for a sequence of time intervals.
- **Use a BI tool to autogenerate a time dimension table**. Some BI tools include time dimension generation functionality that you can use to quickly create a time dimension table.

Self-Referencing Dimension Tables



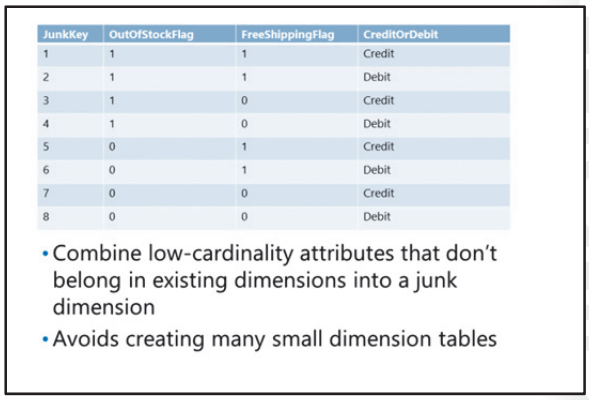
A common requirement in a data warehouse is to support dimensions with parent-child hierarchies. For example, an employee dimension might consist of managers, each of whom has employees reporting to him or her, who in turn might have reports of their own.

Typically, parent-child hierarchies are implemented as self-referencing tables, in which a column in each row is used as a foreign-key reference to a primary-key value in the same table. Some client applications, including SQL Server Analysis Services, are aware of self-joins and can automatically handle parent-child hierarchies in a dimension. For other applications, you might need to implement some custom, recursive logic to enable analysis and reporting of these hierarchies.

When you implement a self-referencing dimension table in a data warehouse, you should think about the following considerations:

- Like all dimension load operations, when records are to be added to the dimension table, the ETL process must look up the alternate key to determine if a record already exists for the entity. However, the alternate key of the parent record must also be looked up to determine the correct surrogate key to use in the foreign key column.
- You may have to deal with a situation where you need to add a record for which the parent record has not yet been loaded.
- Supporting Type 2 SCDs in a self-referencing dimension table can be complex. In a worst case scenario, you might perform a Type 2 change that results in a new row and, therefore, a new surrogate key. You may then need to cascade that Type 2 change to create new rows for all descendants of the entity if the change has not altered the parent-child relationships.

Junk Dimensions



In some reporting and analytical requirements, there are useful attributes for grouping or filtering facts which do not belong in any of the dimensions defined in the dimensional model. If these attributes have low cardinality, when there are only a few discrete values, you can group them into a single dimension table containing miscellaneous analytical values. This kind of dimension table is generally referred to as a "junk dimension" and is used to avoid creating multiple, very small dimension tables.

For example, a sales order dimensional model might include "true or false" indicators for orders where goods were out of stock or where free shipping was provided. There may be a column that stores "credit" or "debit" to indicate the payment method. Instead of creating a dimension table for each of these attributes, you could merge them in every possible combination in a junk dimension table.

Lesson 3
Designing Fact Tables

Fact tables contain the numeric measures that can be aggregated across the dimensions in your dimensional model, and can become extremely large. It is important to design them carefully with reporting and analytical requirements, performance, and manageability in mind.

This lesson discusses common considerations for fact table design.

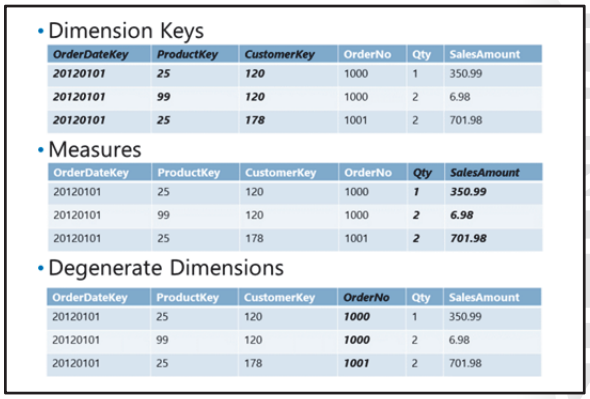
Lesson Objectives

After completing this lesson, you will be able to:

- Determine appropriate columns for a fact table.
- Design a fact table at an appropriate level of grain.
- Describe the types of measure that are stored in a fact table.
- Describe common types of fact table.

Fact Table Columns

A fact table usually consists of the following kinds of columns:



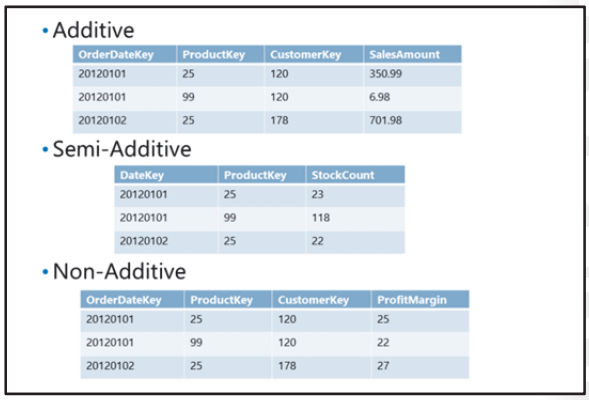
- **Dimension keys**. Fact tables reference dimension tables by storing the surrogate key for each related dimension. In this way, a row in a fact table is conceptually an intersection between the dimension tables to which it relates. For example, recording a sales order placed on a specific date, for a specific product, by a specific customer. You can add foreign key constraints to these columns, which will help the SQL Server query optimizer detect star joins. However, constraints can slow down data load operations, and because the surrogate keys are generated during a controlled ETL process, they do little to enforce referential integrity.
- **Measures**. In most cases, a fact table is primarily used to store numeric measures that can be aggregated by the related dimensions. For example, a row in a fact table recording sales orders might include a column for the sales amount, which can then be aggregated by the dimensions to show sales amount by date, product, or customer. In some cases, a fact table contains no measures and is simply used to indicate that an intersection occurred between the related dimensions. For example, a fact table in a manufacturing dimensional model might record a single row each time a product assembly is completed, indicating the product and date dimension keys. The fact table can then be used to calculate the number of times an assembly of each product was completed per time period by simply counting the distinct rows. A fact table with no numeric measure columns is sometimes referred to as a "factless fact table".

- **Degenerate dimensions**. Sometimes, a fact has associated attributes that are neither keys nor measures, but which can be useful to group or filter facts in a report or analysis. You might include this column in the fact table where client applications can use it as a "degenerate dimension" by which the fact data can be aggregated. In effect, including degenerate dimension columns in a fact table enables it to also be used as a dimension table. Using degenerate dimensions can be a good alternative to using a junk dimension if the analytical attributes are specific to a single fact table.

Note: Unlike most database tables, a fact table does not necessarily require a primary key. Unless you have a business requirement to uniquely identify each row in the fact table, you should avoid creating a unique key column for the fact table and defining a primary key constraint. Facts are generally aggregated, and queries rarely need to individually identify a fact row. In some cases, the combination of dimension keys can uniquely identify a fact row, but this is not guaranteed. For example, a customer could purchase the same product twice in one day.

Types of Measure

Fact tables can contain the following three kinds of measure:

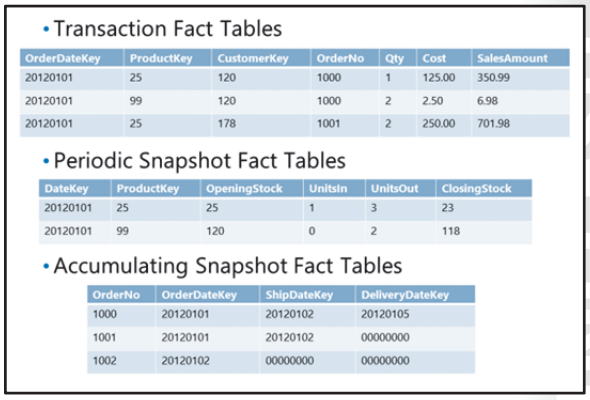


- **Additive measures**. These can be summed across all dimensions. For example, you could use a **SalesAmount** measure in a fact table with **OrderDateKey**, **ProductKey**, and **CustomerKey** dimension keys to calculate total sales amount by time period (such as month), product, or customer.
- **Semi-additive measures**. These can be summed by some dimensions, but not others. Commonly, semi-additive measures cannot be summed by time dimensions. For example, the number of items in stock at the end of each day might be recorded as a **StockCount** measure in a fact table with **DateKey** and **ProductKey** dimension keys that can be used to calculate a total stock count for all products. However, summing the stock count across all the days in a month does not result in a total stock count value for that period. To find out how products are in stock at the end of the month, you must use only the **StockCount** value for the final day.
- **Non-additive measures**. These cannot be summed by any dimension. For example, a fact table for sales orders might include a **ProfitMargin** measure that records the profit margin for each order. However, you cannot calculate the overall margin for any dimension by summing the individual profit margins.

Generally, semi-additive and non-additive measures can be aggregated by using other functions. For example, you could find the minimum stock count for a month or the average profit margin for a product. Understanding the ways in which the measures can be meaningfully aggregated is useful when testing and troubleshooting data warehouse queries and reports.

Types of Fact Table

Generally, data warehouses include fact tables that are one of the following three types:



- **Transaction fact tables**. The most common kind of fact table is a "transaction" fact table, in which each row records a transaction or event at an appropriate grain. For example, a fact table might record sales orders at the line item grain, in which each row records the purchase of a specific item. Transaction fact table measures are usually additive.
- **Periodic snapshot tables**. These record measure values at a specific point in time. For example, a fact table might record the stock movement for each day, including the opening and closing stock count figures. Measures in a periodic snapshot fact table are often semi-additive.
- **Accumulating snapshot fact tables**. These can be used in scenarios where you might want to use a fact table to track the progress of a business process through multiple stages. For example, a fact table might track an order from initial purchase through to delivery by including a date dimension-key field for the order date, shipping date, and delivery date. The **ShipDate** and **DeliveryDate** columns for orders that have been placed but not yet shipped will contain the dimension key for an "Unknown" or "None" row in the time dimension table. These will be updated to reflect the appropriate dimension key as the order is shipped and delivered.

Lesson 4
Physical Design for a Data Warehouse

After designing the logical schema for the data warehouse, you need to implement it as a physical database. This requires careful planning for file placement, data structures such as partitions and indexes, and compression. This lesson discusses considerations for all these aspects of the physical database design.

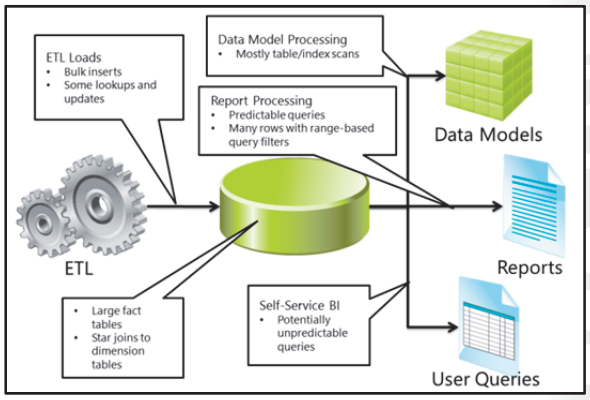
Lesson Objectives

After completing this lesson, you will be able to:

- Describe typical data warehouse I/O activity.
- Plan file placement for a data warehouse.
- Plan partitioning for data warehouse tables.
- Design effective indexes for data warehouse queries.
- Plan data compression for a data warehouse.
- Design views in a data warehouse.

Data Warehouses I/O Activity

Before designing the physical database for the data warehouse, it is useful to consider the types of workload it must support and the data it must store. The database must store potentially very large fact tables with millions of rows, and dimension tables that are often related to fact tables by a single join. Typically, the I/O activity in the database is generated by one of the workloads described in this section or caused by maintenance operations, such as backups.



ETL

ETL processes affect the data warehouse when they load new or updated data into the data warehouse tables. In most cases, the inserts are performed as bulk load operations to minimize logging and constraint checking. The load process may involve some lookup operations to find alternate keys for slowly-changing dimensions, and some update operations for Type 1 dimension changes or data modifications in fact tables where appropriate. Depending on the design of the data structures, ETL load operations might also involve dropping and rebuilding indexes and splitting partitions.

Data models

After each new load, any data models based on the data warehouse must be processed. This involves reading data from the data warehouse tables into the data model and pre-aggregating measures to optimize analysis queries. Depending on the size of the data warehouse and the time window for the processing operation, the entire data model may be completely processed after each load, or an incremental approach may be used, in which only new or modified data is handled.

Because of the volume of data being loaded into the model, the I/O activity typically involves sequential table scans to read entire tables, particularly when performing a full process of the data model.

Reports

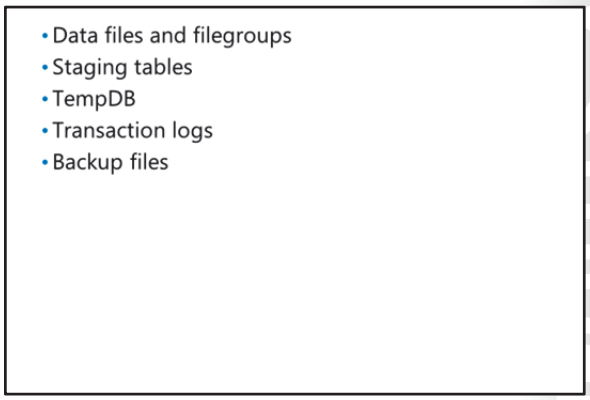
In some scenarios, all reporting is performed against the data models, so it does not affect the data warehouse tables. However, it is common for some reports to query the data warehouse directly. In scenarios where IT-provided reports are supported, the queries are generally predictable and retrieve many rows with range-based query filters, often on a date field.

User queries

If self-service reporting is supported, users may be able to execute queries in the data warehouse or use tools that generate queries on their behalf. Depending on the query expertise of the users, this can result in complex, unpredictable queries.

Consideration for Database Files

In Module 2: *Planning BI Infrastructure*, some key considerations for data warehouse storage hardware were discussed and it was recommended that storage be provided by multiple disks configured as RAID 10 or RAID 5 arrays. This storage is presented to the data warehouse server as multiple logical disks that are sometimes referred to as logical unit numbers (LUNs), though technically a LUN is used to identify a unit of SCSI-based storage. When designing the file placement for your data warehouse, you must decide how best to use these logical disks.



Data files and filegroups

Data files are used to pre-allocate disk storage for database objects. When planning files for a data warehouse, consider the following guidelines:

- Create files with an initial size, based on the eventual size of the objects that will be stored on them. This pre-allocates sequential disk blocks and helps avoid fragmentation.
- Disable autogrowth. If you begin to run out of space in a data file, it is more efficient to explicitly increase the file size by a large amount rather than rely on incremental autogrowth.

Because the logical disks for the database files are typically already configured as RAID 10 or RAID 5 arrays, you do not need to use filegroups to distribute tables across physical disk platters to improve I/O performance. However, you should consider the following guidance for using filegroups in a data warehouse:

- Create at least one filegroup in addition to the primary one, and then set it as the default filegroup so you can separate data tables from system tables.
- Consider creating dedicated filegroups for extremely large fact tables and using them to place those fact tables on their own logical disks.
- If some tables in the data warehouse are loaded on a different schedule from others, consider using filegroups to separate the tables into groups that can be backed up independently.
- If you intend to partition a large fact table, create a filegroup for each one so that older, stable rows can be backed up, and then set as read-only.

Staging tables

Most data warehouses require staging tables to support incremental data loads from the ETL process. In some cases, you might use a separate staging database as well as staging tables in the data warehouse itself. Consider the following recommendations for staging tables:

- If a separate staging database is to be used, create it on a logical disk distinct from the data warehouse files.
- If the data warehouse will include staging tables, create a file and filegroup for them on a logical disk, separate from the fact and dimension tables.
- An exception to the previous guideline is made for staging tables that will be switched with partitions to perform fast loads. These must be created on the same filegroup as the partition with which they will be switched.

TempDB

TempDB is used for temporary objects required for query processing. To avoid fragmentation of data files, place it on a dedicated logical disk and set its initial size based on how much it is likely to be used. You can leave autogrowth enabled, but set the growth increment to be quite large to ensure that performance is not interrupted by frequent growth of TempDB. Additionally, consider creating multiple files for TempDB to help minimize contention during page free space (PFS) scans as temporary objects are created and dropped.

Transaction logs

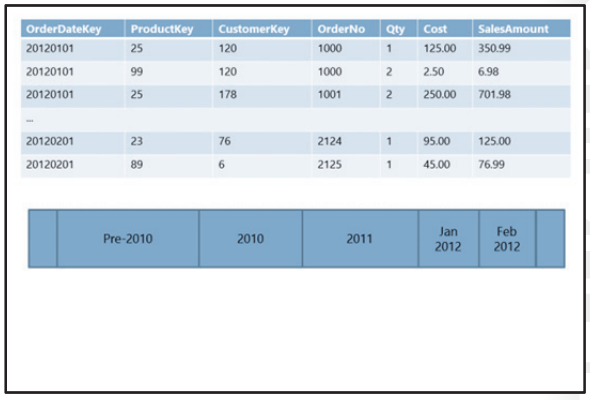
Generally, the transaction mode of the data warehouse, staging database, and TempDB should be set to **Simple** to avoid having to truncate transaction logs. Additionally, most of the inserts in a data warehouse are typically performed as bulk load operations which are not logged. To avoid disk resource conflicts between data warehouse I/O and logging, place the transaction log files for all databases on a dedicated logical disk.

Backup files

You will need to implement a backup routine for the data warehouse, and potentially for a staging database. In most cases, you will back up these databases to disk, so allocate a logical disk for this purpose. You could allocate multiple logical disks and perform a mirrored backup, but because the disks are already configured as RAID 5 or RAID 10 arrays, this would be of little benefit from a performance perspective. Note that the backup files should be copied to offsite storage to provide protection in the case of a complete storage hardware failure or natural disaster.

Table Partitioning

Partitioning a table distributes data based on a function that defines a range of values for each partition. A partition scheme maps partitions to filegroups, and the table is partitioned by applying the partition scheme to the values in a specified column.



Note: For information about how to implement partitioning, see *Partitioned Tables and Indexes* in SQL Server Books Online.

Why use partitioning?

Partitioning a large table can produce the following benefits:

- **Improved query performance**. By partitioning a table across filegroups, you can place specific ranges of data on different disk spindles, which can improve I/O performance. In most data warehouses, the disk storage is already configured as a RAID 10 or RAID 5 array, so this usually has little benefit. However, when using a mix of fast solid state storage for recent, frequently- accessed data, and mechanical disks for older, less queried rows, you can use partitioning to balance disk performance against storage costs. The biggest performance gain from partitioning in a data warehouse is realized when queries return a range of rows that are filtered on the partitioning key. In this case, the query optimizer can eliminate partitions that are not within the filter range, and dramatically reduce the number of rows that need to be read.
- **More granular manageability**. When you partition a large table, you can perform some maintenance operations at the partition level instead of on the whole table. For example, indexes can be created and rebuilt on a per-partition basis, and compression can be applied to individual partitions. It is also possible to back up and restore partitions independently by mapping them to filegroups. This enables you to back up older data once, and then configure the backed-up partitions as read-only. Future backups can be limited to the partitions that contain new or updated data.
- **Improved data load performance**. The biggest benefit of using partitioning in a data warehouse is that it enables you to load many rows quickly by switching a staging table with a partition. This technique dramatically reduces the time taken by ETL data loads, and with the right planning, it can be achieved with minimal requirements to drop or rebuild indexes.

Best practices for partitioning in a data warehouse

When planning a data warehouse, consider the following best practices for partitioning:

- **Partition large fact tables**. Fact tables of around 50 GB or more should usually be partitioned for the reasons described earlier. In general, fact tables benefit from partitioning more than dimension tables.
- **Partition on an incrementing date key**. When defining a partition scheme for a fact table, use a date key that reflects the age of the data as it is incrementally loaded by the ETL process. For example, if a fact table contains sales order data, partitioning on the order date ensures that the most recent orders are in the last partition and the earliest ones are in the first.
- **Design the partition scheme for ETL and manageability**. In a data warehouse, the query performance gains realized by partitioning are small compared to the manageability and data load performance benefits. Ideally, your partitions should reflect the ETL load frequency (for example, monthly, weekly, daily) because this simplifies the load process. However, you may want to merge partitions periodically to reduce their overall number. For example, at the start of each year, you could merge the monthly partitions for the previous year into a single partition for the whole year.
- **Maintain an empty partition at the start and end of the table**. You can use an empty partition at the end of the table to simplify the loading of new rows. When a new set of fact table rows must be loaded, you can place them in a staging table, split the empty partition (to create two empty partitions), and then switch the staged data with the first empty partition. This loads the data into the table and leaves the second empty partition you created at the end of the table ready for the next load. You can use a similar technique to archive or delete obsolete data at the beginning of the table.

 **Note:** Partitioning is only available in SQL Server Enterprise edition. In previous releases of SQL Server Enterprise edition, the number of partitions per table was limited to 1,000. In SQL Server 2012, this limit has been extended to 15,000. On 32-bit systems, you can create a table or index with more than 1,000 partitions, but this is not supported.

Demonstration: Partitioning a Fact Table

This demonstration shows how to create and use a partitioned table.

Demonstration Steps

Create a Partitioned Table

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. Start SQL Server Management Studio and connect to the **MIA-SQL** instance of the database engine by using Windows authentication.
3. Open **Partitions.sql** from the D:\Demofiles\Mod03 folder.
4. Select the code under the comment **Create a database**, and then click **Execute**. This creates a database for the demonstration.
5. Select the code under the comment **Create filegroups**, and then click **Execute**. This creates four filegroups in the demo database.
6. Select the code under the comment **Create partition function and scheme**, and then click **Execute**. This creates a partition function that defines four ranges of values (less than 20000101, 20000101 to 20010100, 20010101 to 20020100, and 20020101 and higher), and a partition scheme that maps these ranges to the FG0000, FG2000, FG2001, and FG2002 filegroups.
7. Select the code under the comment **Create a partitioned table**, and then click **Execute**. This creates a partitioned table on the partition scheme you created previously.
8. Select the code under the comment **Insert data into the partitioned table**, and then click **Execute**. This inserts four records into the table.

View Partition Metadata

1. Select the code under the comment **Query the table**, and then click **Execute**. This retrieves rows from the table and uses the **$PARTITION** function to show which partition the **datekey** value in each row is assigned to. This function is useful for determining which partition of a partition function a specific value belongs in.
2. Select the code under the comment **View filegroups, partitions, and rows**, and then click **Execute**. This code uses system tables to show the partitioned storage and the number of rows in each partition. Note that there are two empty partitions, one at the beginning of the table, and one at the end.

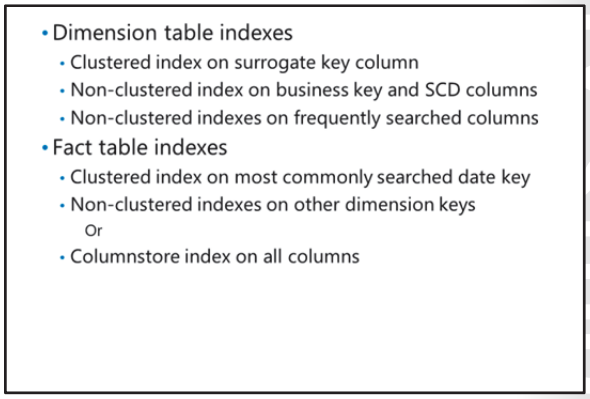
Split a Partition

1. Select the code under the comment **Add a new filegroup and make it the next used**, and then click **Execute**. This creates a new filegroup named FG2003 and adds it to the partition scheme as the next used partition.
2. Select the code under the comment **Split the empty partition at the end**, and then click **Execute**. This creates a new partition for values of 20030101 and higher and assigns it to the next used filegroup (FG2003), leaving an empty partition for values between 20020101 and 20030100.
3. Select the code under the comment **Insert new data**, and then click **Execute**. This inserts two new rows into the partitioned table.
4. Select the code under the comment **View partition metadata**, and then click **Execute**. This shows that the two rows inserted in the previous step are in partition 4, and that partition 5 (on FG2003) is empty.

Merge Partitions

1. Select the code under the comment **Merge the 2000 and 2001 partitions**, and then click **Execute**. This merges the partition that contains the value 20010101 into the previous partition.
2. Select the code under the comment **View partition metadata**, and then click **Execute**. This shows that partition 2 (on FG2000) now contains four rows, and that the partition previously on FG2001 has been removed.
3. Close **Partitions.sql** but keep SQL Server Management Studio open for the next demonstration.

Considerations for Indexes



Most databases use indexes to maximize query performance, and planning them is an important part of the database design process. Before deciding which indexes to create, you need to understand the workloads the database must support, and balance the need for improved query performance against the effect that indexes will have on data inserts and updates, as well as the overhead of maintaining indexes.

At first glance, a data warehouse seems to support mostly read operations, so some inexperienced BI professionals are tempted to create many indexes on all tables to support queries. However, another significant workload in most data warehouses is the regular ETL data load, which can often involve many inserts and updates. Too many indexes can slow down the ETL load process, and the need to periodically reorganize or rebuild indexes can create a significant maintenance overhead.

The first consideration is to determine whether any indexes are required in your data warehouse. It may seem unconventional to consider not creating indexes, but if the volume of fact data is relatively small, and all user access to the data is through a data model that is fully processed after each data load, there may be little performance benefit in maintaining indexes in the data warehouse. However, if your data warehouse does not match this restrictive description, you will probably need to consider creating some indexes. As with any database, the indexes you create depend on the specific queries your data warehouse must support and the need to balance the performance of those queries against data inserts and updates, and index maintenance. However, in most data warehouse scenarios, you should consider the guidelines in this topic as a starting point for index design.

Dimension table indexes

When designing indexes for dimension tables, consider the following guidelines:

- Create a clustered index on the surrogate key column. This column is used to join the dimension table to fact tables, and a clustered index will help the query optimizer minimize the number of reads required to filter fact rows.
- Create a non-clustered index on the alternate key column and include the SCD current flag, start date, and end date columns. This index will improve the performance of lookup operations during ETL data loads that need to handle slowly-changing dimensions.
- Create non-clustered indexes on frequently searched attributes, and consider including all members of a hierarchy in a single index.

Fact table indexes

When designing indexes for a fact table, consider the following guidelines:

- Create a clustered index on the most commonly-searched date key. Date ranges are the most common filtering criteria in most data warehouse workloads, so a clustered index on this key should be particularly effective in improving overall query performance.
- Create non-clustered indexes on other, frequently-searched dimension keys.

Columnstore indexes

SQL Server 2014 supports columnstore indexes, an in-memory solution that uses xVelocity compression technology to organize index data in a column-based format instead of the row-based format used by traditional indexes. Columnstore indexes are specifically designed to enhance the performance of queries against large fact tables joined to smaller dimension tables in a star schema, and can improve the speed of most data warehouse queries significantly. In many cases, you can achieve the same performance improvements or better by replacing the recommended fact table indexes described previously with a single columnstore index that includes all the fact table columns. Some queries do not benefit from columnstore indexes. For example, queries that return an individual row from a dimension table will generally perform better by using a conventional clustered or non-clustered index. However, for most typical data warehouse queries that aggregate many fact rows by one or more dimension attributes, columnstore indexes can be very effective.

 **Note:** For more information about columnstore indexes, go to *Columnstore Indexes* in SQL Server Books Online.

Columnstore indexes can be clustered or non-clustered.

Clustered columnstore indexes

A clustered columnstore index has the following characteristics:

- It can only be created in the Enterprise, Developer, and Evaluation editions of SQL Server 2014.
- It includes all the columns in the table.
- It is the only index on the table.
- It does not store the columns in a sorted order, but rather optimizes storage for compression and performance.
- It can be updated.

 **Note:** Clustered columnstore indexes are new in SQL Server 2014. In SQL Server 2012, only non-clustered columnstore indexes can be created.

Clustered columnstore indexes can be updated, and you can bulk load, insert, update, and delete data in a clustered columnstore indexed table using standard Transact-SQL statements.

Clustered columnstore indexes store the data in compressed columnstore segments, but some data is stored in a rowstore table referred to as the "deltastore" as an intermediary location until it can be compressed and moved into a columnstore segment. The following rules are used to manage data modifications:

- When you use an INSERT statement to insert a new row, it remains in the deltastore until there are enough rows to meet the minimum size for a "rowgroup", which is then compressed and moved into the columnstore segments.

- When you execute a DELETE statement, affected rows in the deltastore are physically deleted, while affected data in the columnstore segments are marked as deleted and the physical storage is only reclaimed when the index is rebuilt.
- When you execute an UPDATE statement, affected rows in the deltastore are updated, while affected rows in the columnstore are marked as deleted and a new row is inserted into the deltastore.

Non-clustered columnstore indexes

A non-clustered columnstore index has the following characteristics:

- It can include some or all the columns in the table.
- It can be combined with other indexes on the same table.
- It cannot be updated. Tables containing a non-clustered columnstore index are read-only.

Non-clustered columnstore indexes are read-only, but given that a typical data warehouse is a static database that is updated periodically through an ETL process, this is less of a limitation than might at first appear. However, administrators do need to plan how to handle updates to data in tables that have non-clustered columnstore indexes.

There are two ways to update non-clustered columnstore indexes:

- Periodically drop the index, perform updates to the table, and then recreate the index. This approach is the simplest way of handling updates, and fits in with the way that many organizations already perform loads into their data warehouses. The disadvantage of this approach is that creating a columnstore index can be time consuming when the base table is very large, and this can be problematic when the window for performing a data load is relatively small.
- Use table partitioning. When you create an index on a partitioned table, SQL Server automatically aligns the index with the table, meaning both are divided up in the same way. When you switch a partition out of the table, the aligned index partition follows. You can use partition switching to perform inserts, updates, merges, and deletes:
    - To perform a bulk insert, partition the table, load new data into a staging table, build a columnstore index on the staging table, and then use partition switching to load the data into the partitioned data warehouse table.
    - For other types of updates, you can switch a partition out of the data warehouse table into a staging table, drop or disable the columnstore index on the staging table, perform the updates, recreate or rebuild the columnstore index on the staging table, and then switch the staging table back into the data warehouse table.

Demonstration: Creating Indexes

This demonstration shows how to create indexes and assess their performance benefits.

Demonstration Steps

Create Indexes on Dimension Tables

1. Ensure that you have completed the previous demonstration in this module.
2. In SQL Server Management Studio, open **Indexes.sql** from the D:\Demofiles\Mod03 folder.
3. Select the code under the comment **Create the data warehouse**, and then click **Execute**. This creates a database for the demonstration.
4. Select the code under the comment **Create the DimDate dimension table**, and then click **Execute**. This creates a time dimension table named **DimDate**.

5. Select the code under the comment **Populate DimDate with values from 2 years ago until the end of this month**, and then click **Execute**. This adds rows to the **DimDate** table.
6. Select the code under the comment **Create indexes on the DimDate table**, and then click **Execute**. This creates a clustered index on the surrogate key column, and non-clustered indexes on commonly-queried attribute columns.
7. Select the code under the comment **Create the DimCustomer table**, and then click **Execute**. This creates a dimension table named **DimCustomer** and inserts some customer data.
8. Select the code under the comment **Create indexes on the DimCustomer table**, and then click **Execute**. This creates a clustered index on the surrogate key column, and non-clustered indexes on commonly-queried attribute columns.
9. Select the code under the comment **Create the DimProduct table**, and then click **Execute**. This creates a dimension table named **DimProduct** and inserts some product data.
10. Select the code under the comment **Create indexes on the DimProduct table**, and then click **Execute**. This creates a clustered index on the surrogate key column, and non-clustered indexes on a commonly-queried attribute column.

View Index Usage and Execution Statistics

1. Select the code under the comment **Create a fact table**, and then click **Execute**. This creates a fact table named **FactOrder** that contains more than 7.5 million rows from the existing data in the dimension tables.
2. On the toolbar, click the **Include Actual Execution Plan** button.
3. Select the code under the comment **View index usage and execution statistics**, and then click **Execute**. This enables statistics messages and queries the tables in the data warehouse to view orders for the previous six months.
4. After query execution completes, in the results pane, click the **Messages** tab. Note the logical reads from each table. The number from the **FactOrder** table should be considerably higher than the dimension tables. Note the CPU time and elapsed time for the query.
5. Click the **Execution Plan** tab, which shows a visualization of the steps the query optimizer used to execute the query. Scroll to the right and to the bottom, and note that a table scan was used to read data from the **FactOrder** table. Then hold the mouse pointer over each of the **Index Scan** icons for the dimension tables to see which indexes were used.
6. Execute the selected code again and compare the results when the data is cached.

Create Indexes on a Fact Table

1. Select the code under the comment **Create traditional indexes on the fact table**, and then click **Execute**. This creates a clustered index on the date dimension key, and non-clustered indexes on the other dimension keys (the operation can take a long time).
2. Select the code under the comment **Empty the cache**, and then click **Execute**. This clears any cached data.
3. Select the code under the comment **Test the traditional indexes**, and then click **Execute**. This executes the same query as earlier.
4. Click the **Messages** tab and compare the number of logical reads for the **FactOrders** table and the CPU and elapsed time values with the previous execution. They should all be lower.
5. Click the **Execution Plan** tab and note that the clustered index on the date key in the fact table was used.

6. Execute the selected code again and compare the results when the data is cached.

Create a Columnstore Index

1. Select the code under the comment **Create a copy of the fact table with no indexes**, and then click **Execute**. This creates an un-indexed copy of the **FactOrder** table named **FactOrderCS**.
2. Select the code under the comment **Create a columnstore index on the copied table**, and then click **Execute**. This creates a columnstore index on all columns in the **FactOrderCS** table.
3. Select the code under the comment **Empty the cache again**, and then click **Execute**. This clears any cached data.
4. Select the code under the comment **Test the columnstore index**, and then click **Execute**. This executes the same query as earlier.
5. Click the **Messages** tab and compare the number of logical reads for the **FactOrdersCS** table and the CPU and elapsed time values with the previous execution. They should all be lower.
6. Click the **Execution Plan** tab and note that the columnstore index on the fact table was used.
7. Execute the selected code again and compare the results when the data is cached.
8. Close **Indexes.sql** but keep SQL Server Management Studio open for the next demonstration.

Data Compression

SQL Server 2014 Enterprise edition supports data compression at both page and row level. Row compression stores all fields in a variable width format and, if possible, reduces the number of bytes used to store each field. Page compression applies the same technique to rows on a page and also identifies redundant values and stores them only once per page. You can apply compression to a table, an index, or a partition.

- Apply page compression on all dimension tables, indexes, and fact table partitions
- If performance becomes CPU-bound, fall back to row compression on the most queried partitions

Data compression in a data warehouse brings the following benefits:

- **Reduced storage requirements**. Although results vary, on average, most data warehouses can be compressed at a ratio of 3.5:1, reducing the amount of disk space required to host the data files by more than two thirds.
- **Improved query performance**. Compression can improve query performance in two ways: Fewer pages must be read from disk, so I/O is reduced, and more data can be stored on a page and cached.

When page or row compression is used, data must be compressed and decompressed by the CPU. Performance gains resulting from compression must be balanced by the increase in CPU workload. However, in most adequately-specified data warehouse servers, the additional workload on CPU is minimal compared to the benefits gained by compressing the data.

Best practices for data compression in a data warehouse

When planning tables, partitions, and indexes in a data warehouse, consider the following best practices for data compression:

- Use page compression on all dimension tables and fact table partitions.
- If performance is CPU-bound, revert to row compression on frequently-accessed partitions.

Demonstration: Implementing Data Compression

This demonstration shows a comparison between an uncompressed data warehouse and an identical compressed one.

Demonstration Steps

Create Uncompressed Tables and Indexes

1. Ensure that you have completed the previous demonstrations in this module.
2. Use Windows Explorer to view the contents of the D:\Demofiles\Mod03 folder, and set the folder window to Details view and resize it if necessary so that you can see the **Size** column.
3. In SQL Server Management Studio, open **Compression.sql** from the D:\Demofiles\Mod03 folder.
4. Select the code under the comment **Create the data warehouse** (from line2 to line 113 in the script), and then click **Execute**. This creates a database with uncompressed tables.
5. While the script is still executing, view the contents of the D:\Demofiles\Mod03 folder and note the increasing size of DemoDW.mdf. This is the data file for the database.

   Note: The log file (DemoDW.ldf) will also be growing, but you can ignore this.
6. When execution is complete (after approximately three minutes), view the final size of DemoDW.mdf and return to SQL Server Management Studio.

Estimate Compression Savings

1. Select the code under the comment **Estimate size saving** (line 119 in the script), and then click **Execute**. This uses the **sp_estimate_data_compression_savings** system stored procedure to compress a sample of the **FactOrder** table (which consists of one clustered and two non-clustered indexes).
2. View the results returned by the stored procedure, noting the current size and estimated compressed size of each index.

Create Compressed Tables and Indexes

1. Select the code under the comment **Create a compressed version of the data warehouse** (from line 125 to line 250 in the script), and then click **Execute**. This creates a database with compressed tables and indexes.
2. While the script is still executing, view the contents of the D:\Demofiles\Mod03 folder and note the increasing size of CompressedDemoDW.mdf. This is the data file for the database.

   Note: The log file (CompressedDemoDW.ldf) will also be growing, but you can ignore this.
3. When execution is complete (after approximately three minutes), compare the final size of CompressedDemoDW.mdf with DemoDW.mdf (the file for the compressed database should be smaller) and return to SQL Server Management Studio.

Compare Query Performance

1. Select the code under the comment **Compare query performance** (from line 255 to line 277 in the script), and then click **Execute**. This executes an identical query in the compressed and uncompressed databases and displays execution statistics.
2. When execution is complete, click the **Messages** tab and compare the statistics for the two queries. The execution time statistics (the second and third set of figures labeled "SQL Server Execution Time") should be similar, and the second query (in the compressed database) should have used considerably less logical reads for each table than the first.

3. Close SQL Server Management Studio.

Using Views to Abstract Base Tables

You can create views in a data warehouse to abstract the underlying fact and dimension tables. Although views are not always necessary, you should consider the following guidelines when planning a data warehouse:

```
CREATE VIEW dw_views.SalesOrder
WITH SCHEMABINDING
AS
SELECT [OrderDateKey]
    ,[ProductKey]
    ,[ShipDateKey]
    ,[CustomerKey]
    ,[OrderNumber]
    ,[OrderQuantity]
    ,[UnitPrice]
    ,[SalesAmount]
FROM [dbo].[FactSalesOrder]
WITH (NOLOCK)
```

- **Create a view for each dimension and fact table, and use the NOLOCK query hint in the view definition**. You can then use these views instead of the base tables for all data access from clients, which will eliminate locking overhead and optimize concurrency.
- **Create views with user-friendly view and column names**. Often, a naming convention (such as prefixing dimension tables with "dim" and fact tables with "fact") is used when creating the tables in a data warehouse. Such naming conventions are useful for database designers and administrators, but they can confuse business users. Creating a layer of views with accessible names makes it easier for users to create their own data models and reports from the data warehouse.
- **Do not include metadata columns in views**. Some columns are used for ETL operations or other administrative tasks, and can be omitted from views that will be consumed by business users. For example, SCD current flag, start date, and end date columns may not be required for end user reporting or data models, so you can create views that do not include them.
- **Create views to combine snowflake dimension tables**. If you have included snowflake dimensions in your dimensional model, create a view for each set of related dimension tables to produce a single logical dimension table.
- **Partition-align indexed views**. SQL Server supports indexed views, which can be partitioned using the same partition scheme as the underlying table. If you use indexed views, you should partition-align them to support partition switching that does not require indexes on the views to be dropped and recreated.
- **Use the SCHEMABINDING option**. This ensures that the underlying tables cannot be dropped or modified in such a way as to invalidate the view unless the view itself is dropped first. The SCHEMABINDING option is a requirement for index views.

Lab: Implementing a Data Warehouse

Scenario

You have gathered analytical and reporting requirements from stakeholders at Adventure Works Cycles. Now you must implement a data warehouse schema to support them.

Objectives

After completing this lab, you will be able to:

- Implement a dimensional star schema.
- Implement a snowflake schema.
- Implement a time dimension.

Estimated Time: 45 Minutes

Virtual machine: 20463C-MIA-SQL

User name: ADVENTUREWORKS\Student

Password: Pa$$w0rd

Exercise 1: Implement a Star Schema

Scenario

Adventure Works Cycles requires a data warehouse to enable information workers and executives to create reports and perform analysis of key business measures. The company has identified two sets of related measures that it wants to include in fact tables. These are separate sales order measures relating to sales to resellers, and Internet sales. The measures will be aggregated by product, reseller (in the case of reseller sales), and customer (for Internet sales) dimensions.

The data warehouse has been partially completed, and you must now add the necessary dimension and fact tables to complete a star schema.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. View a Data Warehouse Schema
3. Create a Dimension Table
4. Create a Fact Table
5. View the Revised Data Warehouse Schema

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. Run **Setup.cmd** in the D:\Labfiles\Lab03\Starter folder as Administrator.

▶ Task 2: View a Data Warehouse Schema

1. Start SQL Server Management Studio and connect to the **MIA-SQL** instance of the SQL Server database engine by using Windows authentication.
2. Create a new database diagram in the **AWDataWarehouse** database (creating the required objects to support database diagrams if prompted). The diagram should include all the tables in the database.

3. In the database diagram, modify the tables so they are shown in standard view, and arrange them so you can view the partially complete data warehouse schema.
4. Save the database diagram as **AWDataWarehouse Schema**.

▶ Task 3: Create a Dimension Table

1. Review the Transact-SQL code in the **DimCustomer.sql** query file in the D:\Labfiles\Lab03\Starter folder. Note that it creates a table named **DimCustomer** in the **AWDataWarehouse** database.
2. Execute the query to create the **DimCustomers** dimension table.

▶ Task 4: Create a Fact Table

1. Review the Transact-SQL code in the **FactInternetSales.sql** query file in the D:\Labfiles\Lab03\Starter folder. Note that it creates a table named **FactInternetSales** in the **AWDataWarehouse** database, and that this table is related to the **DimCustomer** and **DimProduct** tables.
2. Execute the query to create the **FactInternetSales** dimension table.

▶ Task 5: View the Revised Data Warehouse Schema

1. Add the tables that you have created in this exercise to the database diagram that you created.

Note: When adding tables to a diagram, you need to click **Refresh** in the **Add Table** dialog box to see tables you have created or modified since the diagram was initially created.

2. Save the database diagram.
3. Keep SQL Server Management Studio open for the next exercise.

Results: After this exercise, you should have a database diagram in the **AWDataWarehouse** database that shows a star schema consisting of two fact tables (**FactResellerSales** and **FactInternetSales**) and four dimension tables (**DimReseller**, **DimEmployee**, **DimProduct**, and **DimCustomer**).

Exercise 2: Implementing a Snowflake Schema

Scenario

Having created a star schema, you have identified two dimensions that would benefit from being normalized to create a snowflake schema. Specifically, you want to create a hierarchy of related tables for product category, product subcategory, and product. You also want to create a separate geography dimension table that can be shared between the reseller and customer dimensions.

The main tasks for this exercise are as follows:

1. Create Dimension Tables That Form a Hierarchy
2. Create a Shared Dimension table
3. View the Data Warehouse Schema

▶ Task 1: Create Dimension Tables That Form a Hierarchy

1. Review the Transact-SQL code in the **DimProductCategory.sql** query file in the D:\Labfiles\Lab03\Starter folder. Note that it:
   - Creates a table named **DimProductCategory**.
   - Creates a table named **DimProductSubcategory** that has a foreign-key relationship to the **DimProductCategory** table.

- Drops the **ProductSubcategoryName** and **ProductCategoryName** columns from the **DimProduct** table.
- Adds a **ProductSubcategoryKey** column to the **DimProduct** table that has a foreign-key relationship to the **DimProductSubcategory** table.

2. Execute the query to create the dimension tables.

▶ Task 2: Create a Shared Dimension table

1. Review the Transact-SQL code in the **DimGeography.sql** query file in the D:\Labfiles\Lab03\Starter folder. Note that it:
   - Creates a table named **DimGeography**.
   - Drops the **City**, **StateProvinceName**, **CountryRegionCode**, **CountryRegionName**, and **PostalCode** columns from the **DimReseller** table.
   - Adds a **GeographyKey** column to the **DimReseller** table that has a foreign-key relationship to the **DimGeography** table.
   - Drops the **City**, **StateProvinceName**, **CountryRegionCode**, **CountryRegionName**, and **PostalCode** columns from the **DimCustomer** table.
   - Adds a **GeographyKey** column to the **DimCustomer** table that has a foreign-key relationship to the **DimGeography** table.
2. Execute the query to create the dimension table.

▶ Task 3: View the Data Warehouse Schema

1. Delete the tables that you modified in the previous two tasks from the **AWDataWarehouse Schema** diagram (**DimProduct**, **DimReseller**, and **DimCustomer**).
2. Add the new and modified tables that you created in this exercise to the **AWDataWarehouse Schema** diagram and view the revised data warehouse schema, which now includes some snowflake dimensions. You will need to refresh the list of tables when adding tables and you may be prompted to update the diagram to reflect foreign-key relationships.
3. Save the database diagram.

Results: After this exercise, you should have a database diagram in the **AWDataWarehouse** database showing a snowflake schema that contains a dimension consisting of a **DimProduct**, **DimProductSubcategory**, and **DimProductCategory** hierarchy of tables, as well as a **DimGeography** dimension table that is referenced by the **DimCustomer** and **DimReseller** dimension tables.

Exercise 3: Implementing a Time Dimension Table

Scenario

The schema for the Adventure Works data warehouse now contains two fact tables and several dimension tables. However, users need to be able to analyze the fact table measures across consistent time periods. To enable this, you must create a time dimension table.

Users will need to be able to aggregate measures across calendar years (which run from January to December) and fiscal years (which run from July to June). Your time dimension must include the following attributes:

- Date (this should be the business key).
- Day number of week (for example 1 for Sunday, 2 for Monday, and so on).
- Day name of week (for example Sunday, Monday, Tuesday, and so on).
- Day number of month.
- Day number of year.
- Week number of year.
- Month name (for example, January, February, and so on).
- Month number of year (for example, 1 for January, 2 for February, and so on).
- Calendar quarter (for example, 1 for dates in January, February, and March).
- Calendar year.
- Calendar semester (for example, 1 for dates between January and June).
- Fiscal quarter (for example, 1 for dates in July, August, and September).
- Fiscal year.
- Fiscal semester (for example, 1 for dates between July and December).

The main tasks for this exercise are as follows:

1. Create a Time Dimension Table
2. View the Database Schema
3. Populate the Time Dimension Table

▶ Task 1: Create a Time Dimension Table

1. Review the Transact-SQL code in the **DimDate.sql** query file in the D:\Labfiles\Lab03\Starter folder. Note that it:
    - Creates a table named **DimDate**.
    - Adds **OrderDateKey** and **ShipDateKey** columns to the **FactInternetSales** and **FactResellerSales** tables that have foreign-key relationships to the **DimDate** table.
    - Creates indexes on the **OrderDateKey** and **ShipDateKey** foreign-key fields in the **FactInternetSales** and **FactResellerSales** tables.
2. Execute the query to create the dimension table.

▶ Task 2: View the Database Schema

1. Delete the tables that you modified in the previous two tasks from the **AWDataWarehouse Schema** diagram (**FactResellerSales** and **FactInternetSales**).
2. Add the new and modified tables that you created in this exercise to the **AWDataWarehouse Schema** diagram and view the revised data warehouse schema, which now includes a time dimension table named **DimDate**. You will need to refresh the list of tables when adding tables and you may be prompted to update the diagram to reflect foreign-key relationships.
3. Save the database diagram.

▶ Task 3: Populate the Time Dimension Table

1. Review the Transact-SQL code in the **GenerateDates.sql** query file in the D:\Labfiles\Lab03\Starter folder. Note that it:
   - Declares a variable named **@StartDate** with the value 1/1/2000, and a variable named **@EndDate** with the value of the current date.
   - Performs a loop to insert appropriate values for each date between **@StartDate** and **@EndDate** into the **DimDate** table.
2. Execute the script to create the dimension table.
3. When the query has completed, query the **DimDate** table to verify that it now contains time values.
4. Close Visual Studio, saving your work if prompted.

Results: After this exercise, you should have a database that contains a **DimDate** dimension table that is populated with date values from January 1, 2000, to the current date.

Module Review and Takeaways

This module has described considerations for translating business requirements and information about business processes into a dimensional model, and then implementing that model as a data warehouse. Every business is different, and each has its unique challenges and processes. You should use the techniques and guidance in this module as a starting point, but be prepared to adapt typical data warehouse schema elements to match particular business requirements.

Review Question(s)

Question: When designing a data warehouse, is it better or worse to have a strong background in transactional database design?

Module 4

Creating an ETL Solution with SSIS

Contents:

Module Overview

Successful data warehousing solutions rely on the efficient and accurate transfer of data from the various data sources in the business. This is referred to as an extract, transform, and load (ETL) process, a core skill that is required in any data warehousing project.

This module discusses considerations for implementing an ETL process, and then focuses on Microsoft® SQL Server® Integration Services (SSIS) as a platform for building ETL solutions.

Objectives

After completing this module, you will be able to:

- Describe the key features of SSIS.
- Explore source data for an ETL solution.
- Implement a data flow by using SSIS.

Lesson 1
Introduction to ETL with SSIS

There are several ways to implement an ETL solution, but SSIS is the primary ETL tool for SQL Server. Before using it to implement an ETL solution, it is important to understand some of the key features and components.

This lesson describes possible ETL solution options, and then introduces SSIS.

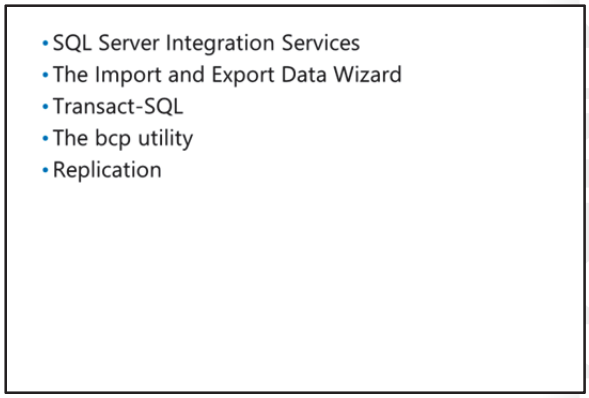
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the options for ETL.
- Describe the key features of SSIS.
- Describe the high-level architecture of an SSIS project.
- Identify key elements of the SSIS design environment.
- Describe strategies for upgrading SSIS solutions from previous versions of SQL Server.

Options for ETL

An ETL solution generally involves the transfer of data from one or more data sources to a destination, often transforming the data structure and values in the process. There are several tools and technologies you can use to accomplish this task, each having specific strengths and weaknesses that you should take into account when choosing the approach for your ETL solution.



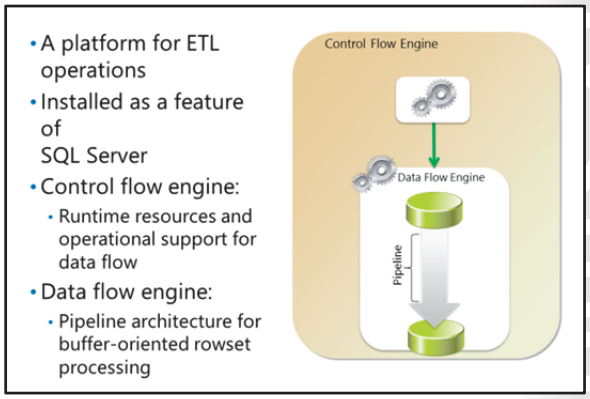
The following list describes some common techniques:

- **SQL Server Integration Services**. This is the primary platform for ETL solutions that are provided with SQL Server, and generally offers the most flexible way to implement an enterprise ETL solution.
- **The Import and Export Data Wizard**. This wizard is included with the SQL Server management tools, and provides a simple way to create an SSIS-based data transfer solution. You should consider using the Import and Export Data Wizard when your ETL solution requires only a few, simple data transfers that do not include any complex transformations in the data flow.
- **Transact-SQL**. Transact-SQL is a powerful language that can enable you to implement complex data transformations while extracting, inserting, or modifying data. Most ETL solutions include some Transact-SQL logic combined with other technologies. In some scenarios, such as when data sources and destinations are co-located, you can implement a complete ETL solution by using only Transact-SQL queries.

- **The bulk copy program (bcp) utility**. This utility provides an interface based on the command line for extracting data from, and inserting data into, SQL Server. The bcp utility provides a versatile tool to create scheduled data extractions and insertions, but its relatively complex syntax and console-based operation make it difficult to create a manageable, enterprise-scale ETL solution on its own.
- **Replication**. SQL Server includes built-in replication functionality that you can use to synchronize data across SQL Server instances. You can also include other relational data sources such as Oracle databases in a replication solution. Replication is a suitable technology for ETL when all data sources are supported in a replication topology, and the data requires minimal transformations.

What Is SSIS?

SSIS is an extensible platform for building complex ETL solutions. It is included with SQL Server and consists of a Microsoft Windows® service that manages the execution of ETL workflows, and several tools and components for developing them. The SSIS service is installed when you select **Integration Services** on the **Feature Selection** page of the SQL Server setup wizard.



Note: After you have installed SSIS, you can use the DCOM Configuration tool (Dcomcnfg.exe) to grant specific permission to use the SSIS 11.0 service.

The SSIS Windows service is primarily a control flow engine that manages the execution of task workflows, that are defined in packages, and can be performed on demand or at scheduled times. When you are developing an SSIS package, the task workflow is referred to as the control flow.

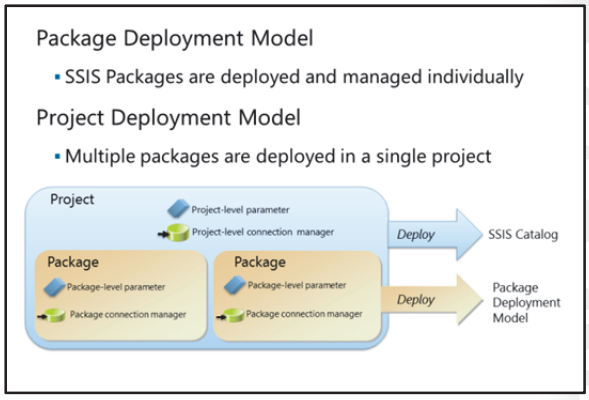
This can include a special type of task to perform data flow operations. SSIS executes these tasks using a data flow engine that encapsulates the data flow in a pipeline architecture. Each step in the Data Flow task operates in sequence on a rowset of data as it passes through the pipeline. The data flow engine uses buffers to optimize the data flow rate, resulting in a high-performance ETL solution.

In addition to the SSIS Windows service, SSIS includes:

- **SSIS Designer**. A graphical design interface for developing SSIS solutions in the Microsoft Visual Studio® development environment. Typically, you start the SQL Server Data Tools application to access this.
- **Wizards**. Graphical utilities you can use to quickly create, configure, and deploy SSIS solutions.
- **Command-line tools**. Utilities you can use to manage and execute SSIS packages.

SSIS Projects and Packages

An SSIS solution usually consists of one or more SSIS projects, each containing at least one SSIS package.



SSIS Projects

In SQL Server 2012, a project is the unit of deployment for SSIS solutions. You can define project-level parameters to enable users to specify run-time settings, and project-level connection managers that reference data sources and destinations used in package data flows. You can then deploy projects to a SSIS catalog in an SQL Server instance, and configure project-level parameter values and connections as appropriate for execution environments. You can use SQL Server Data Tools to create, debug, and deploy SSIS projects.

SSIS Packages

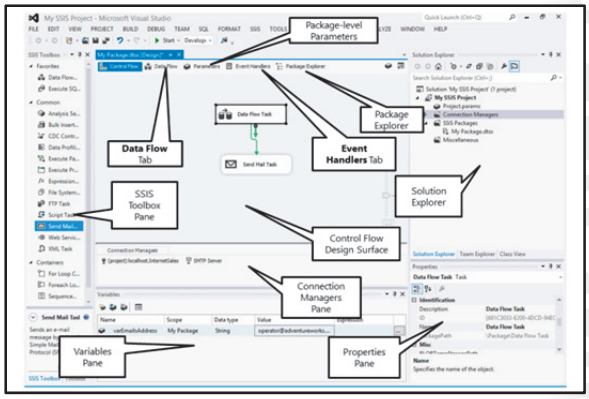
A project contains one or more packages, each defining a workflow of tasks to be executed. The workflow of tasks is referred to as its control flow. A package control flow can include one or more Data Flow tasks, each of which encapsulates its own pipeline. Package-level parameters can be included so that dynamic values are passed to the package at run time.

In previous SSIS releases, deployment was managed at the package level. In SQL Server 2012, you can still deploy individual packages in a package deployment model.

Note: Deployment of SSIS solutions is discussed in more detail in Module 12: *Deploying and Configuring SSIS Packages*.

The SSIS Design Environment

You can use Visual Studio SQL Server Data Tools for business intelligence (BI) to develop SSIS projects and packages. SQL Server Data Tools for BI is an add-in for Microsoft® Visual Studio® and provides a graphical development environment for BI solutions. When you create an Integration Services project, the design environment includes the following elements:

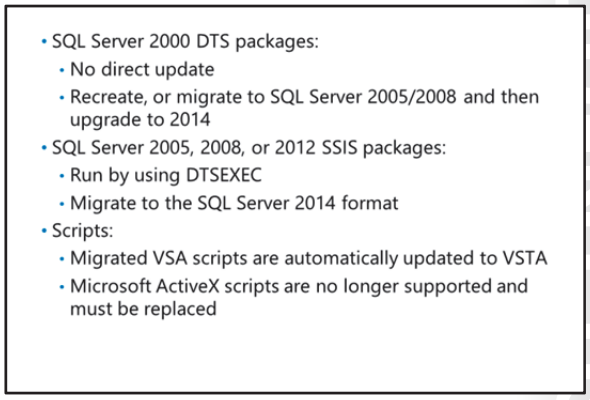


- **Solution Explorer**. A pane in the Visual Studio user interface where you can create and view project-level resources, including parameters, packages, data connection managers, and other shared objects. A solution can contain multiple projects, in which case each project is shown in Solution Explorer.
- **The Properties pane**. A pane in the Visual Studio user interface that you can use to view and edit the properties of the currently-selected object.

- **The Control Flow design surface**. A graphical design surface in SSIS Designer where you can define the workflow of tasks for a package.
- **The Data Flow design surface**. A graphical design surface in SSIS Designer where you can define the pipeline for a Data Flow task within a package.
- **The Event Handlers design surface**. A graphical design surface in SSIS Designer where you can define the workflow for an event handler within a package.
- **Package Explorer**. A tree view of the components within a package.
- **The Connection Managers pane**. A list of the connection managers used in a package.
- **The Variables pane**. A list of variables used in a package. You can display this pane by clicking the **Variables** button at the upper right of the design surface.
- **The SSIS Toolbox**. A collection of components you can add to a package control flow or data flow. You can display this pane by clicking the **SSIS Toolbox** button at the upper right of the design surface or by clicking **SSIS Toolbox** on the **SSIS** menu. Note that this pane is distinct from the standard Visual Studio Toolbox pane.

Upgrading from Previous Versions

If you have developed ETL solutions by using SSIS in SQL Server 2005, 2008, or 2012, or by using Data Transformation Services (DTS) solutions in SQL Server 2000, you should consider how you will include them in an SQL Server 2014 SSIS solution.



SQL Server 2000 DTS Packages

There is no direct upgrade path for DTS packages to SQL Server 2014 SSIS packages, and you cannot run a DTS package in the SQL Server 2014 SSIS run-time engine. To upgrade a DTS-based solution to work with SQL Server 2014, you must re-create the solution by using the latest SSIS tools and components. Alternatively, you can use the SSIS Package Migration Wizard in SQL Server 2005 or 2008 to perform an interim upgrade of the DTS package to SQL Server 2005 or 2008 format, and then upgrade to the SQL Server 2014 format.

SQL Server 2005, 2008, and 2012 SSIS Packages

You can run SSIS packages that were built by using SQL Server 2005, 2008, or 2012 in the SQL Server 2014 SSIS run-time engine by using the DTSEXEC tool. However, you will not be able to take advantage of project-level deployment for packages created in SQL Server 2005 or 2008. To upgrade SSIS packages that were built using SQL Server 2005, SQL Server 2008, or SQL Server 2012 to the SQL Server 2014 format, use the SSIS Package Migration Wizard in SQL Server 2014.

Scripts

SSIS packages can include script tasks to perform custom actions. In previous releases of SSIS, you could implement scripted actions by including a Microsoft® ActiveX® Script task, written in Microsoft® Visual Basic® Scripting Edition (VBScript). You could also employ a Script task written for the .NET Visual Studio for Applications, or VSA runtime, in a control flow. In SQL Server 2014, the ActiveX Script task is no longer supported, and any VBScript-based custom logic must be replaced. In addition, the SQL Server 2014 Script task uses the Visual Studio Tools for Applications (VSTA) runtime, which differs in some details from the VSA runtime used in previous releases. When you use the SSIS Package Migration Wizard to upgrade a package that includes a Script component, the script is automatically updated for the VSTA runtime.

Lesson 2
Exploring Source Data

Now that you understand the basic architecture of SSIS, you can start planning the data flows in your ETL solution. However, before you start implementing an ETL process, you should explore the existing data in the sources that your solution will use. By gaining a thorough knowledge of the source data on which your ETL solution will be based, you can design the most effective SSIS data flows for transferring the data and anticipate any quality issues you may need to resolve in your SSIS packages.

This lesson discusses the value of exploring source data, as well as describing techniques for examining and profiling it.

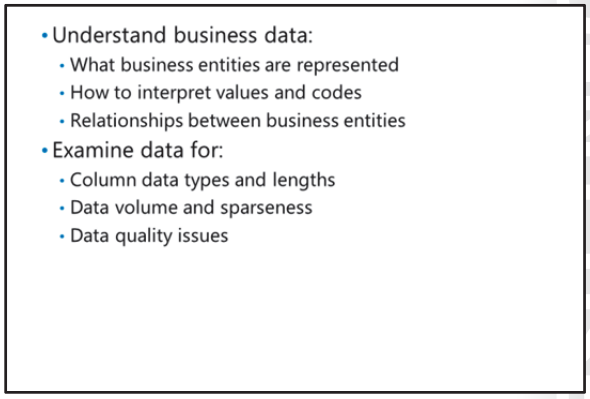
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the value of exploring source data.
- Examine an extract of data from a data source.
- Profile source data by using the Data Profiling SSIS task.

Why Explore Source Data?

The design and integrity of your data warehouse ultimately rely on the data it contains. Before you can design an appropriate ETL process to populate the data warehouse, you must have a thorough knowledge of the source data that your solution will consume.



Specifically, you need to understand:

- The business entities that are represented by the source data, and their attributes. For example, the specific attributes that fully describe a product or a customer entity may be stored in multiple columns, tables, or even databases across the organization.
- How to interpret data values and codes. For example, does a value of **1** in an **InStock** column in a **Products** table mean that the company has a single unit in stock, or does **1** simply indicate the value "true," meaning that there is an unspecified quantity of units in stock?
- The relationships between business entities, and how those relationships are modeled in the data sources.

In addition to understanding the data modeling of the business entities, you also need to examine source data to help identify:

- Column data types and lengths for specific attributes that will be included in data flows. For example, what maximum lengths exist for string values? What formats are used to indicate date, time, and numeric values?
- Data volume and sparseness. For example, how many rows of sales transactions are typically recorded in a single trading day? Are there any attributes that frequently contain null values?

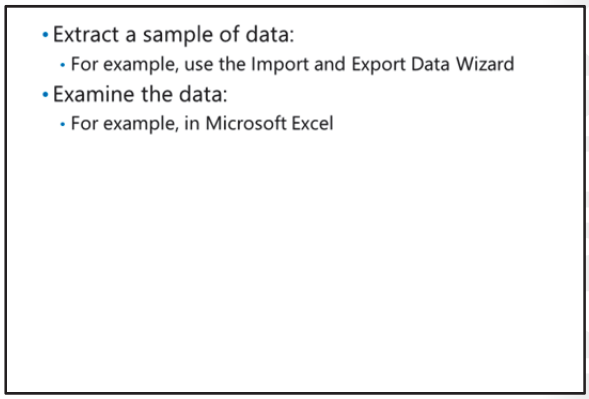
- Data quality issues. For example, are there any obvious data entry errors? Are there commonly-used values that are synonyms for one another?

Finding the answers to questions like these before you implement the ETL solution, can help you anticipate data flow problems and proactively design effective solutions for them.

Examining Source Data

You can explore source data by using several tools and techniques. The following list describes some of the approaches you can use to extract data to examine:

- Running queries against data sources in Microsoft® SQL Server® Management Studio and copying the results to the clipboard.
- Creating an SSIS package with a data flow that extracts a sampling of data or a row count for a specific data source.
- Using the Import and Export Data Wizard to extract a data sample.



After extracting the sample data, you need to examine it. One of the most effective ways to do this is to extract the data in a format you can open in Microsoft Excel®, such as comma-delimited text. Using Excel, you can:

- Sort the data by columns.
- Apply column filters to help identify the range of values used in a particular column.
- Use formulas to calculate minimum, maximum, and average values for numerical columns.
- Search the data for specific string values.

Demonstration: Exploring Source Data

In this demonstration, you will see how to:

- Extract Data with the Import and Export Data Wizard.
- Explore Data in Microsoft Excel®.

Demonstration Steps

Extract Data with the Import and Export Data Wizard

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Demofiles\Mod04 folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. When you are prompted to confirm, click **Yes**, and then wait for the batch file to complete.
4. On the **Start** screen, type **Import and Export**, and then start the **SQL Server 2014 Import and Export Data (64-bit)** app.
5. On the **Welcome to SQL Server Import and Export Wizard** page, click **Next**.

6. On the **Choose a Data Source** page, set the following options, and then click **Next**:
   - **Data source**: SQL Server Native Client 11.0
   - **Server name**: localhost
   - **Authentication**: Use Windows Authentication
   - **Database**: ResellerSales
7. On the **Choose a Destination** page, select the following options, and then click **Next**:
   - **Destination**: Flat File Destination
   - **File name**: D:\Demofiles\Mod04\Top 500 Resellers.csv
   - **Locale**: English (United States)
   - **Unicode**: Unselected
   - **Code page**: 1252 (ANSI – Latin 1)
   - **Format**: Delimited
   - **Text qualifier**: " (this is used to enclose exported text values in quotation marks. This is required because some European address formats include a comma, and these must be distinguished from the commas that are used to separate each column value in the exported text file)
   - **Column names in the first data row**: Selected
8. On the **Specify Table Copy or Query** page, select **Write a query to specify the data to transfer**, and then click **Next**.
9. On the **Provide a Source Query** page, enter the following Transact-SQL code, and then click **Next**:

```
SELECT TOP 500 * FROM Resellers
```

10. On the **Configure Flat File Destination** page, select the following options, and then click **Next**:
    - **Source query**: [Query]
    - **Row delimiter**: {CR}{LF}
    - **Column delimiter**: Comma {,}
11. On the **Save and Run Package** page, select only **Run immediately**, and then click **Next**.
12. On the **Complete the Wizard** page, click **Finish**.
13. When the data extraction has completed successfully, click **Close**.

Explore Source Data in Microsoft Excel®

1. Start Excel and open **Top 500 Resellers.csv** from the D:\Demofiles\Mod04 folder.
2. On the **Home** tab of the ribbon, click any cell that contains data, click **Format as Table**, and then select a table style for the data.
3. In the **Format As Table** dialog box, ensure that the range of cells containing the data is selected and the **My table has headers** check box is selected, and then click **OK**.
4. Adjust the column widths so that you can see all the data.

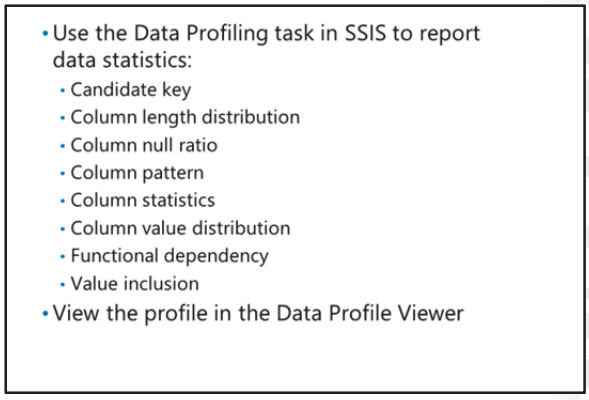
5. View the drop-down filter list for the **CountryRegionCode** column, and note the range of values. Then select only **FR**, and then click OK. Note that the table is filtered to show only the resellers in France. Note also that many of the addresses include a comma. If no text qualifier had been selected in the Import and Export Data Wizard, these commas would have created additional columns in these rows, making the data difficult to examine as a table.
6. Clear the filter for the **CountryRegionCode** column.
7. In a blank cell in column **O**, enter the following formula:

```
=Min(Table1[YearOpened])
```

8. Note that this formula shows the earliest year that a store in this sample data was opened.
9. Close Excel without saving any changes.

Profiling Source Data

In addition to examining samples of source data, you can use the Data Profiling task in an SSIS package to obtain statistics about the data. This can help you understand the structure of the data that you will extract and identify columns where null or missing values are likely. Profiling your source data can help you plan effective data flows for your ETL process.



You can specify multiple profile requests in a single instance of the Data Profiling task. The following kinds of profile request are available:

- **Candidate Key** determines whether you can use a column as a key for the selected table.
- **Column Length Distribution** reports the range of lengths for string values in a column.
- **Column Null Ratio** reports the percentage of null values in a column.
- **Column Pattern** identifies regular expressions that are applicable to the values in a column.
- **Column Statistics** reports statistics such as minimum, maximum, and average values for a column.
- **Column Value Distribution** reports the groupings of distinct values in a column.
- **Functional Dependency** determines if the value of a column is dependent on the value of other columns in the same table.
- **Value Inclusion** reports the percentage of time that a column value in one table matches a column in another.

The Data Profiling task gathers the requested profile statistics and writes them to an XML document. This can be saved as a file for later analysis, or written to a variable for programmatic analysis within the control flow.

To view the profile statistics, you can use the Data Profile Viewer. This is available as a stand-alone tool in which you can open the XML file that the Data Profiling task generates. Alternatively, you can open the Data Profile Viewer window in SQL Server Data Tools, from the **Properties** dialog box of the Data Profiling task, while the package is running in the development environment.

Use the following procedure to collect and view data profile statistics:

1. Create an SSIS project that includes a package.
2. Add an ADO.NET connection manager for each data source that you want to profile.
3. Add the Data Profiling task to the control flow of the package.
4. Configure the Data Profiling task to specify:
   a. The file or variable to which the resulting profile statistic should be written.
   b. The individual profile requests that should be included in the report.
5. Run the package.
6. View the resulting profile statistics in the Data Profile Viewer.

Demonstration: Using the Data Profiling Task

In this demonstration, you will see how to:

- Use the Data Profiling Task.
- View a Data Profiling Report.

Demonstration Steps

Use the Data Profiling Task

1. Ensure you have completed the previous demonstration in this module.
2. Start Visual Studio, and then create a new Integration Services project named **ProfilingDemo** in the D:\Demofiles\Mod04 folder.
3. If the **Getting Started (SSIS)** window is displayed, close it.
4. In Solution Explorer, right-click **Connection Managers** and click **New Connection Manager**. Then add a new **ADO.NET** connection manager with the following settings:
   - **Server name**: localhost
   - **Log on to the server**: Use Windows Authentication
   - **Select or enter a database name**: ResellerSales
5. If the SSIS Toolbox pane is not visible, on the **SSIS** menu, click **SSIS Toolbox**. Then, in the SSIS Toolbox pane, in the **Common** section, double-click **Data Profiling Task** to add it to the Control Flow surface. (Alternatively, you can drag the task icon to the Control Flow surface.)
6. Double-click the **Data Profiling Task** icon on the Control Flow surface.
7. In the **Data Profiling Task Editor** dialog box, on the **General** tab, in the **Destination** property value drop-down list, click **<New File connection...>**.
8. In the **File Connection Manager Editor** dialog box, in the **Usage type** drop-down list, click **Create file**. In the **File** box, type **D:\Demofiles\Mod04\Reseller Sales Data Profile.xml**, and then click **OK**.
9. In the **Data Profiling Task Editor** dialog box, on the **General** tab, set **OverwriteDestination** to **True**.
10. In the **Data Profiling Task Editor** dialog box, on the **Profile Requests** tab, in the **Profile Type** drop-down list, select **Column Statistics Profile Request**, and then click the **RequestID** column.
11. In the **Request Properties** pane, set the following property values. Do not click OK when finished:

- **ConnectionManager**: localhost.ResellerSales
- **TableOrView**: [dbo].[SalesOrderHeader]
- **Column**: OrderDate

12. In the row under the **Column Statistics Profile Request**, add a **Column Length Distribution Profile Request** profile type with the following settings:
    - **ConnectionManager**: localhost.ResellerSales
    - **TableOrView**: [dbo].[Resellers]
    - **Column**: AddressLine1
13. Add a **Column Null Ratio Profile Request** profile type with the following settings:
    - **ConnectionManager**: localhost.ResellerSales
    - **TableOrView**: [dbo].[Resellers]
    - **Column**: AddressLine2
14. Add a **Value Inclusion Profile Request** profile type with the following settings:
    - **ConnectionManager**: localhost.ResellerSales
    - **SubsetTableOrView**: [dbo].[SalesOrderHeader]
    - **SupersetTableOrView**: [dbo].[PaymentTypes]
    - **InclusionColumns**:
        - **Subset side Columns**: PaymentType
        - **Superset side Columns**: PaymentTypeKey
    - **InclusionThresholdSetting**: None
    - **SupersetColumnsKeyThresholdSetting**: None
    - **MaxNumberOfViolations**: 100
15. In the **Data Profiling Task Editor** dialog box, click **OK**.
16. On the **Debug** menu, click **Start Debugging**.

View a Data Profiling Report

1. When the Data Profiling task has completed, with the package still running, double-click **Data Profiling Task**, and then click **Open Profile Viewer**.
2. Maximize the **Data Profile Viewer** window and under the **[dbo].[SalesOrderHeader]** table, click **Column Statistics Profiles**. Then review the minimum and maximum values for the **OrderDate** column.
3. Under the **[dbo].[Resellers]** table, click **Column Length Distribution Profiles** and select the **AddressLine1** column to view the statistics. Click the bar chart for any of the column lengths, and then click the **Drill Down** button (at the right-edge of the title area for the middle pane) to view the source data that matches the selected column length.
4. Close the **Data Profile Viewer** window, and then in the **Data Profiling Task Editor** dialog box, click **Cancel**.
5. On the **Debug** menu, click **Stop Debugging**, and then close Visual Studio, saving your changes if you are prompted.

6. On the Start screen, type **Data Profile**, and then start the **SQL Server 2014 Data Profile Viewer** app. When the app starts, maximize it.
7. Click **Open**, and open **Reseller Sales Data Profile.xml** in the D:\Demofiles\Mod04 folder.
8. Under the **[dbo].[Resellers]** table, click **Column Null Ratio Profiles** and view the null statistics for the **AddressLine2** column. Select the **AddressLine2** column, and then click the **Drill Down** button to view the source data.
9. Under the **[dbo].[SalesOrderHeader]** table, click **Inclusion Profiles** and review the inclusion statistics for the **PaymentType** column. Select the inclusion violation for the payment type value of **0**, and then click the **Drill Down** button to view the source data.

Note: The **PaymentTypes** table includes two payment types, using the value 1 for invoice-based payments and 2 for credit account payments. The Data Profiling task has revealed that for some sales, the value **0** is used, which may indicate an invalid data entry or may be used to indicate some other kind of payment that does not exist in the **PaymentTypes** table.

10. Close the Data Profile Viewer window.

Lesson 3
Implementing Data Flow

After you have thoroughly explored the data sources for your data warehousing solution, you can start to implement an ETL process by using SSIS. This process will consist of one or more SSIS packages, with each containing one or more Data Flow tasks. Data flow is at the core of any SSIS-based ETL solution, so it's important to understand how you can use the components of an SSIS data flow pipeline to extract, transform, and load data.

This lesson describes the various components that are used to implement a data flow, and provides some guidance for optimizing data flow performance.

Lesson Objectives

After completing this lesson, you will be able to:

- Create a connection manager.
- Add a Data Flow task to a package control flow.
- Add a Source component to a data flow.
- Add a destination to a data flow.
- Add transformations to a data flow.
- Optimize data flow performance.

Connection Managers

To extract or load data, an SSIS package must be able to connect to the data source or destination. In an SSIS solution, you define data connections by creating a connection manager for each data source or destination used in the workflow. A connection manager encapsulates the following information, which is used to make a connection to the data source or destination:

- A connection to a data source or destination:
  - Provider (for example, ADO.NET, OLE DB, or flat file)
  - Connection string
  - Credentials
- Project or package level:
  - Project-level connection managers:
    - Can be shared across packages
    - Are listed in Solution Explorer and the Connection Managers pane for packages in which they are used
  - Package-level connection managers:
    - Can be shared across objects in the package
    - Are listed only in the Connection Managers pane for packages in which they are used

- The data provider to be used. For example, you can create a connection manager for a relational database by using an OLE DB or ADO.NET provider. Alternatively, you can create a connection manager for a text file by using a flat file provider.
- The connection string used to locate the data source. For a relational database, the connection string includes the network name of the database server and the name of the database. For a file, the file name and path must be specified.
- The credentials used to access the data source.

You can create a connection manager at the project level or at the package level:

- Project-level connection managers are listed in Solution Explorer and can be shared across multiple packages in the same project. Use project-level connection managers when multiple packages need to access the same data source. To create a project-level connection manager, right-click the Connection Managers folder in Solution Explorer or click the **Project** menu, and then click **New Connection Manager**.
- Package-level connection managers exist only within the package in which they are defined. Both project-level and package-level connection managers used by a package are shown in its Connection Managers pane in the SSIS Designer.

To create a package-level connection manager, right-click in the Connection Managers pane and choose the type you want to create. Alternatively, create a new connection manager in the Properties dialog box of a task, source, destination, or transformation.

Note: When you create a new connection manager, Visual Studio enables you to select connection details that you have created previously, even if they relate to connection managers that do not exist in the current project or have been deleted.

The Data Flow Task

A package defines a control flow for actions that the SSIS run-time engine is to perform. A package control flow can contain several different tasks, and include complex branching and iteration, but the core of any ETL control flow is the Data Flow task.

- The core control flow task in most SSIS packages
- It encapsulates a data flow pipeline
- You define the pipeline for the task on the **Data Flow** tab

To include a data flow in a package control flow, drag the Data Flow task from the SSIS Toolbox pane into the Control Flow surface. Alternatively, you can double-click the Data Flow task icon in the SSIS Toolbox pane and the task will be added to the design surface. After you have added a Data Flow task to the control flow, you can rename it and set its properties in the Properties pane.

Note: This module focuses on the Data Flow task. Other control flow tasks will be discussed in detail in Module 5: *Implementing Control Flow in an SSIS Package*.

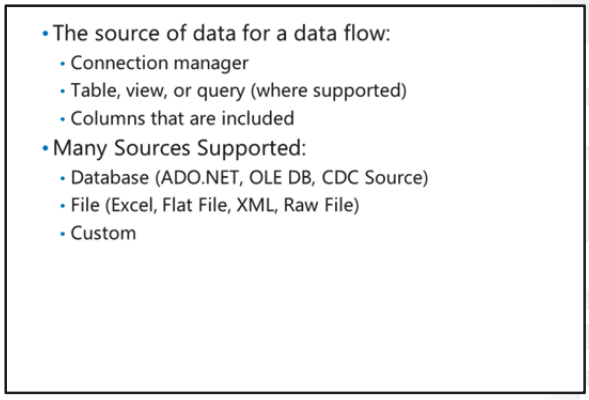
To define the pipeline for the Data Flow task, double-click the task. SSIS Designer will display a design surface where you can add data flow components. Alternatively, click the Data Flow tab in SSIS Designer, and then select the Data Flow task that you want to edit in the drop-down list displayed at the top of the design surface.

A typical data flow pipeline includes one or more data sources, transformations that operate on the data as it flows through the pipeline, and one or more destinations for the data. The pipeline flow is defined by connecting the output from one component to the input of the next.

Data Sources

The starting point for a data flow is a data source, a definition for which includes:

- The connection manager used to connect to the data source.
- The table, view, or query used to extract the data.
- The columns that are included in the output from the data source and passed to the next component in the data flow pipeline.



The following table describes the kinds of data source that SSIS supports:

Databases	
ADO.NET	Any database to which an ADO.NET data provider is installed.
OLE DB	Any database for which an OLE DB provider is installed.
CDC Source	An SQL Server or Oracle database in which change data capture (CDC) has been enabled. CDC is discussed in Module 7: *Implementing an Incremental ETL Process*.
Files	
Excel	A Microsoft Excel® workbook.
Flat file	Data in a text file, such as comma-delimited text.
XML	A file that contains data in XML format.
Raw file	An SSIS-specific binary format file.
Other sources	
Script component	A custom source that is implemented as a script.
Custom	A custom data source that is implemented as a .NET assembly.

In addition to those listed in the table, you can download the following sources from the Microsoft website:

- Oracle
- SAP BI
- Teradata

To add a data source for SQL Server, Excel, a flat file, or Oracle to a data flow, drag the Source Assistant icon from the Favorites section of the SSIS Toolbox pane to the design surface and use the wizard to select or create a connection manager for the source. For other data sources, drag the appropriate icon from the Other Sources section of the SSIS Toolbox pane to the design surface, and then double-click the data source on the design surface to define the connection, data, and output columns.

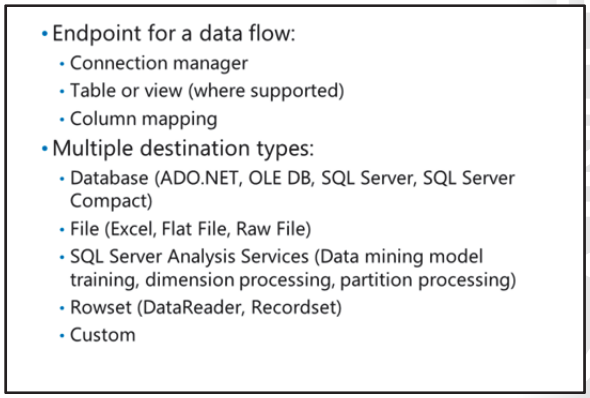
By default, the output from a data source is represented as an arrow at the bottom of the data source icon on the design surface. To create a data flow, you simply drag this arrow and connect it to the next component in the data flow pipeline, which could be a destination or a transformation.

Data Destinations

A destination is an endpoint for a data flow. It has input columns, which are determined by the connection from the previous component in the data flow pipeline, but no output.

A destination definition includes:

- A connection manager for the data store where the data is to be inserted.
- The table or view into which the data must be inserted (where supported).



The following table describes the kinds of destination that SSIS supports:

Databases	
ADO.NET	Any database for which an ADO.NET data provider is installed.
OLE DB	Any database for which an OLE DB provider is installed.
SQL Server	An SQL Server database.
SQL Server Compact	An instance of SQL Server Compact.
Files	
Excel	A Microsoft Excel® workbook.
Flat file	A text file.
Raw file	An SSIS-specific binary format file.
SQL Server Analysis Services	
Data mining model training	Used to build data mining models for data analysis.
Dimension processing	Used to populate a dimension in an online analytical processing (OLAP) cube.
Partition processing	Used to populate a partition in an OLAP cube.
Rowsets	
DataReader	An ADO.NET DataReader interface that can be read by another application.
Recordset	An ADO Recordset interface that can be read by another application.

Other sources	
Script component	A custom destination that is implemented as a script.
Custom	A custom destination that is implemented as a .NET assembly.

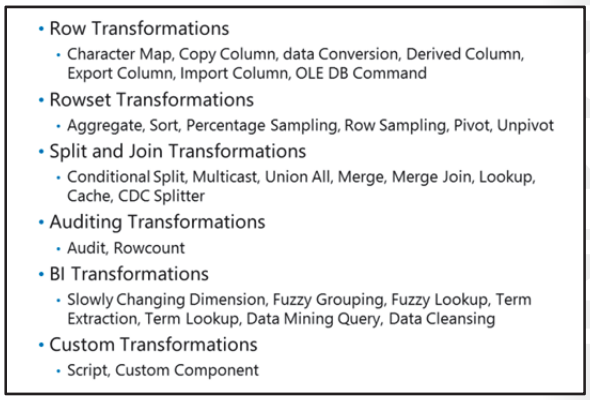
To add an SQL Server, Excel, or Oracle destination to a data flow, drag the Destination Assistant icon from the Favorites section of the SSIS Toolbox pane to the design surface, and then use the wizard to select or create a connection manager. For other kinds of destination, drag the appropriate icon from the Other Destinations section of the SSIS Toolbox pane to the design surface.

After you have added a destination to the data flow, connect the output from the previous component in the data flow to the destination, double-click it, and then edit it to define:

- The connection manager and destination table (if relevant) to be used when loading the data.
- The column mappings between the input columns and the columns in the destination.

Data Transformations

Data transformations enable you to perform operations on rows of data as they pass through the data flow pipeline. Transformations have both inputs and outputs.



The following table lists the transformations that SSIS includes:

Row transformations – update column values or create new columns for each row in the data flow	
Character Map	Applies string functions to column values, such as conversion from lowercase to uppercase.
Copy Column	Creates a copy of a column and adds it to the data flow.
Data Conversion	Converts data of one type to another, for example, numerical values to strings.
Derived Column	Adds a new column based on an expression. For example, you could use an expression to multiply a **Quantity** column by a **UnitPrice** column to create a new **TotalPrice** column.
Export Column	Saves the contents of a column as a file.
Import Column	Reads data from a file and adds it as a column in the data flow.

OLE DB Command	Runs an SQL command for each row in the data flow.
Rowset transformations – create new rowsets	
Aggregate	Creates a new rowset by applying aggregate functions such as SUM.
Sort	Creates a new sorted rowset.
Percentage Sampling	Creates a rowset by randomly selecting a specified percentage of rows.
Row Sampling	Creates a rowset by randomly selecting a specified number of rows.
Pivot	Creates a rowset by condensing multiple records with a single column into a single record with multiple columns.
Unpivot	Creates a rowset by expanding a single record with multiple columns into multiple records with a single column.
Split and Join transformations – merge or branch data flows	
Conditional Split	Splits a single-input rowset into multiple-output rowsets based on conditional logic.
Multicast	Distributes all input rows to multiple outputs.
Union All	Adds multiple inputs into a single output.
Merge	Merges two sorted inputs into a single output.
Merge Join	Joins two sorted inputs to create a single output based on a FULL, LEFT, or INNER join operation.
Lookup	Looks up columns in a data source by matching key values in the input. It creates an output for matched rows and a second output for rows with no matching value in the lookup data source.
Cache	Caches data from a data source to be used by a Lookup transformation.
CDC Splitter	Splits inserts, updates, and deletes from a CDC source into separate data flows. CDC is discussed in Module 7: *Implementing an Incremental ETL Process*.
Auditing transformations – add audit information or count rows	
Audit	Provides execution environment information that can be added to the data flow.
RowCount	Counts the rows in the data flow and writes the result to a variable.
BI transformations – perform BI tasks	
Slowly Changing Dimension	Redirects rows when loading a data warehouse to preserve historical dimension values.
Fuzzy	Uses fuzzy logic to deduplicate rows in the data flow.

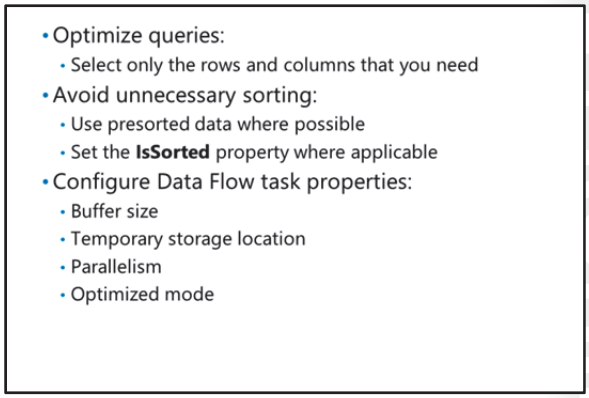
Grouping	
Fuzzy Lookup	Looks up columns in a data source by finding approximate matches for values in the input.
Term Extraction	Extracts nouns or noun phrases from text for statistical analysis.
Term Lookup	Matches terms extracted from text with terms in a reference table.
Data Mining Query	Runs a data mining prediction query against the input to predict unknown column values.
Data Cleansing	Applies a Data Quality Services knowledge base to data as it flows through the pipeline.
Custom transformations – perform custom operations	
Script Component	Runs custom script code for each row in the input.
Custom Component	A custom .NET assembly.

To add a transformation to a workflow, drag it from the **Common** or **Other Transforms** section of the SSIS Toolbox pane to the design surface, and then connect the required inputs to the transformation. Double-click the transformation to configure the specific operation that it will perform, and then define the columns to be included in the outputs from the transformation.

Additional Reading: For more formation about Integration Services Transformations, go to http://go.microsoft.com/fwlink/?LinkID=246724.

Optimizing Data Flow Performance

There are several techniques that you can apply to optimize the performance of a data flow. When you are implementing a data flow, consider the following guidelines:



- Optimize queries. Select only the rows and columns you need to reduce the overall volume of data in the data flow.
- Avoid unnecessary sorting. If you require sorted data from a single data source, sort it during the extraction by using a query with an ORDER BY clause if possible. If subsequent transformations in your data flow rely on sorted data, use the **IsSorted** property of the output to indicate that the data is already sorted.
- Optimize performance. Configure Data Flow task properties using the following properties:
  - **DefaultBufferSize** and **DefaultBufferMaxRows**. Configuring the size of the buffers used by the data flow can significantly improve performance. When there is sufficient memory available, you

should try to achieve a small number of large buffers without incurring any disk paging. The default values for these properties are 10 MB and 10,000 rows respectively.

- **BufferTempStoragePath** and **BLOBTempStoragePath**. Using these properties to locate temporary objects created by the data flow to a fast disk, or spreading them across multiple storage devices, can improve performance.
- **EngineThreads**. Setting the number of threads available to the Data Flow task can improve execution performance, particularly in packages where the **MaxConcurrentExecutables** property has been set to enable parallel execution of the package's tasks across multiple processors.
- **RunInOptimizedMode**. Setting a Data Flow task to run in optimized mode increases performance by removing any columns or components that are not required further downstream in the data flow.

Demonstration: Implementing a Data Flow

In this demonstration, you will see how to:

- Configure a Data Source.
- Use a Derived Column Transformation.
- Use a Lookup Transformation.
- Configure a Destination.

Demonstration Steps

Configure a Data Source

1. Ensure you have completed the previous demonstrations in this module.
2. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance using Windows authentication.
3. In Object Explorer, expand **Databases**, expand **Products**, and expand **Tables**. Then right-click each of the following tables and click **Select Top 1000 Rows** and view the data they contain.
   - **dbo.Product**
   - **dbo.ProductCategory**
   - **dbo.ProductSubcategory**
4. In Object Explorer, under **Databases**, expand **DemoDW**, and expand **Tables**. Then right-click **dbo.DimProduct** and click **Select Top 1000** Rows to verify that this table is empty.
5. Start Visual Studio and create a new Integration Services project named **DataFlowDemo** in the D:\Demofiles\Mod04 folder.
6. If the **Getting Started (SSIS)** window is displayed, close it.
7. In Solution Explorer, expand **SSIS Packages**, right-click **Package.dtsx**, and click **Rename**. Then change the package name to **ExtractProducts.dtsx**.
8. In Solution Explorer, right-click **Connection Managers** and click **New Connection Manager**. Then add a new **OLEDB** connection manager with the following settings:
   - **Server name**: localhost
   - **Log on to the server**: Use Windows Authentication

- **Select or enter a database name**: Products

9. In the **SSIS Toolbox** pane, in the **Favorites** section, double-click **Data Flow Task** to add it to the Control Flow surface. Alternatively, you can drag the task icon to the Control Flow surface.
10. Right-click **Data Flow Task** and click **Rename**. Then change its name to **Extract Products**.
11. Double-click **Extract Products** to switch to the **Data Flow** tab.
12. In the **SSIS Toolbox** pane, in the **Favorites** section, double-click **Source Assistant** to add a source to the Data Flow surface. Alternatively, you can drag the **Source Assistant** icon to the Data Flow surface.
13. In the **Source Assistant - Add New Source** dialog box, in the list of types, click **SQL Server**. In the list of connection managers, click **localhost.Products**, and then click **OK**.
14. Rename the **OLE DB Source** to **Products**, and then double-click it to edit its settings.
15. In the **OLE DB Source Editor** dialog box, on the **Connection Manager** tab, view the list of available tables and views in the drop-down list.
16. Change the data access mode to **SQL Command**, and then enter the following Transact-SQL code:

```
SELECT ProductKey, ProductName FROM Product
```

17. Click **Build Query** to open the **Query Builder** dialog box.
18. In the **Product** table, select the **ProductSubcategoryKey**, **StandardCost**, and **ListPrice** columns, and then click **OK**.
19. In the **OLE DB Source Editor** dialog box, click **Preview** to see a data preview, and then click **Close** to close the preview.
20. In the **OLE DB Source Editor** dialog box, click the **Columns** tab, view the list of external columns that the query has returned and the output columns generated by the data source, and then click **OK**.

Use a Derived Column Transformation

1. In the **SSIS Toolbox** pane, in the **Common** section, double-click **Derived Column** to add a Derived Column transformation to the Data Flow surface, and then position it under the **Products** source. Alternatively, you can drag the **Derived Transformation** icon to the Data Flow surface.
2. Rename the **Derived Column** transformation to **Calculate Profit**.
3. Select the **Products** source, and then drag the blue output arrow to the **Derived Column** transformation.
4. Double-click the **Derived Column** transformation to edit its settings, and then in the **Derived Column Name** box, type **Profit**.
5. Ensure that **<add as new column>** is selected in the **Derived Column** box.
6. Expand the **Column** folder, and then drag the **ListPrice** column to the Expression box.

7. In the Expression box, after [ListPrice], type a minus sign (–), and then drag the StandardCost column to the Expression box to create the following expression:

```
[ListPrice]-[StandardCost]
```

8. Click the **Data Type** box, ensure that it is set to **Currency [DT_CY]**, and then click **OK**.

Use a Lookup Transformation

1. In the **SSIS Toolbox** pane, in the **Common** section, double-click **Lookup** to add a Lookup transformation to the Data Flow surface, and then position it under the **Calculate Profit** transformation. Alternatively, you can drag the **Lookup** icon to the Data Flow surface.
2. Rename the **Lookup** transformation to **Lookup Category**.
3. Select the **Calculate Profit** transformation, and then drag the blue output arrow to the **Lookup Category** transformation.
4. Double-click the **Lookup Category** transformation to edit its settings.
5. In the **Lookup Transformation Editor** dialog box, on the **General** tab, in the **Specify how to handle rows with no matching entries** list, select **Redirect rows to no match output**.
6. In the **Lookup Transformation Editor** dialog box, on the **Connection** tab, ensure that the **localhost.Products** connection manager is selected, and then click **Use results of an SQL query**.
7. Click **Browse**, and then in the D:\Demofiles\Mod04 folder, open the **LookupProductCategories.sql** query.
8. Click **Preview** to view the product category data, note that it includes a **ProductSubcategoryKey** column, and then click **Close** to close the preview.
9. In the **Lookup Transformation Editor** dialog box, on the **Columns** tab, in the **Available Input Columns** list, drag **ProductSubcategoryKey** to **ProductSubCategoryKey** in the **Available Lookup Columns** list.
10. Select the **ProductSubcategoryName** and **ProductCategoryName** columns to add them as new columns to the data flow, and then click **OK**.

Configure a Destination

1. In Solution Explorer, create a new OLE DB connection manager with the following settings:
   - **Server name**: localhost
   - **Log on to the server**: Use Windows Authentication
   - **Select or enter a database name**: DemoDW
2. In the SSIS Toolbox pane, in the Favorites section, double-click **Destination Assistant** to add a destination transformation to the Data Flow surface. Alternatively, you can drag the Destination Assistant icon to the Data Flow surface.
3. In the **Destination Assistant - Add New Destination** dialog box, in the list of types, click **SQL Server**. In the list of connection managers, click **localhost.DemoDW**, and then click OK.
4. Rename the OLE DB destination to **DemoDW** and position it under the **Lookup Category** transformation.
5. Select the **Lookup Category** transformation, and then drag the blue output arrow to the **DemoDW** destination.
6. In the **Input Output Selection** dialog box, in the **Output** list, click **Lookup Match Output**, and then click **OK**.

7. Double-click the **DemoDW** destination to edit its settings, and then in the **Name of the table or the view list**, click **[dbo].[DimProduct]**.
8. In the **OLE DB Destination Editor** dialog box, on the **Mappings** tab, note that input columns are automatically mapped to destination columns with the same name.
9. In the **Available Input Columns** list, drag the **ProductKey** column to the **ProductID** column in the **Available Destination Columns** list, and then click **OK**.
10. In the **SSIS Toolbox** pane, in the **Other Destinations** section, double-click **Flat File Destination** to add a destination transformation to the Data Flow surface, and then position it to the right of the Lookup Category transformation. Alternatively, you can drag the **Flat File Destination** icon to the Data Flow surface.
11. Rename the flat file destination **Uncategorized Products**.
12. Select the **Lookup Category** transformation, and then drag the blue output arrow to the **Uncategorized Products** destination. The **Lookup No Match Output** output is automatically selected.
13. Double-click the **Uncategorized Products** destination to edit its settings, and then click **New**. Then select **Delimited Values** and click **OK**.
14. In the **Flat File Connection Manager Editor** dialog box, name the new connection manager **Unmatched Products** and specify the file name **D:\Demofiles\Mod04\UnmatchedProducts.csv**. Then click **OK**.
15. In the **Flat File Destination Editor** dialog box, click the **Mappings** tab and note that the input columns are mapped to destination columns with the same names, and then click **OK**.
16. On the **Debug** menu, click **Start Debugging**, and observe the data flow as it runs, noting the number of files transferred along each path.
17. When the data flow has completed, on the **Debug** menu, click **Stop Debugging**.
18. Close Visual Studio, saving your changes if you are prompted.
19. Start Excel, and open the **Unmatched Products.csv** flat file in the D:\Demofiles\Mod04 folder. Note that there were no unmatched products.
20. Use SQL Server Management Studio to view the contents of the **DimProduct** table in the **DemoDW** database, and note that the product data has been transferred.
21. Close SQL Server Management Studio without saving any files.

Lab: Implementing Data Flow in an SSIS Package

Scenario

In this lab, you will focus on the extraction of customer and sales order data from the InternetSales database used by the company's e-commerce site, which you must load into the Staging database. This database contains customer data (in a table named Customers), and sales order data (in tables named SalesOrderHeader and SalesOrderDetail). You will extract sales order data at the line item level of granularity. The total sales amount for each sales order line item is then calculated by multiplying the unit price of the product purchased by the quantity ordered. Additionally, the sales order data includes only the ID of the product purchased, so your data flow must look up the details of each product in a separate Products database.

Objectives

After completing this lab, you will be able to:

- Extract and profile source data.
- Implement a data flow.
- Use transformations in a data flow.

Estimated Time: 60 minutes

Virtual machine: 20463C-MIA-SQL

User name: ADVENTUREWORKS\Student

Password: Pa$$w0rd

Exercise 1: Exploring Source Data

Scenario

You have designed a data warehouse schema for Adventure Works Cycles, and now you must design an ETL process to populate it with data from various source systems. Before creating the ETL solution, you have decided to examine the source data so you can understand it better.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Extract and View Sample Source Data
3. Profile Source Data

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the 20463C-MIA-SQL virtual machine, run **Setup.cmd** in the D:\Labfiles\Lab04\Starter folder as Administrator.

▶ Task 2: Extract and View Sample Source Data

1. Use the SQL Server 2014 Import and Export Data Wizard to extract a sample of customer data from the **InternetSales** database on the **localhost** instance of SQL Server to a comma-delimited flat file.
   - Your sample should consist of the first 1,000 records in the **Customers** table.
   - You should use a text qualifier because some string values in the table may contain commas.
2. After you have extracted the sample data, use Excel to view it.

Note: You may observe some anomalies in the data, such as invalid gender codes and multiple values for the same country or region. The purpose of examining the source data is to identify as many of these problems as possible, so that you can resolve them in the development of the ETL solution. You will address the problems in this data in later labs.

▶ Task 3: Profile Source Data

1. Create an Integration Services project named **Explore Internet Sales** in the D:\Labfiles\Lab04\Starter folder.
2. Add an ADO.NET connection manager that uses Windows authentication to connect to the **InternetSales** database on the **localhost** instance of SQL Server.
3. Use a Data Profiling task to generate the following profile requests for data in the **InternetSales** database:
   - Column statistics for the **OrderDate** column in the **SalesOrderHeader** table. You will use this data to find the earliest and latest dates on which orders have been placed.
   - Column length distribution for the **AddressLine1** column in the **Customers** table. You will use this data to determine the appropriate column length to allow for address data.
   - Column null ratio for the **AddressLine2** column in the **Customers** table. You will use this data to determine how often the second line of an address is null.
   - Value inclusion for matches between the **PaymentType** column in the **SalesOrderHeader** table and the **PaymentTypeKey** column in the **PaymentTypes** table. Do not apply an inclusion threshold and set a maximum limit of 100 violations. You will use this data to find out if any orders have payment types that are not present in the table of known payment types.
4. Run the SSIS package and view the report that the Data Profiling task generates in the Data Profile Viewer.

Results: After this exercise, you should have a comma-separated text file that contains a sample of customer data, and a data profile report that shows statistics for data in the **InternetSales** database.

Exercise 2: Transferring Data by Using a Data Flow Task

Scenario

Now that you have explored the source data in the **InternetSales** database, you are ready to start implementing data flows for the ETL process. A colleague has already implemented data flows for reseller sales data, and you plan to model your Internet sales data flows on those.

The main tasks for this exercise are as follows:

1. Examine an Existing Data Flow
2. Create a Data Flow task
3. Add a Data Source to a Data Flow
4. Add a Data Destination to a Data flow
5. Test the Data Flow Task

▶ Task 1: Examine an Existing Data Flow

1. Open the D:\Labfiles\Lab04\Starter\Ex2\ AdventureWorksETL.sln solution in Visual Studio.

2. Open the **Extract Reseller Data.dtsx** package and examine its control flow. Note that it contains two Data Flow tasks.
3. On the **Data Flow** tab, view the **Extract Resellers** task and note that it contains a source named **Resellers** and a destination named **Staging DB**.
4. Examine the **Resellers** source, noting the connection manager that it uses, the source of the data, and the columns that its output contains.
5. Examine the **Staging DB** destination, noting the connection manager that it uses, the destination table for the data, and the mapping of input columns to destination columns.
6. Right-click anywhere on the Data Flow design surface, click **Execute Task**, and then observe the data flow as it runs, noting the number of rows transferred.
7. When the data flow has completed, stop the debugging session.

▶ Task 2: Create a Data Flow task

1. Add a new package to the project and name it **Extract Internet Sales Data.dtsx**.
2. Add a Data Flow task named **Extract Customers** to the new package's control flow.

▶ Task 3: Add a Data Source to a Data Flow

1. Create a new project-level OLE DB connection manager that uses Windows authentication to connect to the **InternetSales** database on the localhost instance of SQL Server.
2. In the **Extract Customers** data flow, add a source that uses the connection manager that you created for the **InternetSales** database, and name it **Customers**.
3. Configure the **Customers** source to extract all columns from the **Customers** table in the **InternetSales** database.

▶ Task 4: Add a Data Destination to a Data Flow

1. Add a destination that uses the existing **localhost.Staging** connection manager to the **Extract Customers** data flow, and then name it **Staging DB**.
2. Connect the output from the **Customers** source to the input of the **Staging DB** destination.
3. Configure the **Staging DB** destination to load data into the **Customers** table in the **Staging** database.
4. Ensure that all columns are mapped, and in particular that the **CustomerKey** input column is mapped to the **CustomerBusinessKey** destination column.

▶ Task 5: Test the Data Flow Task

1. Right-click anywhere on the Data Flow design surface, click **Execute Task**, and then observe the data flow as it runs, noting the number of rows transferred.
2. When the data flow has completed, stop the debugging session.

Results: After this exercise, you should have an SSIS package that contains a single Data Flow task, which extracts customer records from the **InternetSales** database and inserts them into the Staging database.

Exercise 3: Using Transformations in a Data Flow

Scenario

You have implemented a simple data flow to transfer customer data to the staging database. Now you must implement a data flow for Internet sales records. The new data flow must add a new column that contains the total sales amount for each line item (which is derived by multiplying the list price by the quantity of units purchased), and use a product key value to find additional data in a separate Products database. Once again, you will model your solution on a data flow that a colleague has already implemented for reseller sales data.

The main tasks for this exercise are as follows:

1. Examine an Existing Data Flow
2. Create a Data Flow Task
3. Add a Data Source to a Data Flow
4. Add a Derived Column transformation to a data flow
5. Add a Lookup Transformation to a Data Flow
6. Add a Data Destination to a Data Flow
7. Test the Data Flow task

▶ Task 1: Examine an Existing Data Flow

1. Open the D:\Labfiles\Lab04\Starter\Ex3\AdventureWorksETL.sln solution in Visual Studio.
2. Open the **Extract Reseller Data.dtsx** package and examine its control flow. Note that it contains two Data Flow tasks.
3. On the **Data Flow** tab, view the **Extract Reseller Sales** task.
4. Examine the **Reseller Sales** source, noting the connection manager that it uses, the source of the data, and the columns that its output contains.
5. Examine the **Calculate Sales Amount** transformation, noting the expression that it uses to create a new derived column.
6. Examine the **Lookup Product Details** transformation, noting the connection manager and query that it uses to look up product data, and the column mappings used to match data and add rows to the data flow.
7. Examine the **Staging DB** destination, noting the connection manager that it uses, the destination table for the data, and the mapping of input columns to destination columns.
8. Right-click anywhere on the Data Flow design surface, click **Execute Task**, and then observe the data flow as it runs, noting the number of rows transferred.
9. When the data flow has completed, stop the debugging session.

▶ Task 2: Create a Data Flow Task

1. Open the **Extract Internet Sales Data.dtsx** package, and then add a new Data Flow task named **Extract Internet Sales** to its control flow.
2. Connect the pre-existing **Extract Customers** Data Flow task to the new **Extract Internet Sales** task.

▶ Task 3: Add a Data Source to a Data Flow

1. Add a source that uses the existing **localhost.InternetSales** connection manager to the **Extract Internet Sales** data flow, and then name it **Internet Sales**.

2. Configure the **Internet Sales** source to use the Transact-SQL code in the D:\Labfiles\Lab04\Starter\Ex3\InternetSales.sql query file query to extract Internet sales records.

▶ Task 4: Add a Derived Column transformation to a data flow

1. Add a Derived Column transformation named **Calculate Sales Amount** to the **Extract Internet Sales** data flow.
2. Connect the output from the **InternetSales** source to the input of the **Calculate Sales Amount** transformation.
3. Configure the **Calculate Sales Amount** transformation to create a new column named **SalesAmount** containing the **UnitPrice** column value multiplied by the **OrderQuantity** column value.

▶ Task 5: Add a Lookup Transformation to a Data Flow

1. Add a Lookup transformation named **Lookup Product Details** to the **Extract Internet Sales** data flow.
2. Connect the output from the **Calculate Sales Amount** transformation to the input of the **Lookup Product Details** transformation.
3. Configure the **Lookup Product Details** transformation to:
   - Redirect unmatched rows to the no match output.
   - Use the **localhost.Products** connection manager and the **Products.sql** query in the D:\Labfiles\Lab04\Starter\Ex3 folder to retrieve product data.
   - Match the **ProductKey** input column to the **ProductKey** lookup column.
   - Add all lookup columns other than **ProductKey** to the data flow.
4. Add a flat file destination named **Orphaned Sales** to the **Extract Internet Sales** data flow. Then redirect non-matching rows from the **Lookup Product Details** transformation to the **Orphaned Sales** destination, which should save any orphaned records in a comma-delimited file named **Orphaned Internet Sales.csv** in the D:\ETL folder.

▶ Task 6: Add a Data Destination to a Data Flow

1. Add a destination that uses the **localhost.Staging** connection manager to the **Extract Customers** data flow, and name it **Staging DB**.
2. Connect the match output from the **Lookup Product Details** transformation to the input of the **Staging DB** destination.
3. Configure the **Staging DB** destination to load data into the **InternetSales** table in the **Staging** database. Ensure that all columns are mapped. In particular, ensure that the *Key input columns are mapped to the *BusinessKey destination columns.

▶ Task 7: Test the Data Flow task

1. Right-click anywhere on the Data Flow design surface, click **Execute Task**, and then observe the data flow as it runs, noting the number of rows.
2. When the data flow has completed, stop the debugging session.

Results: After this exercise, you should have a package that contains a Data Flow task including Derived Column and Lookup transformations.

Module Review and Takeaways

In this module, you have learned how to explore source data and use SQL Server Integration Services to implement a data flow.

Review Question(s)

Question: How could you determine the range of OrderDate values in a data source to plan a time dimension table in a data warehouse?

Module 5

Implementing Control Flow in an SSIS Package

Contents:

Module Overview

Control flow in SQL Server Integration Services (SSIS) packages enables you to implement complex extract, transform, and load (ETL) solutions that combine multiple tasks and workflow logic. By learning how to implement control flow, you can design robust ETL processes for a data warehousing solution that coordinate data flow operations with other automated tasks.

Objectives

After completing this module, you will be able to:

- Implement control flow with tasks and precedence constraints.
- Create dynamic packages that include variables and parameters.
- Use containers in a package control flow.
- Enforce consistency with transactions and checkpoints.

Lesson 1
Introduction to Control Flow

Control flow in an SSIS package consists of one or more tasks, usually executed as a sequence based on precedence constraints that define a workflow. Before you can implement a control flow, you need to know what tasks are available and how to define a workflow sequence using precedence constraints. You also need to understand how you can use multiple packages to create complex ETL solutions.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the control flow tasks provided by SSIS.
- Define a workflow for tasks by using precedence constraints.
- Use design time features of SSIS to help you develop control flow efficiently.
- Use multiple packages in an SSIS solution.
- Create reusable package templates.

Control Flow Tasks

A control flow consists of one or more tasks. SSIS includes the following control flow tasks that you can use in a package:

- Data Flow Tasks
- Database Tasks
- File and Internet Tasks
- Process Execution Tasks
- WMI Tasks
- Custom Logic Tasks
- Database Transfer Tasks
- Analysis Services Tasks
- SQL Server Maintenance Tasks

Data Flow Tasks	
Data Flow	Encapsulates a data flow that transfers data from a source to a destination.
Database Tasks	
Data Profiling	Generates statistical reports based on a data source.
Bulk Insert	Inserts data into a data destination in a bulk load operation.
Execute SQL	Runs a structured query language (SQL) query in a database.
Execute T-SQL	Runs a Transact-SQL query in a Microsoft® SQL Server® database.
CDC Control	Performs a change data capture (CDC) status management operation. CDC is discussed in Module 7: *Implementing an Incremental ETL Process*.

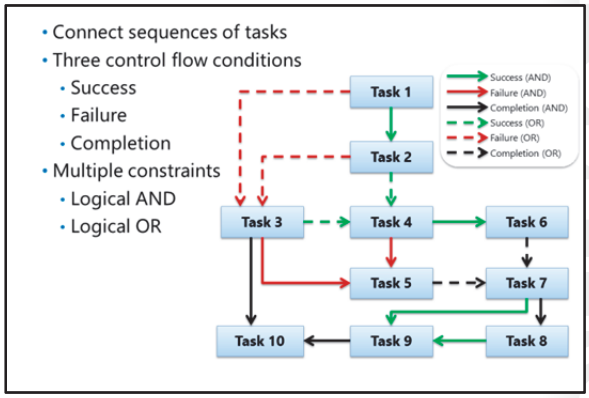
File and Internet Tasks	
File System	Performs file system operations, such as creating folders or deleting files.
FTP	Performs file transfer protocol (FTP) operations, such as copying files.
XML	Performs XML processing operations, such as applying a style sheet.
Web Service	Calls a method on a specific web service.
Send Mail	Sends an email message.
Process Execution Tasks	
Execute Package	Runs a specified SSIS package.
Execute Process	Runs a specified program.
WMI Tasks	
WMI Data Reader	Runs a Windows Management Instrumentation (WMI) query.
WMI Event Watcher	Monitors a specific WMI event.
Custom Logic Tasks	
Script	A Microsoft® Visual Studio® Tools for Applications (VSTA) script.
Custom Task	A custom task implemented as a .NET assembly.
Database Transfer Tasks	
Transfer Database	Transfers a database from one SQL Server instance to another.
Transfer Error Messages	Transfers custom error messages from one SQL Server instance to another.
Transfer Jobs	Transfers SQL Agent jobs from one SQL Server instance to another.
Transfer Logins	Transfers logins from one SQL Server instance to another.
Transfer Master Stored Procedures	Transfers stored procedures in the master database from one SQL Server instance to another.
Transfer SQL Server Objects	Transfers database objects such as tables and views from one SQL Server instance to another.
Analysis Services Tasks	
Analysis Services Execute DDL	Runs a data definition language (DDL) statement in an Analysis Services instance – for example to create a cube.
Analysis Services Processing	Processes an Analysis Services object, such as a cube or data mining model.
Data Mining Query	Runs a prediction query using a data mining model.

SQL Server Maintenance Tasks	
Backup Database	Backs up an SQL Server database.
Check Database Integrity	Checks the integrity of an SQL Server database.
History Cleanup	Deletes out-of-date history data for SQL Server maintenance operations.
Maintenance Cleanup	Deletes files left by maintenance operations.
Notify Operator	Sends a notification by email message, pager message, or network alert to an SQL Agent operator.
Rebuild Index	Rebuilds a specified index on an SQL Server table or view.
Reorganize Index	Reorganizes a specified index on an SQL Server table or view.
Shrink Database	Reduces the size of the specified SQL Server database.
Update Statistics	Updates value distribution statistics for tables and views in an SQL Server database.

To add a task to a control flow, drag it from the SSIS Toolbox to the control flow design surface. Then double-click the task on the design surface to configure its settings.

Precedence Constraints

A control flow usually defines a sequence of tasks to be executed. You define the sequence by connecting tasks with precedence constraints. These constraints evaluate the outcome of a task to determine the flow of execution.



Control Flow Conditions

You can define precedence constraints for one of the following three conditions:

- **Success** – The execution flow to be followed when a task completes successfully. In the control flow designer, success constraints are shown as green arrows.
- **Failure** – The execution flow to be followed when a task fails. In the control flow designer, failure constraints are shown as red arrows.
- **Completion** – The execution flow to be followed when a task completes, regardless of whether it succeeds or fails. In the control flow designer, complete constraints are shown as black arrows.

By using these conditional precedence constraints, you can define a control flow that executes tasks based on conditional logic. For example, you could create a control flow with the following steps:

1. An FTP task downloads a file of sales data to a local folder.

2. If the FTP download succeeds, a data flow task imports the downloaded data into an SQL Server database. However, if the FTP download fails, a Send Mail task notifies an administrator that there's been a problem.
3. When the data flow task completes, regardless of whether it fails or succeeds, a File System task deletes the folder where the customer data file was downloaded.

Using Multiple Constraints

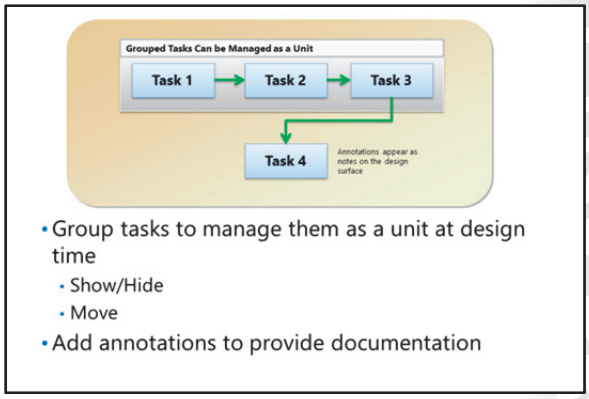
You can connect multiple precedence constraints to a single task. For example, a control flow might include two data flow tasks, and a Send Mail task that you want to use to notify an administrator if something goes wrong. To accomplish this, you could connect a failure precedence constraint from each of the data flow tasks to the Send Mail task. However, you need to determine whether the notification should be sent if either one of the data flow tasks fails, or only if both fail.

By default, when multiple precedence constraints are connected to a single task, a logical AND operation is applied to the precedence condition, meaning that all the precedence constraints must evaluate to True to execute the connected task. In the example above, this means that the Send Mail task would only be executed if both data flow tasks failed. In the control flow designer, logical AND constraints are shown as solid arrows.

You can double-click a precedence constraint to edit and configure it to use a logical OR operation, in which case the connected task is executed if any of the connections evaluates to True. Setting the constraints in the example above to use a logical OR operation would result in the Send Mail task being executed if either (or both) of the data flow tasks failed. In the control flow designer, logical AND constraints are shown as dotted arrows.

Grouping and Annotations

As your control flows become more complex, it can be difficult to interpret the control flow surface. The SSIS Designer includes two features to help SSIS developers work more efficiently.



Grouping Tasks

You can group multiple tasks on the design surface to manage them as a single unit. A task grouping is a design time only feature and has no effect on run-time behavior. With a grouped set of tasks, you can:

- Move the tasks around the design surface as a single unit.
- Show or hide the individual tasks to make the best use of space on the screen.

To create a group of tasks, select which ones you want by dragging around or clicking them while holding the CTRL key. Right-click any of the selected tasks and click **Group**.

Adding Annotations

You can add annotations to the design surface to document your workflow. An annotation is a text-based note that you can use to describe important features of your package design. To add an annotation, right-click the design surface, click **Add Annotation**, and then type the annotation text.

 **Note:** You can add annotations to the Control Flow design surface, the Data Flow design surface, and the Event Handler design surface.

Demonstration: Implementing Control Flow

In this demonstration, you will see how to:

- Add Tasks to a Control Flow.
- Use Precedence Constraints to Define a Control Flow.

Demonstration Steps

Add Tasks to a Control Flow

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Demofiles\Mod05 folder, run **Setup.cmd** as Administrator.
3. Start Visual Studio and open **ControlFlowDemo.sln** from the D:\Demofiles\Mod05 folder.
4. In Solution Explorer, double-click **Control Flow.dtsx**.
5. If the SSIS Toolbox is not visible, on the **SSIS** menu, click **SSIS Toolbox**. Then, from the SSIS Toolbox, drag a **File System Task** to the control flow surface.
6. Double-click the File System Task and configure the following settings:
    - **Name**: Delete Files
    - **Operation**: Delete directory content
    - **SourceConnection**: A new connection with a **Usage type** of **Create folder**, and a **Folder** value of **D:\Demofiles\Mod05\Demo**.
7. From the SSIS Toolbox, drag a second **File System Task** to the control flow surface. Then double-click the File System Task and configure the following settings:
    - **Name**: Delete Folder
    - **Operation**: Delete directory
    - **SourceConnection**: Demo
8. From the SSIS Toolbox, drag a third **File System Task** to the control flow surface. Then double-click the File System Task and configure the following settings:
    - **Name**: Create Folder
    - **Operation**: Create directory
    - **UseDirectoryIfExists**: True
    - **SourceConnection**: Demo
9. From the SSIS Toolbox, drag a fourth **File System Task** to the control flow surface. Then double-click the File System Task and configure the following settings:
    - **Name**: Copy File
    - **Operation**: Copy file
    - **DestinationConnection**: Demo

- **OverwriteDestination**: True
- **SourceConnection**: A new connection with a **Usage type** of **Existing file**, and a **File** value of **D:\Demofiles\Mod05\Demo.txt**

10. From the SSIS Toolbox, drag a **Send Mail** task to the control flow surface. Then double-click the Send Mail task and configure the following settings:
    - **Name** (on the **General** tab): Send Failure Notification
    - **SmtpConnection** (on the **Mail** tab): Create a new SMTP connection manager with a **Name** property of **Local SMTP Server** and an **SMTP Server** property of **localhost**. Use the default values for all other settings
    - **From** (on the **Mail** tab): demo@adventureworks.msft
    - **To** (on the Mail tab): student@adventureworks.msft
    - **Subject** (on the **Mail** tab): Control Flow Failure
    - **MessageSource** (on the **Mail** tab): A task failed

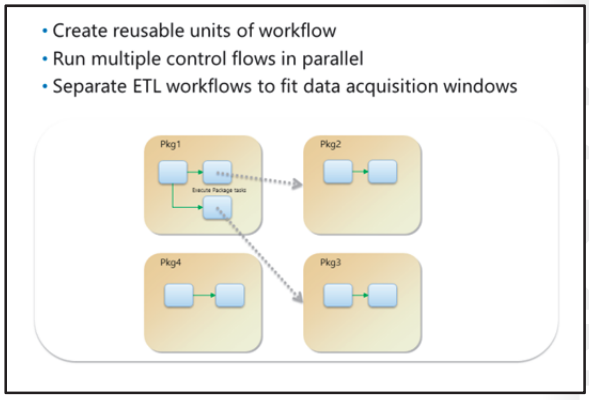
Use Precedence Constraints to Define a Control Flow

1. Select the **Delete Files** task and drag its green arrow to the **Delete Folder** task. Then connect the **Delete Folder** task to the **Create Folder** task and the **Create Folder** task to the **Copy File** task.
2. Connect each of the file system tasks to the **Send Failure Notification** task.
3. Right-click the connection between **Delete Files** and **Delete Folder**, and then click **Completion**.
4. Right-click the connection between **Delete Folder** and **Create Folder** and click **Completion**.
5. Click the connection between the **Delete Files** task and the **Send Failure Notification** task to select it. Then hold the **Ctrl** key and click each connection between the remaining file system tasks and the **Send Failure Notification** task while holding the **Ctrl** key to select them all.
6. Press **F4** and in the **Properties** pane, set the **Value** property to **Failure**.
7. Click anywhere on the control flow surface to clear the current selection, and then double-click any of the red constraints connected to the **Send Failure Notification** task. Then in the **Precedence Constraint Editor** dialog box, in the **Multiple constraints** section, select **Logical OR. One constraint must evaluate to True**, and click **OK**. Note that all connections to the **Send Failure Notification** task are now dotted to indicate that a logical OR operation is applied.
8. Right-click the control flow surface next to the **Send Failure Notification** task and click **Add Annotation**. Then type **Send an email message if a task fails**.
9. Select the **Delete Files** and **Delete Folder** tasks, then right-click either of them and click **Group**. Drag the group to rearrange the control flow so you can see that the **Delete Folder** task is still connected to the **Create Folder** task.
10. On the **Debug** menu, click **Start Debugging** to run the package, and note that the **Delete Files** and **Delete Folder** tasks failed because the specified folder did not previously exist. This caused the **Send Failure Notification** task to be executed.
11. You can view the email message that was sent by the **Send Failure Notification** task in the C:\inetpub\mailroot\Drop folder. Double-click it to open with Outlook.
12. In Visual Studio, on the **Debug** menu, click **Stop Debugging**, and then run the package again. This time all the file system tasks should succeed because the folder was created during the previous execution. Consequently, the **Send Failure Notification** task is not executed.
13. Stop debugging and close Visual Studio. Save the solution files if prompted.

Using Multiple Packages

While you can implement an SSIS solution that includes only one package, most enterprise solutions include multiple packages. By dividing your solution into multiple packages, you can:

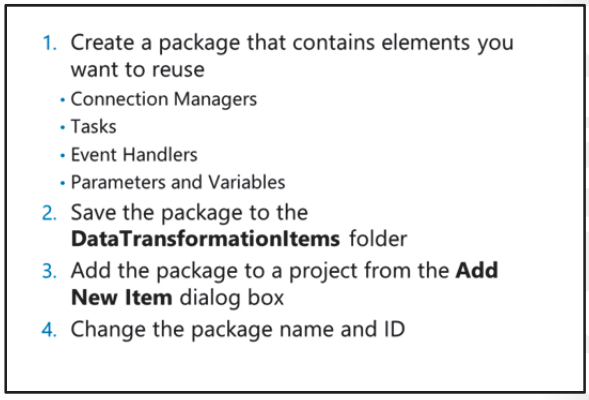
- Create reusable units of workflow that can be used multiple times in a single ETL process.
- Run multiple control flows in parallel, taking advantage of multi-processing computers and improving the overall throughput of your ETL processes.
- Separate data extraction workflows to suit data acquisition windows.



You can execute each package independently, as well as using the Execute Package task to run one package from another.

Creating a Package Template

SSIS developers often need to create multiple similar packages. To make the development process more efficient, you can adopt the following procedure to create a package template that you can reuse to create multiple packages with pre-defined objects and settings.



1. Create a package that includes the elements you want to reuse. These elements can include:
   - Connection Managers
   - Tasks
   - Event Handlers
   - Parameters and Variables
2. Save the package to the **DataTransformationItems** folder on your development workstation.
3. By default, this folder is located at C:\Program Files (x86)\Microsoft Visual Studio <*version*>\Common7\IDE\PrivateAssemblies\ProjectItems\DataTransformationProject.
4. When you want to reuse the package, add a new item to the project and select the package in the **Add New Item** dialog box.
5. Change the **Name** and **ID** properties of the new package to avoid naming conflicts.

Lesson 2
Creating Dynamic Packages

You can use variables, parameters, and expressions to make your SSIS packages more dynamic. For example, rather than hard-coding a database connection string or file path in a data source, you can create a package that sets the value dynamically at run time. This produces a more flexible and reusable solution and helps mitigate differences between the development and production environments.

This lesson describes how you can create variables and parameters, and use them in expressions.

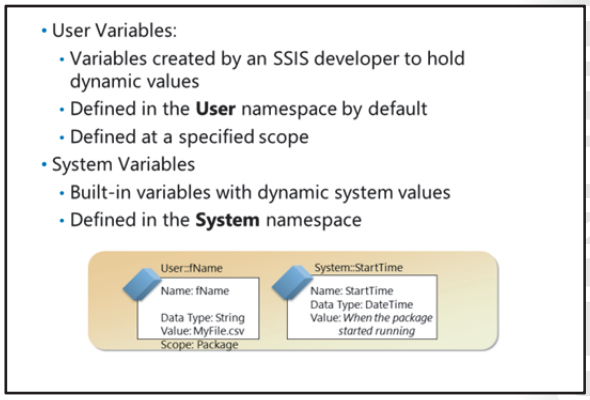
Lesson Objectives

After completing this lesson, you will be able to:

- Create variables in an SSIS solution.
- Create parameters in an SSIS solution.
- Use expressions in an SSIS solution.

Variables

You can use variables to store values that a control flow uses at run time. Variable values can change as the package is executed to reflect run-time conditions. For example, a variable used to store a file path might change depending on the specific server on which the package is running. You can use variables to:



- Set property values for tasks and other objects.
- Store an iterator or enumerator value for a loop.
- Set input and output parameters for an SQL query.
- Store results from an SQL query.
- Implement conditional logic in an expression.

SSIS packages can contain user and system variables.

User Variables

You can define user variables to store dynamic values that your control flow uses. To create a variable, view the Variables pane in SSIS Designer and click the Add Variable button. For each user variable, you can specify the following properties:

- **Name** – A name for the variable. The combination of name and namespace must be unique within the package. Note that variable names are case-sensitive.
- **Scope** – The scope of the variable. Variables can be accessible throughout the whole package, or scoped to a particular container or task. You cannot set the scope in the Variable pane, it is determined by the object selected when you create the variable.
- **Data Type**. – The type of data the variable will hold, for example string, datetime, or decimal.
- **Value** – The initial value of the variable.

- **Namespace** – The namespace within which the variable name is unique. By default, user variables are defined in the **User** namespace, but you can create additional namespaces as required.
- **Raise Change Event** – Causes an event to be raised when the variable value changes. You can then implement an event handler to perform some custom logic.
- **IncludeInDebugDump** – Causes the variable value to be included in debug dump files.

System Variables

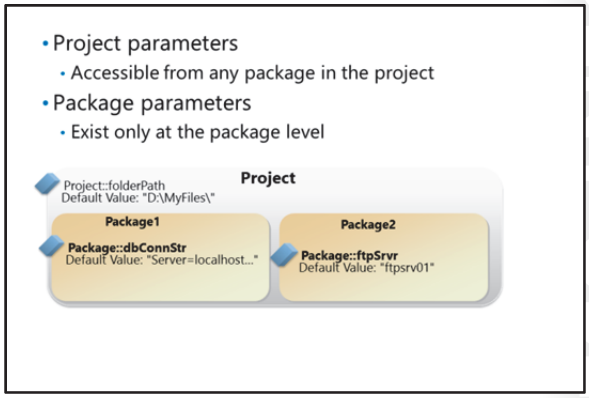
System variables store information about the running package and its objects, and are defined in the System namespace. Some useful system variables include:

- **MachineName** – The computer on which the package is running.
- **PackageName** – The name of the package that is running.
- **StartTime** – The time that the package started running.
- **UserName** – The user who started the package.

Note: For a full list of system variables, refer to the SQL Server Integration Services documentation in SQL Server Books Online.

Parameters

You can use parameters to pass values to a project or package at run time. When you define a parameter, you can set a default value, which can be overridden when the package is executed in a production environment. For example, you could use a parameter to specify a database connection string for a data source, using one value during development, and a different value when the project is deployed to a production environment.



Parameters have three kinds of value:

- **Design default value** – A default value assigned to the parameter in the design environment.
- **Server default value** – A default value assigned to the parameter during deployment. This value overrides the design default value.
- **Execution value** – A value for a specific execution of a package. This value overrides both the server and design default values.

When the project is deployed to an SSIS Catalog, administrators can define multiple environments and specify server default parameter values for each environment.

SSIS supports two kinds of parameter:

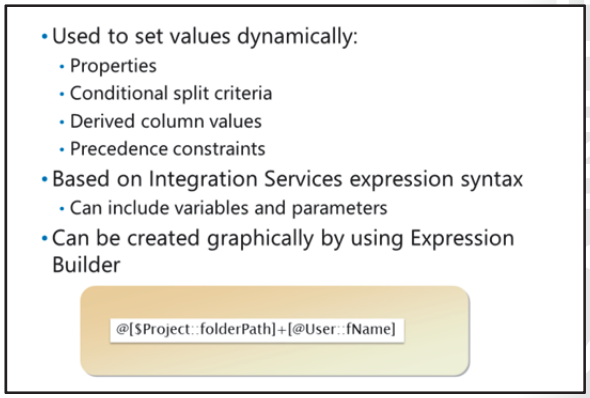
- Project parameters, which are defined at the project level and can be used in any packages within the project.
- Package parameters, which are scoped at the package level and are only available within the package for which they are defined.

Note: Parameters are only supported in the project deployment model. When the legacy deployment model is used, you can set dynamic package properties by using package configurations. Deployment is discussed in Module 12: *Deploying and Configuring SSIS Packages*.

Expressions

SSIS provides a rich expression language that you can use to set values for numerous elements in an SSIS package, including:

- Properties.
- Conditional Split transformation criteria.
- Derived Column transformation values.
- Precedence constraint conditions.



Expressions are based on Integration Services expression syntax, which uses similar functions and keywords to common programming languages like Microsoft® C#. Expressions can include variables and parameters, enabling you to set values dynamically based on specific run-time conditions.

For example, you could use an expression in a data flow task to specify the location of a file to be used as a data source.

The following sample code shows an expression that concatenates a parameter containing a folder path and a variable containing a file name to produce a full file path:

An SSIS Expression

```
@[$Project::folderPath]+[@User::fName]
```

Note that variable names are prefixed with an **@** symbol, and that square brackets are used to enclose identifier names in order to support identifiers with names containing spaces. Also, note that the fully-qualified parameter and variable names are used, including the namespace, and the parameter name is prefixed with a **$** symbol.

You can type expressions, or in many cases you can create them by using the Expression Builder. This is a graphical tool that enables you to create expressions by dragging in variables, parameters, constants, and functions. The Expression Builder automatically adds prefixes and text qualifiers for variables and parameters, simplifying the task of creating complex expressions.

Demonstration: Using Variables and Parameters

In this demonstration, you will see how to:

- Create a Variable.
- Create a Parameter.
- Use Variables and Parameters in an Expression.

Demonstration Steps

Create a Variable

1. Ensure you have completed the previous demonstration in this module.
2. Start Visual Studio and open the **VariablesAndParameters.sln** solution in the D:\Demofiles\Mod05 folder.
3. In Solution Explorer, double-click **Control Flow.dtsx**.
4. On the **View** menu, click **Other Windows**, and click **Variables**.
5. In the **Variables** pane, click the **Add Variable** button and add a variable with the following properties:
    - **Name**: fName
    - **Scope**: Control Flow
    - **Data type**: String
    - **Value**: Demo1.txt

Create a Parameter

1. In Solution Explorer, double-click **Project.parameters**.
2. In the **Project.params [Design]** window, click the **Add Parameter** button and add a parameter with the following properties:
    - **Name**: folderPath
    - **Data type**: String
    - **Value**: D:\Demofiles\Mod05\Files\
    - **Sensitive**: False
    - **Required**: True
    - **Description**: Folder containing text files

Note: Be sure to include the trailing "\" in the **Value** property.

3. Save all files and close the **Project.params [Design]** window.

Use a Variable and a Parameter in an Expression

1. On the **Control Flow.dtsx** package design surface, in the **Connection Managers** pane, click the **Demo.txt** connection manager and press **F4**.
2. In the **Properties** pane, in the **Expressions** property box, click the ellipsis (**...**) button. Then in the **Property Expressions Editor** dialog box, in the **Property** box, select **ConnectionString** and in the **Expression** box, click the ellipsis (**...**) button.

3. In the **Expression Builder** dialog box, expand the **Variables and Parameters** folder, and drag the **$Project::folderPath** parameters to the **Expression** box. Then in the **Expression** box, type a plus (**+**) symbol. Then drag the **User::fName** variable to the **Expression** box to create the following expression:

```
@[$Project::folderPath]+[@User::fName]
```

4. In the **Expression Builder** dialog box, click **Evaluate Expression** and verify that the expression produces the result **D:\Demofiles\Mod05\Files\Demo1.txt**. Then click **OK** to close the **Expression Builder** dialog box, and in the **Property Expressions Editor** dialog box, click **OK**.
5. Run the project, and when it has completed, stop debugging and close Visual Studio.
6. View the contents of the D:\Demofiles\Mod05\Demo folder and verify that **Demo1.txt** has been copied.

Lesson 3
Using Containers

You can create containers in SSIS packages to group related tasks together or define iterative processes. Using containers in packages helps you create complex workflows and a hierarchy of execution scopes that you can use to manage package behavior.

This lesson describes the kinds of containers that are available and how to use them in an SSIS package control flow.

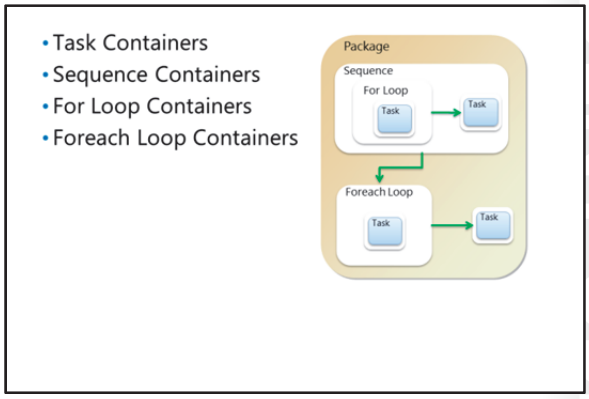
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the types of container available in an SSIS package.
- Use a Sequence container to group related tasks.
- Use a For Loop container to repeat a process until a specific condition is met.
- Use a Foreach Loop container to process items in an enumerated collection.

Introduction to Containers

SSIS packages can contain the following kinds of containers:



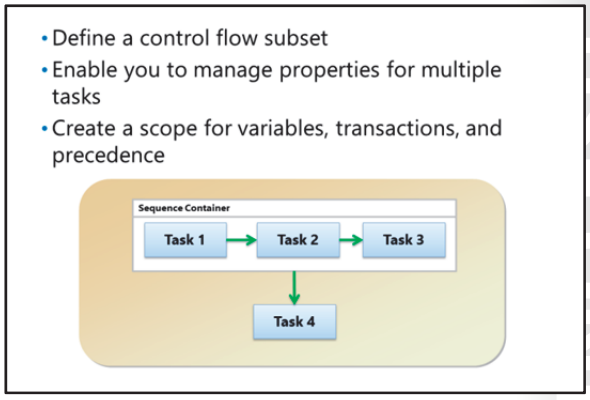
- **Task containers** – Each control flow task has its own implicit container.
- **Sequence containers** – You can group tasks and other containers into a sequence container. This creates an execution hierarchy and enables you to set properties at the container level that apply to all elements within the container.
- **For Loop containers** – You can use a For Loop container to perform an iterative process until a specified condition is met. For example, you could use a For Loop container to execute the same task a specific number of times.
- **Foreach Loop containers** – You can use a Foreach Loop container to perform an iterative task that processes each element in an enumerated collection. For example, you could use a Foreach Loop container to execute a data flow task that imports data from each file in a specified folder into a database.

Containers can be start or endpoints for precedence constraints and you can nest containers within other containers.

Sequence Containers

You can use a sequence container to group tasks and other containers together, and define a subset of the package control flow. By using a sequence container, you can:

- Manage properties for multiple tasks as a unit.
- Disable a logical subset of the package for debugging purposes.
- Create a scope for variables.
- Manage transactions at a granular level.



To create a sequence container, drag the Sequence Container icon from the SSIS Toolbox pane to the design surface. Then drag the tasks and other containers you want to include into the sequence container.

Note: In the design environment, the sequence container behaves similarly to a grouped set of tasks. However, unlike a group, a sequence container exists at run time and its properties can affect the behavior of the control flow.

Demonstration: Using a Sequence Container

In this demonstration, you will be see how to use a sequence container.

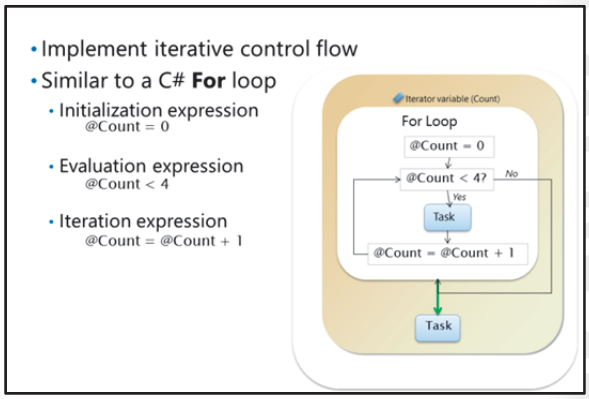
Demonstration Steps

Use a Sequence Container

1. Ensure you have completed the previous demonstrations in this module.
2. Start Visual Studio and open the **SequenceContainer.sln** solution in the D:\Demofiles\Mod05 folder.
3. In Solution Explorer, double-click **Control Flow.dtsx**.
4. Right-click the **Group** indicator around the **Delete Files** and **Delete Folder** tasks and click **Ungroup** to remove it.
5. Drag a **Sequence Container** from the SSIS Toolbox to the control flow design surface.
6. Right-click the precedence constraint that connects **Delete Files** to **Send Failure Notification**, and click **Delete**. Then delete the precedence constraints connecting the **Delete Folder** to **Send Failure Notification** and **Create Folder**.
7. Click and drag around the **Delete Files** and **Delete Folder** tasks to select them both, and then drag into the sequence container.
8. Drag a precedence constraint from the sequence container into **Create Folder**. Then right-click the precedence constraint and click **Completion**.
9. Drag a precedence constraint from the sequence container to **Send Failure Notification**. Then right-click the precedence constraint and click **Failure**.
10. Run the package and view the results, then stop debugging.

11. Click the sequence container and press **F4**. Then in the **Properties** pane, set the **Disable** property to **True**.
12. Run the package again and note that neither of the tasks in the sequence container is executed. Then stop debugging and close Visual Studio.

For Loop Containers



You can use a For Loop container to repeat a portion of the control flow until a specific condition is met. For example, you could run a task a specified number of times.

Conceptually, a For Loop container behaves similarly to a For Loop construct in common programming languages such as Microsoft® C#. A For Loop container uses the following expression-based properties to determine the number of iterations it performs:

- An optional initialization expression, which sets a counter variable to an initial value.
- An evaluation expression that typically evaluates a counter variable in order to exit the loop when it matches a specific value.
- An iteration expression that typically modifies the value of a counter variable.

To use a For Loop container in a control flow, drag the For Loop Container icon from the SSIS Toolbox to the control flow surface, and then double-click it to set the expression properties required to control the number of loop iterations. Then drag the tasks and containers you want to repeat into the For Loop container on the control flow surface.

Demonstration: Using a For Loop Container

In this demonstration, you will be see how to use a For Loop container.

Demonstration Steps

Use a For Loop Container

1. Ensure you have completed the previous demonstrations in this module.
2. Start Visual Studio and open the **ForLoopContainer.sln** solution in the D:\Demofiles\Mod05 folder.
3. In Solution Explorer, double-click **Control Flow.dtsx**.
4. If the Variables window is not open, on the **View** menu, click **Other Windows**, and click **Variables**. Then add a variable with the following properties:
   - **Name**: counter
   - **Scope**: Control Flow
   - **Data type**: int32
   - **Value**: 0
5. From the SSIS Toolbox, drag a **For Loop Container** to the control flow design surface.

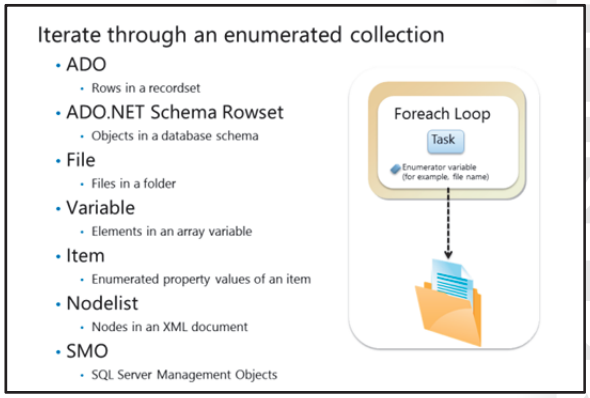
6. Double-click the For Loop container and set the following properties. Then click **OK**:
   - **InitExpression**: @counter = 1
   - **EvalExpression**: @counter < 4
   - **AssignExpression**: @counter = @counter + 1
7. From the SSIS Toolbox, drag an **Execute Process Task** into the For Loop container.
8. Double-click the **Execute Process Task** and set the following properties, then click **OK**:
   - **Name** (on the **General** tab): Open File
   - **Executable** (on the **Process** tab): Notepad.exe
   - **Expressions** (on the **Expressions** tab): Use the **Property Expressions Editor** to set the following expression for the **Arguments** property:

```
@[$Project::folderPath] + "Demo" + (DT_WSTR,1)@[User::counter] + ".txt"
```

9. Drag a precedence constraint from the **For Loop Container** to the **Sequence Container** and rearrange the control flow if necessary.
10. Run the package, and note that the For Loop starts Notepad three times, opening the text file with the counter variable value in its name (Demo1.txt, Demo2.txt, and Demo3.txt). Close Notepad each time it opens, and when the execution is complete, stop debugging.
11. Close Visual Studio, saving the solution files if prompted.

Foreach Loop Containers

You can use a Foreach Loop container to perform an iterative process on each item in an enumerated collection. SSIS supports the following enumerators in a Foreach Loop container:



- **ADO** – You can use this enumerator to loop through elements of an ADO object, for example records in a Recordset.
- **ADO.NET Schema Rowset** – You can use this enumerator to iterate through objects in an ADO.NET schema, for example tables in a dataset or rows in a table.
- **File** – You can use this enumerator to iterate through files in a folder.
- **Variable** – You can use this enumerator to iterate through elements in a variable that contains an array.
- **Item** – You can use this enumerator to iterate through a property collection for an SSIS object.
- **Nodelist** – You can use this enumerator to iterate through elements and attributes in an XML document.
- **SMO** – You can use this enumerator to iterate through a collection of SQL Server Management Objects.

To use a Foreach Loop container in a control flow:

1. Drag the **Foreach Loop Container** icon from the SSIS Toolbox to the control flow surface.
2. Double-click the Foreach Loop container and select the enumerator you want to use. Each enumerator has specific properties you need to set, for example the File enumerator requires the path to the folder containing the files you want to iterate through.
3. Specify the variable in which you want to store the enumerated collection value during each iteration.
4. Drag the tasks you want to perform during each iteration into the Foreach Loop container and configure their properties appropriately to reference the collection value variable.

Demonstration: Using a Foreach Loop Container

In this demonstration, you will see how to use a Foreach Loop Container.

Demonstration Steps

Use a Foreach Loop Container

1. Ensure you have completed the previous demonstrations in this module.
2. Start Visual Studio and open the **ForeachLoopContainer.sln** solution in the D:\Demofiles\Mod05 folder.
3. In Solution Explorer, double-click **Control Flow.dtsx**.
4. From the SSIS Toolbox, drag a **Foreach Loop Container** to the control flow design surface. Then double-click the Foreach loop container to view the **Foreach Loop Editor** dialog box.
5. On the **Collection** tab, in the **Enumerator** list, select **Foreach File Enumerator**, and in the **Expressions** box, click the ellipsis (**…**) button. Then in the **Property Expressions Editor** dialog box, in the **Property** list, select **Directory** and in the **Expression** box click the ellipsis (**…**) button.
6. In the **Expression Builder** dialog box, expand the **Variables and Parameters** folder and drag the **$Project::folderPath** parameter to the **Expression** box to specify that the loop should iterate through files in the folder referenced by the **folderPath** project parameter. Then click **OK** to close the Expression Builder, and click **OK** again to close the Property Expression Editor.
7. In the **Foreach Loop Editor** dialog box, on the **Collection** tab, in the **Retrieve file name** section, select **Name and extension** to return the file name and extension for each file the loop finds in the folder.
8. In the **Foreach Loop Editor** dialog box, on the **Variable Mappings** tab, in the **Variable** list, select **User::fName** and in the **Index** column select **0** to assign the file name of each file found in the folder to the **fName** variable. Then click **OK**.
9. Remove the precedence constraints that are connected to and from the **Copy File** task, and then drag the **Copy File** task into the **Foreach Loop Container**.
10. Create a precedence constraint from the **Create Folder** task to the **Foreach Loop Container**, and a precedence constraint from the **Foreach Loop Container** to the **Send Failure Notification** task. Then right-click the constraint between the **Foreach Loop Container** and the **Send Failure Notification** task and click **Failure**.
11. Run the package, closing each instance of Notepad as it opens. When the package execution has completed, stop debugging and close Visual Studio, saving the solution files if prompted.
12. Verify that the D:\Demofiles\Mod05\Demo folder contains each of the files in the D:\Demofiles\Mod05\Files folder.

Lab A: Implementing Control Flow in an SSIS Package

Scenario

You are implementing an ETL solution for Adventure Works Cycles and must ensure that the data flows you have already defined are executed as a workflow that notifies operators of success or failure by sending an email message. You must also implement an ETL solution that transfers data from text files generated by the company's financial accounting package to the data warehouse.

Objectives

After completing this lab, you will be able to:

- Use tasks and precedence constraints.
- Use variables and parameters.
- Use containers.

Estimated Time: 60 minutes

Virtual machine: 20463C-MIA-SQL

User name: ADVENTUREWORKS\Student

Password: Pa$$w0rd

Exercise 1: Using Tasks and Precedence in a Control Flow

Scenario

You have implemented data flows to extract data and load it into a staging database as part of the ETL process for your data warehousing solution. Now you want to coordinate these data flows by implementing a control flow that notifies an operator of the outcome of the process.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. View a Control Flow
3. Add Tasks to a Control Flow
4. Test the Control Flow

▶ Task 1: Prepare the Lab Environment

1. Ensure the 20463C-20463C-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. Run **Setup.cmd** in the D:\Labfiles\Lab05A\Starter folder as Administrator.

▶ Task 2: View a Control Flow

1. Use Visual Studio to open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab05A\Starter\Ex1 folder.
2. Open the **Extract Reseller Data.dtsx** package and examine its control flow. Note that it contains two **Send Mail** tasks – one that runs when either the **Extract Resellers** or **Extract Reseller Sales** tasks fail, and one that runs when the **Extract Reseller Sales** task succeeds.
3. Examine the settings for the precedence constraint connecting the **Extract Resellers** task to the **Send Failure Notification** task to determine the conditions under which this task will be executed.

4. Examine the settings for the **Send Mail** tasks, noting that they both use the **Local SMTP Server** connection manager.
5. Examine the settings of the **Local SMTP Server** connection manager.
6. On the **Debug** menu, click **Start Debugging** to run the package, and observe the control flow as the task executes. Then, when the task has completed, on the **Debug** menu, click **Stop Debugging**.
7. In the C:\inetpub\mailroot\Drop folder, double-click the most recent file to open it in Outlook. Then read the email message and close Outlook.

▶ Task 3: Add Tasks to a Control Flow

1. Open the **Extract Internet Sales Data.dtsx** package and examine its control flow.
2. Add a **Send Mail** task to the control flow, configure it with the following settings, and create a precedence constraint that runs this task if the **Extract Internet Sales** task succeeds:
   - **Name**: Send Success Notification
   - **SmtpConnection**: A new SMTP Connection Manager named **Local SMTP Server** that connects to the **localhost** SMTP server
   - **From**: ETL@adventureworks.msft
   - **To**: Student@adventureworks.msft
   - **Subject**: Data Extraction Notification
   - **MessageSourceType**: Direct Input
   - **MessageSource**: The Internet Sales data was successfully extracted
   - **Priority**: Normal
3. Add a second **Send Mail** task to the control flow, configure it with the following settings, and create a precedence constraint that runs this task if either the **Extract Customers** or **Extract Internet Sales** task fails:
   - **Name**: Send Failure Notification
   - **SmtpConnection**: The **Local SMTP Server** connection manager you created previously
   - **From**: ETL@adventureworks.msft
   - **To**: Student@adventureworks.msft
   - **Subject**: Data Extraction Notification
   - **MessageSourceType**: Direct Input
   - **MessageSource**: The Internet Sales data extraction process failed
   - **Priority**: High

▶ Task 4: Test the Control Flow

1. Set the **ForceExecutionResult** property of the **Extract Customers** task to **Failure**. Then run the package and observe the control flow.
2. When package execution is complete, stop debugging and verify that the failure notification email message has been delivered to the C:\inetpub\mailroot\Drop folder. You can double-click the email message to open it in Outlook.
3. Set the **ForceExecutionResult** property of the **Extract Customers** task to **None**. Then run the package and observe the control flow.

4. When package execution is complete, stop debugging and verify that the success notification email message has been delivered to the C:\inetpub\mailroot\Drop folder.
5. Close Visual Studio when you have completed the exercise.

Results: After this exercise, you should have a control flow that sends an email message if the **Extract Internet Sales** task succeeds, or sends an email message if either the **Extract Customers** or **Extract Internet Sales** tasks fail.

Exercise 2: Using Variables and Parameters

Scenario

You need to enhance your ETL solution to include the staging of payments data that is generated in comma-separated value (CSV) format from a financial accounts system. You have implemented a simple data flow that reads data from a CSV file and loads it into the staging database. You must now modify the package to construct the folder path and file name for the CSV file dynamically at run time instead of relying on a hard-coded name in the data flow task settings.

The main tasks for this exercise are as follows:

1. View a Control Flow
2. Create a Variable
3. Create a Parameter
4. Use a Variable and a Parameter in an Expression

▶ Task 1: View a Control Flow

1. View the contents of the D:\Accounts folder and note the files it contains. In this exercise, you will modify an existing package to create a dynamic reference to one of these files.
2. Open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab05A\Starter\Ex2 folder.
3. Open the **Extract Payment Data.dtsx** package and examine its control flow. Note that it contains a single data flow task named **Extract Payments**.
4. View the **Extract Payments** data flow and note that it contains a flat file source named **Payments File**, and an OLE DB destination named **Staging DB**.
5. View the settings of the **Payments File** source and note that it uses a connection manager named **Payments File**.
6. In the **Connection Managers** pane, double-click **Payments File**, and note that it references the Payments.csv file in the D:\Labfiles\Lab05A\Starter\Ex2 folder. This file has the same data structure as the payments file in the D:\Accounts folder.
7. Run the package, and stop debugging when it has completed.
8. On the **Execution Results** tab, find the following line in the package execution log:

```
[Payments File [2]] Information: The processing of the file
"D:\Labfiles\Lab05A\Starter\Ex2\Payments.csv" has started
```

▶ Task 2: Create a Variable

1. Add a variable with the following properties to the package:
   - **Name**: fName
   - **Scope**: Extract Payments Data
   - **Data type**: String
   - **Value**: Payments - US.csv

Note that the value includes a space on either side of the "-" character.

▶ Task 3: Create a Parameter

1. Add a project parameter with the following settings:
   - **Name**: AccountsFolderPath
   - **Data type**: String
   - **Value**: D:\Accounts\
   - **Sensitive**: False
   - **Required**: True
   - **Description**: Path to accounts files

Note: Be sure to include the trailing "\" in the **Value** property.

▶ Task 4: Use a Variable and a Parameter in an Expression

1. Set the **Expressions** property of the **Payments File** connection manager in the **Extract Payment Data** package so that the ConnectionString property uses the following expression:

```
@[$Project::AccountsFolderPath]+ @[User::fName]
```

2. Run the package and view the execution results to verify that the data in the D:\Accounts\Payments - US.csv file was loaded.
3. Close Visual Studio when you have completed the exercise.

Results: After this exercise, you should have a package that loads data from a text file based on a parameter that specifies the folder path where the file is stored, and a variable that specifies the file name.

Exercise 3: Using Containers

Scenario

You have created a control flow that loads Internet sales data and sends a notification email message to indicate whether the process succeeded or failed. You now want to encapsulate the data flow tasks for this control flow in a sequence container so you can manage them as a single unit.

You have also successfully created a package that loads payments data from a single CSV file based on a dynamically-derived folder path and file name. Now you must extend this solution to iterate through all the files in the folder and import data from each one.

The main tasks for this exercise are as follows:

1. Add a Sequence Container to a Control Flow

2. Add a Foreach Loop Container to a Control Flow

▶ Task 1: Add a Sequence Container to a Control Flow

1. Open the **AdventureWorksETL** solution in the D:\Labfiles\Lab05A\Starter\Ex3 folder.
2. Open the **Extract Internet Sales Data.dtsx** package and modify its control flow so that:
   - The **Extract Customers** and **Extract Internet Sales** tasks are contained in a **Sequence** container named **Extract Customer Sales Data**.
   - The **Send Failure Notification** task is executed if the **Extract Customer Sales Data** container fails.
   - The **Send Success Notification** task is executed if the **Extract Customer Sales Data** container succeeds.
3. Run the package to verify that it successfully completes both data flow tasks in the sequence and then executes the **Send Success Notification** task.

▶ Task 2: Add a Foreach Loop Container to a Control Flow

1. In the **AdventureWorksETL** solution, open the **Extract Payment Data.dtsx** package.
2. Move the existing **Extract Payments Data Flow** task into a new **Foreach Loop Container**.
3. Configure the **Foreach Loop Container** so that it loops through the files in the folder referenced by the **AccountsFolderPath** parameter, adding each file to the **fName** variable.
4. Run the package and count the number of times the Foreach Loop is executed.
5. When execution has completed, stop debugging and view the results to verify that all files in the D:\Accounts folder were processed.
6. Close Visual Studio when you have completed the exercise.

Results: After this exercise, you should have one package that encapsulates two data flow tasks in a sequence container, and another that uses a Foreach Loop to iterate through the files in a folder specified in a parameter and uses a data flow task to load their contents into a database.

Lesson 4
Managing Consistency

SSIS solutions are generally used to transfer data from one location to another. Often, the overall SSIS solution can include multiple data flows and operations, and it may be important to ensure that the process always results in data that is in a consistent state, even if some parts of the process fail.

This lesson discusses techniques for ensuring data consistency when packages fail.

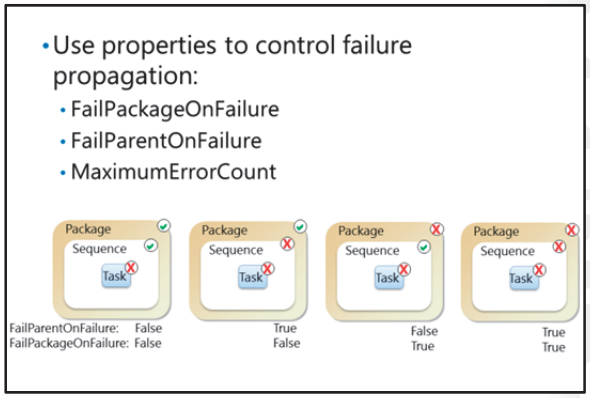
Lesson Objectives

After completing this lesson, you will be able to:

- Configure failure behavior.
- Use transactions.
- Use checkpoints.

Configuring Failure Behavior

An SSIS package control flow can contain nested hierarchies of containers and tasks. You can use the following properties to control how a failure in one element of the control flow determines the overall package outcome:

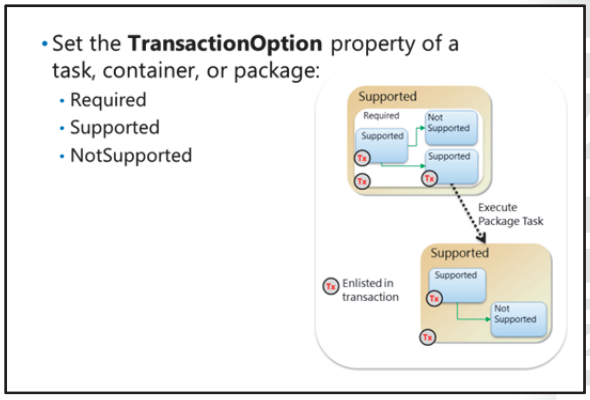


- **FailPackageOnFailure** – When set to **True**, the failure of the task or container results in the failure of the package in which it is defined. The default value for this property is **False**.
- **FailParentOnFailure** – When set to **True**, the failure of the task or container results in the failure of its container. If the item with this property is not in a container, then its parent is the package, in which case this property has the same effect as the **FailPackageOnFailure** property. When setting this property on a package executed by an **Execute Package** task in another package, a value of **True** causes the calling package to fail if this package fails. The default value for this property is **False**.
- **MaximumErrorCount** – This property specifies the maximum number of errors that can occur before the item fails. The default value for this property is **1**.

You can use these properties to achieve fine-grained control of package behavior in the event of an error that causes a task to fail.

Using Transactions

Transactions ensure that all data changes in a control flow either succeed or fail as a single, atomic unit of work. When tasks are enlisted in a transaction, a failure of any single task causes all tasks to fail, ensuring that the data affected by the control flow remains in a consistent state with no partial data modifications.



A task, container, or package's participation in a transaction is determined by its **TransactionOption** property, which you can set to one of three possible values:

- **Required** – this executable requires a transaction, and will create a new one if none exists.
- **Supported** – this executable will enlist in a transaction if its parent is participating in one.
- **NotSupported** – this executable does not support transactions and will not enlist in an existing transaction.

SSIS Transactions rely on the Microsoft Distributed Transaction Coordinator (MSDTC), a system component that coordinates transactions across multiple data sources. An error will occur if an SSIS package attempts to start a transaction when the MSDTC service is not running.

SSIS supports multiple concurrent transactions within a single hierarchy of packages, containers, and tasks, but does not support nested transactions. To understand how multiple transactions behave in a hierarchy, consider the following facts:

- If a container with a **TransactionOption** value of **Required** includes a container with a **TransactionOption** of **NotSupported**, the child container will not participate in the parent transaction.
- If the child container includes a task with a **TransactionOption** value of **Supported**, the task will not participate in the existing transaction.
- If the child container contains a task with a **TransactionOption** value of **Required**, the task will start a new transaction. However, the new transaction is unrelated to the existing transaction, and the outcome of one transaction will have no effect on the other.

Demonstration: Using a Transaction

In this demonstration, you will see how to use a transaction.

Demonstration Steps

Use a Transaction

1. If you did not complete the previous demonstrations in this module, ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**. Then, in the D:\Demofiles\Mod05 folder, run **Setup.cmd** as administrator.
2. Start SQL Server Management Studio and connect to the **localhost** database engine instance using Windows authentication.

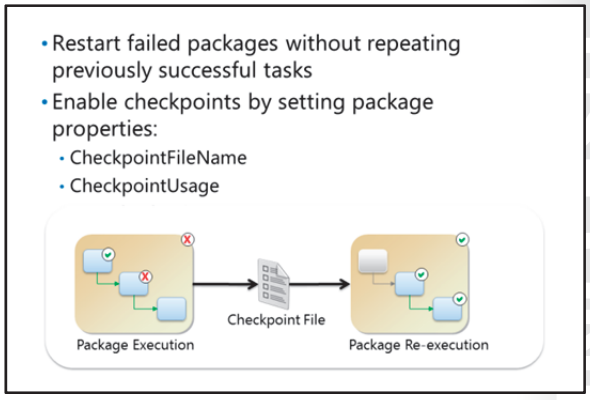
3. In Object Explorer, expand **Databases**, **DemoDW**, and **Tables**.
4. Right-click **dbo.StagingTable** and click **Select Top 1000 Rows** to verify that it contains product data.
5. Right-click **dbo.ProductionTable** and click **Select Top 1000 Rows** to verify that it is empty.
6. Start Visual Studio and open the **Transactions.sln** solution in the D:\Demofiles\Mod05 folder.
7. In Solution Explorer, double-click **Move Products.dtsx**. Note that the control flow consists of a data flow task named **Copy Products** that moves products from a staging table to a production table, and an SQL Command task named **Update Prices** that sets the product price.
8. On the **Debug** menu, click **Start Debugging** to run the package and note that the **Update Prices** task fails. Then on the **Debug** menu click **Stop Debugging**.
9. In SQL Server Management Studio, select the top 1,000 rows from the **dbo.ProductionTable** tables, noting that it now contains product data but the prices are all set to 0.00. You want to avoid having products with invalid prices in the production table, so you need to modify the SSIS package to ensure that, when the price update task fails, the production table remains empty.
10. Click **New Query**, enter the following Transact-SQL code, and then click **Execute**. This deletes all rows in the **dbo.ProductionTable** table:

```
TRUNCATE TABLE DemoDW.dbo.ProductionTable;
```

11. In Visual Studio, click anywhere on the control flow surface and press **F4**. Then in the **Properties** pane, set the **TransactionOption** property to **Required**.
12. Click the **Copy Products** task, and in the **Properties** pane, set the **FailPackageOnFailure** property to **True** and ensure the **TransactionOption** property is set to **Supported**.
13. Repeat the previous step for the **Update Prices** task.
14. Run the package and note that the **Update Prices** task fails again. Then stop debugging.
15. In SQL Server Management Studio, select the top 1,000 rows from the **dbo.ProductionTable** table, noting that it is empty, even though the **Copy Products** task succeeded. The transaction has rolled back the changes to the production table because the **Update Prices** task failed.
16. In Visual Studio, double-click the **Update Prices** task and change the **SQLStatement** property to **UPDATE ProductionTable SET Price = 100**. Then click **OK**.
17. Run the package and note that all tasks succeed. Then stop debugging and close Visual Studio.
18. In SQL Server Management Studio, select the top 1,000 rows from the **dbo.ProductionTable** table, noting that it now contains products with valid prices.

Using Checkpoints

Another way you can manage data consistency is to use checkpoints. Checkpoints enable you to restart a failed package after the issue that caused it to fail has been resolved. Any tasks that were previously completed successfully are ignored, and the execution resumes at the point in the control flow where the package failed. While checkpoints do not offer the same level of atomic consistency as a transaction, they can provide a useful solution when a control flow includes a long-running or resource-intensive task that you do not wish to repeat unnecessarily, such as downloading a large file from an FTP server.



Checkpoints work by saving information about work in progress to a checkpoint file. When a failed package is restarted, the checkpoint file is used to identify where to resume execution in the control flow. To enable a package to use checkpoints, you must set the following properties:

- **CheckpointFileName** –The full file path where you want to save the checkpoint file.
- **SaveCheckpoints** – A Boolean value used to specify whether or not the package should save checkpoint information to the checkpoint file.
- **CheckpointUsage** – An enumeration with one of the following values:
  - **Always**: The package will always look for a checkpoint file when starting. If none exists, the package will fail with an error.
  - **Never**: The package will never use a checkpoint file to resume execution and will always begin execution with the first task in the control flow.
  - **IfExists**: If a checkpoint file exists, the package will use it to resume where it failed previously. If no checkpoint file exists, the package will begin execution with the first task in the control flow.

Demonstration: Using a Checkpoint

In this demonstration, you will see how to use a checkpoint.

Demonstration Steps

Use a Checkpoint

1. If you did not complete the previous demonstrations in this module, ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**. Then, in the D:\Demofiles\Mod05 folder, run **Setup.cmd** as administrator.
2. Start SQL Server Management Studio and connect to the **localhost** database engine instance using Windows authentication.
3. In Object Explorer, expand **Databases**, expand **DemoDW**, and expand **Tables**.
4. Right-click **dbo.StagingTable** and click **Select Top 1000 Rows** to verify that it contains product data.

5. Start Excel and open **Products.csv** in the D:\Demofiles\Mod05 folder. Note that it contains details for three more products. Then close Excel.
6. Start Visual Studio and open the **Checkpoints.sln** solution in the D:\Demofiles\Mod05 folder.
7. In Solution Explorer, double-click **Load Data.dtsx**. Note that the control flow consists of a file system task to create a folder, a second file system task to copy the products file to the new folder, and a data flow task that loads the data in the products file into the staging table.
8. Click anywhere on the control flow surface to select the package, and press **F4**. Then in the **Properties** pane, set the following properties:
   - **CheckpointFileName**: D:\Demofiles\Mod05\Checkpoint.chk
   - **CheckpointUsage**: IfExists
   - **SaveCheckpoints**: True
9. Set the **FailPackageOnFailure** property for all three tasks in the control flow to **True**.
10. On the **Debug** menu, click **Start Debugging** to run the package and note that the **Load to Staging Table** task fails. Then on the **Debug** menu click **Stop Debugging**.
11. In the D:\Demofiles\Mod05 folder, note that a file named **Checkpoint.chk** has been created, and that the file system tasks that succeeded have created a folder named **Data** and copied the **Products.csv** file into it.
12. In Visual Studio, view the **Data Flow** tab for the **Load to Staging Table** task, and double-click the **Derive Columns** transformation. Then change the expression for the **NewPrice** column to **100** and click **OK**.
13. View the **Control Flow** tab, and then run the package. Note that the **Create Folder** and **Copy File** tasks, which succeeded previously, are not re-executed. Only the **Load to Staging Table** task is executed.
14. Stop debugging, and verify that the **Checkpoint.chk** file has been deleted now that the package has been executed successfully.
15. In SQL Server Management Studio, select the top 1,000 rows from the **dbo.StagingTable**, and note that it now contains data about six products.
16. Close SQL Server Management Studio and Visual Studio.

Lab B: Using Transactions and Checkpoints

Scenario

You are concerned that the Adventure Works ETL data flow might fail, leaving you with a partially-loaded staging database. To avoid this, you intend to use transactions and checkpoints to ensure data integrity.

Objectives

After completing this lab, you will be able to:

- Use transactions.
- Use checkpoints.

Estimated Time: 30 minutes

Virtual machine: 20463C-MIA-SQL

User name: ADVENTUREWORKS\Student

Password: Pa$$w0rd

Exercise 1: Using Transactions

Scenario

You have created an SSIS package that uses two data flows to extract, transform, and load Internet sales data. You now want to ensure that package execution always results in a consistent data state, so that if any of the data flows fail, no data is loaded.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. View the Data in the Database
3. Run a Package to Extract Data
4. Implement a Transaction
5. Observe Transaction Behavior

▶ Task 1: Prepare the Lab Environment

1. Ensure the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. Run **Setup.cmd** in the D:\Labfiles\Lab05B\Starter folder as Administrator.

▶ Task 2: View the Data in the Database

1. Start SQL Server Management Studio and connect to the **localhost** database engine instance by using Windows authentication.
2. In the **Staging** database, view the contents of the **dbo.Customers** and **dbo.InternetSales** tables to verify that they are both empty.

▶ Task 3: Run a Package to Extract Data

1. Use Visual Studio to open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab05B\Starter\Ex1 folder.
2. Open the **Extract Internet Sales Data.dtsx** package and examine its control flow.

3. Run the package, noting that the **Extract Customers** task succeeds, but the **Extract Internet Sales** task fails. When execution is complete, stop debugging.
4. In SQL Server Management Studio, verify that the **dbo.InternetSales** table is still empty, but the **dbo.Customers** table now contains customer records.
5. In SQL Server Management Studio, execute the following Transact-SQL query to reset the staging tables:

```
TRUNCATE TABLE Staging.dbo.Customers;
```

▶ Task 4: Implement a Transaction

1. Configure the **Extract Customer Sales Data** sequence container in the **Extract Internet Sales Data.dtsx** package so that it requires a transaction.
2. Ensure that the **Extract Customers** and **Extract Internet Sales** tasks both support transactions, and configure them so that if they fail, their parent also fails.

▶ Task 5: Observe Transaction Behavior

1. Run the **Extract Internet Sales Data.dtsx** package, noting once again that the **Extract Customers** task succeeds, but the **Extract Internet Sales** task fails. Note also that the **Extract Customer Sales Data** sequence container fails. When execution is complete, stop debugging.
2. In SQL Server Management Studio, verify that both the **dbo.InternetSales** and **dbo.Customers** tables are empty.
3. View the data flow for the **Extract Internet Sales** task, and modify the expression in the **Calculate Sales Amount** derived column transformation to remove the text "/ (OrderQuantity % OrderQuantity)". The completed expression should match the following code sample:

```
UnitPrice * OrderQuantity
```

4. Run the **Extract Internet Sales Data.dtsx** package, noting that the **Extract Customers** and **Extract Internet Sales** tasks both succeed. When execution is complete, stop debugging.
5. In SQL Server Management Studio, verify that both the **dbo.InternetSales** and **dbo.Customers** tables contain data.
6. Close Visual Studio when you have completed the exercise.

Results: After this exercise, you should have a package that uses a transaction to ensure that all data flow tasks succeed or fail as an atomic unit of work.

Exercise 2: Using Checkpoints

Scenario

You have created an SSIS package that uses two data flows to extract, transform, and load reseller sales data. You now want to ensure that if any task in the package fails, it can be restarted without re-executing the tasks that had previously succeeded.

The main tasks for this exercise are as follows:

1. View the Data in the Database
2. Run a Package to Extract Data
3. Implement Checkpoints
4. Observe Checkpoint Behavior

▶ Task 1: View the Data in the Database

1. Use SQL Server Management Studio to view the contents of the **dbo.Resellers** and **dbo.ResellerSales** tables in the **Staging** database on the **localhost** database engine instance.
2. Verify that both of these tables are empty.

▶ Task 2: Run a Package to Extract Data

1. Open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab05B\Starter\Ex2 folder.
2. Open the **Extract Reseller Data.dtsx** package and examine its control flow.
3. Run the package, noting that the **Extract Resellers** task succeeds, but the **Extract Reseller Sales** task fails. When execution is complete, stop debugging.
4. In SQL Server Management Studio, verify that the **dbo.ResellerSales** table is still empty, but the **dbo.Resellers** table now contains reseller records.
5. In SQL Server Management Studio, execute the following Transact-SQL query to reset the staging tables:

```
TRUNCATE TABLE Staging.dbo.Resellers;
```

▶ Task 3: Implement Checkpoints

1. Set the following properties of the **Extract Reseller Data** package:
   - **CheckpointFileName**: D:\ETL\CheckPoint.chk
   - **CheckpointUsage**: IfExists
   - **SaveCheckpoints**: True
2. Configure the properties of the **Extract Resellers** and **Extract Reseller Sales** tasks so that if they fail, the package also fails.

▶ Task 4: Observe Checkpoint Behavior

1. View the contents of the D:\ETL folder and verify that no file named CheckPoint.chk exists.
2. Run the **Extract Reseller Sales Data.dtsx** package, noting once again that the **Extract Resellers** task succeeds, but the **Extract Reseller Sales** task fails. When execution is complete, stop debugging.
3. View the contents of the D:\ETL folder and verify that a file named CheckPoint.chk has been created.
4. In SQL Server Management Studio, verify that the **dbo.ResellerSales** table is still empty, but the **dbo.Resellers** table now contains reseller records.

5. View the data flow for the **Extract Reseller Sales** task, and modify the expression in the **Calculate Sales Amount** derived column transformation to remove the text "/ (OrderQuantity % OrderQuantity)". The completed expression should match the following code sample:

```
UnitPrice * OrderQuantity
```

6. Run the Extract Reseller Sales Data.dtsx package, noting the Extract Resellers task is not re-executed, and package execution starts with the Extract Reseller Sales task, which failed on the last attempt. When execution is complete, stop debugging.
7. In SQL Server Management Studio, verify that the **dbo.ResellerSales** table now contains data.
8. Close Visual Studio when you have completed the exercise.

Results: After this exercise, you should have a package that uses checkpoints to enable execution to be restarted at the point of failure on the previous execution.

Module Review and Takeaways

In this module, you have learned how to implement control flow in an SSIS package, and how to use transactions and checkpoints to ensure data integrity when a package fails.

Review Question(s)

Question: You want Task 3 to run if Task 1 or Task 2 fails. How can you accomplish this?

Question: Which container should you use to perform the same task once for each file in a folder?

Question: Your package includes an FTP task that downloads a large file from an FTP folder, and a data flow task that inserts data from the file into a database. The data flow task may fail because the database is unavailable, in which case you plan to run the package again after bringing the database online. How can you avoid downloading the file again when the package is re-executed?

Module 6

Debugging and Troubleshooting SSIS Packages

Contents:

Module Overview

As you develop more complex SQL Server Integration Services (SSIS) packages, it is important to be familiar with the tools and techniques you can use to debug package execution and handle any errors. This module describes how you can debug packages to find the cause of errors that occur during execution. It then discusses the logging functionality, built into SSIS that you can use to log events for troubleshooting purposes. Finally, the module describes common approaches for handling errors in control flow and data flow.

Objectives

After completing this module, you will be able to:

- Debug an SSIS package.
- Implement logging for an SSIS package.
- Handle errors in an SSIS package.

Lesson 1
Debugging an SSIS Package

When you are developing an application, misconfiguration of tasks or data flow components, or errors in variable definitions or expressions, can lead to unexpected behavior. Even if you develop your package perfectly, there are many potential problems that might arise during execution, such as a missing or misnamed file, or an invalid data value. It is therefore important to be able to use debugging techniques to find the cause of these problems, and formulate a solution.

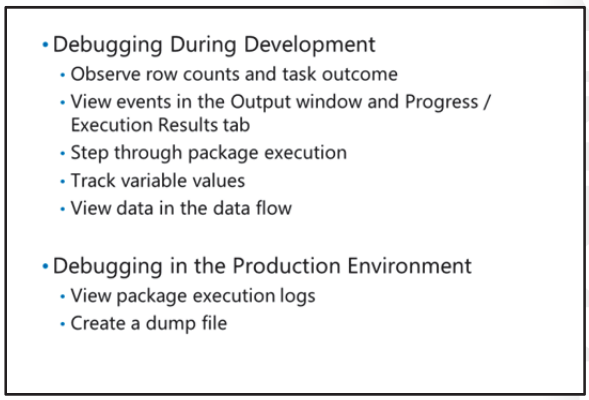
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the tools and techniques for debugging SSIS packages.
- View package execution events.
- Use breakpoints to pause package execution.
- View variable values and status while debugging.
- Use data viewers to view data flow values while debugging.

Overview of SSIS Debugging

Debugging is the process of finding the source of problems that occur during package execution, either during development or in a package that has been deployed to a production environment.



Debugging During Development

At design time, SSIS developers can use a variety of Visual Studio debugging techniques to find problems in control flow and data flow processes. These techniques include:

- Observing row counts and task outcome indicators when running packages in the debugging environment.
- Viewing events that are recorded during package execution. These events are shown in the **Progress** tab during execution, and in the **Execution Results** tab after execution. Events are also shown in the **Output** window during and after each execution.
- Stepping through package execution by setting breakpoints that pause execution at specific points in the control flow.
- Viewing variable values while debugging.
- Viewing the rows that pass through the data flow pipeline by attaching data viewers to data flow paths.

Note: Microsoft® Visual Studio® includes a number of debugging windows and tools that are primarily designed for debugging software solutions with programming languages such as Microsoft® Visual C#®. This lesson focuses on the debugging tools in the Visual Studio environment that is provided with the SQL Server Data Tools for BI add-in and designed for debugging SSIS packages.

Debugging in the Production Environment

It is common for problems to occur during execution after a package has been deployed to the production environment. In this scenario, if the package source project is available, you can use the techniques previously described. However, you can also debug the package by examining any log files it is configured to generate, or by using the dtexec or dtutil utilities to generate a dump file. These files contain information about system variable values and settings that you can use to diagnose a problem with a package.

Viewing Package Execution Events

You can think of a package execution as a sequence of events generated by the tasks and containers in the package control flow. When you run a package in debug mode in the development environment, these events are recorded and displayed in two locations. To run a package in debug mode, you can use any of the following techniques:

- Package execution is a sequence of events generated by tasks and containers
- During debugging, events are shown:
  - Progress / Execution Results Tab
  - Output Window

- On the **Debug** menu, click **Start Debugging**.
- Click the **Start Debugging** button on the toolbar.
- Press **F5**.

The Progress / Execution Results Tab

During execution, the Progress tab of the SSIS package designer shows a hierarchical view of the package and its containers and tasks, displaying information about events that occur during execution. When execution is complete, the tab is renamed Execution Results and shows the entire event tree for the completed execution.

You can enable or disable the display of messages on the Progress tab by toggling Debug Progress Reporting on the SSIS menu. Disabling progress reporting can help improve performance when debugging complex packages.

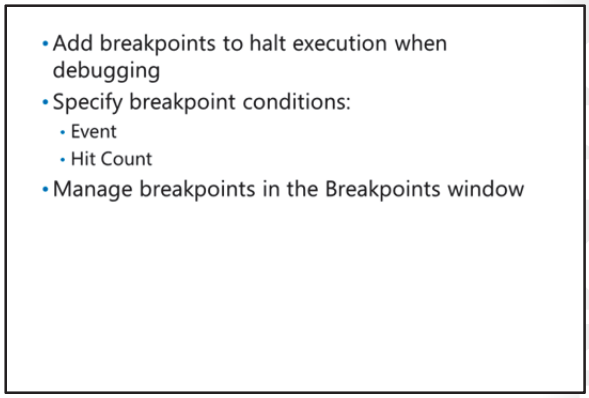
The Output Window

The Output window shows the list of events that occur during execution. After execution is complete, you can review the Output window to find details of the events that occurred.

The Output window and the Progress / Execution Results tab are useful resources for troubleshooting errors during package execution. As an SSIS developer, you should habitually review the events in these windows when debugging packages.

Breakpoints

To troubleshoot a problem with an event in a package, you can use breakpoints to pause execution at the stage in the control flow where the error occurs.



You can create a breakpoint for events raised by any container or task in a control flow. The simplest way to create a breakpoint is to select the task or container where you want to pause execution, and on the **Debug** menu, click **Toggle Breakpoint**. This adds a breakpoint at the **OnPreExecute** event of the selected task or container, which is the first event a task or container raises during package execution.

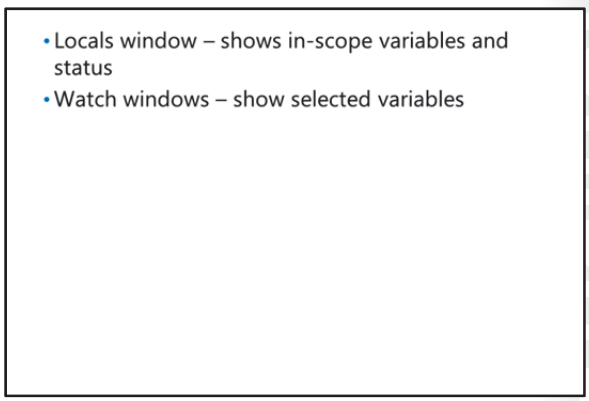
For greater control of when a breakpoint pauses execution, you can right-click any task or container and click **Edit Breakpoints** to display the **Set Breakpoints** dialog box for the task or container. In this dialog box you can:

- Enable breakpoints for any event supported by the task or container.
- Specify a **Hit Count Type** and **Hit Count Value** to control how often the event is ignored before the breakpoint pauses execution. You can set the **Hit Count Type** to one of the following settings:
  - **Always** – The **Hit Count Value** is ignored and execution is always paused at this event.
  - **Hit count equals** – Execution is paused when the event has been raised the number of times specified in the **Hit Count** property.
  - **Hit greater or equal** – Execution is paused when the event has been raised the number of times specified in the **Hit Count** property or more.
  - **Hit count multiple** – Execution is paused when the event has been raised a number of times that is a multiple of the **Hit Count** property or more.

You can view and manage all the breakpoints that are set in a package in the **Breakpoints** window. You can display this window by clicking the **Debug** menu, clicking **Windows**, and clicking **Breakpoints**.

Variable and Status Windows

When you have used a breakpoint to pause package execution, it can be useful to view the current values assigned to variables, parameters, and other system settings. SQL Server Data Tools provides two windows you can use to observe these values while debugging.



The Locals Window

The Locals window is a pane in the SQL Server Data Tools environment that lists all the system settings, variables, and parameters that are currently in scope. You can use this window to find current values for these settings, variables, and parameters in the execution context.

To view the Locals window when package execution is paused by a breakpoint, click Windows on the Debug menu, and then click Locals.

Watch Windows

If you want to track specific variable or parameter values while debugging, you can add a watch for each value you want to track. Watched values are shown in watch windows named Watch 1, Watch 2, Watch 3, and Watch 4. However, in most SSIS debugging scenarios, only Watch 1 is used.

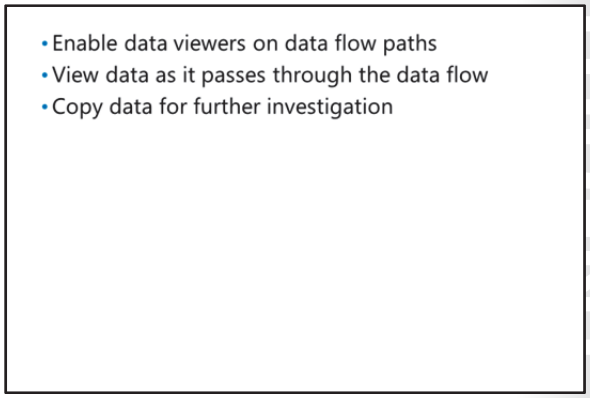
To display a watch window while debugging, on the Debug menu, click Windows, click Watch, and then click the watch window you want to display.

To add a value to the Watch 1 window, right-click the variable or parameter you want to track in the Locals window, and click Add Watch.

To add a variable or parameter to another watch window, drag it from the Locals window to the watch window in which you want it to be displayed.

Data Viewers

Most SSIS packages are designed primarily to transfer data. When debugging a package, it can be useful to examine the data as it passes through the data flow. Data viewers provide a way to view the data rows as they pass along data flow paths between sources, transformations, and destinations.



Enabling a Data Viewer

To enable a data viewer, right-click a data flow path on the Data Flow tab and click Enable Data Viewer. Alternatively, you can double-click a data flow path on the Data Viewer tab of the Data Flow Path Editor dialog box. Using the dialog box also enables you to select specific columns to be included in the data viewer.

Viewing Data in the Data Flow

A data viewer behaves like a breakpoint, and pauses execution at the data flow path on which it is defined. When a data viewer pauses execution, a window containing the data in the data flow path is displayed, enabling you to examine the data at various stages. After examining the data, you can resume execution by clicking the green Continue arrow button in the data viewer window. If you no longer require the data viewer, you can remove it by clicking the Detach button in the data viewer window.

Copying Data from a Data Viewer

The data viewer window includes a Copy button, which you can use to copy the contents of the data viewer to the Microsoft® Windows® clipboard. When a data flow contains a large number of rows, it can be useful to copy the contents of a data viewer and paste the data into a tool such as Microsoft® Excel® for further examination.

Demonstration: Debugging a Package

In this demonstration, you will see how to:

- Add a Breakpoint.
- View Variables while Debugging.
- Enable a Data Viewer.

Demonstration Steps

Add a Breakpoint

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are started, and log onto 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Demofiles\Mod06 folder, run **Setup.cmd** as Administrator.
3. Start Visual Studio and open the **Debugging.sln** solution in the D:\Demofiles\Mod06 folder.
4. In Solution Explorer, double-click **Debugging Demo.dtsx**. This package includes a control flow that performs the following tasks:
    - Copies a text file using a variable named **User::sourceFile** to determine the source path and a variable named **User::copiedFile** to determine the destination path.
    - Uses a data flow to extract the data from the text file, convert columns to appropriate data types, and load the resulting data into a database table.
    - Deletes the copied file.
5. On the **Debug** menu, click **Start Debugging**, note that the first task fails, and on the **Debug** menu, click **Stop Debugging**.
6. Click the **Copy Source File** task and on the **Debug** menu, click **Toggle Breakpoint**. Right-click the **Copy Source File** task and click **Edit Breakpoints**. Note that you can use this dialog box to control the events and conditions for breakpoints in your package. When you toggle a breakpoint, by default it is enabled for the **onPreExecute** event with a **Hit Count Type** value of **Always**. Then click **OK**.
7. Start debugging and note that execution stops at the breakpoint.

View Variables while Debugging

1. With execution stopped at the breakpoint, on the **Debug** menu, click **Windows**, and click **Locals**.
2. In the **Locals** pane, expand **Variables** and find the **user:copiedFile** variable, then right-click it and click **Add Watch**. The **Watch 1** pane is then shown with the **user::copiedFile** variable displayed.
3. Click the **Locals** pane and find the **user:sourceFile** variable, then right- click it and click **Add Watch**. The **Watch 1** pane is then shown with the **user::copiedFile** and **user:sourceFile** variables displayed.
4. Note that the value of the **user:sourceFile** variable is D:\\Demofiles\\Mod06\\Products.txt ("\" is used as an escape character, so "\\" is used to indicate a literal "\" string), and in the D:\Demofiles\Mod06 folder, note that the file is actually named Products.csv.
5. Stop debugging, and on the SSIS menu, click **Variables**. Then in the **Variables** pane, change the value for the **sourceFile** variable to D:\Demofiles\Mod06\Products.csv.
6. Start debugging and observe the variable values in the **Watch 1** pane. Note that the **sourceFile** variable now refers to the correct file.
7. On the **Debug** menu, click **Continue** and note that the **Load Data** task fails. Then stop debugging.

Enable a Data Viewer

1. Double-click the **Load Data** task to view the data flow design surface.
2. Right-click the data flow path between **Products File** and **Data Conversion**, and click **Enable Data Viewer**.
3. Double-click the data flow path between **Products File** and **Data Conversion**, and in the **Data Flow Path Editor** dialog box, click the **Data Viewer** tab. Note that you can use this tab to enable the data viewer and specify which columns should be included, and that by default, all columns are included. Then click **OK**.
4. Click the **Control Flow** tab and verify that a breakpoint is still enabled on the **Copy Source File** task. Then on the **Debug** menu, click **Start Debugging**.
5. When execution stops at the breakpoint, on the **Debug** menu, click **Continue**.
6. When the data viewer window is displayed, resize it so you can see the data it contains, and note that the **Price** column for the second row contains a "-" character instead of a number.
7. In the data viewer window, click **Copy Data**. Then click the green continue button in the data viewer window.
8. When execution stops because the data flow task has failed, on the **Debug** menu, click **Stop Debugging**.
9. Start Excel, and create a new blank workbook.
10. With cell A1 selected, on the **Home** tab of the ribbon, click **Paste**. Then view the data you have pasted from the data viewer.
11. Close Excel without saving the workbook, and close Visual Studio.

Lesson 2
Logging SSIS Package Events

Visual Studio debugging tools can be extremely useful when developing a package. However, after a package is in production, it can be easier to diagnose a problem if the package provides details in a log of the events that occurred during execution. In addition to using a log for troubleshooting, you might want to record details of package execution for auditing or performance benchmarking purposes. Planning and implementing a suitable logging solution is an important part of developing a package, and SSIS includes built-in functionality to help you accomplish this.

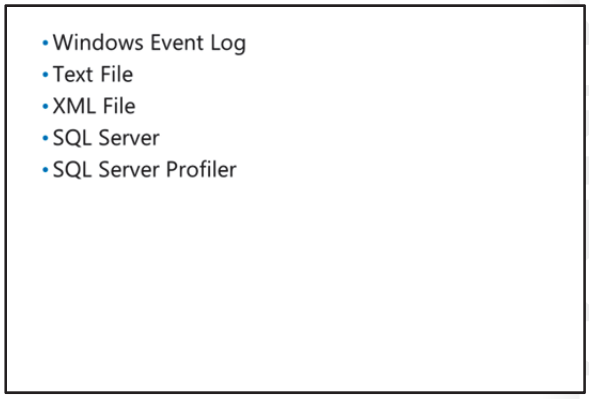
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the log providers available in SSIS.
- Describe the events that can be logged and the schema for logging information.
- Implement logging in an SSIS package.
- View logged events.

SSIS Log Providers

The SSIS logging architecture supports the recording of event information to one or more logs. Each log is accessed through a log provider that determines its type and the connection details used to access it.



SSIS includes the following log providers:

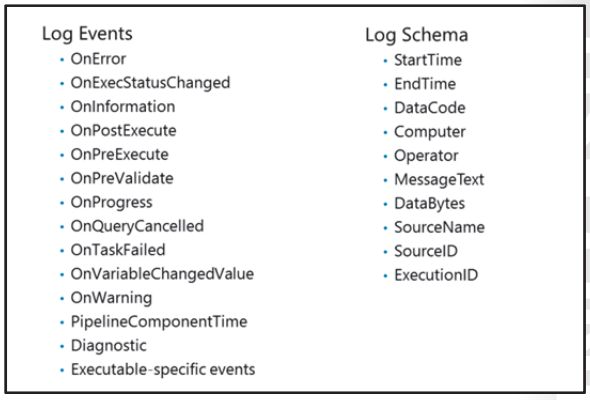
- **Windows Event Log** – Logs event information in the **Application** Windows event log. No connection manager is required for this log provider.
- **Text File** – Logs event information to a text file specified in a file connection manager.
- **XML File** – Logs event information to an XML file specified in a file connection manager.
- **SQL Server** – Logs event information in the **sysssislog** system table in a Microsoft® SQL Server® database, which is specified in an OLE DB connection manager.
- **SQL Server Profiler** – Logs event information in a .trc file that can be examined in SQL Server profiler. The location of the .trc file is specified in a file connection manager. This log provider is only available in 32-bit execution environments.

Additionally, software developers can use the Microsoft .NET Framework to develop custom log providers.

When deciding which log providers to include in your logging solution, you should generally try to comply with standard logging procedures in the existing IT infrastructure environment. For example, if administrators in the organization typically use the Windows Event Log as the primary source of troubleshooting information, you should consider using it for your SSIS packages. When using files or SQL Server tables for logging, you should also consider the security of the log, which may contain sensitive information.

Log Events and Schema

Having determined the log providers you want to use, you can select the events for which you want to create log entries and the details you want to include in them.



Log Events

SSIS logging supports the following events:

- **OnError** – This event is raised when an error occurs.
- **OnExecStatusChanged** – This event is raised when a task is paused or resumed.
- **OnInformation** – This event is raised during validation and execution to report information.
- **OnPostExecute** – This event is raised when execution of an executable has completed.
- **OnPreExecute** – This event is raised before an executable starts running.
- **OnPreValidate** – This event is raised when validation of an executable begins.
- **OnProgress** – This event is raised to indicate execution progress for an executable.
- **OnQueryCancelled** – This event is raised when execution is cancelled.
- **OnTaskFailed** – This event is raised when a task fails.
- **OnVariableChangedValue** – This event is raised when a variable has its value changed.
- **OnWarning** – This event is raised when a warning occurs.
- **PipelineComponentTime** – This event is raised to indicate the processing time for each phase of a data flow component.
- **Diagnostic** – This event is raised to provide diagnostic information.
- **Executable-specific events** – Some containers and tasks provide events that are specific to the executable. For example, a Foreach Loop container provides an event that is raised at the start of each loop iteration.

While it may be tempting to log every event, you should consider the performance overhead incurred by the logging process. The choice of events to log depends on the purposes of the logging solution. For example, if your goal is primarily to provide troubleshooting information when exceptions occur, you should consider logging the **OnError**, **OnWarning**, and **OnTaskFailed** events. If your log will be used for auditing purposes, you might want to log the **OnInformation** event, and if you want to use your log to measure package performance, you may consider logging the **OnProgress** and **PipelineComponentTime** events.

Log Schema

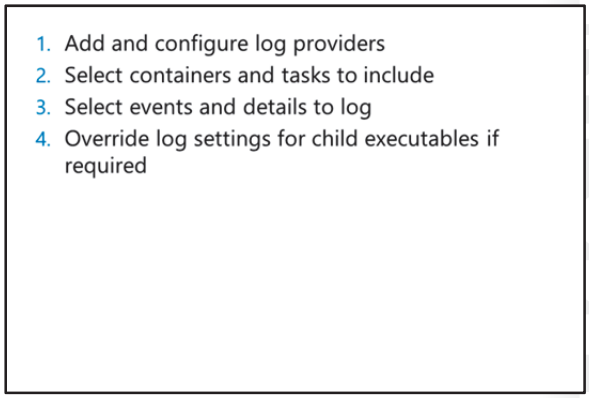
The specific details that can be logged for each event are defined in the SSIS log schema. This schema includes the following values:

- **StartTime** – When the executable started running.
- **EndTime** – When the executable finished.

- **DataCode** – An integer value indicating the execution result:
  - **0**: Success
  - **1**: Failure
  - **2**: Completed
  - **3**: Cancelled
- **Computer** – The name of the computer on which the package was executed.
- **Operator** – The Windows account that initiated package execution.
- **MessageText** – A message associated with the event.
- **DataBytes** – A byte array specific to the log entry.
- **SourceName** – The name of the executable.
- **SourceID** – The unique identifier for the executable.
- **ExecutionID** – A unique identifier for the running instance of the package.

You can choose to include all elements of the schema in your log, or select individual values to reduce log size and performance overhead. However, the **StartTime**, **EndTime**, and **DataCode** values are always included in the log.

Implementing SSIS Logging



Packages can be thought of as a hierarchy of containers and tasks, with the package itself as the root of the hierarchy. You can configure logging at the package level in the hierarchy, and by default, all child containers and tasks inherit the same logging settings. If required, you can override inherited logging settings for any container or task in the hierarchy. For example, you might choose to log only **OnError** events to the Windows Event Log provider at the package level and inherit these settings for most child containers and tasks, but configure a data flow task within the package to log **OnInformation** and Diagnostic events to the SQL Server log provider.

To implement logging for an SSIS package, in SQL Server Data Tools, with the package open in the designer, on the SSIS menu, click Logging to display the Configure SSIS Logs dialog box. Then perform the following steps:

1. Add and configure log providers. On the **Providers and Logs** tab of the dialog box, add the log providers you want to use. For providers other than the one for Windows Event Log, you must also specify a connection manager that defines the file or SQL Server instance where you want to write the log information.
2. Select containers and tasks to include. Select the package container, which by default selects all child containers and tasks with inherited log settings. You can then de-select individual containers and tasks that you do not want to include.

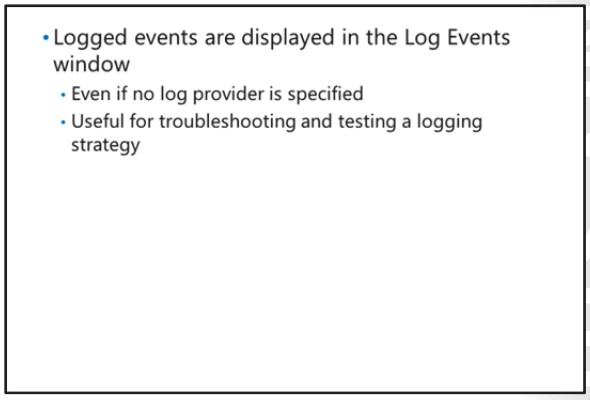
3. Select events and details to log. On the **Details** tab of the dialog box, select the events you want to include in the log. By default, all schema fields are logged for the selected events, but you can click the **Advanced** button to choose individual fields.
4. Override log settings for child executables if required. If you want to specify individual logging settings for a specific child executable, select the executable in the **Containers** tree and specify the log provider, events, and details you want to use for that executable.

Viewing Logged Events

You can view logged events in Visual Studio by displaying the Log Events window. When logging is configured for a package, this window shows the selected log event details, even when no log provider is specified.

The Log Events window is a useful tool for troubleshooting packages during development, and also for testing and debugging logging configuration.

To display the Log Events window, on the **SSIS** menu, click **Log Events**.



Demonstration: Logging Package Execution

In this demonstration, you will see how to:

- Configure SSIS Logging.
- View Logged Events.

Demonstration Steps

Configure SSIS Logging

1. Ensure you have completed the previous demonstration in this module.
2. Start Visual Studio and open the **Logging.sln** solution in the D:\Demofiles\Mod06 folder.
3. In Solution Explorer, double-click **Logging Demo.dtsx**.
4. On the **SSIS** menu, click **Logging**. If an error message is displayed, click **OK**.
5. In the **Configure SSIS Logs: Logging Demo** dialog box, in the **Provider type** list, select **SSIS log provider for Windows Event Log** and click **Add**. Then select **SSIS log provider for SQL Server** and click **Add**.
6. In the **Configuration** column for the SSIS log provider for SQL Server, select the **(local).DemoDW** connection manager. Note that the Windows Event Log provider requires no configuration.
7. In the **Containers** tree, check the checkbox for **Logging Demo**, and then with **Logging Demo** selected, on the **Providers and Logs** tab, check the checkbox for the **SSIS log provider for Windows Event Log**.
8. With **Logging Demo** selected, on the **Details** tab, select the **OnError** and **OnInformation** events.

9. Click the **Providers and Logs** tab, and in the **Containers** tree, clear the checkbox for **Load Data**, and then click the checkbox again to check it. This enables you to override the inherited logging settings for the **Load Data** task.
10. With **Load Data** selected, on the **Providers and Logs** tab, check the checkbox for the **SSIS log provider for SQL Server**.
11. With **Load Data** selected, on the **Details** tab, select the **OnError** and **OnInformation** events, and then click **Advanced** and clear the **Operator** column for the two selected events. Then click **OK**.

View Logged Events

1. On the **Debug** menu, click **Start Debugging**. Then, when the **Load Data** task fails, on the **Debug** menu click **Stop Debugging**.
2. On the **SSIS** menu, click **Log Events**. This shows the events that have been logged during the debugging session (if the log is empty, re-run the package and then view the Log Events window again).
3. On the Start screen, type **Event** and run the **Event Viewer** app. Then expand **Windows Logs**, and click **Application**. Note the log entries with a source of **SQLISPackage120**. These are the logged events for the package.
4. Start SQL Server Management Studio and connect to the **localhost** instance of the database engine by using Windows authentication.
5. In Object Explorer, expand **Databases**, expand **DemoDW**, expand **Tables**, and expand **System Tables**. Then right-click **dbo.sysssislog** and click **Select Top 1000 Rows**.
6. View the contents of the table, noting that the **Operator** column is empty.
7. Close SQL Server Management Studio without saving any files, then close Event Viewer and Visual Studio.

Lesson 3
Handling Errors in an SSIS Package

No matter how much debugging you perform, or how much information you log during package execution, exceptions that cause errors can occur in any ETL process. For example, servers can become unavailable, files can be renamed or deleted, and data sources can include invalid entries. A good SSIS solution includes functionality to handle errors by performing compensating tasks and continuing with execution wherever possible, or by ensuring that temporary resources are cleaned up and operators notified where execution cannot be continued.

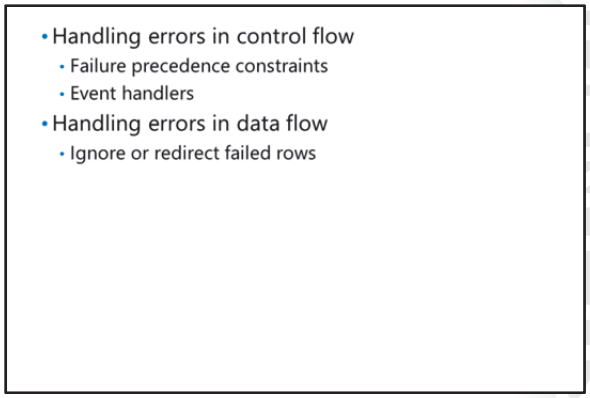
Lesson Objectives

After completing this lesson, you will be able to:

- Describe approaches for handling errors in an SSIS package.
- Implement event handlers.
- Handle errors in data flows.

Introduction to Error Handling

Errors can occur at any stage in the execution of a package, and SSIS provides several ways to handle them and take corrective action if possible.



Handling Errors in Control Flow

You can use the following techniques to handle errors in package control flow:

- Use **Failure** precedence constraints to redirect control flow when a task fails. For example, you can use a **Failure** precedence constraint to execute another task that performs a compensating alternative action to allow the control flow to continue, or to delete any temporary files and send an email notification to an operator.
- Implement event handlers to execute a specific set of tasks when an event occurs in the control flow. For example, you could implement an event handler for the **OnError** event of the package, and include tasks to delete files and send email notifications in the **OnError** event handler.

Note: Precedence constraints are discussed in Module 5: *Implementing Control Flow in an SSIS Package*. The remainder of this lesson focuses on using event handlers to handle errors in control flow.

Handling Errors in Data Flow

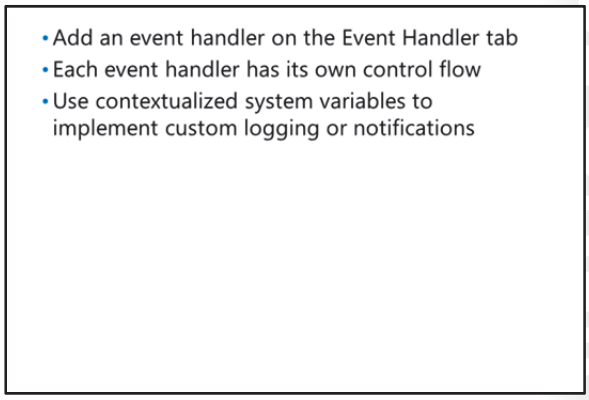
Errors in data flow can often be caused by invalid or unexpected data values in rows being processed by the data flow pipeline. SSIS data flow components provide the following configuration options for handling rows that cause errors:

- Fail the task if any rows cause an error.
- Ignore errors and continue the data flow.

- Redirect rows that cause an error to the error output of the data flow component.

Implementing Event Handlers

You can add an event handler for the events supported by each executable in the package. To add an event handler to a package, click the **Event Handlers** tab, select the executable and event for which you want to implement a handler, and click the hyperlink on the design surface. Doing this creates a new control flow surface on which you can define the control flow for the event handler. Event handlers can be used for all kinds of control flow tasks, and are not specific to handling errors. However, the **OnError** and **OnTaskFailed** events are commonly used for handling error conditions.

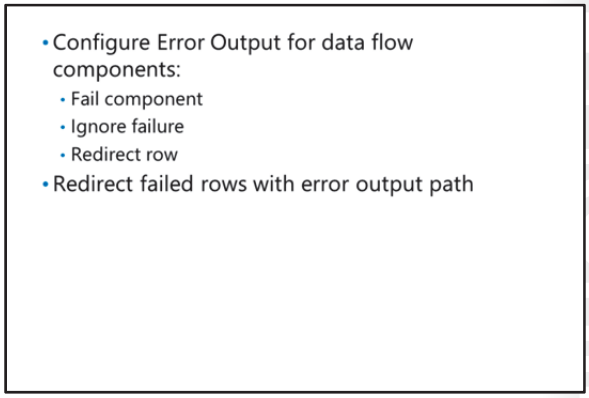


The system variables and configuration values available to tasks in your event handler are specific to the context of the event. For example, the **System::ErrorDescription** variable is populated with an error message during the **OnError** event.

Because a package is a hierarchy of containers and tasks, each with their own events, you need to consider where best to handle each possible error condition. For example, you can handle a task-specific error in the task's own **OnError** event or, depending on the **MaxErrors** property of containers and the package itself, the error could trigger **OnError** events further up the package hierarchy, where you could also handle the error condition. In general, if you anticipate specific errors that you can resolve or compensate for and continue execution, you should implement the **OnError** event handler for the task or container where the error is likely to occur. You should use the **OnError** event handler of the package to catch errors that cannot be resolved, and use it to perform clean-up tasks and notify operators.

Handling Data Flow Errors

Data flow components participate in a data flow pipeline through which rows of data are passed along data flow paths. Errors can occur in data flow for a number of reasons, including:



- Rows that contain data of data type that is incompatible with a transformation or destination, such as a decimal field in a text file that is mapped to an integer column in a destination.
- Rows that contain invalid data values, such as a text file that contains a date field with an invalid date value.
- Rows that contain data that will be truncated by a transformation or destination, such as a 50-character text field that is loaded into a table where the mapped column has a maximum length of 40 characters.
- Rows that contain data values that will cause an exception during a transformation, such as a numerical field with a value of zero that is used as a divisor in a derived column transformation.

By default, rows that cause an error result in the failure of the data flow component. However, you can configure many data flow components to ignore rows that contain errors, or to redirect them to the error output data flow path of the component. Ignoring or redirecting failed rows enables the data flow to complete for all other rows, and if you have chosen to redirect failed rows, you can use transformations to attempt to correct the invalid data values, or save the failed rows to a file or table for later analysis.

When configuring error output for a data flow component, you can specify different actions for truncations and errors. For example, you could choose to ignore truncations, but redirect rows that contain other errors. Additionally, some components enable you to specify different actions for each column in the data flow. You could ignore errors in one column while redirecting rows that have an invalid value in another.

Redirected rows include all the input columns for the data flow component, and two additional columns:

- **ErrorCode** – The numeric code for the error.
- **ErrorColumn** – The original number of the column that caused the error.

Demonstration: Handling Errors

In this demonstration, you will see how to:

- Implement an Event Handler.
- Redirect Failed Rows.

Demonstration Steps

Implement an Event Handler

1. Ensure you have completed the previous demonstration in this module.
2. Start Visual Studio and open the **Error Handling.sln** solution in the D:\Demofiles\Mod06 folder.
3. In Solution Explorer, double-click **Error Handling Demo.dtsx**, and then click the **Event Handlers** tab.
4. On the **Event Handlers** tab, in the **Executable** list, ensure **Error Handling Demo** is selected, and in the **Event Handler** list, ensure **OnError** is selected. Then click the hyperlink in the middle of the design surface.
5. In the SSIS Toolbox, double-click **Send Mail Task**, and then on the design surface, right-click **Send Mail Task**, click **Rename**, and change the name to **Notify administrator**.
6. Double-click **Notify administrator**, and on the **Mail** tab of the **Send Mail Task Editor** dialog box, configure the following properties:
    - **SmtpConnection**: A new connection to the **localhost** SMTP server with default settings
    - **From**: etl@adventureworks.msft
    - **To**: administrator@adventureworks.msft
    - **Subject**: An error has occurred
7. In the **Send Mail Task Editor** dialog box, on the **Expressions** tab, click the ellipsis (**…**) button in the **Expressions** box. Then in the **Property Expressions Editor** dialog box, in the **Property** list select **MessageSource**, and in the **Expression** box, click the ellipsis (**…**) button.
8. In the **Expression Builder** dialog box, expand **Variables and Parameters**, expand **System Variables**, and drag **System:ErrorDescription** to the **Expression** box. Then click **OK**.

9. In the **Property Expressions Editor** dialog box, in the row under the **MessageSource** property, in the **Property** list select **FileAttachments**. Then in the **Expression** box, click the ellipsis (**…**) button.
10. In the **Expression Builder** dialog box, expand **Variables and Parameters**, and drag **User::sourceFile** to the **Expression** box. Then click **OK**.
11. In the **Property Expressions Editor** dialog box, click **OK**. Then in the **Send Mail Task Editor** dialog box, click **OK**.
12. Click the **Control Flow** tab, and then on the **Debug** menu, click **Start Debugging**. When the **Load Data** task fails, click the **Event Handlers** tab to verify that the **OnError** event handler has been executed, and then on the **Debug** menu, click **Stop Debugging**.
13. In the C:\inetpub\mailroot\Drop folder, double-click the most recent email message to open it in Outlook and view its contents. Then close Outlook.

Redirect Failed Rows

1. In Visual Studio, click the **Data Flow** tab of the **Error Handling Demo.dtsx** package, and in the **Data Flow Task** drop-down list, ensure **Load Data** is selected.
2. Double-click **Data Conversion**, and then in the **Data Conversion Transformation Editor** dialog box, click **Configure Error Output**.
3. In the **Configure Error Output** dialog box, click the **Error** cell for the **Numeric ProductID** column, and then hold the **Ctrl** key and click the **Error** cell for the **Numeric Price** column so that both cells are selected. Then in the **Set this value to the selected cells** list, select **Redirect row** and click **Apply**.
4. Click **OK** to close the **Configure Error Output** dialog box, and then click **OK** to close the **Data Conversion Transformation Editor** dialog box.
5. In the SSIS Toolbox, in the **Other Destinations** section, double-click **Flat File Destination**. Then on the design surface, right-click **Flat File Destination**, click **Rename**, and change the name to **Invalid Rows**.
6. Move **Invalid Rows** to the right of **Data Conversion**, then click **Data Conversion** and drag the red data path from **Data Conversion** to **Invalid Rows**. In the **Configure Error Output** dialog box, verify that the **Numeric ProductID** and **Numeric Price** columns both have an **Error** value of **Redirect row**, and click **OK**.
7. Double-click **Invalid Rows**, and next to the **Flat File connection manager** drop-down list, click **New**.
8. In the **Flat File Format** dialog box, ensure **Delimited** is selected and click **OK**. Then in the **Flat File Connection Manager Editor** dialog box, in the **Connection manager name** box, type **Invalid Rows CSV File**, in the **File** name box type **D:\Demofiles\Mod06\InvalidRows.csv**, and click **OK**.
9. In the **Flat File Destination Editor** dialog box, click the **Mappings** tab and note that the input columns include the columns from the data flow, an **ErrorCode** column, and an **ErrorColumn** column. Then click **OK**.
10. Click the **Control Flow** tab and on the **Debug** menu, click **Start Debugging**. Note that all tasks succeed, and on the **Data Flow** tab note the row counts that pass through each data flow path. Then on the **Debug** menu, click **Stop Debugging** and close Visual Studio.
11. Start Excel and open **InvalidRows.csv** in the D:\Demofiles\Mod06 folder. View the rows that were redirected, and then close Excel without saving the workbook.

Lab: Debugging and Troubleshooting an SSIS Package

Scenario

The ETL process for Adventure Works Cycles occasionally fails when extracting data from text files generated by the company's financial accounts system. You plan to debug the ETL process to identify the source of the problem and implement a solution to handle any errors.

Objectives

After completing this lab, you will be able to:

- Debug an SSIS package.
- Log SSIS package execution.
- Implement an event handler in an SSIS package.
- Handle errors in a data flow.

Estimated Time: 60 minutes

Virtual machine: 20463C-MIA-SQL

User name: ADVENTUREWORKS\Student

Password: Pa$$w0rd

Exercise 1: Debugging an SSIS Package

Scenario

You have developed an SSIS package to extract data from text files exported from a financial accounts system, and load the data into a staging database. However, while developing the package you have encountered some errors, and you need to debug it to identify the cause.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Run an SSIS Package
3. Add a Breakpoint
4. Add a Data Viewer
5. View Breakpoints
6. Observe Variables while Debugging
7. View Data Copied from a Data Viewer

▶ Task 1: Prepare the Lab Environment

1. Ensure the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as ADVENTUREWORKS\Student with the password Pa$$w0rd.
2. Run **Setup.cmd** in the D:\Labfiles\Lab06\Starter folder as Administrator.

▶ Task 2: Run an SSIS Package

1. Open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab06\Starter\Ex1 folder.
2. Open the **Extract Payment Data.dtsx** package and examine its control flow.
3. Run the package, noting that it fails. When execution is complete, stop debugging.

▶ Task 3: Add a Breakpoint

1. Add a breakpoint to the Foreach Loop container in the Extract Payment Data.dtsx package.
2. Select the appropriate event to ensure that execution is always paused at the beginning of every iteration of the loop.

▶ Task 4: Add a Data Viewer

1. In the data flow for the **Extract Payments** task, add a data viewer to the data flow path between the **Payments File** source and the **Staging DB** destination.

▶ Task 5: View Breakpoints

1. View the **Breakpoints** window, and note that it contains the breakpoint you defined on the Foreach Loop container and the data viewer you added to the data flow.

▶ Task 6: Observe Variables while Debugging

1. Start debugging the package, and note that execution pauses at the first breakpoint.
2. View the **Locals** window and view the configuration values and variables it contains.
3. Add the following variables to the **Watch 1** window:
   - **User::fName**
   - **$Project::AccountsFolderPath**
4. Continue execution and note that it pauses at the next breakpoint, which is the data viewer you defined in the data flow. Note also that the data viewer window contains the contents of the file indicated by the **User::fName** variable in the **Watch 1** window.
5. Continue execution, observing the value of the **User::fName** variable and the contents of the data viewer window during each iteration of the loop.
6. When the **Extract Payments** task fails, note the value of the **User::fName** variable, and copy the contents of the data viewer window to the clipboard. Then stop debugging and close Visual Studio.

▶ Task 7: View Data Copied from a Data Viewer

1. Start Excel and paste the data you copied from the data viewer window into a worksheet.
2. Examine the data and try to determine the errors it contains.

Results: After this exercise, you should have observed the variable values and data flows for each iteration of the loop in the Extract Payment Data.dtsx package. You should also have identified the file that caused the data flow to fail and examined its contents to find the data errors that triggered the failure.

Exercise 2: Logging SSIS Package Execution

Scenario

You have debugged the Extract Payments Data.dtsx package and found some errors in the source data. Now you want to implement logging to assist in diagnosing future errors when the package is deployed in a production environment.

The main tasks for this exercise are as follows:

1. Configure SSIS Logs
2. View Logged Events

▶ Task 1: Configure SSIS Logs

1. Open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab06\Starter\Ex2 folder.
2. Open the **Extract Payment Data.dtsx** package and configure logging with the following settings:
   - Enable logging for the package, and inherit loggings settings for all child executables.
   - Log package execution events to the Windows Event Log.
   - Log all available details for the **OnError** and **OnTaskFailed** events.

▶ Task 2: View Logged Events

1. Run the package in Visual Studio, and note that it fails. When execution is complete, stop debugging.
2. View the **Log Events** window and note the logged events it contains. If no events are logged, re-run the package and look again. Then close Visual Studio.
3. View the **Application** Windows event log in the Event Viewer administrative tool.

Results: After this exercise, you should have a package that logs event details to the Windows Event Log.

Exercise 3: Implementing an Event Handler

Scenario

You have debugged the Extract Payments Data.dtsx package and observed how errors in the source data can cause it to fail. You now want to implement an error handler that copies the invalid source data file to a folder for later examination, and notifies an administrator.

The main tasks for this exercise are as follows:

1. Create an Event Handler
2. Add a File System Task
3. Add a Send Mail Task
4. Test the Event Handler

▶ Task 1: Create an Event Handler

1. Open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab06\Starter\Ex3 folder.
2. In the **Extract Payment Data.dtsx** package, create an event handler for the **OnError** event of the **Extract Payment Data** executable.

▶ Task 2: Add a File System Task

1. Add a file system task to the control flow for the **OnError** event handler, and name it **Copy Failed File**.
2. Configure the **Copy Failed File** task as follows:
   - The task should perform a **Copy File** operation.
   - Use the **Payments File** connection manager as the source connection.
   - Create a new connection manager to create a file named D:\ETL\FailedPayments.csv for the destination.
   - The task should overwrite the destination file if it already exists.

3. Configure the connection manager you created for the destination file to use the following expression for its **ConnectionString** property. Instead of the name configured in the connection manager, the copied file is named using a combination of the unique package execution ID and the name of the source file:

```
"D:\\ETL\\" + @[System::ExecutionInstanceGUID] + @[User::fName]
```

▶ Task 3: Add a Send Mail Task

1. Add a send mail task to the control flow for the **OnError** event handler, and name it **Send Notification**.
2. Connect the **Copy Failed File** task to the **Send Notification** task with a **Completion** precedence constraint.
3. Configure the **Notification** task as follows:
   - Use the **Local SMTP Server** connection manager.
   - Send a high-priority email message from etl@adventureworks.msft to student@adventureworks.msft with the subject "An error occurred".
   - Use the following expression to set the **MessageSource** property:

```
@[User::fName] + " failed to load. " + @[System::ErrorDescription]
```

▶ Task 4: Test the Event Handler

1. Run the package in Visual Studio and verify that the event handler is executed. Then close Visual Studio, saving your changes if prompted.
2. Verify that the source file containing invalid data is copied to the D:\ETL folder with a name similar to {1234ABCD-1234-ABCD-1234-ABCD1234}Payments - EU.csv.
3. View the contents of the C:\inetpub\mailroot\Drop folder, and verify that an email was sent for each error that occurred. You can view the messages by double-clicking them in Outlook.

Results: After this exercise, you should have a package that includes an event handler for the **OnError** event. The event handler should create a copy of files that contain invalid data and send an email message.

Exercise 4: Handling Errors in a Data Flow

Scenario

You have implemented an error handler that notifies an operator when a data flow fails. However, you would like to handle errors in the data flow so that only rows containing invalid data are not loaded, and the rest of the data flow succeeds.

The main tasks for this exercise are as follows:

1. Redirect Data Flow Errors
2. View Invalid Data Flow Rows

▶ Task 1: Redirect Data Flow Errors

1. Open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab06\Starter\Ex4 folder.
2. Open the **Extract Payments Data.dtsx** package and view the data flow for the **Extract Payments** task.
3. Configure the error output of the **Staging DB** destination to redirect rows that contain an error.
4. Add a flat file destination to the data flow and name it Invalid Rows. Then configure the Invalid Rows destination as follows:
   - Create a new connection manager named **Invalid Payment Records** for a delimited file named **D:\ETL\InvalidPaymentsLog.csv**.
   - Do not overwrite data in the text file if it already exists.
   - Map all columns, including **ErrorCode** and **ErrorColumn** to fields in the text file.

▶ Task 2: View Invalid Data Flow Rows

1. Run the package in debug mode and note that it succeeds. When execution is complete, stop debugging and close Visual Studio.
2. Use Excel to view the contents of the InvalidPaymentsLog.csv file in the D:\ETL folder and note the rows that contain invalid values.

Results: After this exercise, you should have a package that includes a data flow where rows containing errors are redirected to a text file.

Module Review and Takeaways

In this module, you have learned how to debug SSIS packages and how to use logging and error handling to ensure the reliability of an ETL process.

Review Question(s)

Question: You have executed a package in Visual Studio, and a task failed unexpectedly. Where can you review information about the package execution to help determine the cause of the problem?

Question: You have configured logging with the SSIS log provider for SQL Server. Where can you view the logged event information?

Question: You suspect a data flow is failing because some values in a source text file are too long for the columns in the destination. How can you handle this problem?

Module 7

Implementing a Data Extraction Solution

Contents:

Module Overview

A data warehousing solution generally needs to refresh the data warehouse at regular intervals to reflect new and modified data in the source systems on which it is based. It is important to implement a refresh process that has a minimal impact on network and processing resources, and which enables you to retain historical data in the data warehouse while reflecting changes and additions to business entities in transactional systems.

This module describes the techniques you can use to implement an incremental data warehouse refresh process.

Objectives

After completing this module, you will be able to:

- Plan data extraction.
- Extract modified data.

Lesson 1
Planning Data Extraction

Most data warehousing solutions use an incremental ETL process to refresh the data warehouse with new and modified data from source systems. Implementing an effective incremental ETL process presents a number of challenges, for which common solution designs have been identified. By understanding some of the key features of an incremental ETL process, you can design an effective data warehouse refresh solution that meets your analytical and reporting needs while maximizing performance and resource efficiency.

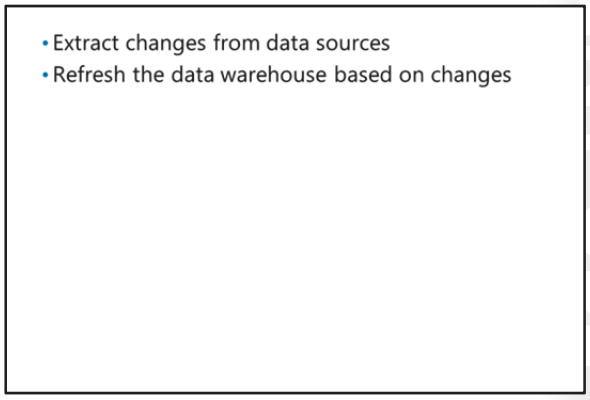
Lesson Objectives

After completing this lesson, you will be able to:

- Describe a typical data warehouse refresh scenario.
- Describe considerations for implementing an incremental ETL process.
- Describe common ETL architectures.
- Plan data extraction windows.
- Plan transformations.

Overview of Data Warehouse Load Cycles

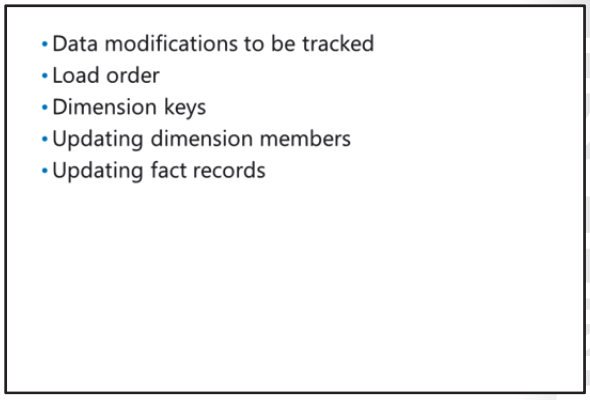
A typical data warehousing solution includes a regular refresh of the data in the data warehouse to reflect new and modified data in the source systems on which it is based. For each load cycle, data is extracted from the source systems, usually to a staging area, and then loaded into the data warehouse. The frequency of the refresh process depends on how up to date the analytical and reporting data in the data warehouse needs to be. In some cases, you might choose to implement a different refresh cycle for each group of related data sources.



In rare cases, it can be appropriate to replace the data warehouse data with fresh data from the data sources during each load cycle. However, a more common approach is to use an incremental ETL process to extract only rows that have been inserted or modified in the source systems. Rows are then inserted or updated in the data warehouse to reflect the extracted data. This reduces the volume of data being transferred, minimizing the effect of the ETL process on network bandwidth and processing resources.

Considerations for Incremental ETL

When planning an incremental ETL process, there are a number of factors that you should consider:



Data Modifications to Be Tracked

One of the primary considerations for planning an incremental ETL process is to identify the kinds of data modifications you need to track in source systems. Specifically, you should consider the following kinds of modifications:

- **Inserts** – for example, new sales transactions or new customer registrations.
- **Updates** – for example, a change of a customer's telephone number or address.
- **Deletes** – for example, the removal of a discontinued item from a product catalog.

Most data warehousing solutions include inserted and updated records in refresh cycles. However, you must give special consideration to deleted records because propagating deletions to the data warehouse results in the loss of historical reporting data.

Load Order

A data warehouse can include dependencies between tables. For example, rows in a fact table generally include foreign key references to rows in dimension tables, and some dimension tables include foreign key references to subdimension tables. For this reason, you should generally design your incremental ETL process to load subdimension tables first, then dimension tables, and finally fact tables. If this is not possible, you can load inferred members as minimal placeholder records for dimension members that are referenced by other tables and which will be loaded later.

Note: Inferred members are normally used to create a placeholder record for a missing dimension member referenced by a fact record. For example, the data to be loaded into a fact table for sales orders might include a reference to a product for which no dimension record has yet been loaded. In this case, you can create an inferred member for the product that contains the required key values but null columns for all other attributes. You can then update the inferred member record on a subsequent load of product data.

Dimension Keys

The keys used to identify rows in dimension tables are usually independent from the business keys used in source systems, and are referred to as surrogate keys. When loading data into a data warehouse, you need to consider how you will identify the appropriate dimension key value to use in the following scenarios:

- Determining whether or not a staged record represents a new dimension member or an update to an existing one, and if it is an update, applying the update to the appropriate dimension record.
- Determining the appropriate foreign key values to use in a fact table that references a dimension table, or in a dimension table that references a subdimension table.

In many data warehouse designs, the source business key for each dimension member is retained as an alternative key in the data warehouse, and can therefore be used to look up the corresponding dimension key. In other cases, dimension members must be found by matching a unique combination of multiple columns.

Updating Dimension Members

When refreshing dimension tables, you must consider whether changes to individual dimension attributes will have a material effect on historical reporting and analysis. Dimension attributes can be categorized as one of three kinds:

- **Fixed** – the attribute value cannot be changed. For example, you might enforce a rule that prevents changes to a product name after it has been loaded into the dimension table.
- **Changing** – the attributes value can change without affecting historical reporting and analytics. For example, a customer's telephone number might change, but it is unlikely that any historical business reporting or analytics will aggregate measures by telephone number. The change can therefore be made without the need to retain the previous telephone number.
- **Historical** – the attribute value can change, but the previous value must be retained for historical reporting and analysis. For example, a customer might move from Edinburgh to New York, but reports and analysis must associate all sales that occurred to that customer before the move with Edinburgh, and all sales after the move with New York.

Updating Fact Records

When refreshing the data warehouse, you must consider whether you will allow updates to fact records. Often, your data warehouse design will only contain complete fact records, so no incomplete records will be loaded. However in some cases, you might want to include an incomplete fact record in the data warehouse that will be updated during a later refresh cycle.

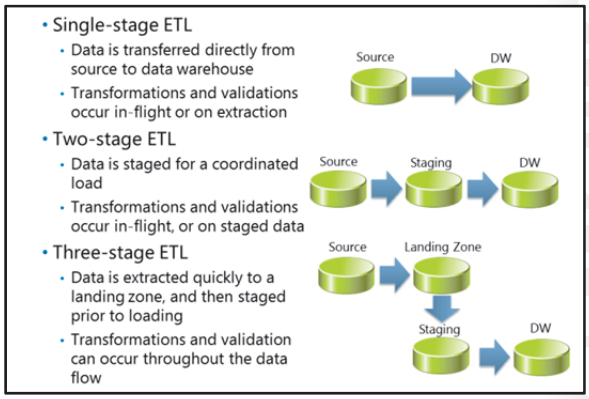
For example, you might choose to include a fact record for a sales order where the sale has been completed, but the item has not yet been delivered. If the record includes a column for the delivery date, you might initially store a null value in this column, and then update the record during a later refresh after the order has been delivered.

While some data warehousing professionals allow updates to the existing record in the fact table, other practitioners prefer to support changes by deleting the existing fact record and inserting a new one. In most cases, the delete operation is actually a logical deletion that is achieved by setting a bit value on a column that indicates whether the record is active or not, rather than actually deleting the record from the table.

Common ETL Data Flow Architectures

Fundamentally, ETL is concerned with data flow from source systems to the data warehouse. The data flow process can be performed directly from source to target, or in stages. Factors that affect the choice of data flow architecture include:

- The number of data sources.
- The volume of data to be transferred.
- The complexity of validation and transformation operations to be applied to the data.



- How frequently data is generated in source systems, and how long it is retained.
- Suitable times to extract source data while minimizing the impact on performance for users.

Single-stage ETL

In a very small business intelligence (BI) solution with few data sources and simple requirements, it may be possible to copy data from data sources to the data warehouse in a single data flow. Basic data validations (such as checking for NULL fields or specific value ranges) and transformations (such as concatenating multiple fields into a single field, or looking up a value from a key) can either be performed during extraction (for example, in the Transact-SQL statement used to retrieve data from a source database) or in-flight (for example, by using transformation components in an SSIS data flow task).

Two-stage ETL

In many cases, a single-stage ETL solution is not suitable because of the complexity or volume of data being transferred. Additionally, if multiple data sources are used, it is common to synchronize loads into the data warehouse to ensure consistency and integrity across fact and dimension data from different sources, and to minimize the performance impact of the load operations on data warehouse activity. If the data is not ready to extract from all systems at the same time, or if some sources are only available at specific times when others are not available, a common approach is to stage the data in an interim location before loading into the data warehouse.

Typically, the structure of the data in the staging area is similar to the source tables, which minimizes the extraction query complexity and duration in the source systems. When all source data is staged, it can then be conformed to the data warehouse schema during the load operation, either as it is extracted from the staging tables or during the data flow to the data warehouse.

Staging the data also provides a recovery point for data load failures and enables you to retain extracted data for audit and verification purposes.

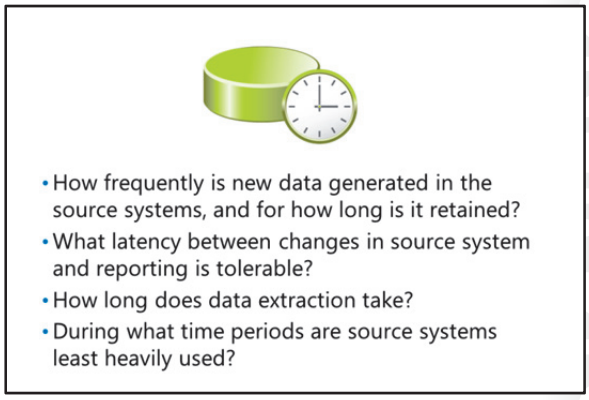
Three-stage ETL

A two-stage data flow architecture can reduce the extraction overhead and source systems, enabling a coordinated load of data from multiple sources. However, performing validation and transformations during the data flow into the data warehouse can affect load performance, and cause the load to negatively affect data warehouse activity. When large volumes of data must be loaded into the data warehouse, it is important to minimize load times by preparing the data as much as possible before performing the operation.

For BI solutions that involve loading large volumes of data, a three-stage ETL process is recommended. In this data flow architecture, the data is initially extracted to tables that closely match the source system schemas—often referred to as a "landing zone." From here, the data is validated and transformed as it is loaded into staging tables that more closely resemble the target data warehouse tables. Finally, the conformed and validated data can be loaded into the data warehouse tables.

Planning Extraction Windows

To help you determine when to perform the data extraction process, consider the following questions:



How frequently is new data generated in the source systems, and for how long is it retained?

Some business applications generate only a few transactions per day, and store the details permanently. Others generate transient feeds of data that must be captured in real time. The volume of changes and storage interval of the source data will determine the frequency of extraction required to support the business requirements.

What latency between changes in source systems and reporting is tolerable?

Another factor in planning data extraction timings is the requirement for the data warehouse to be kept up to date with changes in the source systems. If real-time (or near real-time) reporting must be supported, data must be extracted and loaded into the data warehouse as soon as possible after each change. Alternatively, if all reporting and analysis is historical, you may be able to leave a significant period of time (for example, a month) between data warehouse loads. However, note that you do not need to match data extractions one-to-one with data loads. If less overhead is created in the data source by a nightly extraction of the day's changes than a monthly extraction, you might choose to stage the data nightly, and then load it into the data warehouse in one operation at the end of the month.

How long does data extraction take?

Perform a test extraction and note the time taken to extract a specific number of rows. Then, based on how many new and modified rows are created in a particular period, estimate the time an extraction would take if performed hourly, daily, weekly, or at any other interval that makes sense, based on your answers to the first two questions.

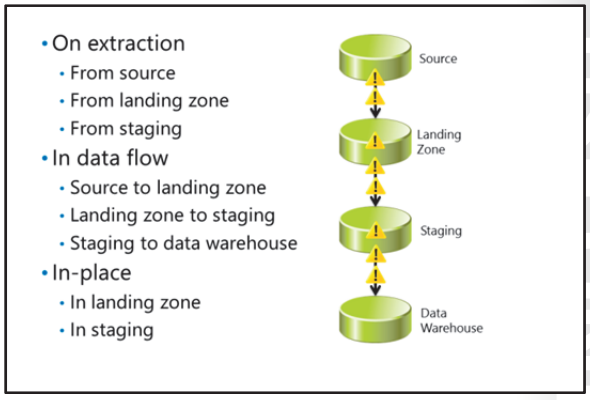
During what time periods are source systems least heavily used?

Some data sources may be available only during specific periods, and others might be too heavily used during business hours to support the additional overhead of an extraction process. You must work closely with the administrators and users of the data sources to identify the ideal data extraction time periods for each source.

After you consider these questions for all source systems, you can start to plan extraction windows for the data. Note that it is common to have multiple sources with different extraction windows, so that the elapsed time to stage all the data might be several hours or even days.

Planning Transformations

When planning an ETL solution, you must consider the transformations that need to be applied to the data to validate it and make it conform to the target table schemas.



Where to Perform Transformations

One consideration for transformations is where they should be applied during the ETL process.

Performing transformations on extraction

If the data sources support it, you can perform transformations in the queries used to extract data. For example, in an SQL Server data source, you can use joins, ISNULL expressions, CAST and CONVERT expressions, and concatenation expressions in the SELECT query used to extract the data. In an enterprise BI solution, this technique can be used during the following extractions:

- Extraction from the source system.
- Extraction from the landing zone.
- Extraction from the staging area.

Performing transformations in the data flow

You can use SSIS data flow transformations to transform data during the data flow. For example, you can use lookups, derived column transformations, and custom scripts to validate and modify rows in a data flow. You can also use merge and split transformations to combine or create multiple data flow paths. In an enterprise BI solution, this technique can be used during the following data flows:

- Source to landing zone.
- Landing zone to staging.
- Staging to data warehouse.

Performing transformations in-place

In some cases, it might make sense to transfer data from sources into one or more database tables, and then perform UPDATE operations to modify the data in-place before the next phase of the ETL data flow. For example, you might extract data from one source and stage it, and then update coded values based on data from another source that is extracted during a later extraction window. In an enterprise BI solution, this technique can be used in the following locations:

- Landing zone tables.
- Staging tables.

Guidelines for choosing where to perform transformations

Although there is no single correct place in the data flow to perform transformations, consider the following guidelines for designing your solutions:

- Minimize the extraction workload on source systems. This enables you to extract the data in the shortest time possible with minimal adverse effect on business processes and applications using the data source.

- Perform validations and transformations in the data flow as soon as possible. This enables you to remove or redirect invalid rows and unnecessary columns early in the extraction process, reducing the amount of data being transferred across the network.
- Minimize the time it takes to load the data warehouse tables. This enables you to get the new data into production as soon as possible and perform the load with minimal adverse effect on data warehouse users.

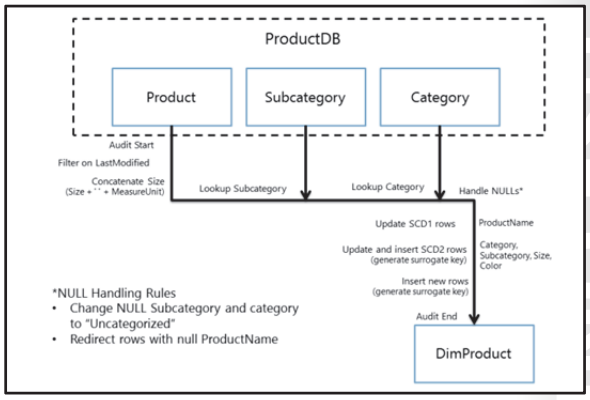
How to Perform Transformations

You can use Transact-SQL statements to transform or validate columns during extraction or in-place. Alternatively, you can use SSIS data flow transformations to modify the data during the data flow. The following table lists some typical validation and transformation scenarios, together with information about how to use Transact-SQL or data flow transformations to implement a solution.

Scenario	Transact-SQL	Data flow transformations
Data type conversion	Use the CAST or CONVERT function.	Use the **Data Conversion** transformation.
Concatenation	Concatenate fields in the SELECT clause of the query.	Use the **Derived Column** transformation.
Replacing NULL values	Use the ISNULL function.	Use the **Derived Column** transformation with an expression containing the **ReplaceNull** function.
Looking up related values where referential integrity is enforced	Use an INNER JOIN.	Use the **Lookup** transformation.
Looking up related values where referential integrity is not enforced	Use an OUTER JOIN, and optionally use ISNULL to replace null values where no matching rows exist.	Use the **Lookup** transformation with the **Ignore failure** option, and then add a transformation later in the data flow to handle null values (either by replacing them with a **Derived Column** or redirecting them with a **Condition Split**). Alternatively, use the **Redirect rows to no match output** option and handle the nulls before using a **Merge** transformation to return the fixed rows to the main data flow.

 **Note:** Some people who are unfamiliar with SSIS make the erroneous assumption that the data flow processes rows sequentially, and that data flow transformations are inherently slower than set-based transformations performed with Transact-SQL. However, the SSIS pipeline performs set-based operations on buffered batches of rows, and is designed to provide high-performance transformation in data flows.

Documenting Data Flows



An important part of designing an ETL solution is to document the data flows you need to implement. The diagrams and notes that document your data flows are commonly referred to as "source-to-target" documentation, that typically starts with a simple high-level diagram for each table in the data warehouse. The diagram shows the source tables from which the data warehouse table fields originate and the validation and transformations that must be applied during the data flow.

As a general rule, use a consistent diagramming approach for each table, and include as much detail about validation rules, transformations, and potential issues as you can. It is common for these high-level diagrams to start simple and be refined as the ETL design evolves.

As your ETL design is refined, you will start to develop a clear idea of what fields will be extracted, generated, and validated at each stage of the data flow. To help document the lineage of the data as it flows from the source to the data warehouse tables, you can create detailed source-to-target mappings that show detailed information for the fields at each stage.

A common way to create a source-to-target mapping is to use a spreadsheet divided into a set of columns for each stage in the data flow. Start with the fields in the target table, and then work backward to determine the required staging, landing zone, and source fields, along with any validation rules and transformations that must be applied. The goal is to create a single document in which the origins of a field in the target table can be traced back across a row to its source.

Like high-level data flow diagrams, many BI professionals have adopted different variations of source-to-target mapping. The organization in which you are working may not have a standard format for this kind of documentation. It's important, therefore, to use a consistent format that is helpful during ETL design and easy to understand for anyone who needs to troubleshoot or maintain the ETL system in the future.

Lesson 2
Extracting Modified Data

An incremental ETL process starts by extracting data from source systems. To avoid including unnecessary rows of data in the extraction, the solution must be able to identify records that have been inserted or modified since the last refresh cycle, and limit the extraction to those records.

This lesson describes a number of techniques for identifying and extracting modified records.

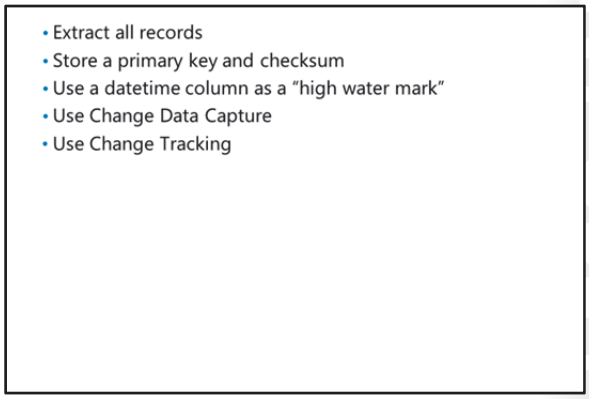
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the common options for extracting modified records.
- Implement an ETL solution that extracts modified rows based on a DateTime column.
- Configure the Change Data Capture feature in the SQL Server Enterprise Edition database engine.
- Implement an ETL solution that extracts modified rows by using the Change Data Capture feature.
- Use the CDC Control Task and data flow components to extract Change Data Capture records.
- Configure the Change Tracking feature in the Microsoft® SQL Server® database engine.
- Implement an ETL solution that extracts modified rows by using Change Tracking.

Options for Extracting Modified Data

There are a number of commonly-used techniques employed to extract data as part of a data warehouse refresh cycle.



Extract All Records

The simplest solution is to extract all source records and load them to a staging area, before using them to refresh the data warehouse. This technique works with all data sources and ensures that the refresh cycle includes all inserted, updated, and deleted source records. However, this technique can require the transfer and storage of large volumes of data, making it inefficient and impractical for many enterprise data warehousing solutions.

Store a Primary Key and Checksum

Another solution is to store the primary key of all previously-extracted rows in a staging table along with a checksum value that is calculated from source columns in which you want to detect changes. For each refresh cycle, your ETL process can extract source records for which the primary key is not recorded in the table of previous extractions, as well as rows where the checksum value calculated from the columns in the source record does not match the checksum recorded during the previous extraction. Additionally, any primary keys recorded in the staging table that no longer exist in the source represent deleted records.

This technique limits the extracted records to those that have been inserted or modified since the previous refresh cycle. However, for large numbers of rows, the overhead of calculating a checksum to compare with each row can significantly increase processing requirements.

Use a Datetime Column as a "High Water Mark"

Tables in data sources often include a column to record the date and time of the initial creation and last modification to each record. If your data source includes such a column, you can log the date and time of each refresh cycle and compare it with the last modified value in the source data records to identify records that have been inserted or modified. This technique is commonly referred to as using the date and time of each extraction as a "high water mark" because of its similarity to the way a tide or flood can leave an indication of the highest water level.

Use Change Data Capture

Change Data Capture (CDC) is a feature of SQL Server Enterprise Edition that uses transaction log sequence numbers (LSNs) to identify insert, update, and delete operations that have occurred within a specified time period. To use CDC, your ETL process must store the date and time or LSN of the last extraction as described for the high water mark technique. However, it is not necessary for tables in the source database to include a column that indicates the date and time of the last modification.

CDC is an appropriate technique when:

- The data source is a database in the enterprise edition of an SQL Server 2008 or later.
- You need to extract a complete history that includes each version of a record that has been modified multiple times.

Use Change Tracking

Change Tracking is another SQL Server technology you can use to record the primary key of records that have been modified and extract records based on a version number that is incremented each time a row is inserted, updated, or deleted. To use Change Tracking, you must log the version that is extracted, and then compare the logged version number to the current version in order to identify modified records during the next extraction.

Change Tracking is an appropriate technique when:

- The data source is an SQL Server 2008 or later database.
- You need to extract the latest version of a row that has been modified since the previous extraction, but you do not need a full history of all interim versions of the record.

Considerations for Handling Deleted Records

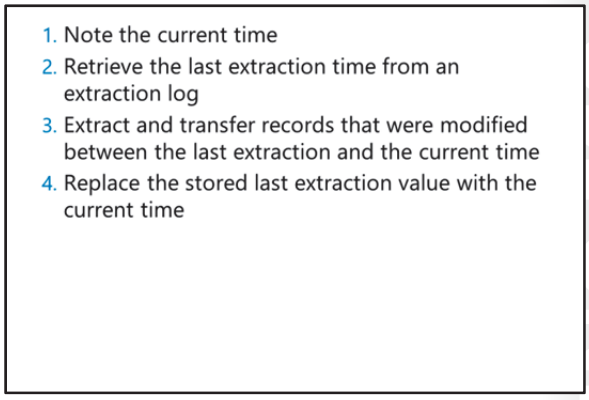
If you need to propagate record deletions in source systems to the data warehouse, you should consider the following guidelines

- You need to be able to identify which records have been deleted since the previous extraction. One way to accomplish this is to store the keys of all previously-extracted records in the staging area and compare them to the values in the source database as part of the extraction process. Alternatively, Change Data Capture and Change Tracking both provide information about deletions, enabling you to identify deleted records without maintaining the keys of previously extracted records.
- If the source database supports logical deletes by updating a Boolean column, to indicate that the record is removed, then deletions are conceptually just a special form of update. You can implement custom logic in the extraction process to treat data updates and logical deletions separately if necessary.

Extracting Rows Based on a Datetime Column

If your data source includes a column to indicate the date and time each record was inserted or modified, you can use the high water mark technique to extract modified records. The high-level steps your ETL process must perform to use the high water mark technique are:

1. Note the current time.
2. Retrieve the date and time of the previous extraction from a log table.
3. Extract records where the modified date column is later than the last extraction time, but before or equal to the current time you noted in step 1. This disregards any insert or update operations that have occurred since the start of the extraction process.
4. In the log, update the last extraction date and time with the time you noted in step 1.



Demonstration: Using a Datetime Column

In this demonstration, you will see how to use a datetime column to extract modified data.

Demonstration Steps

Use a Datetime Column to Extract Modified Data

1. Ensure 20463C-MIA-DC and 20463C-MIA-SQL are started, and log onto 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Demofiles\Mod07 folder, run **Setup.cmd** as Administrator.
3. Start SQL Server Management Studio and connect to the **localhost** instance of the database engine using Windows authentication.
4. In Object Explorer, expand **Databases**, **DemoDW**, and **Tables**. Note that the database includes tables in three schemas (**src**, **stg**, and **dw**) to represent the data sources staging database, and data warehouse in an ETL solution.
5. Right-click each of the following tables and click **Select Top 1000 Rows**:
   - **stg.Products** – this table is used for staging product records during the ETL process, and is currently empty.
   - **stg.ExtractLog** – this table logs the last extraction date for each source system.
   - **src.Products** – this table contains the source data for products, including a **LastModified** column that records when each row was last modified.
6. Start Visual Studio and open the **IncrementalETL.sln** solution in the D:\Demofiles\Mod07 folder. Then in Solution Explorer, double-click the **Extract Products.dtsx** SSIS package.
7. On the **SSIS** menu, click **Variables**, and note that the package contains two user variables named **CurrentTime** and **LastExtractTime**.
8. On the control flow surface, double-click **Get Current Time**, and note that the expression in this task sets the **CurrentTime** user variable to the current date and time. Then click **Cancel**.
9. Double-click **Get Last Extract Time**, and note the following configuration settings. Then click **Cancel**:

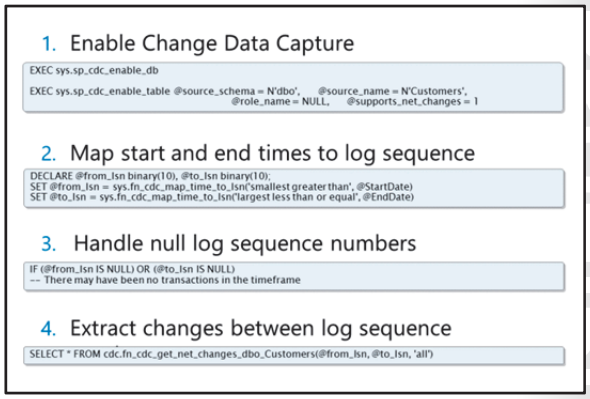
- On the **General** tab, the **ResultSet** property is set to return a single row, and **SQLStatement** property contains a query to retrieve the maximum **LastExtractTime** value for the products source in the **stg.ExtractLog** table.
- On the **Result Set** tab, the **LastExtractTime** value in the query results row is mapped to the **LastExtractTime** user variable.

10. Double-click **Stage Products** to view the data flow surface, and then double-click **Products Source** and note that the SQL command used to extract products data includes a WHERE clause that filters the query results. Then click **Parameters**, and note that the parameters in the Transact-SQL query are mapped to the **LastExtractTime** and **CurrentTime** variables.
11. Click **Cancel** in all dialog boxes and then click the **Control Flow** tab.
12. On the control flow surface, double-click **Update Last Extract Time**, and note the following configuration settings. Then click **Cancel**:
    - On the **General** tab, the **SQLStatement** property contains a Transact-SQL UPDATE statement that updates the **LastExtractTime** in the **stg.ExtractLog** table.
    - On the **Parameter Mapping** tab, the **CurrentTime** user variable is mapped to the parameter in the Transact-SQL statement.
13. In SQL Server Management Studio, open **Modify Products.sql** in the D:\Demofiles\Mod07 folder. Then execute the script to modify some rows in the **src.Products** table.
14. In Visual Studio, on the **Debug** menu, click **Start Debugging**. Then, when package execution is complete, on the **Debug** menu, click **Stop Debugging**.
15. In SQL Server Management Studio, right-click each of the following tables and click Select Top 1000 Rows:
    - **stg.ExtractLog** – Note that the **LastExtractTime** for the **Products** data source has been updated.
    - **stg.Products** – Note the rows that have been extracted from the **src.Products** table.
16. Minimize SQL Server Management Studio and Visual Studio.

Change Data Capture

The CDC feature in SQL Server Enterprise Edition provides a number of functions and stored procedures that you can use to identify modified rows. To use CDC, perform the following high-level steps:

1. Enable CDC in the data source. You must enable CDC for the database, and for each table in the database where you want to monitor changes.



The following Transact-SQL code sample shows how to use the **sp_cdc_enable_db** and **sp_cdc_enable_table** system stored procedures to enable CDC in a database and monitor data modifications in the **dbo.Customers** table:

```
EXEC sys.sp_cdc_enable_db
EXEC sys.sp_cdc_enable_table @source_schema = N'dbo', @source_name = N'Customers',
@role_name = NULL, @supports_net_changes = 1
```

2. In the ETL process used to extract the data, map start and end times (based on the logged date and time of the previous extraction and the current date and time) to log sequence numbers.

   The following Transact-SQL code sample shows how to use the **fn_cdc_map_time_to_lsn** system function to map Transact-SQL variables named **@StartDate** and **@EndDate** to log sequence numbers:

```
DECLARE @from_lsn binary(10), @to_lsn binary(10);
SET @from_lsn = sys.fn_cdc_map_time_to_lsn('smallest greater than', @StartDate)
SET @to_lsn = sys.fn_cdc_map_time_to_lsn('largest less than or equal', @EndDate)
```

3. Include logic to handle errors if either of the log sequence numbers is null. This can happen if no changes have occurred in the database during the specified time period.

   The following Transact-SQL code sample shows how to check for null log sequence numbers:

```
IF (@from_lsn IS NULL) OR (@to_lsn IS NULL)
There may have been no transactions in the timeframe
```

4. Extract records that have been modified between the log sequence numbers. When you enable CDC for a table, SQL Server generates table-specific system functions that you can use to extract data modifications to that table. The following Transact-SQL code sample shows how to use the fn_cdc_get_net_changes_dbo_Customers system function to retrieve rows that have been modified in the dbo.Customers table:

```
SELECT * FROM cdc.fn_cdc_get_net_changes_dbo_Customers(@from_lsn, @to_lsn, 'all')
```

Note: For detailed information about the syntax of the CDC system functions and stored procedures used in these code samples, go to SQL Server Books Online.

Demonstration: Using Change Data Capture

In this demonstration, you will see how to:

- Enable Change Data Capture.
- Use Change Data Capture to Extract Modified Data.

Demonstration Steps

Enable Change Data Capture

1. Ensure you have completed the previous demonstration in this module.
2. Maximize SQL Server Management Studio, and in Object Explorer, in the **DemoDW** database, right-click the **src.Customers** table and click **Select Top 1000 Rows**. This table contains source data for customers.

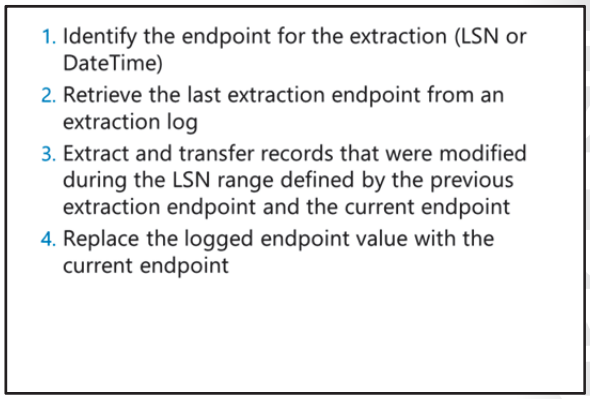
3. Open **Using CDC.sql** in the D:\Demofiles\Mod07 folder, and in the code window, select the Transact-SQL code under the comment **Enable CDC**, and then click **Execute**. This enables CDC in the **DemoDW** database, and starts logging modifications to data in the **src.Customers** table.
4. Select the Transact-SQL code under the comment **Select all changed customer records since the last extraction**, and then click **Execute**. This code uses CDC functions to map dates to log sequence numbers, and retrieve records in the **src.Customers** table that have been modified between the last logged extraction in the **stg.ExtractLog** table, and the current time. There are no changed records because no modifications have been made since CDC was enabled.

Use Change Data Capture to Extract Modified Data

1. Select the Transact-SQL code under the comment **Insert a new customer**, and then click **Execute**. This code inserts a new customer record.
2. Select the Transact-SQL code under the comment **Make a change to a customer**, and then click **Execute**. This code updates a customer record.
3. Select the Transact-SQL code under the comment **Now see the net changes**, and then click **Execute**. This code uses CDC functions to map dates to log sequence numbers, and retrieve records in the **src.Customers** table that have been modified between the last logged extraction in the **stg.ExtractLog** table, and the current time. Two records are returned.
4. Wait ten seconds. Then select the Transact-SQL code under the comment **Check for changes in an interval with no database activity**, and then click **Execute**. Because there has been no activity in the database during the specified time interval, one of the log sequence numbers is null. This demonstrates the importance of checking for a null log sequence number value when using CDC.
5. Minimize SQL Server Management Studio.

Extracting Data with Change Data Capture

To extract data from a CDC-enabled table in an SSIS-based ETL solution, you can create a custom control flow that uses the same principles as the high water mark technique described earlier. The general approach is to establish the range of records to be extracted based on a minimum and maximum log sequence number (LSN), extract those records, and log the endpoint of the extracted range to be used as the starting point for the next extraction.



You can choose to log the high water mark as an LSN or a datetime value that can be mapped to an LSN by using the **fn_cdc_map_time_to_lsn** system function.

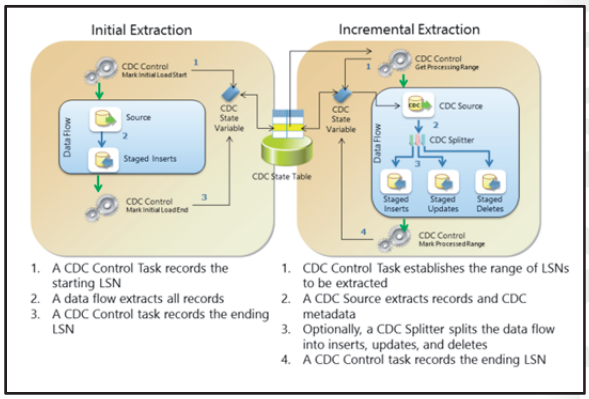
The following procedure describes one way to create an SSIS control flow for extracting CDC data:

1. Use an Expression task to assign the current time to a datetime variable.
2. Use an SQL Command task to retrieve the logged datetime value that was recorded after the previous extraction.

3. Use a Data Flow task in which a source employs the **fn_cdc_map_time_to_lsn** system function to map the current time and previously-extracted time to the corresponding LSNs, and then uses the **cdc.fn_cdc_get_net_changes_capture_instance** function to extract the data that was modified between those LSNs. You can use the **_$operation** column in the resulting dataset to split the records into different data flow paths for inserts, updates, and deletes.
4. Use an SQL Command task to update the logged datetime value to be used as the starting point for the next extraction.

The CDC Control Task and Data Flow Components

To make it easier to implement packages that extract data from CDC-enabled sources, SSIS includes CDC components that abstract the underlying CDC functionality. The CDC components included in SSIS are:



- **CDC Control Task** – A control flow task that you can use to manage CDC state, providing a straightforward way to track CDC data extraction status.
- **CDC Source** – A data flow source that uses the CDC state logged by the CDC Control task to extract a range of modified records from a CDC-enabled data source.
- **CDC Splitter** – A data flow transformation that splits output rows from a CDC Source into separate data flow paths for inserts, updates, and deletes.

All the CDC components in SSIS require the use of ADO.NET connection managers to the CDC-enabled data source and the database where CDC state is to be stored.

Performing an Initial Extraction with the CDC Control Task

When using the CDC Control Task to manage extractions from a CDC-enabled data source, it is recommended practice to create a package that will be executed once to perform the initial extraction. This package should contain the following control flow:

1. A CDC Control Task configured to perform the **Mark initial load start** operation. This writes an encoded value, including the starting LSN to a package variable, and optionally persists it to a state tracking table in a database.
2. A data flow that extracts all rows from the source and loads them into a destination—typically a staging table. This data flow does not require CDC-specific components.
3. A second CDC Control Task configured to perform the **Mark initial load end** operation. This writes an encoded value, including the ending LSN to a package variable, and optionally persists it to a state tracking table in a database.

Performing Incremental Extractions with the CDC Control Task

After the initial extraction has been performed, subsequent extractions should use an SSIS package with the following control flow:

1. A CDC Control Task configured to perform the **Get processing range** operation. This establishes the range of records to be extracted and writes an encoded representation to a package variable, which can also be persisted to a state tracking table in a database.
2. A data flow that uses a CDC Source, using the encoded value in the CDC state package variable to extract modified rows from the data source.
3. Optionally, the data flow can include a CDC Splitter task, which uses the **_$operation** column in the extracted rowset to redirect inserts, updates, and deletes to separate data flow paths. These can then be connected to appropriate destinations for staging tables.
4. A second CDC Control Task configured to perform the **Mark processed range** operation. This writes an encoded value, including the ending LSN to a package variable, and optionally persists it to a state tracking table in a database. This value is then used to establish the starting point for the next extraction.

Demonstration: Using CDC Components

In this demonstration, you will see how to use the CDC Control Task to:

- Perform an Initial Extraction.
- Extract Changes.

Demonstration Steps

Perform an Initial Extraction

1. Ensure you have completed the previous demonstrations in this module.
2. Maximize SQL Server Management Studio and open the **CDC Components.sql** script file in the D:\Demofiles\Mod07 folder.
3. Note that the script enables CDC for the **src.Shippers** table. Then click **Execute**.
4. In Object Explorer, right-click each of the following tables in the **DemoDW** database, and click **Select Top 1000 Rows** to view their contents:
   - **src.Shippers** – This table should contain four records.
   - **stg.ShipperDeletes** – This table should be empty.
   - **stg.ShipperInserts** – This table should be empty.
   - **stg.ShipperUpdates** – This table should be empty.
5. Maximize Visual Studio, in which the IncrementalETL.sln solution should be open, and in Solution Explorer, double-click the **Extract Initial Shippers.dtsx** SSIS package. Note that the CDC Control tasks in the control flow contain errors, which you will resolve.
6. Double-click the **Mark Initial Load Start** CDC Control task, and in its editor, set the following properties. Then click OK:
   - **SQL Server CDC database ADO.NET connection manager**: localhost DemoDW ADO NET.
   - **CDC control operation**: Mark initial load start.
   - **Variable containing the CDC state**: Click **New** and create a new variable named **CDC_State**.

- **Automatically store state in a database table**: Selected.
- **Connection manager for the database where the state is stored**: localhost DemoDW ADO NET.
- **Table to use for storing state**: Click **New**, and then click **Run** to create the **cdc_states** table.
- **State name**: CDC_State.

7. Double-click the **Extract All Shippers** data flow task, and on the Data Flow surface, note that an ADO.NET source named **Shippers** is used to extract all rows from the **src.Shippers** table, and an ADO.NET destination named **Shipper Inserts** is used to load the extracted rows into the **stg.ShipperInserts** table.
8. On the Control Flow tab, double-click the **Mark Initial Load End CDC Control** task and set the following properties. Then click OK:
   - **SQL Server CDC database ADO.NET connection manager**: localhost DemoDW ADO NET.
   - **CDC control operation**: Mark initial load end (make sure you do *not* select Mark initial load start).
   - **Variable containing the CDC state**: User::CDC_State (the variable you created earlier).
   - **Automatically store state in a database table**: Selected.
   - **Connection manager for the database where the state is stored**: localhost DemoDW ADO NET.
   - **Table to use for storing state**: [dbo].[cdc_states] (the table you created earlier).
   - **State name**: CDC_State.
9. On the **Debug** menu, click **Start Debugging** and wait for the package execution to complete. Then on the **Debug** menu, click **Stop Debugging**.
10. In SQL Server Management Studio, right-click the **Tables** folder for the **DemoDW** database and click **Refresh**. Note that a table named **dbo.cdc_states** has been created.
11. Right-click **dbo.cdc_states** and click **Select Top 1000 Rows** to view the logged **CDC_State** value (which should begin "ILEND…").
12. Right-click **stg.ShipperInserts** and click **Select Top 1000 Rows** to verify that the initial set of shippers has been extracted.

Extract Changes

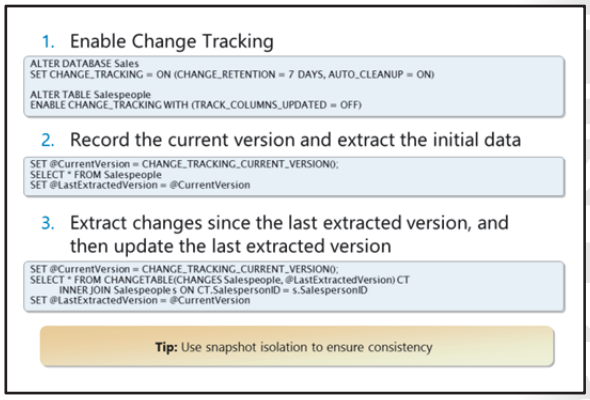
1. In SQL Server Management Studio, open **Update Shippers.sql** in the D:\Demofiles\Mod07 folder.
2. Select the code under the comment **Cleanup previous extraction**, noting that it truncates the **stg.ShipperInserts** table, and click **Execute**.
3. In Visual Studio, in Solution Explorer, double-click **Extract Changed Shippers.dtsx**.
4. On the **Control Flow** tab, double-click the **Get Processing Range** CDC Control task. Note it gets the processing range, storing it in the **CDC_State** variable and the **cdc_states** table. Then click **Cancel**.
5. Double-click the **Extract Modified Shippers** data flow task and on the **Data Flow** tab, view the properties of the **Shipper CDC Records** CDC Source component, noting that it extracts modified records based on the range stored in the **CDC_State** variable.
6. Note that **CDC Splitter** transformation has three outputs, one each for inserts, updates, and deletes. Each of these is connected to an ADO.NET destination that loads the records into the **stg.ShipperInserts**, **stg.ShipperUpdates**, and **stg.ShipperDeletes** tables respectively.

7. On the **Control Flow** tab, double-click the **Mark Processed Range** CDC Control task and note it updates the **CDC_State** variable and the **cdc_states** table when the extraction is complete. Then click **Cancel**.
8. On the **Debug** menu, click **Start Debugging**. When execution is complete, stop debugging and then double-click the **Extract Modified Shippers** data flow task and note that no rows are transferred, because the source data is unchanged since the initial extraction.
9. In SQL Server Management Studio, in the Update Shippers.sql script file, select the code under the comment **Modify source data**, noting that it performs an INSERT, an UPDATE, and a DELETE operation on the **src.Shippers** table, and click **Execute**.
10. In Visual Studio, on the **Debug** menu, click **Start Debugging**. When execution is complete, stop debugging and view the **Extract Modified Shippers** data flow task and note the number of rows transferred (if no rows were transferred, stop debugging and re-run the package). When three rows have been transferred (one to each output of the CDC Splitter transformation), stop debugging and close Visual Studio.
11. In SQL Server Management Studio, right-click each of the following tables and click **Select Top 1000 Rows** to view their contents. Each table should contain a single row:
    - stg.ShipperDeletes
    - stg.ShipperInserts
    - stg.ShipperUpdates
12. Minimize SQL Server Management Studio.

Change Tracking

The Change Tracking feature in SQL Server provides a number of functions and stored procedures that you can use to identify modified rows. To use Change Tracking, perform the following high-level steps:

1. Enable Change Tracking in the data source. You must enable Change Tracking for the database, and for each table in the database for which you want to monitor changes.



The following Transact-SQL code sample shows how to enable Change Tracking in a database named Sales and monitor data modifications in the Salespeople table. Note that you can choose to track which columns were modified, but the change table only contains the primary key of each modified row—not the modified column values:

```
ALTER DATABASE Sales
SET CHANGE_TRACKING = ON (CHANGE_RETENTION = 7 DAYS, AUTO_CLEANUP = ON)
ALTER TABLE Salespeople
ENABLE CHANGE_TRACKING WITH (TRACK_COLUMNS_UPDATED = OFF)
```

2. For the initial data extraction, record the current version (which by default is 0), and extract all rows in the source table. Then log the current version as the last extracted version.

   The following Transact-SQL code sample shows how to use the Change_Tracking_Current_Version system function to retrieve the current version, extract the initial data, and assign the current version to a variable so it can be stored as the last extracted version:

```
SET @CurrentVersion = CHANGE_TRACKING_CURRENT_VERSION();
SELECT * FROM Salespeople
SET @LastExtractedVersion = @CurrentVersion
```

3. For subsequent refresh cycles, extract changes that have occurred between the last extracted version and the current one. The following Transact-SQL code sample shows how to determine the current version, use the Changetable system function in a query that joins the primary key of records in the change table to records in the source table, and update the last extracted version:

```
SET @CurrentVersion = CHANGE_TRACKING_CURRENT_VERSION();
SELECT * FROM CHANGETABLE(CHANGES Salespeople, @LastExtractedVersion) CT
    INNER JOIN Salespeople s ON CT.SalespersonID = s.SalespersonID
SET @LastExtractedVersion = @CurrentVersion
```

   When using Change Tracking, a best practice is to enable snapshot isolation in the source database and use it to ensure that any modifications occurring during the extraction do not affect records that were modified between the version numbers that define the lower and upper bounds of your extraction range.

Note: For detailed information about the syntax of the Change Tracking system functions used in these code samples, go to SQL Server Books Online.

Demonstration: Using Change Tracking

In this demonstration, you will see how to:

- Enable Change Tracking.
- Use Change Tracking.

Demonstration Steps

Enable Change Tracking

1. Ensure you have completed the previous demonstrations in this module.
2. Maximize SQL Server Management Studio, and in Object Explorer, in the **DemoDW** database, right-click the **src.Salespeople** table and click **Select Top 1000 Rows**. This table contains source data for sales employees.

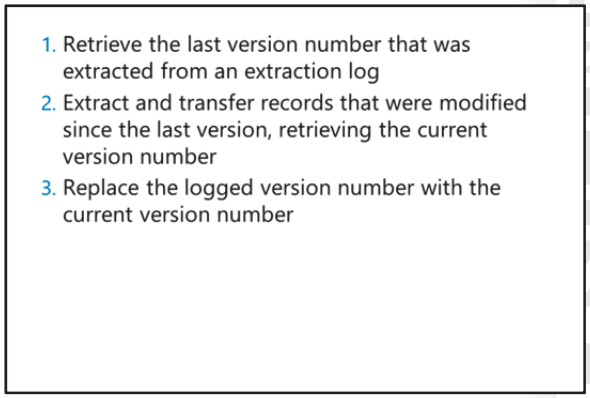
3. Open **Using CT.sql** in the D:\Demofiles\Mod07 folder. Then select the Transact-SQL code under the comment **Enable Change Tracking**, and click **Execute**. This enables Change Tracking in the **DemoDW** database, and starts logging changes to data in the **src.Salespeople** table.
4. Select the Transact-SQL code under the comment **Obtain the initial data and log the current version number,** and then click **Execute**. This code uses the CHANGE_TRACKING_CURRENT_VERSION function to determine the current version, and retrieves all records in the **src.Salespeople** table.

Use Change Tracking

1. Select the Transact-SQL code under the comment **Insert a new salesperson**, and then click **Execute**. This code inserts a new customer record.
2. Select the Transact-SQL code under the comment **Update a salesperson**, and then click **Execute**. This code updates the customer record.
3. Select the Transact-SQL code under the comment **Retrieve the changes between the last extracted and current versions**, and then click Execute. This code retrieves the previously-extracted version from the **stg.ExtractLog** table, determines the current version, uses the CHANGETABLE function to find records in the **src.Salespeople** table that have been modified since the last extracted version, and then updates the last extracted version in the **stg.ExtractLog** table.
4. Close SQL Server Management Studio without saving any changes.

Extracting Data with Change Tracking

You can create an SSIS package that uses the Change Tracking feature in SQL Server in a similar way to the high water mark technique described earlier in this lesson. The key difference is that, rather than storing the date and time of the previous extraction, you must store the Change Tracking version number that was extracted, and update this with the current version during each extract operation.



A typical control flow for extracting data from a Change Tracking-enabled data source includes the following elements:

1. An SQL Command that retrieves the previously extracted version from a log table and assigns it to a variable.
2. A data flow that contains a source to extract records that have been modified since the previously extracted version and return the current version.
3. An SQL Command that updates the logged version number with the current version.

Lab: Extracting Modified Data

Scenario

You have developed SSIS packages that extract data from various data sources and load it into a staging database. However, the current solution extracts all source records each time the ETL process is run. This results in unnecessary processing of records that have already been extracted and consumes a sizeable amount of network bandwidth to transfer a large volume of data. To resolve this problem, you must modify the SSIS packages to extract only data that has been added or modified since the previous extraction.

Objectives

After completing this lab, you will be able to:

- Use a datetime column to extract modified rows.
- Use Change Data Capture to extract modified rows.
- Use the CDC Control Task to extract modified rows.
- Use Change Tracking to extract modified rows.

Estimated Time: 60 minutes

Virtual machine: 20463C-MIA-SQL

User name: ADVENTUREWORKS\Student

Password: Pa$$w0rd

Exercise 1: Using a Datetime Column to Incrementally Extract Data

Scenario

The **InternetSales** and **ResellerSales** databases contain source data for your data warehouse. The sales order records in these databases include a **LastModified** date column that is updated with the current date and time when a row is inserted or updated. You have decided to use this column to implement an incremental extraction solution that compares record modification times to a logged extraction date and time in the staging database. This restricts data extractions to rows that have been modified since the previous refresh cycle.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. View Extraction Data
3. Examine an Incremental Data Extraction
4. Define Variables for Extraction Times
5. Modify a Data Source to Filter Data
6. Add a Task to Update the Extraction Log
7. Test the Package

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. Run **Setup.cmd** in the D:\Labfiles\Lab07\Starter folder as Administrator.

▶ Task 2: View Extraction Data

1. Start SQL Server Management Studio and connect to the **MIA-SQL** database instance by using Windows authentication.
2. In the **Staging** database, view the contents of the **dbo.ExtractLog** table, noting that it contains the date and time of previous extractions from the **InternetSales** and **ResellerSales** databases. This is initially set to January 1st 1900.
3. In the **InternetSales** database, view the contents of the **dbo.SalesOrderHeader** table and note that the **LastModified** column contains the date and time that each record was inserted or modified.

▶ Task 3: Examine an Incremental Data Extraction

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab07\Starter\Ex1 folder. Then open the **Extract Reseller Data.dtsx** SSIS package.
2. View the variables defined in the package, and note that they include two DateTime variables named **CurrentTime** and **ResellerSalesLastExtract**.
3. Examine the tasks in the package and note the following:
   - The **Get Current Time** task uses the GETDATE() function to assign the current date and time to the **CurrentTime** variable.
   - The **Get Last Extract Time** task uses a Transact-SQL command to return a single row containing the **LastExtract** value for the **ResellerSales** data source from the **dbo.ExtractLog** table in the **Staging** database, and assigns the **LastExtract** value to the **ResellerSalesLastExtract** variable.
   - The **Extract Reseller Sales** data flow task includes a data source named **Reseller Sales** that uses a WHERE clause to extract records with a **LastModified** value between two parameterized values. The parameters are mapped to the **ResellerSalesLastExtract** and **CurrentTime** variables.
   - The **Update Last Extract Time** task updates the **LastExtract** column for the **ResellerSales** data source in the **dbo.ExtractLog** table with the **CurrentTime** variable.
4. Run the **Extract Reseller Data.dtsx** package. When it has completed, stop debugging, and in SQL Server Management Studio, verify that the **ExtractLog** table in the **Staging** database has been updated to reflect the most recent extraction from the **ResellerSales** source.

▶ Task 4: Define Variables for Extraction Times

1. In Visual Studio, open the **Extract Internet Sales Data.dtsx** SSIS package, and add DateTime variables named **CurrentTime** and **InternetSalesLastExtract**.
2. Add an **Expression** task named **Get Current Time** to the **Extract Customer Sales Data** sequence in the control flow of the package, and configure it to apply the following expression:

```
@[User::CurrentTime] = GETDATE()
```

3. Add an **Execute SQL** task named **Get Last Extract Time** to the **Extract Customer Sales Data** sequence in the control flow of the package, and set the following configuration properties:
   - On the **General** tab, set the **Connection** property to **localhost.Staging**.
   - On the **General** tab, set the **SQLStatement** to the following Transact-SQL query:

```
SELECT MAX(LastExtract) LastExtract
FROM ExtractLog
WHERE DataSource = 'InternetSales'
```

   - On the **General** tab, set the **ResultSet** property to **Single row**.

- On the **Result Set** tab, add a result that maps the **LastExtract** column in the result set to the **User::InternetSalesLastExtract** variable.

4. Connect the precedence constraints of the new tasks so that the **Get Current Time** task runs first and is followed by the **Get Last Extract Time** task, and then the **Extract Customers** task.

▶ Task 5: Modify a Data Source to Filter Data

1. On the data flow tab for the **Extract Internet Sales** data flow task, make the following changes to the **Internet Sales** source:
   - Add the following WHERE clause to the query in the SQL Command property:

```
WHERE LastModified > ?
AND LastModified <= ?
```

   - Specify the following input parameter mappings:
     - **Parameter0**: User::InternetSalesLastExtract
     - **Parameter1**: User:CurrentTime

▶ Task 6: Add a Task to Update the Extraction Log

1. On the control flow tab, add an **Execute SQL** task named **Update Last Extract Time** to the **Extract Customer Sales Data** sequence in the control flow of the **Extract Internet Sales Data.dtsx** package, and set the following configuration properties:
   - On the **General** tab, set the **Connection** property to **localhost.Staging**.
   - On the **General** tab set the **SQLStatement** property to the following Transact-SQL query:

```
UPDATE ExtractLog
SET LastExtract = ?
WHERE DataSource = 'InternetSales'
```

   - On the **Parameter Mapping** tab, add the following parameter mapping:
     - **Variable Name**: User::CurrentTime
     - **Direction**: Input
     - **Data Type**: DATE
     - **Parameter Name**: 0
     - **Parameter Size**: -1
2. Connect the precedence constraint of the **Extract Internet Sales** task to the **Update Last Extract Time** task.

▶ Task 7: Test the Package

1. View the **Extract Internet Sales** data flow and then start debugging the package and note the number of rows transferred. When package execution is complete, stop debugging.
2. In SQL Server Management Studio, view the contents of the **dbo.ExtractLog** table in the **Staging** database and verify that the **LastExtract** column for the **InternetSales** data source has been updated.
3. View the contents of the **dbo.InternetSales** table and note the rows that have been extracted.
4. In Visual Studio, debug the package again and verify that this time no rows are transferred in the **Extract Internet Sales** data flow (because no data has been modified since the previous extraction). When package execution is complete, stop debugging.

5. Close Visual Studio.

Results: After this exercise, you should have an SSIS package that uses the high water mark technique to extract only records that have been modified since the previous extraction.

Exercise 2: Using Change Data Capture

Scenario

The Internet Sales database contains a Customers table that does not include a column to indicate when records were inserted or modified. You plan to use the Change Data Capture feature of SQL Server Enterprise Edition to identify records that have changed between data warehouse refresh cycles, and restrict data extractions to include only modified rows.

The main tasks for this exercise are as follows:

1. Enable Change Data Capture
2. Create a Stored Procedure to Retrieve Modified Rows
3. Use the Stored Procedure in a Data Flow
4. Test the Package

▶ Task 1: Enable Change Data Capture

1. In SQL Server Management Studio, execute Transact-SQL statements to enable Change Data Capture in the **InternetSales** database, and monitor net changes in the **Customers** table. You can use the **Enable CDC.sql** file in the D:\Labfiles\Lab07\Starter\Ex2 folder to accomplish this.
2. Open the **Test CDC.sql** file in the D:\Labfiles\Lab07\Starter\Ex2 folder and examine it.
3. Execute the code under the comment **Select all changed customer records between 1/1/1900 and today** and note that no rows are returned because no changes have occurred since Change Data Capture was enabled.
4. Execute the code under the comment **Make a change to all customers (to create CDC records)** to modify data in the **Customers** table.
5. Execute the code under the comment **Now see the net changes** and note that all customer records are returned because they have all been modified within the specified time period while Change Data Capture was enabled.

▶ Task 2: Create a Stored Procedure to Retrieve Modified Rows

1. In SQL Server Management Studio, execute a Transact-SQL statement that creates a stored procedure named **GetChangedCustomers** in the **InternetSales** database. The stored procedure should perform the following tasks. You can execute the **Create SP.sql** file in the D:\Labfiles\Lab07\Starter\Ex2 folder to accomplish this:
   - Retrieve the log sequence numbers for the dates specified in **StartDate** and **EndDate** parameters.
   - If neither of the log sequence numbers is null, return all records that have changed in the **Customers** table.
   - If either of the log sequence numbers is null, return an empty rowset.
2. Test the stored procedure by running the following query:

```
USE InternetSales
```

```
GO
EXEC GetChangedCustomers '1/1/1900', '1/1/2099';
GO
```

▶ Task 3: Use the Stored Procedure in a Data Flow

1. Reset the staging database by running the Transact-SQL code in the **Reset Staging.sql** file in the D:\Labfiles\Lab07\Starter\Ex2 folder.
2. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab07\Starter\Ex2 folder. Then open the **Extract Internet Sales Data.dtsx** SSIS package.
3. On the **Data Flow** tab for the **Extract Customers** task, modify the **Customers** source to execute the following Transact-SQL command, and map the **@StartDate** and **@EndDate** input parameters to the **User::InternetSalesLastExtract** and User::CurrentTime variables:

```
EXEC GetChangedCustomers ?, ?
```

▶ Task 4: Test the Package

1. View the **Extract Customers** data flow, and then start debugging the package and note the number of rows transferred. When package execution is complete, stop debugging.
2. In SQL Server Management Studio, view the contents of the **dbo.ExtractLog** table in the **Staging** database and verify that the **LastExtract** column for the **InternetSales** data source has been updated.
3. View the contents of the **dbo.Customers** table in the **Staging** database and note the rows that have been extracted.
4. Debug the package again and verify that no rows are transferred in the **Extract Customers** data flow this time. When package execution is complete, stop debugging and close Visual Studio.

Results: After this exercise, you should have a database in which Change Data Capture has been enabled, and an SSIS package that uses a stored procedure to extract modified rows based on changes monitored by Change Data Capture.

Exercise 3: Using the CDC Control Task

Scenario

The **HumanResources** database contains an **Employee** table in which employee data is stored. You plan to use the Change Data Capture feature of SQL Server Enterprise Edition to identify modified rows in this table. You also plan to use the CDC Control Task in SSIS to manage the extractions from this table by creating a package to perform the initial extraction of all rows, and a second package that uses the CDC data flow components to extract rows that have been modified since the previous extraction.

The main tasks for this exercise are as follows:

1. Enable Change Data Capture
2. View Staging Tables
3. Create Connection Managers for CDC Components
4. Create a Package for Initial Data Extraction
5. Test Initial Extraction
6. Create a Package for Incremental Data Extraction

7. Test Incremental Extraction

▶ Task 1: Enable Change Data Capture

1. Enable Change Data Capture in the **HumanResources** database, and monitor net changes in the **Employee** table. You can use the **Enable CDC.sql** file in the D:\Labfiles\Lab07\Starter\Ex3 folder to accomplish this.

▶ Task 2: View Staging Tables

1. Verify that the following tables in the **Staging** database are empty:
   - dbo.EmployeeDeletes
   - dbo.EmployeeInserts
   - dbo.EmployeeUpdates

▶ Task 3: Create Connection Managers for CDC Components

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab07\Starter\Ex3 folder.
2. Note that the project already contains an OLE DB connection manager for the **Staging** database, but the CDC components in SSIS require an ADO.NET connection manager. You will therefore need to create ADO.NET connection managers for the **HumanResources** and **Staging** databases.
3. Create a new, project-level connection manager that produces an ADO.NET connection to the **HumanResources** database on the **localhost** server by using Windows authentication. After the connection manager has been created, rename it to **localhost.HumanResources.ADO.NET.conmgr**.
4. Create another new, project-level connection manager that creates an ADO.NET connection to the **Staging** database on the **localhost** server by using Windows authentication. After the connection manager has been created, rename it to **localhost.Staging.ADO.NET.conmgr**.

▶ Task 4: Create a Package for Initial Data Extraction

1. Add a new SSIS package named **Extract Initial Employee Data.dtsx** to the project.
2. Add a **CDC Control Task** to the control flow, and rename the task to **Mark Initial Load Start**. Then configure the **Mark Initial Load Start** task as follows:
   - **SQL Server CDC database ADO.NET connection manager**: localhost HumanResources ADO NET.
   - **CDC control operation**: Mark initial load start.
   - **Variable containing the CDC state**: A new variable named **CDC_State** in the **Extract Initial Employee Data** container.
   - **Automatically store state in a database table**: Selected.
   - **Connection manager for the database where the state is stored**: localhost Staging ADO NET.
   - **Table to use for storing state**: A new table named **[dbo].[cdc_states]** in the **Staging** database.
   - **State name**: CDC_State.
3. Add a Data Flow Task named **Extract Initial Employee Data** to the control flow, connecting the success precedence constraint from the **Mark Initial Load Start** task to the **Extract Initial Employee Data** task.
4. In the **Extract Initial Employee Data** data flow, create an ADO.NET Source named **Employees**, with the following settings:

- **ADO.NET connection manager**: localhost HumanResources ADO NET.
- **Data access mode**: Table or view.
- **Name of the table or view**: dbo"."Employee".

5. Connect the data flow from the **Employees** source to a new ADO.NET Destination named **Employee Inserts** with the following settings:
   - **Connection manager**: localhost Staging ADO NET.
   - **Use a table or view**: "dbo"."EmployeeInserts".
   - **Mappings**: Map all available input columns to destination columns of the same name.
6. On the control flow, add a second CDC Control Task named **Mark Initial Load End** and connect the success precedence constraint from the **Extract Initial Employee Data** task to the **Mark Initial Load End** task. Then configure the **Mark Initial Load End** task as follows:
   - **SQL Server CDC database ADO.NET connection manager**: localhost HumanResources ADO NET.
   - **CDC control operation**: Mark initial load end*.
   - **Variable containing the CDC state**: User:: CDC_State.
   - **Automatically store state in a database table**: Selected.
   - **Connection manager for the database where the state is stored**: localhost Staging ADO NET.
   - **Table to use for storing state**: [dbo].[cdc_states].
   - **State name**: CDC_State.

*Be careful not to select "Mark initial control start".

7. When you have completed the control flow, save the package.

▶ Task 5: Test Initial Extraction

1. Start debugging the **Extract Initial Employee Data.dtsx** package. When package execution is complete, stop debugging.
2. In SQL Server Management Studio, view the contents of the **dbo.EmployeeInserts** table in the **Staging** database to verify that the employee records have been transferred.
3. Refresh the view of the tables in the **Staging** database, and verify that a new table named **dbo.cdc_states** has been created. This table should contain an encoded string that indicates the CDC state.

▶ Task 6: Create a Package for Incremental Data Extraction

1. In Visual Studio, add a new SSIS package named **Extract Changed Employee Data.dtsx** to the **AdventureWorksETL** project.

2. Add a CDC Control Task to the control flow of the new package, and rename the task to **Get Processing Range**. Configure the **Get Processing Range** task as follows:
   - **SQL Server CDC database ADO.NET connection manager**: localhost HumanResources ADO NET.
   - **CDC control operation**: Get processing range.
   - **Variable containing the CDC state**: A new variable named **CDC_State** in the **Extract Changed Employee Data** container.
   - **Automatically store state in a database table**: Selected.
   - **Connection manager for the database where the state is stored**: localhost Staging ADO NET.
   - **Table to use for storing state**: [dbo].[cdc_states].
   - **State name**: CDC_State.
3. Add a Data Flow Task named **Extract Changed Employee Data** to the control flow, connecting the success precedence constraint from the **Get Processing Range** task to the **Extract Changed Employee Data** task.
4. In the **Extract Changed Employee Data** data flow, add a **CDC Source** (in the **Other Sources** section of the SSIS Toolbox) named **Employee Changes**, with the following settings:
   - **ADO.NET connection manager**: localhost HumanResources ADO NET.
   - **CDC enabled table**: [dbo].[Employee].
   - **Capture instance**: dbo_Employee.
   - **CDC processing mode**: Net.
   - **Variable containing the CDC state**: User::CDC_State.
5. Connect the data flow from the **Employee Changes** source to a **CDC Splitter** transformation (in the **Other Transforms** section of the SSIS Toolbox).
6. Add an ADO.NET Destination named **Employee Inserts** below and to the left of the CDC Splitter transformation. Connect the **InsertOutput** data flow output from the CDC Splitter transformation to the **Employee Inserts** destination. Then configure the **Employee Inserts** destination as follows:
   - **Connection manager**: localhost Staging ADO NET.
   - **Use a table or view**: "dbo"."EmployeeInserts".
   - **Mappings**: Map all available input columns other than **_$start_lsn**, **_$operation**, and **_$update_mask** to destination columns of the same name.
7. Add an ADO.NET Destination named **Employee Updates** directly below the CDC Splitter transformation, and connect the **UpdateOutput** data flow output from the CDC Splitter transformation to the **Employee Updates** destination. Then configure the **Employee Updates** destination as follows:
   - **Connection manager**: localhost Staging ADO NET.
   - **Use a table or view**: "dbo"."EmployeeUpdates".
   - **Mappings**: Map all available input columns other than **_$start_lsn**, **_$operation**, and **_$update_mask** to destination columns of the same name.
8. Add an ADO.NET Destination named **Employee Deletes** below and to the right of the CDC Splitter transformation. Connect the **DeleteOutput** data flow output from the CDC Splitter transformation to the **Employee Deletes** destination. Then configure the **Employee Deletes** destination as follows:

- **Connection manager**: localhost Staging ADO NET.
- **Use a table or view**: "dbo"."EmployeeDeletes".
- **Mappings**: Map all available input columns other than **_$start_lsn**, **_$operation**, and **_$update_mask** to destination columns of the same name.

9. On the control flow, add a second CDC Control Task named **Mark Processed Range**, and connect the success precedence constraint from the **Extract Changed Employee Data** task to the **Mark Processed Range** task. Then, configure the **Mark Processed Range** task as follows:
   - **SQL Server CDC database ADO.NET connection manager**: localhost HumanResources ADO NET.
   - **CDC control operation**: Mark processed range.
   - **Variable containing the CDC state**: User:: CDC_State.
   - **Automatically store state in a database table**: Selected.
   - **Connection manager for the database where the state is stored**: localhost Staging ADO NET.
   - **Table to use for storing state**: [dbo].[cdc_states].
   - **State name**: CDC_State.
10. When you have completed the control flow, save the package.

▶ Task 7: Test Incremental Extraction

1. In Visual Studio, view the control flow for the **Extract Changed Employee Data.dtsx** package and start debugging.
2. When execution has completed, view the **Extract Changed Employee Data** data flow to verify that no rows were extracted (because the data is unchanged since the initial extraction).
3. Maximize SQL Server Management Studio, open the **Change Employees.sql** Transact-SQL script file in the D:\Labfiles\Lab07\Starter\Ex3 folder, and execute the script to:
   - Truncate the **dbo.EmployeeInserts**, **dbo.EmployeeUpdates**, and **dbo.EmployeeDeletes** tables in the **Staging** database.
   - Insert a new record in the **Employee** table in the **HumanResources** database.
   - Update employee 281 to change the **Title** column value.
   - Delete employee 273.
4. In Visual Studio, start debugging again.
5. When execution has completed, view the **Extract Changed Employee Data** data flow to verify that three rows were extracted and split into one insert, one update, and one delete. Then stop debugging.

Note: If no rows were transferred, wait for a few seconds, and then run the package again.

6. In SQL Server Management Studio, view the contents of the **dbo.EmployeeDeletes**, **dbo.EmployeeInserts**, and **dbo.EmployeeUpdates** tables in the **Staging** database to verify that they contain the inserted, updated, and deleted rows respectively.
7. Close Visual Studio and minimize SQL Server Management Studio when you are finished.

Results: After this exercise, you should have a **HumanResources** database in which Change Data Capture has been enabled, and an SSIS package that uses the CDC Control to extract the initial set of employee records. You should also have an SSIS package that uses the CDC Control and CDC data flow components to extract modified employee records based on changes recorded by Change Data Capture.

Exercise 4: Using Change Tracking

Scenario

The **ResellerSales** database contains a **Resellers** table that does not include a column to indicate when records were inserted or modified. You plan to use the Change Tracking feature of SQL Server to identify records that have changed between data warehouse refresh cycles, and restrict data extractions to include only modified rows.

The main tasks for this exercise are as follows:

1. Enable Change Tracking
2. Create a Stored Procedure to Retrieve Modified Rows
3. Modify a Data Flow to use the Stored Procedure
4. Test the Package

▶ Task 1: Enable Change Tracking

1. Enable Change Tracking in the **ResellerSales** database, and track changes in the **Resellers** table. You do not need to track which columns were modified. You can use the **Enable CT.sql** file in the D:\Labfiles\Lab07\Starter\Ex4 folder to accomplish this.
2. Use the **Test CT.sql** file in the D:\Labfiles\Lab07\Starter\Ex4 folder to perform the following tasks:
   - Get the current change tracking version number.
   - Retrieve all data from the **Resellers** table.
   - Store the current version number as the previously- retrieved version.
   - Update **Resellers** table.
   - Get the new current version number.
   - Get all changes between the previous and current versions.
   - Store the current version number as the previously- retrieved version.
   - Update the **Resellers** table again.
   - Get the new current version number.
   - Get all changes between the previous and current versions.
3. View the results returned by the script and verify that:
   - The first resultset shows all reseller records.
   - The second resultset indicates that the previously-retrieved version was numbered 0, and the current version is numbered 1.
   - The third resultset indicates that the previously-retrieved version was numbered 1, and the current version is numbered 2.

▶ Task 2: Create a Stored Procedure to Retrieve Modified Rows

1. In SQL Server Management Studio, execute a Transact-SQL statement that enables snapshot isolation and creates a stored procedure named **GetChangedResellers** in the **ResellerSales** database. The stored procedure should perform the following tasks (you can use the Create SP.sql file in the D:\Labfiles\Lab07\Starter\Ex4 folder to accomplish this):
   - Set the isolation level to snapshot.
   - Retrieve the current change tracking version number.
   - If the **LastVersion** parameter is -1, assume that no previous versions have been retrieved, and return all records from the **Resellers** table.
   - If the **LastVersion** parameter is not -1, retrieve all changes between **LastVersion** and the current version.
   - Update the **LastVersion** parameter to the current version, so the calling application can store the last version retrieved for next time.
   - Set the isolation level back to read "committed".
2. Test the stored procedure by running the following query:

```
USE ResellerSales
GO
DECLARE @p BigInt = -1;
EXEC GetChangedResellers @p OUTPUT;
SELECT @p LastVersionRetrieved;
EXEC GetChangedResellers @p OUTPUT;
```

▶ Task 3: Modify a Data Flow to use the Stored Procedure

1. In SQL Server Management Studio open the **Reset Staging.sql** file in the D:\Labfiles\Lab07\Starter\Ex4 folder and execute it.
2. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab07\Starter\Ex4 folder. Then open the **Extract Reseller Data.dtsx** SSIS package.
3. Add a decimal variable named **PreviousVersion** to the package.
4. Add an Execute SQL task named **Get Previously Extracted Version** to the control flow, and configure it to return a single row resultset by executing the following Transact-SQL statement in the **Staging** database, mapping the **LastVersion** column in the result set to the **User::PreviousVersion** variable:

```
SELECT MAX(LastVersion) LastVersion
FROM ExtractLog
WHERE DataSource = 'ResellerSales'
```

5. Add an Execute SQL task named **Update Previous Version** to the control flow, and configure it to execute the following Transact-SQL statement in the **Staging** database, mapping the **User:PreviousVersion** variable to parameter 0 in the query:

```
UPDATE ExtractLog
SET LastVersion = ?
WHERE DataSource = 'ResellerSales'
```

6. Make the necessary changes to the precedence constraint connections in the control flow so that:
   - The **Get Previously Extracted Version** task is executed after the **Get Last Extract Time** task and before the **Extract Resellers** task.

- The **Update Previous Version** task is executed after the **Update Last Extract Time** task and before the **Send Success Notification** task.

7. Modify the **Resellers** source in the **Extract Resellers** data flow to retrieve reseller data by executing the following Transact-SQL statement, mapping the **@LastVersion** input/output parameter to the **User::PreviousVersion** variable:

```
EXEC GetChangedResellers ? OUTPUT
```

▶ Task 4: Test the Package

1. View the **Extract Resellers** data flow, and then start debugging the package and note the number of rows transferred. When package execution is complete, stop debugging.
2. In SQL Server Management Studio, view the contents of the **dbo.ExtractLog** table in the **Staging** database and verify that the **LastVersion** column for the **ResellerSales** data source has been updated.
3. View the contents of the **dbo.Resellers** table and note the rows that have been extracted. Then close SQL Server Management Studio without saving any files.
4. In Visual Studio, debug the package again and verify that no rows are transferred in the Extract Resellers data flow. When package execution is complete, stop debugging.
5. Close Visual Studio.

Results: After this exercise, you should have a database in which Change Tracking has been enabled, and an SSIS package that uses a stored procedure to extract modified rows based on changes recorded by Change Tracking.

Module Review and Takeaways

In this module, you have learned how to plan and implement an ETL solution that extracts data incrementally from data sources.

Review Question(s)

Question: What should you consider when choosing between Change Data Capture and Change Tracking?

Module 8

Loading Data into a Data Warehouse

Contents:

Module Overview

A data warehousing solution loads data into the data warehouse at periodic intervals. Loading data into data warehouse tables presents some challenges, because of the need to maintain historic versions of some data and a requirement to minimize the performance impact on reporting workloads as the data warehouse is loaded.

This module describes the techniques you can use to implement a data warehouse load process.

Objectives

After completing this module, you will be able to:

- Describe the considerations for planning data loads.
- Use SQL Server Integration Services (SSIS) to load new and modified data into a data warehouse.
- Use Transact-SQL techniques to load data into a data warehouse.

Lesson 1
Planning Data Loads

A key challenge in loading a data warehouse is minimizing the time it takes to load a large volume of data. Data warehouse loads often involve tens or even hundreds of thousands of rows, and although many organizations can support loading during periods of low use or inactivity, the operation must still be optimized to complete within the available load window.

This lesson describes considerations to help you plan an efficient data load process.

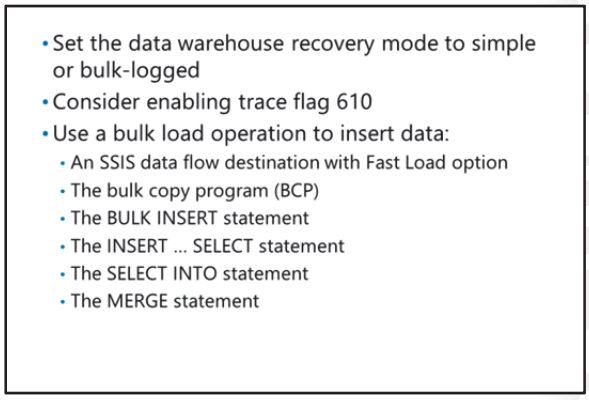
Lesson Objectives

After completing this lesson, you will be able to:

- Use minimal logging to improve data load performance.
- Describe considerations for loading indexed tables.
- Describe key features of slowly changing dimensions.

Minimizing Logging

One way in which load times can be reduced is to minimize database transaction logging during the load operations. SQL Server uses a write-ahead transaction log to log transactional activity in the database, and this logging overhead can affect load performance. However, you can use "minimally logged" operations, in which only extent allocations and metadata changes are logged, to reduce the adverse impact of logging. In most cases, using the TABLOCK query hint causes the database engine to use minimal logging if it is supported by the operation being performed and the destination table.



Set the data warehouse recovery mode to simple or bulk-logged

The first step in ensuring that logging is minimized is to set the recovery mode of the data warehouse database to **simple** or **bulk-logged**. This ensures that minimal logging can be used for operations that support it. Note that setting the recovery mode affects your ability to perform a transaction log backup, so by making this configuration change, you are creating a constraint for your data warehouse backup strategy. However, transactional activity is rare in a data warehouse, and in most cases, a regime that includes full and differential backups is likely to be the most appropriate solution.

Consider enabling trace flag 610

Trace flag 610 was introduced in SQL Server 2008 and controls logging behavior for indexed tables. When this trace flag is enabled, tables that contain data and have a clustered index can support minimal logging for inserts, which can significantly improve load performance for high volumes. Additionally, minimal logging is used for indexed tables wherever possible, even if the TABLOCK hint is not specified. However, you should test performance and behavior with this trace flag before using it in a production data warehouse.

Use a bulk load operation to insert data

To take advantage of minimal logging, you must use a bulk load operation to insert data.

SSIS data flow destinations and the fast load option

If you are loading data into the data warehouse table from an SSIS data flow, select the **Fast Load** option in the data flow destination. This configures the INSERT statement used by the destination to include the TABLOCK hint and causes the operation to be minimally logged if possible. When choosing a data flow destination for a bulk load operation into an SQL Server database, consider the following guidelines:

- Use an **SQL Server Destination** component when both the source data and the SSIS service are hosted on the same server as the destination tables. This destination provides the fastest bulk load performance for SQL Server and supports bulk load options to fine-tune load behavior. You can use this destination only if the SSIS service is running on the same server as the data warehouse into which the data is being loaded.
- If SSIS is hosted on a different computer from the data warehouse, use an **OLE DB Destination** component. This destination supports bulk load, though some additional configuration may be required to support bulk loading of ordered data into a clustered index. Ordered loads into clustered indexes are discussed in the next topic.

The bulk copy program (BCP)

The bulk copy program (BCP) uses the SQL Server bulk application programming interface (API) and supports bulk load operations. Additionally, using SQL Server native format files can improve load performance.

The BULK INSERT statement

You can use the Transact-SQL BULK INSERT statement to load data from a text file. The statement runs within the SQL Server database engine process and uses a bulk load operation to insert the data.

The INSERT … SELECT Statement

You can use a Transact-SQL INSERT statement with a SELECT clause to insert data into a target table from a staging table.

For example, the following code inserts data from a staging table named **StagedOrders** into a fact table named **FactOrders**, looking up surrogate keys in dimensions tables named **DimDate** and **DimProduct**:

Using an INSERT statement with a SELECT clause

```
INSERT INTO FactOrders WITH (TABLOCK)
    (OrderDateKey, ProductKey, Quantity, SalesPrice)
SELECT d.DateKey, p.ProductKey, s.Quantity, s.SalesPrice
FROM StagedOrders AS s
JOIN DimDate AS d
ON s.DateAltKey = d.DateAltKey
JOIN DimProduct AS p
ON s.ProductAltKey = p.ProductAltKey;
```

The SELECT INTO Statement

The SELECT INTO statement creates a new table from data retrieved by a query. Although this technique can occasionally be useful for staging, it is not generally used to load data into a data warehouse because it creates a new table instead of using an existing one.

The MERGE Statement

The MERGE statement combines insert and update operations to merge changes from one table into another. In a data warehouse loading scenario, it can be used to load new rows and perform type 1 updates to a dimension table.

The following code example shows the MERGE statement being used to load a data warehouse table:

Using the MERGE statement

```
MERGE INTO DimProduct WITH (TABLOCK) AS tgt
USING
-- Query to return staged data
  (SELECT ProductAltKey, ProductName, Color
   FROM StagedProducts) AS src (ProductAltKey, ProductName, Color)
   -- Match staged records to existing dimension records
ON (src.ProductAltKey = tgt.ProductAltKey)
-- If a record for this product already exists, update it
WHEN MATCHED THEN
  UPDATE
  SET ProductName = src.ProductName,
      Color = src.Color
-- If not, insert a new record
WHEN NOT MATCHED THEN
  INSERT (ProductAltKey, ProductName, Color)
  VALUES (src.ProductAltKey, src.ProductName, src.Color);
```

The MERGE statement is discussed in greater depth later in this module.

 **Additional Reading:** For more information about non-logged and minimally-logged operations, go to *The Data Loading Performance Guide* at http://msdn.microsoft.com/en-us/library/dd425070(v=sql.100).

Considerations for Indexed Tables

When loading tables on which indexes have been defined, there are some additional considerations that your ETL process must take into account.

- Consider dropping and recreating indexes for large volumes of new data
- Sort data by the clustering key and specify the ORDER hint
- Nonclustered Columnstore indexes make the table read-only

Consider dropping and recreating indexes for large volumes of new data

If you must load a large volume of data into an indexed table, you might find that dropping and recreating the indexes incurs less overhead than inserting data into the table with the indexes in place. Use the table on the next page as a guideline for deciding whether to consider dropping indexes before a bulk insert. If the volume of new data relative to the existing data in the table exceeds the threshold for the provided index scenario, consider dropping the indexes before loading the data.

Indexes	New data relative to existing table size
Clustered index only	30%
Clustered index plus one non-clustered index	25%
Clustered index plus two non-clustered indexes	25%
Single non-clustered index only	100%
Two non-clustered indexes	60%

 **Additional Reading:** For more information about when to drop an index to optimize a data load, go to *Guidelines for Optimizing Bulk Import* at http://msdn.microsoft.com/en-us/library/ms177445.aspx. This article is the source of the information in the preceding table.

Sort data by the clustering key and specify the ORDER hint

When using the BULK INSERT statement, if the table you are loading has a clustered index that you do not intend to drop, and the data to be loaded is already sorted by the clustering column, specify the ORDER hint. This eliminates the internal sort operation that usually occurs when inserting data into a clustered index.

The following code example shows how to use the ORDER hint:

Using the ORDER hint

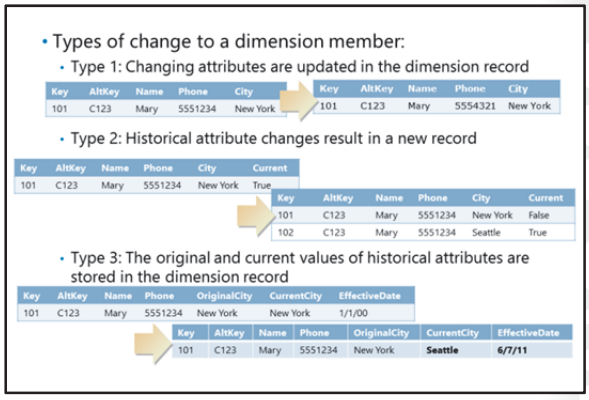
```
BULK INSERT Accounts FROM 'D:\Data\Accounts.csv'
WITH (
  FIELDTERMINATOR = ','
  , ROWTERMINATOR = '\n'
  , TABLOCK
  , ORDER(TransactionDate)
)
```

If you are using the INSERT … SELECT statement, you cannot specify the ORDER hint, but the database engine detects that the data is ordered by the clustering key and optimizes the insert operation accordingly. If you are using an SSIS SQL Server destination, you can specify bulk load options, including order columns, on the **Advanced** tab of the **SQL Destination Editor** dialog box. For OLE DB destinations, you must view the **Component Properties** tab of the **Advanced Editor** for the destination, and add the ORDER(*ColumnName*) hint to the **FastLoadOptions** property.

Nonclustered Columnstore indexes make the table read-only

If you have designed tables with nonclustered columnstore indexes, you should bear in mind that a table with a columnstore index cannot be modified. To load data into a table with a columnstore index, you must either drop the index before loading the new data, or load the data into a new table with the same schema as the indexed table and use a UNION clause to present the indexed historical rows and non-indexed updateable rows as a single logical table to client applications. If you decide to use two tables, you should periodically drop the columnstore index and merge the updateable data into the historical table. Alternatively, consider partitioning the table and using the partition switch technique described later in this module.

Slowly Changing Dimensions



When updating dimension tables, you need to apply the appropriate logic, depending on the kind of dimension attributes being modified. Changes to fixed attributes are not supported, so any updates to columns containing them must either be discarded or cause an error. However, modifications to changing and historical attributes must be supported, and a widely used set of techniques for handling these changes has been identified.

Slowly changing dimensions change over time, while retaining historical attributes for reporting and analysis. Changes to dimension members are usually categorized as the following types:

- Type 1 – Changing attributes are updated in the existing record and the previous value is lost.
- Type 2 – Historical attribute changes result in a new record in the dimension table, representing a new version of the dimension member. A column is used to indicate which version of the dimension member is the current one, either with a flag value to indicate the current record, or by storing the date and time when each version becomes effective. This technique enables you to store a complete history of all versions of the dimension member.
- Type 3 – Historical attribute changes are stored in the existing record, in which the original value is also retained and a column indicates the date on which the new value becomes effective. This technique enables you to store the original and latest versions of the dimension member.

Lesson 2
Using SSIS for Incremental Loads

After your incremental ETL process has transferred source data to the staging location, it must load the data into the data warehouse. One of the main challenges when refreshing data in a data warehouse is identifying which staged records are new dimension members or facts that need to be inserted, and which records represent modifications that require rows to be updated.

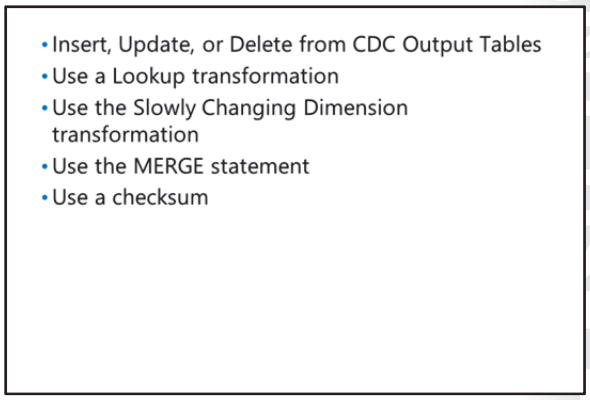
Lesson Objectives

This lesson describes a number of common techniques for performing an incremental load of a data warehouse. After completing this lesson, you will be able to:

- Describe the common options for incrementally loading a data warehouse.
- Load data from staging tables created by using the CDC source and CDC Splitter data flow components.
- Use a Lookup transform to differentiate between new records and updates of existing ones.
- Use the Slowly Changing Dimension transformation to apply type 1 and type 2 changes to a dimension.
- Use the Transact-SQL MERGE statement to insert and update data in a single query.

Options for Incrementally Loading Data

There are a number of commonly-used techniques for loading incremental changes to a data warehouse. The specific technique you should use to load a particular dimension or fact table depends on a number of factors. These include performance, the need to update existing records as well as insert new ones, the need to retain historical dimension attributes, and the location of the staging and data warehouse tables.



Insert, Update, or Delete Data Based on CDC Output Tables

If you use the CDC Splitter to stage modified source data into operation-specific tables, you can create an SSIS package that uses the business keys or unique column combinations in the staging tables to apply the appropriate changes to associated tables in the data warehouse. In most cases, delete operations in the source are applied as logical delete operations in the data warehouse, in which a deleted flag is set instead of actually deleting the matching record.

Use a Lookup Transformation

You can use a Lookup transformation to determine whether a matching record exists in the data warehouse for a record that has been extracted from the data sources. You can then use the no match output of the Lookup transformation to create a data flow for new records that need to be inserted into the data warehouse. Optionally, you can also use the match output of the Lookup transformation to create a data flow that updates existing data warehouse records with new values from the extracted ones.

Use the Slowly Changing Dimension Transformation

The Slowly Changing Dimension transformation enables you to create a complex data flow that inserts new dimension members. It applies type 1 or type 2 changes to existing dimension members, depending on which attributes have been updated. In many data warehousing scenarios, the Slowly Changing Dimension transformation provides an easy-to-implement solution for refreshing dimension tables. However, its performance can be limited for ETL processes with extremely large numbers of rows, for which you might need to create a custom solution for slowly changing dimensions.

Use the MERGE Statement

The MERGE statement is a Transact-SQL construct that you can use to perform insert, update, and delete operations in the same statement. It works by matching rows in a source rowset with rows in a target table, and taking appropriate action to merge the source with the target.

The MERGE statement is appropriate when the source or staging tables and the data warehouse tables are implemented in SQL Server databases and the ETL process is able to execute the MERGE statement using a connection through which all source and target tables can be accessed. In practical terms, this requires that:

- The staging tables and data warehouse are co-located in the same SQL Server database.
- The staging and data warehouse tables are located in multiple databases in the same SQL Server instance, and the credentials used to execute the MERGE statement have appropriate user rights in both databases.
- The staging tables are located in a different SQL Server instance than the data warehouse, but a linked server has been defined enabling the MERGE statement to access both databases. The performance of the MERGE statement over the linked server connection is acceptable.

Use a Checksum

You can use the columns in the staged dimension records to generate a checksum value, and then compare this with a checksum generated from the historic attributes in the corresponding dimension table to identify rows that require a type 2 or type 3 change. When combined with a Lookup transformation to identify new or modified rows, this technique can form the basis of a custom solution for slowly changing dimensions.

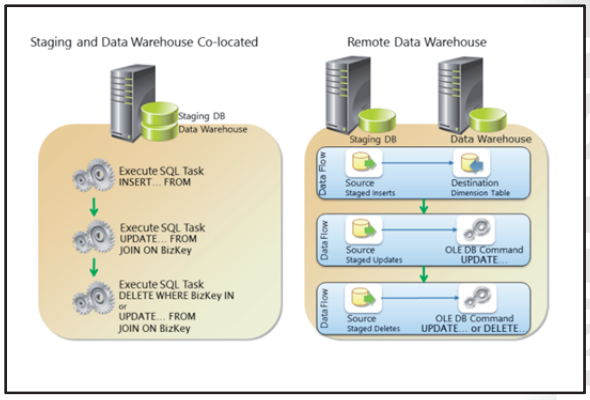
Considerations for Deleting Data Warehouse Records

If you need to propagate record deletions in source systems to the data warehouse, you should consider the following guidelines:

- In most cases, you should use a logical deletion technique in which you indicate that a record is no longer valid, by setting a Boolean column value. It is not common practice to physically delete records from a data warehouse unless you have a compelling business reason to discard all historical information relating to them.
- The techniques you can use to delete records (or mark them as logically deleted) when loading data depend on how you have staged deleted records.
    - If the staging tables for a dimension or fact table contain all valid records (not just those that have been modified since the previous refresh cycle), you can delete any existing records in the data warehouse that do not exist in the staging tables.
    - If the staged data indicates logical deletions in the form of a Boolean column value, and you need to apply logical deletes in the data warehouse, you can treat these deletions as updates.
    - If the keys of records to be deleted are stored separately from new and updated records in the staging database, you may want to perform two distinct load operations for each dimension or fact table—one to load new and updated records, and another for deletions.

Using CDC Output Tables

As described earlier in this module, you can use the CDC Splitter or a custom data flow to stage CDC insert, update, or delete records in separate tables based on the _$operation value in the CDC output. If your data extraction process has taken this approach, you can apply the appropriate changes to the corresponding data warehouse tables. The specific control flow tasks you can use to perform this type of data load depend on the relative locations of the staging tables and the data warehouse.



Loading a Data Warehouse from a Co-Located Staging Database

If the staging tables and the data warehouse are co-located in the same database or server instance, or can be connected by using a linked server, you can use SQL Command control flow tasks to execute set-based Transact-SQL statements that load the data warehouse tables.

For example, you could use the following Transact-SQL statements to load data from CDC output tables into a co-located data warehouse table:

Inserting, updating, and deleting data based on co-located CDC output tables

```
-- Insert new records
INSERT INTO dw.DimProduct (ProductAltKey, ProductName, Description, Price)
SELECT ProductBizKey, ProductName, Description, Price
FROM stg.ProductInserts;

-- Update modifiedrecords
UPDATE dw.DimProduct
SET     dw.DimProduct.ProductName = stg.ProductUpdates.ProductName,
        dw.DimProduct.Description = stg.ProductUpdates.Description,
        dw.DimProduct.Price = stg.ProductUpdates.Price
FROM dw.DimProduct JOIN stg.ProductUpdates
ON dw.DimProduct.ProductAltKey = stg.ProductUpdates.ProductBizKey;

-- Mark deleted records as deleted
UPDATE dw.DimProduct
SET dw.DimProduct.Deleted = 1
FROM dw.DimProduct JOIN stg.ProductDeletes
ON dw.DimProduct.ProductAltKey = stg.ProductDeletes.ProductBizKey;
```

You could use a DELETE statement to remove deleted records from the data warehouse table. However, deleting records in this way can result in loss of historical analytical and reporting data. This can be further complicated by the presence of foreign key constraints between fact and dimension tables. A more common approach is to perform a logical delete by setting a column to a Boolean value to indicate that the record has been deleted from the source system.

Loading a Remote Data Warehouse

If the data warehouse is stored on a different server from the staging database, and no linked server connection is available, you can apply the necessary updates in the staging tables to the data warehouse by creating a data flow for each operation.

To load records from an inserts staging table, create a data flow that includes a source component to extract records from the staging table, and a destination that maps the extracted column to the appropriate data warehouse table.

To apply the changes recorded in an updates staging table to a data warehouse table, create a data flow that includes a source component to extract records from the staging table and an OLE DB Command transformation to execute an UPDATE statement that sets the changeable data warehouse table columns to the corresponding staging table values. This is based on a join between the business key in the staging table and the alternative key in the data warehouse table.

To apply deletes to data warehouse tables based on records in a deletes staging table, create a data flow that includes a source component to extract records from the staging table and an OLE DB Command transformation to execute a DELETE statement that matches the business key in the staging table to the alternative key in the data warehouse table. Alternatively, use the OLE DB command to perform a logical delete by executing an UPDATE statement that sets the deleted flag to 1 for records where the business key in the staging table matches the alternative key in the data warehouse table.

Demonstration: Using CDC Output Tables

In this demonstration, you will see how to load data from CDC output tables.

Demonstration Steps

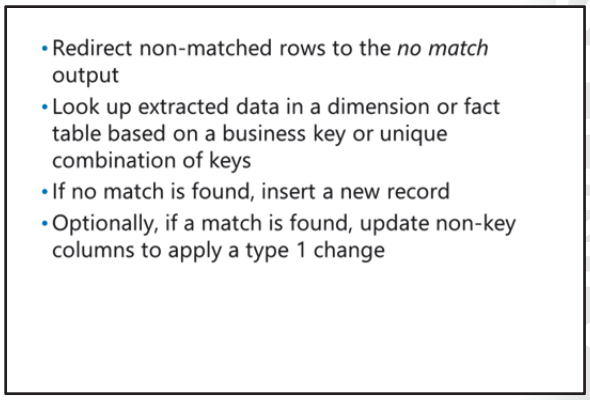
Load Data from CDC Output Tables

1. Ensure 20463C-MIA-DC and 20463C-MIA-SQL are started, and log onto 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Demofiles\Mod08 folder, run **Setup.cmd** as Administrator.
3. Start SQL Server Management Studio and connect to the **localhost** instance of the SQL Server database engine by using Windows authentication.
4. In Object Explorer, expand **Databases**, expand **DemoDW**, and expand **Tables**. Then right-click each of the following tables and click **Select Top 1000 Rows**:
   - **dw.DimShipper**. This is the dimension table in the data warehouse.
   - **stg.ShipperDeletes**. This is the table of records that have been deleted in the source system.
   - **stg.ShipperInserts**. This is the table of new records in the source system.
   - **stg.ShipperUpdates**. This is the table of rows that have been updated in the source system.
5. Start Visual Studio and open the **IncrementalETL.sln** solution in the D:\Demofiles\Mod08 folder. Then in Solution Explorer, double-click the **Load Shippers.dtsx** SSIS package.
6. On the control flow surface, double-click the **Load Inserted Shippers** Execute SQL task. Note that the SQL Statement inserts data into **dw.DimShippers** from the **stg.ShipperInserts** table. Then click **Cancel**.
7. On the control flow surface, double-click the **Load Updated Shippers** Execute SQL task. Note that the SQL Statement updates data in **dw.DimShippers** with new values from the **stg.ShipperUpdates** table. Then click **Cancel**.
8. On the control flow surface, double-click the **Load Deleted Shippers** data flow task. On the data flow surface, note that the task extracts data from the **stg.ShipperDeletes** table, and then uses an OLE DB Command transformation to update the **Deleted** column in **dw.DimShippers** for the extracted rows.
9. On the **Debug** menu, click **Start Debugging**, and observe the control flow as it executes. When execution is complete, on the **Debug** menu, click **Stop Debugging** and minimize Visual Studio.
10. In SQL Server Management Studio, right-click the **dw.DimShipper** table and click **Select Top 1000 Rows** to review the changes that have been made.

11. Minimize SQL Server Management Studio.

The Lookup Transformation

To use a Lookup transformation when loading a data warehouse table, connect the source output that extracts the staged data to the Lookup transformation and apply the following configuration settings:



- Redirect non-matched rows to the no match output.
- Look up the primary key column or columns in the dimension or fact table you want to refresh by matching it to one or more input columns from the staged data. If the staged data includes a business key column and the business key is stored as an alternative key in the data warehouse table, match one to the other. Alternatively, you can match a combination of columns that uniquely identifies a fact or dimension member.
- Connect the no match output from the Lookup transformation to a data flow that ultimately inserts new records into the data warehouse.
- To update existing data warehouse records with modified values in the staged data, connect the match output of the Lookup transformation to a data flow that uses an OLE DB Command transformation. This is based on the primary key you retrieved in the Lookup transformation.

Note: The Lookup transformation uses an in-memory cache to optimize performance. If the same data set is to be used in multiple lookup operations, you can persist the cache to a file and use a Cache Connection Manager to reference it. This further improves performance by decreasing the time it takes to load the cache, but results in lookup operations against a data set that might not be as up-to-date as the data in the database. For more information about configuring caching for the Lookup transformation, go to *Lookup Transformation* in SQL Server Books Online.

Demonstration: Using the Lookup Transformation

In this demonstration, you will see how to:

- Use a Lookup Transformation to Insert Rows.
- Use a Lookup Transformation to Insert and Update Rows.

Demonstration Steps

Use a Lookup Transformation to Insert Rows

1. Ensure you have completed the previous demonstrations in this module.
2. Maximize Visual Studio, and in Solution Explorer, double-click the **Load Geography.dtsx** SSIS package.

3. On the control flow surface, double-click **Load Geography Dimension** to view the data flow surface. Then, on the data flow surface, double-click **Staged Geography Data**. Note that the SQL command used by the OLE DB source extracts geography data from the **stg.Customers** and **stg.Salespeople** tables, and then click **Cancel**.
4. On the data flow surface, double-click **Lookup Existing Geographies** and note the following configuration settings of the Lookup transformation. Then click **Cancel**:
   - On the **General** tab, unmatched rows are redirected to the no-match output.
   - On the **Connection** tab, the data to be matched is retrieved from the **dw.DimGeography** table.
   - On the **Columns** tab, the **GeographyKey** column is retrieved for rows where the input columns are matched.
5. On the data flow surface, note that the data flow arrow connecting **Lookup Existing Geographies** to **New Geographies** represents the *no match* data flow.
6. Double-click **New Geographies**, and note that the rows in the no match data flow are inserted into the **dw.DimGeography** table. Then click **Cancel**.
7. On the **Debug** menu, click **Start Debugging**, and observe the data flow as it executes. Note that while four rows are extracted from the staging tables, only one does not match an existing record. The new record is loaded into the data warehouse, and the rows that match existing records are discarded. When execution is complete, on the **Debug** menu, click **Stop Debugging**.

Use a Lookup Transformation to Insert and Update Rows

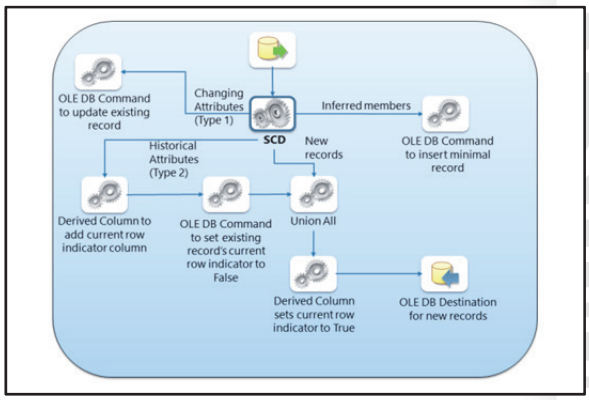
1. In Visual Studio, in Solution Explorer, double-click the **Load Products.dtsx** SSIS package. Then on the control flow surface, double-click **Load Product Dimension** to view the data flow surface.
2. On the data flow surface, double-click **Staged Products**, note that the SQL command used by the OLE DB source extracts product data from the **stg.Products** table, and then click **Cancel**.
3. On the data flow surface, double-click **Lookup Existing Products** and note the following configuration settings of the Lookup transformation. Then click **Cancel**:
   - On the **General** tab, unmatched rows are redirected to the no-match output.
   - On the **Connection** tab, the data to be matched is retrieved from the **dw.DimProduct** table.
   - On the **Columns** tab, the **ProductKey** column is retrieved for rows where the **ProductBusinessKey** column in the staging table matches the **ProductAltKey** column in the data warehouse dimension table.
4. On the data flow surface, note that the data flow arrow connecting **Lookup Existing Products** to **Insert New Products** represents the *no match* data flow. The data flow arrow connecting **Lookup Existing Products** to **Update Existing Products** represents the *match* data flow.
5. Double-click **Insert New Products**, and note that the rows in the no match data flow are inserted into the **dw.DimProduct** table. Then click **Cancel**.
6. Double-click **Update Existing Products**, and note the following configuration settings. Then click **Cancel**:
   - On the **Connection Managers** tab, the OLE DB Command transformation connects to the **DemoDW** database.
   - On the **Component Properties** tab, the **SQLCommand** property contains a parameterized Transact-SQL statement that updates the **ProductName**, **ProductDescription**, and **ProductCategoryName** columns for a given **ProductKey**.

- On the **Column Mapping** tab, the **ProductName**, **ProductDescription**, **ProductCategoryName**, and **ProductKey** input columns from the match data flow are mapped to the parameters in the SQL command.

7. On the **Debug** menu, click **Start Debugging**, and observe the data flow as it executes. Note the number of rows extracted from the staging tables, and how the **Lookup** transformation splits these rows to insert new records and update existing ones.
8. When execution is complete, on the **Debug** menu, click **Stop Debugging**. Then minimize Visual Studio.

The Slowly Changing Dimension Transformation

The Slowly Changing Dimension transformation provides a wizard you can use to generate a complex data flow to handle inserts and updates for a dimension table. Using the Slowly Changing Dimension wizard, you can specify:



- The columns containing keys that can be used to look up existing dimension records in the data warehouse.
- The non-key columns that are fixed, changing, or historic attributes.
- Whether or not changes to a fixed column should produce an error or be ignored.
- The column in the dimension table that should be used to indicate the current version of a dimension member for which historic attributes have changed over time.
- Whether or not the staged data includes inferred members for which a minimal record should be inserted. Inferred members are identified based on the value of a specified column in the source.

After completing the wizard, the Slowly Changing Dimension transformation generates a data flow that includes the following:

- A path to insert new dimension records.
- A path to update dimension records where a changing attribute has been modified. This is an implementation of a type 1 change.
- A path to update the current record indicator and insert a new record for dimension members where a historic attribute has been modified. This is an implementation of a type 2 change.
- If specified, a path to insert minimal records for inferred members in the source data.

Demonstration: Implementing a Slowly Changing Dimension

In this demonstration, you will see how to:

- View an SCD Data Flow.
- Create an SCD Data Flow.

Demonstration Steps

View an SCD Data Flow

1. Ensure you have completed the previous demonstrations in this module.
2. Maximize Visual Studio, and in Solution Explorer, double-click the **Load Salespeople.dtsx** SSIS package.
3. On the control flow surface, double-click **Load Salesperson Dimension** to view the data flow surface. Note the following details about the data flow:
    - Staged salespeople records are extracted from the **stg.Salesperson** table.
    - The **Lookup Geography Key** transformation retrieves the **GeographyKey** value for the salesperson based on the **PostalCode**, **City**, **Region**, and **Country** column values.
    - **Salesperson SCD** is a slowly changing dimension transformation that generates multiple data flow paths for historical attribute updates requiring the insertion of a new record, new dimension member records, and changing attribute updates.
4. On the **Debug** menu, click **Start Debugging** and observe the data flow as it executes. Note that the staged data includes one new salesperson, and a salesperson record with a modified historical attribute. This results in two new records in the data warehouse.
5. When execution is complete, on the **Debug** menu, click **Stop Debugging**.

Create an SCD Data Flow

1. Maximize SQL Server Management Studio and in Object Explorer, right- click the **stg.Customers** table and click **Select Top 1000 Rows**. This table contains staged customer data.
2. Right-click the **dw.DimCustomers** table and click **Select Top 1000 Rows**. This table contains customer dimension data. Note that the staged data includes two new customers, one with a changed email address, and another who has moved from New York to Seattle.
3. In Visual Studio, in Solution Explorer, double-click the **Load Customers.dtsx** SSIS package. Then on the control flow surface, double-click **Load Customer Dimension** and note the following details about the data flow:
    - Staged customer records are extracted from the **stg.Customer** table.
    - The **Lookup Geography Key** transformation retrieves the **GeographyKey** value for the customer based on the **PostalCode**, **City**, **Region**, and **Country** column values.
4. In the SSIS Toolbox, drag a **Slowly Changing Dimension** to the data flow surface, below the **Lookup Geography Key** transformation. Then right-click **Slowly Changing Dimension**, click **Rename**, and change the name to **Customer SCD**.
5. Click **Lookup Geography Key**, and then drag the blue data flow arrow to **Customer SCD**. In the **Input Output Selection** dialog box, in the **Output** drop-down list, select **Lookup Match Output**, and then click **OK**.
6. Double-click **Customer SCD**, and then complete the Slowly Changing Dimension Wizard, specifying the following configuration settings:
    - On the **Select a Dimension Table and Keys** page, in the **Connection manager** drop-down list, select **localhost.DemoDW**, and in the **Table or view** drop-down list, select **[dw].[DimCustomer]**. Then specify the following column mappings, leaving the input column and key type for the **CurrentRecord** dimension column blank:
        - **CustomerID**: CustomerAltKey (Business Key)
        - **CustomerEmail**: CustomerEmail (Not a key column).

        - **GeographyKey**: CustomerGeographyKey (Not a key column).
        - **CustomerName**: CustomerName (Not a key column).
    - On the **Slowly Changing Dimension Columns** page, specify the following change types:
        - **CustomerEmail**: Changing Attribute.
        - **CustomerName**: Changing Attribute.
        - **CustomerGeographyKey**: Historical Attribute.
    - On the **Fixed and Changing Attribute Options** page, leave both options clear.
    - On the **Historical Attribute Options** page, select **Use a single column to show current and expired records**. Then, in the **Column to indicate current record** drop-down list, select **CurrentRecord**. In the **Value when current** drop-down list, select **True**, and in the **Expiration value** drop-down list, select **False**.
    - On the **Inferred Dimension Members** page, uncheck the **Enable inferred member support** option.
7. When you have completed the wizard, note the data flow it has created.
8. On the **Debug** menu, click **Start Debugging** and observe the data flow as it executes. Note that the staged data includes two new customers. One customer record has a modified changing attribute where their email address has been amended, the other has a modified historical attribute, having moved to a new geographical location.
9. When execution is complete, on the **Debug** menu, click **Stop Debugging**. Then close Visual Studio and minimize SQL Server Management Studio.

Lesson 3
Using Transact-SQL Loading Techniques

SSIS provides a range of techniques that you can use to load data into a data warehouse. They often involve executing a Transact-SQL statement in the staging database or in the data warehouse itself.

SQL Server supports Transact-SQL statements such as MERGE, and the SWITCH statement for partitioned tables that can often provide a high-performance solution for loading data warehouse tables.

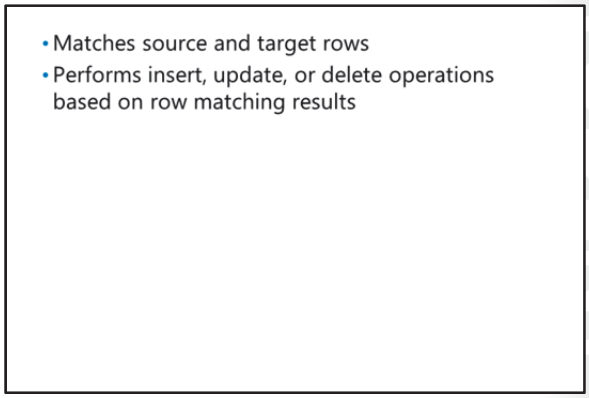
Lesson Objectives

After completing this lesson, you will be able to:

- Use the MERGE statement to load a table with new and updated records in a single statement.
- Use the SWITCH statement to load a partitioned table.

The MERGE Statement

The MERGE statement combines insert and update operations to merge changes from one table into another. In a data warehouse loading scenario, it can be used to load new and updated rows in dimension and fact tables. When used to load a data warehouse table, a MERGE statement includes the following components:



- An INTO clause containing the name of the target table being loaded.
- A USING clause containing a query that defines the source data to be loaded into the target table. This is usually executed against tables in a staging database, often using JOIN clauses to look up dimension keys in the data warehouse.
- An ON clause specifying the criteria used to match rows in the source rowset with rows in the target table.
- A WHEN MATCHED clause specifying the action to be taken for rows in the target table that match rows in the source rowset—often an UPDATE statement to apply a type 1 change.
- A WHEN NOT MATCHED clause specifying the action to be taken when no matches are found in the target table for rows in the source rowset. This is usually an INSERT statement to add a new record to the data warehouse.

The following code example shows how the MERGE statement can be used to load new records and perform type 1 updates in a dimension table:

Applying type 1 updates with MERGE

```
MERGE INTO DimProduct WITH (TABLOCK) AS tgt
USING
-- Query to return staged data
  (SELECT ProductAltKey, ProductName, Color
   FROM StagedProducts) AS src (ProductAltKey, ProductName, Color)
   -- Match staged records to existing dimension records
ON (src.ProductAltKey = tgt.ProductAltKey)
-- If a record for this product already exists, update it
WHEN MATCHED THEN
  UPDATE
  SET ProductName = src.ProductName,
      Color = src.Color
-- If not, insert a new record
WHEN NOT MATCHED THEN
  INSERT (ProductAltKey, ProductName, Color)
  VALUES (src.ProductAltKey, src.ProductName, src.Color);
```

Additionally, when combined with the OUTPUT clause, you can use the **$action** metadata column to detect updates, and implement type 2 changes.

The following code example shows how to perform type 2 updates with the MERGE statement:

Applying type 2 updates with the MERGE statement

```
INSERT INTO DimCustomer WITH (TABLOCK)
  (CustomerAltKey, Name, City, CurrentFlag, StartDate, EndDate)
SELECT CustomerAltKey, Name, City, 1, getdate(), NULL
FROM
  (MERGE INTO DimCustomer AS tgt
   USING
    -- Query to return staged customer data
    (SELECT CustomerAltKey, Name, City
     FROM [StagedCustomer)
     AS src (CustomerAltKey, Name, City)
   -- Match staged customers to existing (current) dimension records
   ON (src.CustomerAltKey = tgt.CustomerAltKey AND tgt.CurrentFlag = 1)
   -- If a current record for this customer already exists, mark it as a type 2 change
   WHEN MATCHED THEN
     UPDATE
     SET tgt.CurrentFlag = 0, tgt.EndDate = getdate()
   -- If not, insert a new record
   WHEN NOT MATCHED THEN
     INSERT (CustomerAltKey, Name, City, CurrentFlag, StartDate, EndDate)
     VALUES (CustomerAltKey, Name, City, 1, getdate(), NULL)
  -- Now output the records you've inserted or updated
  OUTPUT $action, CustomerAltKey, Name, City)
  AS Type2Changes(MergeAction, CustomerAltKey, Name, City)
-- filter them so you insert new records for the type 2 updates.
WHERE MergeAction = 'UPDATE';
```

When you implement an incremental ETL process with SQL Server Integration Services, you can use an SQL Command task in the control flow to execute a MERGE statement. However, you must ensure that the connection manager assigned to the SQL Command task provides access to the source and target tables.

Demonstration: Using the MERGE Statement

In this demonstration, you will see how to use the MERGE statement.

Demonstration Steps

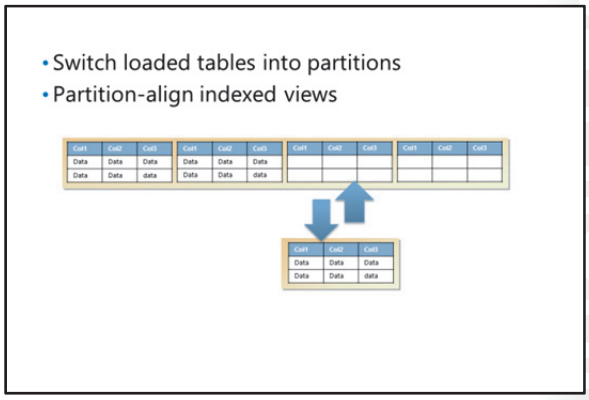
Use the MERGE Statement

1. Ensure you have completed the previous demonstrations in this module.
2. Maximize SQL Server Management Studio and in Object Explorer. Right-click the **stg.SalesOrders** table in the **DemoDW** database and click **Select Top 1000 Rows**. This table contains staged sales order data.
3. Right-click the **dw.FactSalesOrders** table and click **Select Top 1000 Rows**. This table contains sales order fact data. Note that the staged data includes three order records that do not exist in the data warehouse fact table (with OrderNo and ItemNo values of 1005 and 1; 1006 and 1; and 1006 and 2 respectively), and one record that does exist but for which the Cost value has been modified (OrderNo 1004, ItemNo 1).
4. Open the **Merge Sales Orders.sql** file in the D:\Demofiles\Mod08 folder and view the Transact-SQL code it contains, noting the following details:
   - The MERGE statement specified the **DemoDW.dw.FactSalesOrders** table as the target.
   - A query that returns staged sales orders and uses joins to look up dimension keys in the data warehouse is specified as the source.
   - The target and source tables are matched on the **OrderNo** and **ItemNo** columns.
   - Matched rows are updated in the target.
   - Unmatched rows are inserted into the target.
5. Click **Execute** and note the number of rows affected.
6. Right click the **dw.FactSalesOrders** table and click **Select Top 1000 Rows**. Then compare the contents of the table with the results of the previous query you performed in step 4.
7. Minimize SQL Server Management Studio.

Loading Partitioned Tables

Fact tables are commonly partitioned to simplify management and data loads. When using partitioned fact tables, you should consider the following guidelines for your ETL data load processes:

- Switch loaded tables into partitions.
- Partition-align indexed views.



Switch loaded tables into partitions

After you create partitioned fact tables, you can optimize data load operations by switching a loaded table into an empty partition. This technique can be used to load a partition from a table that:

- Has the same schema as the partition, including column names and data types.
- Has the same indexes as the partition, including columnstore indexes.
- Has the same compression setting as the partition.

- Has a check constraint that uses the same criteria as the partition function.
- Is stored on the same filegroup as the partition.

To use this technique to load new data into a partition, maintain an empty partition at the end of the table. The lower bound of the partition range for the empty partition should be the date key value for the next set of data to be loaded. The basic technique to load new data into a partition uses the following procedure:

1. If each partition is stored on its own filegroup, add a filegroup to the database and set it as the next used filegroup for the partition scheme.
2. Split the empty partition at the end of the table, specifying the key for the *upper* bound of the data to be loaded. This creates one empty partition for the new data, and another to be maintained at the end of the table for the next load cycle.
3. Create a table on the same filegroup as the second to last, empty partition, with the same columns and data types as the partitioned table. For fastest load performance, create this table as a heap (a table with no indexes).
4. Bulk insert the staged data into the load table you created in the previous step.
5. Add a constraint that checks that the partitioning key column values are within the range of the target partition to the load table.
6. Add indexes to the load table that match those on the partitioned table.
7. Switch the partition and the load table.
8. Drop the load table.

This technique works best when the table is partitioned on a date key that reflects the data warehouse load cycle, so each new load is performed into a new partition. However, it can also be used when partitions do not match load intervals.

- When partitions are based on more frequent intervals than load cycles (for example, each partition holds a week's worth of data, but the data is loaded monthly), you can switch multiple load tables into multiple partitions.
- When partitions are based on less frequent intervals than load cycles (for example, each partition holds a month's worth of data, but the data is loaded daily), you can:
  - Create a new partition for the load, and then merge it with the previous partition.
  - Switch out a partially-loaded partition, drop the indexes on the partially-populated load table, insert the new rows, recreate the indexes, and switch the partition back in. This technique can also be used for late arriving facts (rows that belong in partitions that have previously been loaded) and updates.

 **Additional Reading:** For more information about loading partitioned fact tables, go to *Loading Bulk Data into a Partitioned Fact Table* at http://technet.microsoft.com/en-us/library/cc966380.aspx.

Partition-align indexed views

If you plan to use indexed views in the data warehouse, align the indexes to the partitions on the underlying table. When indexed views are partition-aligned, you can switch partitions without having to drop and recreate the indexes on the views.

Demonstration: Loading a Partitioned Table

In this demonstration, you will see how to:

- Split a Partition.
- Create a Load Table.
- Switch a Partition.

Demonstration Steps

Split a Partition

1. Ensure you have completed the previous demonstrations in this module.
2. Maximize SQL Server Management Studio and open the **Load Partitions.sql** file in the D:\Demofiles\Mod08 folder.
3. Select the code under the comment **Create a partitioned table**, and then click **Execute**. This creates a database with a partitioned fact table, on which a columnstore index has been created.
4. Select the code under the comment **View partition metadata**, and then click **Execute**. This shows the partitions in the table with their starting and ending range values, and the number of rows they contain. Note that the partitions are shown once for each index (or for the heap if no clustered index exists). Note that the final partition (4) is for key values of 20020101 or higher and currently contains no rows.
5. Select the code under the comment **Add a new filegroup and make it the next used**, and then click **Execute**. This creates a filegroup, and configures the partition scheme to use it for the next partition to be created.
6. Select the code under the comment **Split the empty partition at the end**, and then click **Execute**. This splits the partition function to create a new partition for keys with the value 20030101 or higher.
7. Select the code under the comment **View partition metadata again**, and then click **Execute**. This time the query is filtered to avoid including the same partition multiple times. Note that the table now has two empty partitions (4 and 5).

Create a Load Table

1. Select the code under the comment **Create a load table**, and then click **Execute**. This creates a table on the same filegroup as partition 4, with the same schema as the partitioned table.
2. Select the code under the comment **Bulk load new data**, and then click **Execute**. This inserts the data to be loaded into the load table (in a real solution, this would typically be bulk loaded from staging tables).
3. Select the code under the comment **Add constraints and indexes**, and then click **Execute**. This adds a check constraint to the table that matches the partition function criteria, and a columnstore index that matches the index on the partitioned table.

Switch a Partition

1. Select the code under the comment **Switch the partition**, and then click **Execute**. This switches the load table with the partition on which the value 20020101 belongs. Note that the required partition number is returned by the $PARTITION function.
2. Select the code under the comment **Clean up and view partition metadata**, and then click **Execute**. This drops the load table and returns the metadata for the partitions. Note that partition 4 now contains two rows that were inserted into the load table.

Lab: Loading a Data Warehouse

Scenario

You are ready to start developing the SSIS packages that load data from the staging database into the data warehouse.

Objectives

After completing this lab, you will be able to:

- Load data from CDC output tables.
- Use a Lookup transformation to load data.
- Use the Slowly Changing Dimension transformation.
- Use the MERGE statement.

Estimated Time: 60 minutes

Virtual machine: 20463C-MIA-SQL

User name: ADVENTUREWORKS\Student

Password: Pa$$w0rd

Exercise 1: Loading Data from CDC Output Tables

Scenario

The staging database in your ETL solution includes tables named **EmployeeInserts**, containing employee records that have been inserted in the employee source system, **EmployeeUpdates**, containing records modified in the employee source system, and **EmployeeDeletes** containing records that have been deleted in the employee source system. You must use these tables to load and update the **DimEmployee** dimension table, which uses a **Deleted** flag to indicate records that have been deleted in the source system.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Create a Data flow for Inserts
3. Create a Data Flow for Updates
4. Create a Data Flow for Deletes
5. Test the Package

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. Run **Setup.cmd** in the D:\Labfiles\Lab08\Starter folder as Administrator.

▶ Task 2: Create a Data flow for Inserts

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab08\Starter\Ex1 folder.
2. In Solution Explorer, note that a connection manager for the **AWDataWarehouse** database has been created.
3. Add a new SSIS package named **Load Employee Data.dtsx** to the project.

4. Add a Data Flow Task named **Insert Employees** to the control flow.
5. In the **Insert Employees** data flow, create an OLE DB source named **Staged Employee Inserts** that extracts data from the **[dbo].[EmployeeInserts]** table in the **Staging** database.
6. Connect the **Staged Employees** source to a new OLE DB destination named **New Employees** that uses fast load option to load data into the **DimEmployee** table in the **AWDataWarehouse** database.

▶ Task 3: Create a Data Flow for Updates

1. On the control flow surface of the **Load Employee Data.dtsx** package, connect the success precedence constraint of the **Insert Employees** data flow task to a new Data Flow Task named **Update Employees**.
2. In the **Update Employees** data flow, create an OLE DB source named **Staged Employee Updates** that extracts data from the **[dbo].[EmployeeUpdates]** table in the **Staging** database.
3. Connect the data flow from the **Staged Employee Updates** source to a new OLE DB Command transformation named **Update Existing Employees** that executes the following Transact-SQL statement in the **AWDataWarehouse** database:

```
UPDATE dbo.DimEmployee
SET          FirstName = ?, LastName = ?, EmailAddress = ?, Title = ?, HireDate = ?
WHERE EmployeeAlternateKey = ?
```

   Use the following column mappings:

   - **FirstName**: Param_0
   - **LastName**: Param_1
   - **EmailAddress**: Param_2
   - **Title**: Param_3
   - **HireDate**: Param_4
   - **EmployeeID**: Param_5

▶ Task 4: Create a Data Flow for Deletes

1. On the control flow surface of the **Load Employee Data.dtsx** package, connect the success precedence constraint of the **Update Employees** data flow task to a new one named **Delete Employees**.
2. In the **Delete Employees** data flow, create an OLE DB source named **Staged Employee Updates** that extracts data from the **[dbo].[EmployeeDeletes]** in the **Staging** database.
3. Connect the data flow from the **Staged Employee Deletes** source to a new OLE DB Command transformation named **Delete Existing Employees** that executes the following Transact-SQL statement in the **AWDataWarehouse** database, mapping the **EmployeeID** column to **Param_0**:

```
UPDATE dbo.DimEmployee
SET Deleted = 1
WHERE EmployeeAlternateKey = ?
```

▶ Task 5: Test the Package

1. In Visual Studio, start debugging the **Load Employee Data.dtsx** package, and when execution is complete, view the data flow surface for each of the data flow tasks, noting the numbers of rows processed in each task.
2. When you have finished, stop debugging and close Visual Studio.

Results: After this exercise, you should have an SSIS package that uses data flows to apply inserts, updates, and logical deletes in the data warehouse, based on staging tables extracted by the CDC Control task and data flow components.

Exercise 2: Using a Lookup Transformation to Insert or Update Dimension Data

Scenario

Another BI developer has partially implemented an SSIS package to load product data into a hierarchy of dimension tables. You must complete this package by creating a data flow that uses a lookup transformation to determine whether a product dimension record already exists, and then insert or update a record in the dimension table accordingly.

The main tasks for this exercise are as follows:

1. View Data Flows
2. Create a Data Flow
3. Add a Lookup Transformation for Parent Keys
4. Add a Lookup Transformation for Product Records
5. Add a Destination for New Products
6. Add an OLE DB Command for Updated Product Records
7. Test the Package

▶ Task 1: View Data Flows

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab08\Starter\Ex2 folder. Then open the **Load Products Data.dtsx** SSIS package.
2. View the data flow for the **Load Product Category Dimension** task, and note the following:
   - The **Staged Product Category Data** source extracts product category data from the **InternetSales** and **ResellerSales** tables in the **Staging** database.
   - The **Lookup Existing Product Categories** task retrieves the **ProductCategoryKey** value for product categories that exist in the **DimProductCategory** table in the **AWDataWarehouse** database by matching the product category business key in the staging database to the product category alternative key in the data warehouse.
   - The **Lookup No Match Output** data flow path from the **Lookup Existing Product Categories** task connects to the **New Product Categories** destination, and the **Lookup Match Output** data flow path connects to the **Update Existing Product Categories** task.
   - The **New Product Categories** destination loads new product category records into the **DimProductCategory** table.
   - The **Update Existing Product Categories** task executes a Transact-SQL statement to update the **ProductCategoryName** column in the **DimProductCategory** table for an existing row based on the **ProductCategoryKey**.
3. View the data flow for the **Load Product Subcategory Dimension** task, and note that this data flow inserts or updates product category dimension data using a similar approach to the **Load Product Category Dimension** data flow. Additionally, it has a lookup task to retrieve the

ProductCategoryKey in **AWDataWarehouse** for the parent category, which should have already been loaded.

▶ Task 2: Create a Data Flow

1. Add a Data Flow task named **Load Product Dimension** to the control flow of the **Load Products Data.dtsx** package, and connect the success precedence constraint from the **Load Product Subcategory Dimension** task to the **Load Product Dimension** task.
2. In the data flow for the **Load Product Dimension** data flow task, add an OLE DB source named **Staged Product Data** that uses the following Transact-SQL command to retrieve data from the **Staging** database:

```
SELECT DISTINCT ProductSubcategoryBusinessKey, ProductBusinessKey, ProductName,
StandardCost, Color, ListPrice, Size, Weight, Description
FROM dbo.InternetSales
UNION
SELECT DISTINCT ProductSubcategoryBusinessKey, ProductBusinessKey, ProductName,
StandardCost, Color, ListPrice, Size, Weight, Description
FROM dbo.ResellerSales
```

▶ Task 3: Add a Lookup Transformation for Parent Keys

1. In the **Load Product Dimension** data flow, add a Lookup transformation named **Lookup Parent Subcategory**, connect the output data flow path from the **Staged Product Data** source to it, and configure it as follows:
   - The component should fail if there are rows with no matching entries.
   - The components should look up rows in the **[dbo].[DimProductSubcategory]** table in the **AWDataWarehouse** database by matching the **ProductSubcategoryBusinessKey** column to the **ProductSubcategoryAlternateKey** lookup column.
   - Each matching lookup should return the **ProductSubCategoryKey** lookup column and add it to the data flow.

▶ Task 4: Add a Lookup Transformation for Product Records

1. In the **Load Product Dimension** data flow, add a Lookup transformation named **Lookup Existing Products**, connect the **Lookup Match Output** data flow path from the **Lookup Parent Subcategory** transformation to it, and configure as follows:
   - Rows with no matching entries should be redirected to the *no match* output.
   - The components should look up rows in the **[dbo].[DimProduct]** table in the **AWDataWarehouse** database by matching the **ProductBusinessKey** column to the **ProductAlternateKey** lookup column.
   - Each matching lookup should return the **ProductKey** lookup column and add it to the data flow.

▶ Task 5: Add a Destination for New Products

1. In the **Load Product Dimension** data flow, connect the **Lookup No Match** output from the **Lookup Existing Products** transformation to a new OLE DB destination named **New Products** that loads unmatched product records into the **DimProduct** table in the **AWDataWarehouse** database.
2. Use the following column mappings when loading the data:
   - <ignore>: ProductKey
   - ProductBusinessKey: ProductAlternateKey
   - ProductName: ProductName

- StandardCost: StandardCost
- Color: Color
- ListPrice: ListPrice
- Size: Size
- Weight: Weight
- Description: Description
- ProductSubcategoryKey: ProductSubcategoryKey

▶ Task 6: Add an OLE DB Command for Updated Product Records

1. In the **Load Product Dimension** data flow, connect the **Lookup Match Output** data flow path from the **Lookup Existing Products** transformation to a new OLE DB Command transformation named **Update Existing Products**.
2. Configure the Update Existing Products transformation to use the following Transact-SQL command to update the **DimProduct** table in the **AWDataWarehouse** database:

```
UPDATE dbo.DimProduct
SET  ProductName = ?, StandardCost = ?, Color = ?, ListPrice = ?, Size = ?,
Weight = ?, Description = ?, ProductSubcategoryKey = ?
WHERE ProductKey = ?
```

3. Use the following column mappings for the query:
   - ProductName: Param_0
   - StandardCost: Param_1
   - Color: Param_2
   - ListPrice: Param_3
   - Size: Param_4
   - Weight: Param_5
   - Description: Param_6
   - ProductSubcategoryKey: Param_7
   - ProductKey: Param_8

▶ Task 7: Test the Package

1. With the **Load Product Dimension** data flow visible, start debugging the package and verify that all rows flow to the **New Products** destination (because the data warehouse contained no existing product records). When package execution is complete, stop debugging.
2. Debug the package again and verify that all rows flow to the **Update Existing Products** transformation this time (because all staged product records were loaded to the data warehouse during the previous execution, so they all match existing records). When package execution is complete, stop debugging and close Visual Studio.

Results: After this exercise, you should have an SSIS package that uses a Lookup transformation to determine whether product records already exist, and updates them or inserts them as required.

Exercise 3: Implementing a Slowly Changing Dimension

Scenario

You have an existing SSIS package that uses a Slowly Changing Dimension transformation to load reseller dimension records into a data warehouse. You want to examine this package and then create a new one that uses a Slowly Changing Dimension transformation to load customer dimension records into the data warehouse.

The main tasks for this exercise are as follows:

1. Execute a Package to Load a Non-Changing Dimension
2. Observe a Data Flow for a Slowly Changing Dimension
3. Implement a Slowly Changing Dimension Transformation
4. Test the Package

▶ Task 1: Execute a Package to Load a Non-Changing Dimension

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab08\Starter\Ex3 folder.
2. Open the **Load Geography Data.dtsx** package and review the control flow and data flow defined in it. This package includes a simple data flow to load staged geography data into the data warehouse. Only new rows are loaded, and rows that match existing data are discarded.
3. Start debugging and observe the package execution as it loads geography data into the data warehouse. When package execution has completed, stop debugging.

▶ Task 2: Observe a Data Flow for a Slowly Changing Dimension

1. Open the **Load Reseller Data.dtsx** SSIS package.
2. Examine the data flow for the **Load Reseller Dimension** task, and note the following features:
   - The **Staged Reseller Data** source extracts data from the **Resellers** table in the **Staging** database.
   - The **Lookup Geography Key** transformation looks up the geography key for the reseller in the **DimGeography** table in the **AWDataWarehouse** database.
   - The **Reseller SCD** is a slowly changing dimension transformation that has generated the remaining transformations and destinations. You can double-click the Reseller SCD transformation to view the wizard used to configure the slowly changing dimension, and then click **Cancel** to avoid making any unintentional changes.
   - The **Reseller SCD** transformation maps the **ResellerBusinessKey** input column to the **ResellerAlternateKey** dimension column and uses it as a business key to identify existing records.
   - The **Reseller SCD** transformation treats **AddressLine1**, **AddressLine2**, **BusinessType**, **GeographyKey**, and **NumberEmployees** as historical attributes, **Phone** and **ResellerName** as changing attributes, and **YearOpened** as a fixed attribute.
3. Start debugging and observe the data flow as the dimension is loaded. When package execution is complete, stop debugging.

▶ Task 3: Implement a Slowly Changing Dimension Transformation

1. Open the **Load Customer Data.dtsx** SSIS package and view the data flow for the **Load Customer Dimension** task. Note that the data flow already contains a source named **Staged Customer Data**, which extracts customer data from the **Staging** database, and a Lookup transformation named

Lookup Geography Key, which retrieves a **GeographyKey** value from the **AWDataWarehouse** database.

2. Add a Slowly Changing Dimension transformation named **Customer SCD** to the data flow and connect the **Lookup Match Output** data flow path from the **Lookup Geography Key** transformation to the **Customer SCD** transformation.
3. Use the Slowly Changing Dimension wizard to configure the data flow to load the **DimCustomer** table in the **AWDataWarehouse** database.
   - Map input columns to dimension columns with the same name.
   - Map the **CustomerBusinessKey** input column to the **CustomerAlternateKey** dimension column, and use this column as the business key.
   - Do not map the **CurrentRecord** dimension column to any input column.
   - Specify the following slowly changing dimension columns:

Dimension Columns	Change Type
AddressLine1	Historical attribute
AddressLine2	Historical attribute
BirthDate	Changing attribute
CommuteDistance	Historical attribute
EmailAddress	Changing attribute
FirstName	Changing attribute
Gender	Historical attribute
GeographyKey	Historical attribute
HouseOwnerFlag	Historical attribute
LastName	Changing attribute
MaritalStatus	Historical attribute
MiddleName	Changing attribute
NumberCarsOwned	Historical attribute
Occupation	Historical attribute
Phone	Changing attribute
Suffix	Changing attribute
Title	Changing attribute

   - Use the **CurrentRecord** column to show current and expired records, using the value **True** for current records and **False** for expired records.
   - Do not enable inferred member support.

▶ Task 4: Test the Package

1. Debug the package and verify that all rows pass through the **New Output** data flow path. When package execution is complete, stop debugging.
2. Debug the package again and verify that no rows pass through the **New Output** data flow path, because they already exist and no changes have been made. When package execution is complete, stop debugging.
3. Use SQL Server Management Studio to execute the **Update Customers.sql** script in the **localhost** instance of the database engine. This script updates two records in the staging database, changing one customer's phone number and another's marital status.
4. In Visual Studio, debug the package again and verify that one row passes through the **Historical Attribute Inserts Output** data flow path, and another passes through the **Changing Attributes Updates Output**. When package execution is complete, stop debugging.
5. Close Visual Studio.

Results: After this exercise, you should have an SSIS package that uses a Slowly Changing Dimension transformation to load data into a dimension table.

Exercise 4: Using the MERGE Statement

Scenario

Your staging database is located on the same server as the data warehouse and you want to take advantage of this colocation of data and use the MERGE statement to insert and update staged data into the Internet sales fact table. An existing package already uses this technique to load data into the reseller sales fact table.

The main tasks for this exercise are as follows:

1. Examine a Control Flow that uses the MERGE Statement
2. Create a Package that Uses the MERGE Statement
3. Test the Package

▶ Task 1: Examine a Control Flow that uses the MERGE Statement

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab08\Starter\Ex4 folder. Then open the **Load Reseller Sales Data.dtsx** SSIS package.
2. Examine the configuration of the **Merge Reseller Sales** task and note the following details:
   - The task uses the **localhost.Staging** connection manager to connect to the **Staging** database.
   - The task executes a Transact-SQL MERGE statement that retrieves reseller sales and related dimension keys from the **Staging** and **AWDataWarehouse** databases. It then matches these records with the **FactResellerSales** table based on the **SalesOrderNumber** and **SalesOrderLineNumber** columns, updates rows that match, and inserts new records for rows that do not.
3. Start debugging to run the package and load the reseller data. When package execution is complete, stop debugging.

▶ Task 2: Create a Package that Uses the MERGE Statement

1. Add a new SSIS package named **Load Internet Sales Data.dtsx**.

2. Add an Execute SQL Task named **Merge Internet Sales Data** to the control flow of the **Load Internet Sales Data.dtsx** package.
3. Configure the **Merge Internet Sales Data** task using the **localhost.Staging** connection manager and execute a MERGE statement that retrieves Internet sales and related dimension keys from the **Staging** and **AWDataWarehouse** databases. It also matches these records with the **FactInternetSales** table based on the **SalesOrderNumber** and **SalesOrderLineNumber** columns, updates rows that match, and inserts new records for rows that do not. To accomplish this, you can use the code in the **Merge Internet Sales.sql** script file in the D:\Labfiles\Lab08\Starter\Ex4 folder.

▶ Task 3: Test the Package

1. View the control flow tab and start debugging the package, observing the execution of the **Merge Internet Sales Data** task. When execution is complete, stop debugging.
2. Close Visual Studio.

Results: After this exercise, you should have an SSIS package that uses an Execute SQL task to execute a MERGE statement that inserts or updates data in a fact table.

Module Review and Takeaways

In this module, you have learned how to plan and implement a solution for loading data into a data warehouse.

Review Question(s)

Question: What should you consider when deciding whether or not to use the MERGE statement to load staging data into a data warehouse*?*

Module 9

Enforcing Data Quality

Contents:

Module Overview

Ensuring the high quality of data is essential if the results of data analysis are to be trusted. SQL Server 2014 includes Data Quality Services (DQS) to provide a computer-assisted process for cleansing data values, as well as identifying and removing duplicate data entities. This process reduces the workload of the data steward to a minimum while maintaining human interaction to ensure accurate results.

Objectives

After completing this module, you will be able to:

- Describe how DQS can help you manage data quality.
- Use DQS to cleanse your data.
- Use DQS to match data.

Lesson 1
Introduction to Data Quality

Data quality is a major concern for anyone building a data warehousing solution. In this lesson, you will learn about the kinds of data quality issue that must be addressed in a data warehousing solution, and how SQL Server Data Quality Services can help.

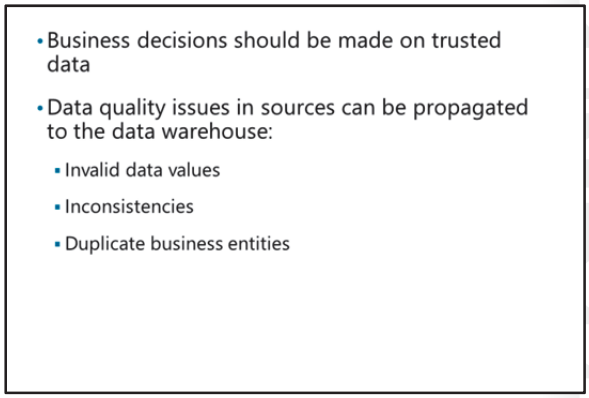
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the need for data quality management.
- Describe the features and components of DQS.
- Describe the features of a knowledge base.
- Describe the features of a domain.
- Explain how reference data can be used in a knowledge base.
- Create a DQS knowledge base.

What Is Data Quality, and Why Do You Need It?

As organizations consume more data from more sources, the need for data quality management has become increasingly common. Data quality is especially important in a Business Intelligence (BI) solution, because the reports and analysis generated from data in the data warehouse can form the basis of important decisions. Business users must be able to trust the data they use to make these decisions.



Data Quality Issues

Common data quality issues include:

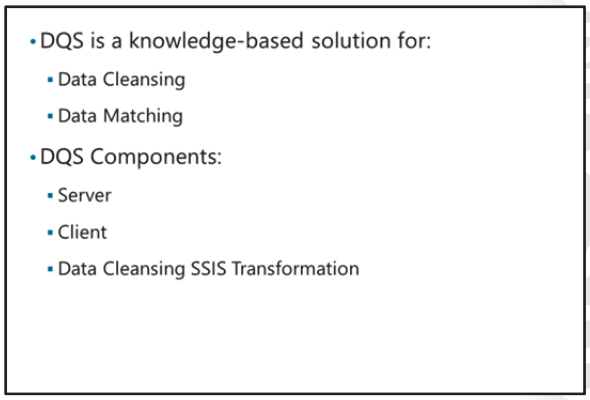
- **Invalid data values** – for example, an organization might categorize its stores as "wholesale" or "retail". However, a user might have an application that allows free-form data entry to create a store with a category of "reseller" instead of "retail", or they might accidentally type "whalesale" instead of "wholesale". Any analysis or reporting that aggregates data by store type will then produce inaccurate results because of the additional, invalid categories.
- **Inconsistencies** – for example, an organization might have one application for managing customer accounts in which US states are stored using two-letter codes (such as "WA" for Washington), and a second application that stores supplier addresses with a full state name (such as "California"). When data from both these systems is loaded, your data warehouse will contain inconsistent values for states.

- **Duplicate business entities** – for example, a customer relationship management system might contain records for Jim Corbin, Jimmy Corbin, James Corbin, and J Corbin. If the address and telephone number for these customers are all the same, it might be reasonable to assume that all the records relate to the same customer. Of course, it's also possible that Jim Corbin has a wife named Jennifer and a son called James, so you must be confident that you have matched the records appropriately before deduplicating the data.

Data Quality Services Overview

DQS is a knowledge-based solution for managing data quality. With DQS, you can perform the following kinds of data quality management:

- **Data Cleansing** – identifying invalid or inconsistent data values and correcting them.
- **Data Matching** – finding duplicate data entities.

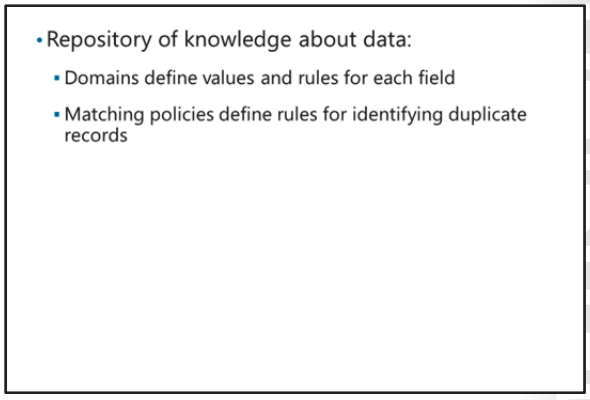


DQS is installed from the SQL Server 2014 installation media, and consists of the following components:

- **Data Quality Services Server** – a service that uses a knowledge base to apply data quality rules to data. The server must be installed on the same instance as the data that you wish to analyze. Two SQL Server catalogs are installed, allowing you to monitor, maintain, back up, and perform other administrative tasks from within SQL Server Management Studio. DQS_MAIN includes stored procedures, the DQS engine, and published knowledge bases. DQS_ PROJECT includes data required for knowledge base management and data quality project activities.
- **Data Quality Client** – a wizard-based application that data stewards (typically business users) can use to create and manage data quality services knowledge bases and perform data quality services tasks. The client can either be installed on the same computer as the DQS server or used remotely.
- **Data Cleansing SSIS Transformation** – a data flow transformation for SQL Server Integration Services (SSIS) that you can use to cleanse data as it passes through a data flow pipeline.

What Is a Knowledge Base?

DQS enables you to improve data quality by creating a knowledge base about the data, and then applying the rules it contains to perform data cleansing and matching. A knowledge base stores all the knowledge related to a specific aspect of the business. For example, you could maintain one knowledge base for a customer database and another for a product database.



Each knowledge base contains:

- Domains that define valid values and correction rules for data fields.
- Matching policies that define rules for identifying duplicate data entities.

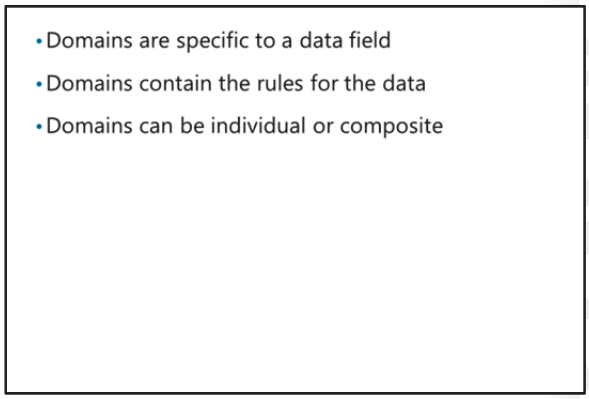
Knowledge bases are usually created and maintained by data stewards, who are often business users with particular expertise in a specific area.

DQS provides a basic knowledge base that includes domains for US address data, such as states and cities. You can use it to learn about data quality services and as a starting point for your own knowledge bases.

What Is a Domain?

Domains are central to a DQS knowledge base. Each domain identifies the possible values and rules for a data field (that is, a column in a dataset). The values for each domain are categorized as:

- **Valid** – for example, valid values for a US State domain might include "California" or "CA".
- **Invalid** – for example, invalid values for a US State domain might include "8".
- **Error** – for example, a common error for a US State domain might be "Calfornia" (with a missing "i").



Values can be grouped as synonyms. For example, you might group "California", "CA", and "Calfornia" as synonyms for California. You can specify a leading value to which all synonyms should be corrected. For example, you could configure the domain so that instances of "CA" and "Calfornia" are automatically corrected to "California".

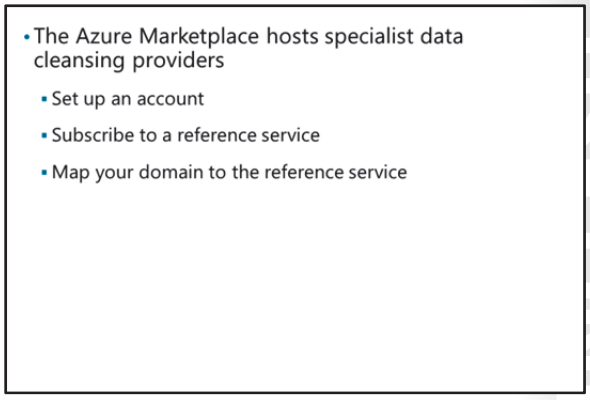
In addition to defining the values for a domain, you can create domain rules that validate new data values. For example, you could create a rule to ensure that all values in an Age domain are numbers or that all values in an Email Address domain include a "@" character. You can also specify standardization settings for a text-based domain to enforce correct capitalization. This enables you to ensure that cleansed text values have consistent formats.

Often, you can create domains to represent the most granular level of your data, for example First Name, but the actual unit of storage comprises multiple domains, for example Full Name. In this example, you can combine the First Name and Last Name domains to form a Full Name composite domain. Composite domains are also used for address fields comprising of a combination of address, city, state, postal code, and country data. Another use of composite domains is a rule that combines data from multiple domains. For example, you can verify that the string "98007" in a Postal Code domain corresponds to the string "Bellevue" in a city domain.

Matching can be performed on the individual domains that comprise the composite domain, but not on the composite domain itself.

What Is a Reference Data Service?

Many data quality problems are outside the core specialization of the organization in which they are being used. For example, your organization might be an Internet retailer that is shipping goods based on incorrect address data, a core data problem that produces high unnecessary costs. You may have made your website as user-friendly as possible, but there might still be an unacceptably high number of incorrectly-addressed orders.



To cleanse data that is outside the knowledge of your organization, you can subscribe to third-party Reference Data Service (RDS) providers. Using the Windows Azure Data Market, it is straightforward to subscribe to an RDS service and use it to validate and cleanse your data.

To continue the example, using the Windows Azure Data Market, you could purchase a subscription to an address verification service. You can then send data there for it to be verified and cleansed, reducing incorrect address information and, therefore, cutting down your postage costs.

To use RDS to cleanse your data, you must follow these steps:

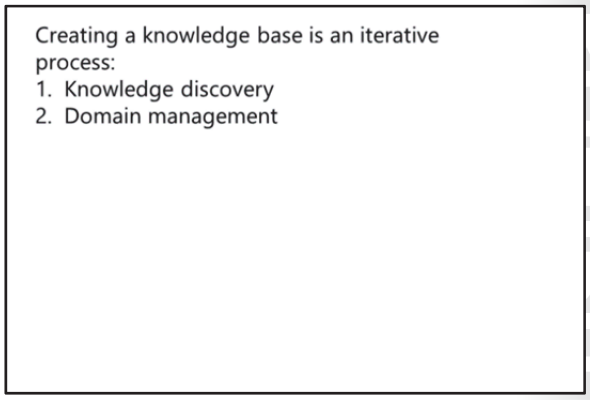
1. Create a free account key at the Windows Azure Marketplace.
2. Subscribe to a free or paid-for RDS provider's service at the Marketplace.
3. Configure the reference data service details in DQS.
4. Map your domain to the RDS service.
5. Finally, you can use the knowledge base containing the domain that maps to the RDS service to cleanse the data.

One of the key advantages of using the Azure Data Market to provide DQS services is that the cost of the data service is typically based on the number of times you use it per month. This allows you to scale up at busy times and reduce costs when the business is quieter.

Creating a Knowledge Base

Building a DQS knowledge base is an iterative process that involves the following steps:

1. Knowledge Discovery – using existing data to identify domain values.
2. Domain Management – categorizing discovered values as valid, invalid, or errors, specifying synonyms and leading values, defining correction rules, and other domain configuration tasks.



The data steward can create the initial knowledge base from scratch, base it on an existing one, or import one from a data file. The knowledge discovery process is then used to identify data fields that need to be managed, map these fields to domains in the knowledge base (which can be created during knowledge discovery if required), and identify values for these fields.

After the knowledge base has been populated by the knowledge discovery process, the data steward manages the domains to control how DQS validates and corrects data values. Additionally, domain management may include configuring reference data services, or setting up term-based or cross-field relationships.

Creating a knowledge base is not a one-time activity. A data steward will continually use the knowledge discovery and domain management processes to enhance the knowledge base and manage the quality of new data values and domains.

Demonstration: Creating a Knowledge Base

In this demonstration, you will see how to create a DQS knowledge base.

Demonstration Steps

Create a Knowledge Base

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and log into the 20463C-MIA-SQL virtual machine as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**. Then, in the D:\Demofiles\Mod09 folder, right-click **Setup.cmd** and click **Run as administrator**. When prompted, click **Yes**.
2. On the task bar, click **SQL Server 2014 Data Quality Client**. When prompted, enter the server name **MIA-SQL**, and click **Connect**.
3. In SQL Server Data Quality Services, in the **Knowledge Base Management** section, click **New Knowledge Base** and create a knowledge base named **Demo KB** from the existing **DQS Data** one. Select the **Domain Management** activity, and click **Next**.
4. Select the **US - State** domain. Then, on the **Domain Properties** tab, change the domain name to **State**.
5. On the **Domain Values** tab, note the existing values. The leading value for each state is the full state name. Other possible values that should be corrected to the leading value are indented beneath each leading value.
6. Click **Finish**, and then when prompted to publish the knowledge base, click **No**.

Perform Knowledge Discovery

1. In SQL Server Data Quality Services, under **Recent Knowledge Base**, click **Demo KB** and then click **Knowledge Discovery**.
2. On the **Map** page, in the **Data Source** drop-down list, select **Excel File**, in the **Excel File** box, browse to **D:\Demofiles\Mod09\Stores.xls**. In the **Worksheet** drop-down list, ensure **Sheet1$** is selected, and ensure **Use first row as header** is selected. This worksheet contains a sample of store data that needs to be cleansed.
3. In the **Mappings** table, in the **Source Column** list, select **State (String)**, and in the **Domain** list, select **State**.
4. In the **Mappings** table, in the **Source Column** list, select **City (String)**, and then click the **Create a domain** button and create a domain named **City** with the default properties.

5. Repeat the previous step to map the **StoreType (String)** source column to a new domain named **StoreType**.
6. Click **Next**, and then on the **Discover** page, click **Start** and wait for the knowledge discovery process to complete. When the process has finished, note that 11 new **City** and **StoreType** records were found and that there were three unique **City** values, five unique **State** values, and four unique **StoreType** values. Then click **Next**.
7. On the **Manage Domain Values** page, with the **City** domain selected, note the new values that were discovered.
8. Select the **State** domain and note that no new values were discovered. Then clear the **Show Only New** checkbox and note that all possible values for the **State** domain are shown, and the **Frequency** column indicates that the data included **California**, **CA**, **Washington**, and **WA**.
9. Select the **StoreType** domain and note the values discovered.
10. In the list of values, click **Retail**, hold the Ctrl key and click **Resale**, and click the **Set selected domain values as synonyms** button. Then right-click **Retail** and click **Set as Leading**.
11. In the list of values, note that **Whalesale** has a red spelling checker line. Then right-click **Whalesale**, and click **Wholesale** in the list of suggested spelling corrections. Note that the **Type** for the **Whalesale** value changes to **Error** and the **Correct to** value is automatically set to **Wholesale**.
12. Click **Finish**. If prompted to review more values, click **No**. When prompted to publish the knowledge base, click **No**.

Perform Domain Management

1. In SQL Server Data Quality Services, under **Recent Knowledge Base**, click **Demo KB**, and then click **Domain Management**.
2. In the **Domain** list, select **StoreType**. Then view the **Domain Values** tab and note that the values discovered in the previous task are listed with appropriate leading values and correction behavior.
3. Click the **Add new domain value** button, and then enter the value **Reseller**.
4. Click the **Retail** leading value, hold the Ctrl key and click the new **Reseller** value. Click the **Set selected domain values as synonyms** button (which, depending on the screen resolution, may be in a drop-down list at the end of the toolbar above the table). Note that **Reseller** becomes a valid value that is corrected to the **Retail** leading value.
5. Click **Finish**, and when prompted to publish the knowledge base, click **Publish**. Then, when publishing is complete, click **OK**.

Lesson 2
Using Data Quality Services to Cleanse Data

One of the major tasks for a data quality management solution is to cleanse data by validating and correcting domain values. This lesson describes how you can use DQS to cleanse data and review the results.

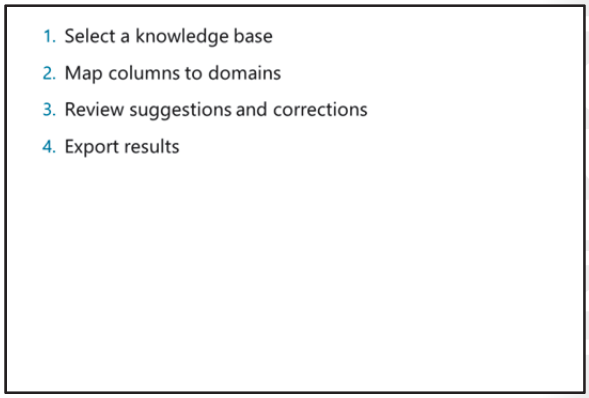
Lesson Objectives

After completing this lesson, you will be able to:

- Create a data cleansing project.
- View cleansed data.
- Use the Data Cleansing transformation in an SSIS data flow.

Creating a Data Cleansing Project

Data stewards can use the Data Quality Client application to create a data cleansing project that applies the knowledge in a knowledge base to data in an SQL Server database or an Excel workbook.

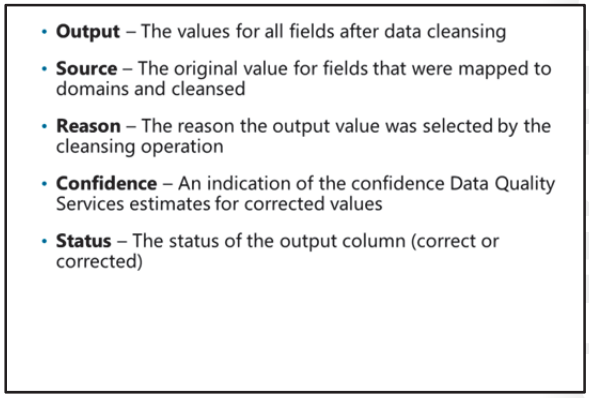


When creating a data cleansing project, the data steward must:

1. Select the knowledge base to use and specify that the action to be performed is cleansing.
2. Select the source containing the data to be cleansed and map the columns in it to the domains in the knowledge base.
3. Run the data cleansing process, and then review the suggestions and corrections generated by DQS. The data steward can then approve or reject the suggestions and corrections.
4. Export the cleansed data to a database table, comma-delimited file, or Excel workbook.

Viewing Cleansed Data

The output from a data cleansing project includes the cleansed data as well as additional information about the corrections made by DQS. The output columns are named by combining the name of the domain and the type of data. For example, the cleansed output for a domain named **State** is stored in a column named **State_Output**.



Cleansed data output includes the following column types:

- **Output** – the values for all fields after data cleansing. All fields in the original data source

generate output columns, even those not mapped to domains in the knowledge base (in which case they contain the original data values).

- **Source** – the original value for fields that were mapped to domains and cleansed.
- **Reason** – the reason the output value was selected by the cleansing operation. For example, a valid value might be corrected to a leading value defined for the domain, or DQS might have applied a cleansing algorithm and suggested a corrected value.
- **Confidence** – an indication of the confidence DQS estimates for corrected values. For values corrected to leading values defined in the knowledge base, this is usually 1 (or 100%). When DQS uses a cleansing algorithm to suggest a correction, the confidence is a value between 0 and 1.
- **Status** – the status of the output column. A value of "correct" indicates that the original value was already correct, and a value of "corrected" indicates that DQS changed the value.

Demonstration: Cleansing Data

In this demonstration, you will see how to use DQS to cleanse data.

Demonstration Steps

Create a Data Cleansing Project

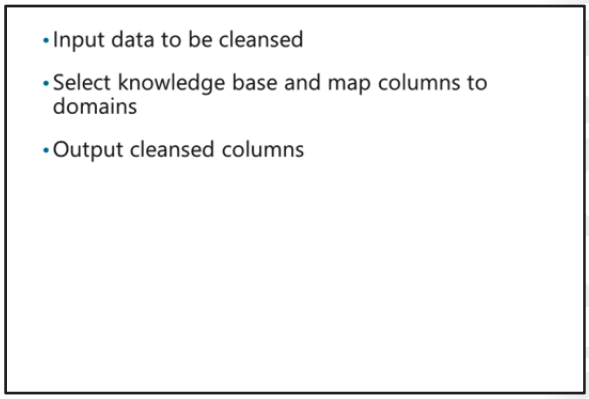
1. Ensure you have completed the previous demonstration in this module.
2. If it is not already running, start the SQL Server 2014 Data Quality Client application, and connect to **MIA-SQL**.
3. In SQL Server Data Quality Services, in the **Data Quality Projects** section, click **New Data Quality Project**, and create a new project named **Cleansing Demo** based on the **Demo KB** knowledge base. Ensure the **Cleansing** activity is selected, and then click **Next**.
4. On the **Map** page, in the **Data Source** list, ensure **SQL Server** is selected. Then in the **Database** list, click **DemoDQS**, and in the **Table/View** list, click **Stores**.
5. In the **Mappings** table, map the **City (varchar)**, **State (varchar)**, and **StoreType (varchar)** source columns to the **City**, **State**, and **StoreType** domains. Then click **Next**.
6. On the **Cleanse** page, click **Start**. Then, when the cleansing process has completed, view the data in the **Profiler** tab, noting the number of corrected and suggested values for each domain, and click **Next**.
7. On the **Manage and View Results** page, ensure that the **City** domain is selected, and on the **Suggested** tab, note that **DQS** has suggested correcting the value **New Yrk** to **New York**. Click the **Approve** option to accept this suggestion, and then click the **Corrected** tab to verify that the value has been corrected.
8. Click the **State** and **StoreType** domains in turn, and on the **Corrected** tab, note the corrections that have been applied, based on the values defined in the knowledge base. Then click **Next**.
9. On the **Export** page, view the output data preview. Then under **Export cleansing results**, in the **Destination Type** list, select **Excel File**. In the **Excel file name** box type **D:\Demofiles\Mod09\CleansedStores.xls**, ensure that **Standardize Output** is selected, ensure that the **Data and Cleansing Info** option is selected, and click **Export**.
10. When the file download has completed, click **Close**. Then click **Finish**, and close SQL Server Data Quality Services.

View Cleansed Data

1. Open D:\Demofiles\Mod09\CleansedStores.xls in Excel.
2. Note that the output includes the following types of column:
   - **Output** – the values for all fields after data cleansing.
   - **Source** – the original value for fields that were mapped to domains and cleansed.
   - **Reason** – the reason the output value was selected by the cleansing operation.
   - **Confidence** – an indication of the confidence DQS estimates for corrected values.
   - **Status** – the status of the output column (correct or corrected).
3. Close Excel without saving any changes.

Using the Data Cleansing Data Flow Transformation

In addition to creating data cleansing projects to operate interactively, you can use the Data Cleansing transformation to work in an SSIS data flow. Using the Data Cleansing transformation enables you to automate data cleansing as a part of the extract, transform, and load (ETL) processes used to populate your data warehouse.



To add the Data Cleansing transformation to a data flow in an SSIS package, perform the following steps:

1. Add the Data Cleansing transformation to the data flow and drag a data flow connection, from a source or transformation containing the data you want to cleanse, to the input of the Data Cleansing transformation.
2. Edit the settings of the Data Cleansing transformation to connect to the data quality server, specify the knowledge base you want to use, and map the data flow input columns to domains in the knowledge base.
3. Drag the output from the Data Cleansing transformation to the next transformation or destination in the data flow, and map the output columns from the Data Cleansing transformation to the appropriate input columns. The output columns from the Data Cleansing transformation are the same as those generated by an interactive data cleansing project.

Lab A: Cleansing Data

Scenario

You have created an ETL solution for the Adventure Works data warehouse, and invited some data stewards to validate the process before putting it into production. The data stewards have noticed some data quality issues in the staged customer data, and requested that you provide a way for them to cleanse data so that the data warehouse is based on consistent and reliable data. The data stewards have provided you with an Excel workbook containing some examples of the issues found in the data.

Objectives

After completing this lab, you will be able to:

- Create a DQS knowledge base.
- Use DQS to cleanse data.
- Incorporate data cleansing into an SSIS data flow.

Lab Setup

Estimated Time: 30 Minutes

Virtual machine: 20463C-MIA-SQL

User name: ADVENTUREWORKS\Student

Password: Pa$$w0rd

The setup script for this lab does *not* delete any existing DQS knowledge bases or projects. It is recommended that students create snapshots for the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines before starting so that the lab environment can be returned to the starting point if necessary.

Exercise 1: Creating a DQS Knowledge Base

Scenario

You have integrated data from many sources into your data warehouse, and this has provided several benefits. However, users have observed some quality issues with the data, which you must correct.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. View Existing Data
3. Create a Knowledge Base
4. Perform Knowledge Discovery

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. Run Setup.cmd in the D:\Labfiles\Lab09\Starter folder as Administrator.

▶ Task 2: View Existing Data

1. Open D:\Labfiles\Lab09\Starter\Sample Customer Data.xls in Excel and examine the worksheets in the workbook.
   - Note that there are multiple names for the same country on the **CountryRegion** and **StateProvince** worksheets.
   - Note that there are multiple names for the same state on the **StateProvince** worksheets.
   - Note that some customers do not have a gender code of **F** or **M** on the **Gender** worksheet.
2. On the **Sample Customer Data** worksheet, apply column filters to explore the data further and view the source records for the anomalous data.
3. Close Excel without saving any changes to the workbook.

▶ Task 3: Create a Knowledge Base

1. Start the Data Quality Client application and connect to **MIA-SQL**.
2. Create a new knowledge base with the following properties:
   - **Name**: Customer KB.
   - **Description**: Customer data knowledge base.
   - **Create Knowledge Base From**: Existing Knowledge Base (DQS Data).
   - **Select Activity**: Domain Management.
3. View the domain values for the **Country/Region**, **Country/Region (two-letter leading)**, and **US - State** domains.
4. Change the name of the **US – State** domain to **State**.
5. Create a domain with the following properties:
   - **Domain Name**: Gender
   - **Description**: Male or female
   - **Data Type**: String
   - **Use Leading Values**: Selected
   - **Normalize String**: Selected
   - **Format Output to**: Upper Case
   - **Language**: English
   - **Enable Speller**: Selected
   - **Disable Syntax Error Algorithms**: Not selected
6. View the domain values for the **Gender** domain, and notice that null is allowed.
7. Add new domain values for **F**, **M**, **Female**, and **Male** to the **Gender** domain.
8. Set **F** and **Female** as synonyms, with **F** as the leading value.
9. Set **M** and **Male** as synonyms, with **M** as the leading value.
10. Finish editing the knowledge base, but do not publish it.

▶ Task 4: Perform Knowledge Discovery

1. Open the **Customer KB** Knowledge base for knowledge discovery.

2. Use the **Sample Customer Data$** worksheet in the **Sample Customer Data.xls** Excel workbook in D:\Labfiles\Lab09\Starter as the source data for mapping. Use the first row as the header.
3. In the **Mappings** table, select the following:

Source Column	Domain
CountryRegionCode (String)	Country/Region (two-letter leading)
CountryRegionName (String)	Country/Region
StateProvinceName (String)	State
Gender (String)	Gender

4. Start the discovery process, and when it is complete, view the new values that have been discovered for the **State** domain, and set **New South Wales** and **NSW** as synonyms with **New South Wales** as the leading value. In the alphabetically ordered list of values, click **New South Wales** first, press Ctrl key, click **NSW** to select them both, and then click **Set selected domain values as synonyms**.
5. View the new values that have been discovered for the **Country/Region (two-letter leading)** domain, and mark the value **UK** as an error that should be corrected to **GB**.
6. View the new values that have been discovered for the **Gender** domain, and mark the value **W** as invalid and correct it to **F**.
7. View the new values that have been discovered for the **Country/Region** domain, and remove the filter that causes the list to show only new values.
8. Set **United States** and **America** as synonyms with **United States** as the leading value. In the alphabetically ordered list of values, click **United States** first, press Ctrl and click **America** to select them both, and then click **Set selected domain values as synonyms**.
9. Set **United Kingdom** and **Great Britain** as synonyms with **United Kingdom** as the leading value. In the alphabetically ordered list of values, click **United Kingdom** first, press Ctrl, click **Great Britain** to select them both, and then click **Set selected domain values as synonyms**.
10. Finish and publish the knowledge base.

Results: After this exercise, you should have created a knowledge base and performed knowledge discovery.

Exercise 2: Using a DQS Project to Cleanse Data

Scenario

Now you have a published knowledge base, you can use it to perform data cleansing in a data quality project.

The main tasks for this exercise are as follows:

1. Create a Data Quality Project

▶ Task 1: Create a Data Quality Project

1. Create a new data quality project with the following properties:
   - **Name**: Cleanse Customer Data

- **Description**: Apply Customer KB to customer data
- **Use knowledge base**: Customer KB
- **Select Activity**: Cleansing

2. On the **Map** page select the **InternetSales** SQL Server database, then select the **Customers** table. Then in the **Mappings** table, select the following mappings:

Source Column	Domain
CountryRegionCode (nvarchar)	Country/Region (two-letter leading)
CountryRegionName (nvarchar)	Country/Region
StateProvinceName (nvarchar)	State
Gender (nvarchar)	Gender

3. Start the cleansing process, review the source statistics in the **Profiler** pane, and then on the **Manage and View Results** page, note that DQS has found the value **Astralia**, which is likely to be a typographical error, and suggested it be corrected to **Australia** on the **Suggested** tab of the **Country** domain.
4. Approve the suggested correction, and note that it is now listed on the **Corrected** tab. Then view the corrected values for the **Country/Region**, **Country/Region (two-letter leading)**, **Gender**, and **State** domains.
5. On the **Export** page, view the output data, and then export the data and cleansing info to an Excel file named **D:\Labfiles\Lab09\Starter\CleansedCustomers.xls**.
6. When the export is complete, finish the project and view the results of the cleansing process in Excel.

Results: After this exercise, you should have used a DQS project to cleanse data and export it as an Excel workbook.

Exercise 3: Using DQS in an SSIS Package

Scenario

You are happy with the data cleansing capabilities of DQS and the results are accurate enough to be automated. You will edit an SSIS package to include a data cleansing component as part of a dataflow.

The main tasks for this exercise are as follows:

1. Add a DQS Cleansing Transformation to a Data Flow

2. Test the Package

▶ Task 1: Add a DQS Cleansing Transformation to a Data Flow

1. Open the D:\Labfiles\Lab09\Starter\AdventureWorksETL.sln solution in Visual Studio.
2. Open the **Extract Internet Sales Data.dtsx** SSIS package and, if it is not already visible, display the SSIS Toolbox (which is available on the **SSIS** menu).
3. Open the **Extract Customers** data flow task, add a **DQS Cleansing** transformation, and rename it to **Cleanse Customer Data**.
4. Remove the data flow between **Customers** and **Staging DB** and add a data flow from **Customers** to **Cleanse Customers**.
5. Configure the following settings for **Cleanse Customer Data**:
   - Create a new Data Quality connection manager for the **MIA-SQL** server and the **Customer KB** knowledge base.
   - Specify the following mapping with the default source, output, and status alias values:

Input Column	Domain
Gender	Gender
StateProvinceName	State
CountryRegionCode	Country/Region (two-letter leading)
CountryRegionName	Country/Region

6. Standardize the output.
7. Connect the output data flow from **Cleanse Customer Data** to **Staging DB** and change the following column mappings in the **Staging DB** destination (leaving the remaining existing mappings as they are):

Input Column	Destination Column
Gender_Output	Gender
StateProvinceName_Output	StateProvinceName
CountryRegionCode_Output	CountryRegionCode
CountryRegionName_Output	CountryRegionName

▶ Task 2: Test the Package

1. Debug the package and observe the **Extract Customers** data flow as it executes, noting the number of rows processed by the **Cleanse Customer Data** transformation.
2. When package execution has completed, stop debugging and close Visual Studio (note that the entire package may take some time to complete after the **Extract Customers** data flow has finished).

Results: After this exercise, you should have created and tested an SSIS package that cleanses data.

Lesson 3
Using Data Quality Services to Match Data

As well as cleansing data, you can use DQS to identify duplicate data entities. The ability to match data entities is useful when you need to deduplicate data to eliminate errors in reports and analysis caused by the same entity being counted more than once.

This lesson explains how to create a matching policy, and then use it to find duplicate data entities in a data matching project.

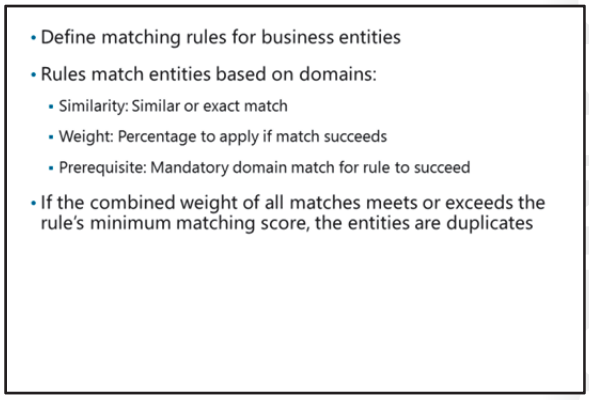
Lesson Objectives

After completing this lesson, you will be able to:

- Create a matching policy.
- Create a data matching project.
- View data matching results.

Creating a Matching Policy

A data warehouse is almost always composed of data from multiple sources and at least some of this is often provided by third parties. Also, there are likely to be many transactions relating to the same customer or product, but unless you have a system which only allows existing customers to buy existing products, duplication is likely.



For example, suppose you sell books on the Internet. To buy a book, a customer must first register with their name, address, username, and password. Customers often forget their details and, although there is a Forgotten Username/Password link, many choose to register again. You can implement a constraint to stop anyone having the same username, but there may still be many instances when a single customer registers two or more times. This duplication can cause problems for data analysis because you will have more customers in your system than in reality. There might be more of a particular gender, more between certain age brackets, or more in a particular geographic area, than occur in reality. The duplicate entries will also affect sales per customer analysis, which will return lower-than-accurate results.

By providing a constraint to prevent duplicate usernames, you have gone some way to preventing duplication, but as you can see from the example, this will only slightly reduce the problem. You could enforce unique names, but that would prevent customers with common names. You could enforce unique names at the same address, but that would block someone with the same name as a partner or child.

Matching Policies

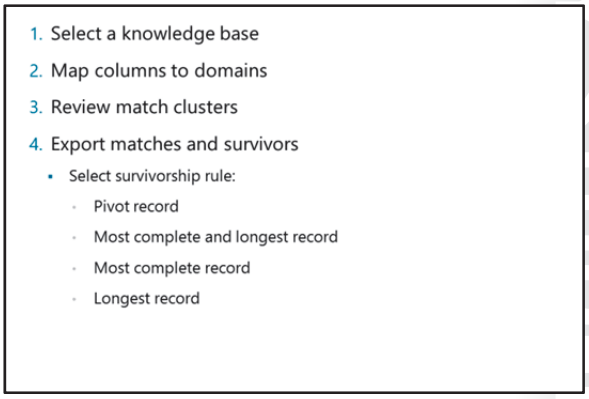
DQS can use a matching policy to assess the likelihood of records being duplicates. In cases with a high likelihood of duplication, the potential duplicates are assessed by a data steward before any changes are made. A data steward can add a matching policy to a knowledge base and create rules that help determine whether multiple data records represent the same business entity. A data matching rule compares one or more domains across records and applies weighted comparisons to identify matches. For each domain in the matching rule, the data steward defines the following settings:

- **Similarity** – you can specify that the rule should look for similar values based on fuzzy logic comparison, or an exact match.
- **Weight** – a percentage score to apply if a domain match succeeds.
- **Prerequisite** – indicates that this particular domain must match for the records to be considered duplicates.

For each rule, the data steward specifies a minimum matching score. When the matching process occurs, the individual weightings for each successful domain match comparison are added together. If the total is equal to or greater than the minimum matching score, and all prerequisite domains match, the records are considered to be duplicates.

Creating a Data Matching Project

Data stewards can use the Data Quality Client application to create a data matching project that applies the knowledge in a knowledge base to data in an SQL Server database or an Excel workbook.



When creating a data matching project, the data steward must:

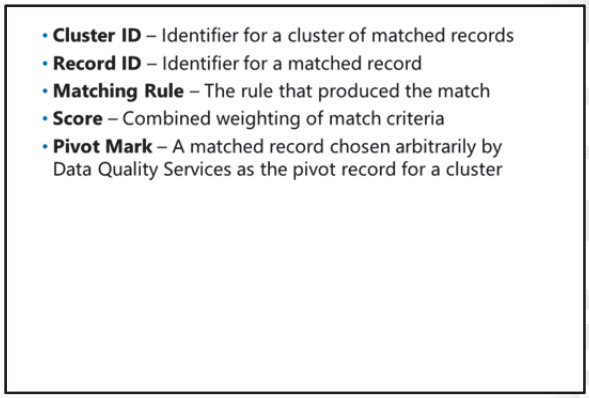
1. Select the knowledge base to use and specify that the action to be performed is matching.
2. Select the data source containing the data to be matched and map the columns in it to the knowledge base domains.
3. Run the data matching process then review the clusters of matched records that DQS identifies, based on the knowledge base matching policies.
4. Export the matched data to a database table, comma-delimited file, or Excel workbook. Additionally, you can specify a survivorship rule that eliminates duplicate records and exports the surviving records. You can specify the following rules for survivorship:
    - **Pivot record** – a record chosen arbitrarily by DQS in each cluster of matched records.
    - **Most complete and longest record** – the record that has fewest missing data values and the longest values in each field.
    - **Most complete record** – the record that has fewest missing data values.
    - **Longest record** – the record containing the longest values in each field.

Viewing Data Matching Results

After the data matching process is complete, you can view and export the following results:

- **Matches** – the original dataset plus additional columns that indicate clusters of matched records.
- **Survivors** – the resulting dataset, with duplicate records eliminated, based on the selected survivorship rule.

When you export matches, the results include the original data and the following columns:



- **Cluster ID** – a unique identifier for a cluster of matched records.
- **Record ID** – a unique identifier for each matched record.
- **Matching Rule** – the rule that produced the match.
- **Score** – the combined weighting of the matched domains as defined in the matching rule.
- **Pivot Mark** – a matched record chosen arbitrarily by DQS as the pivot record for a cluster.

Demonstration: Matching Data

In this demonstration, you will see how to use DQS to match data.

Demonstration Steps

Create a Matching Policy

1. Ensure you have completed the previous demonstrations in this module.
2. Start the SQL Server 2014 Data Quality Client application, and connect to **MIA-SQL**.
3. In SQL Server Data Quality Services, under **Recent Knowledge Base**, click **Demo KB**, and then click **Matching Policy**.
4. On the **Map** page, in the **Data Source** drop-down list, select **Excel File**, in the **Excel File** box, browse to **D:\Demofiles\Mod09\Stores.xls**. In the **Worksheet** drop-down list, ensure **Sheet1$** is selected, and ensure **Use first row as header** is selected. This worksheet contains a sample of store data that needs to be matched.
5. In the **Mappings** table, map the **City (String)**, **State (String)**, and **StoreType (String)** source columns to the **City**, **State**, and **StoreType** domains.
6. In the **Mappings** table, in the **Source Column** list, select **PhoneNumber (String)**, and then click the **Create a domain** button and create a domain named **PhoneNumber** with the default properties.
7. Repeat the previous step to map the **StreetAddress (String)** source column to a new domain named **StreetAddress**.
8. Click the **Add a column mapping** button to create a new row in the mapping table, and then repeat the previous step to map the **StoreName(String)** source column to a new domain named **StoreName**. Then click **Next**.

9. On the **Matching Policy** page, click the **Create a matching rule** button. Then in the **Rule Details** section, change the rule name to **Is Same Store**.
10. In the **Rule Editor** table, click the **Add a new domain element** button. Then in the **Domain** column, ensure that **StoreName** is selected. In the **Similarity** column, ensure that **Similar** is selected, in the **Weight** column, enter **20**, and leave the **Prerequisite** column unselected.
11. Repeat the previous steps to add the following rules:
    - **StreetAddress**: Similar: 20%: Not a prerequisite
    - **City**: Exact: 20%: Not a prerequisite
    - **PhoneNumber**: Exact: 30%: Not a prerequisite
    - **StoreType**: Similar: 10%: Not a prerequisite
    - **State**: Exact: 0%: Prerequisite selected
12. Click **Start**, wait for the matching process to complete, and note that one match is detected in the sample data (Store 1 is the same as Store One). Then click **Next**.
13. On the **Matching Results** page, view the details in the **Profiler** tab, and then click **Finish**. When prompted to publish the knowledge base, click **Publish**, and when publishing is complete, click **OK**.

Create a Data Matching Project

1. In SQL Server Data Quality Services, in the **Data Quality Projects** section, click **New Data Quality Project**, and create a new project named **Matching Demo** based on the **Demo KB** knowledge base. Ensure the **Matching** activity is selected, and then click **Next**.
2. On the **Map** page, in the **Data Source** list, ensure **SQL Server** is selected. Then in the **Database** list, click **DemoDQS**, and in the **Table/View** list, click **Stores**.
3. In the **Mappings** table, map the **City (varchar)**, **PhoneNumber (varchar)**, **State (varchar)**, **StoreName (varchar)**, **StoreType (varchar)**, and **StreetAddress (varchar)** source columns to the **City**, **PhoneNumber**, **State**, **StoreName**, **StoreType**, and **StreetAddress** domains. Then click **Next**.

   Note: When the **Mappings** table is full, click **Add a column mapping** to add an additional row.
4. On the **Matching** page, click **Start**, and when matching is complete, note that two matches were detected (Store 1 is the same as Store One and Store 16 is the same as Store Sixteen). Then click **Next**.
5. On the **Export** page, in the **Destination Type** drop-down list, select **Excel File**. Then select the following content to export:
    - **Matching Results**: D:\Demofiles\Mod09\MatchedStores.xls
    - **Survivorship Results**: D:\Demofiles\Mod09\SurvivingStores.xls
6. Select the **Most complete record** survivorship rule, and click **Export**. Then when the export has completed successfully, click **Close**.
7. Click **Finish** and close SQL Server Data Quality Services.

View Data Matching Results

1. Open D:\Demofiles\Mod09\MatchedStores.xls in Excel. Note this file contains all the records in the dataset with additional columns to indicate clusters of matched records. In this case, there are two clusters, each containing two matches.
2. Open D:\Demofiles\Mod09\SurvivingStores.xls in Excel. Note this file contains the records that were selected to survive the matching process. The data has been deduplicated by eliminating duplicates and retaining only the most complete record.

3. Close Excel without saving any changes.

Lab B: Deduplicating Data

Scenario

You have created a DQS knowledge base and used it to cleanse customer data. However, data stewards are concerned that the staged customer data may include duplicate entries. For records to be considered a match, the following criteria must be true:

- The **Country/Region** column must be an exact match.
- A total matching score of 80 or higher based on the following weightings must be achieved:
  - An exact match of the **Gender** column has a weighting of 10.
  - An exact match of the **City** column has a weighting of 20.
  - An exact match of the **EmailAddress** column has a weighting of 30.
  - A similar **FirstName** column value has a weighting of 10.
  - A similar **LastName** column value has a weighting of 10.
  - A similar **AddressLine1** column value has a weighting of 20.

Objectives

After completing this lab, you will be able to:

- Add a matching policy to a DQS knowledge base.
- Use DQS to match data.

Estimated Time: 30 Minutes

Virtual machine: 20463C-MIA-SQL

User name: ADVENTUREWORKS\Student

Password: Pa$$w0rd

Exercise 1: Creating a Matching Policy

Scenario

You have implemented a data cleansing solution for data being staged. However, you have identified that the staged data contains multiple records for the same business entity. You want to use a data matching solution to deduplicate the data.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Create a Matching Policy

▶ Task 1: Prepare the Lab Environment

1. Complete the previous lab in this module.
2. Run **LabB.cmd** in the D:\Labfiles\Lab09\Starter folder as Administrator.

▶ Task 2: Create a Matching Policy

1. Start the Data Quality Client application and connect to the **MIA-SQL** server.
2. Open the **Customer KB** knowledge base for the **Matching Policy** activity.

3. Use the **Sheet1$** worksheet in the **D:\Labfiles\Lab09\Starter\Sample Staged Data.xls** Excel file as the data source for the matching policy.
4. On the **Map** page, map the columns in the Excel worksheet to the following new domains:

Source Column	Domain
FirstName (String)	A new domain named **FirstName** with a **String** data type.
LastName (String)	A new domain named **LastName** with a **String** data type.
AddressLine1(String)	A new domain named **AddressLine1** with a **String** data type.
City (String)	A new domain named **City** with a **String** data type.
EmailAddress (String)	A new domain named **EmailAddress** with a **String** data type.

5. Add more column mappings and map the following fields to existing domains:

Source Column	Domain
Gender (String)	Gender
StateProvinceName (String)	State
CountryRegionCode (String)	Country/Region (two-letter heading)
CountryRegionName (String)	Country/Region

6. On the **Matching Policy** page, create a matching rule with the following details:
   - **Rule name**: Is Same Customer
   - **Description**: Checks for duplicate customer records
   - **Min. matching score**: 80

Domain	Similarity	Weight	Prerequisite
Country/Region	Exact		Selected
Gender	Exact	10	Unselected
City	Exact	20	Unselected
EmailAddress	Exact	30	Unselected
FirstName	Similar	10	Unselected
LastName	Similar	10	Unselected
AddressLine1	Similar	20	Unselected

7. Start the matching process and, when it has finished, review the matches found by DQS, noting that there are duplicate records for three customers.
8. When you have finished, publish the knowledge base.

Results: After this exercise, you should have created a matching policy and published the knowledge base.

Exercise 2: Using a DQS Project to Match Data

Scenario

You will now create a data quality project to apply the matching rules from the previous exercise. After this process is complete, you will have exported a deduplicated set of data. You will finally apply the deduplication results in the staging database by executing Transact-SQL statements.

The main tasks for this exercise are as follows:

1. Create a Data Quality Project for Matching Data
2. Review and Apply Matching Results

▶ Task 1: Create a Data Quality Project for Matching Data

1. In SQL Server Data Quality Services, create a new data quality project with the following details:
   - **Name**: Deduplicate Customers
   - **Description**: Identify customer matches
   - **Use knowledge base**: Customer KB
   - **Select Activity**: Matching
2. Using the **Customers** table in the **Staging** SQL Server database as the data source, map the following columns to domains in the knowledge base:

Source Column	Domain
FirstName (nvarchar)	FirstName
LastName (nvarchar)	LastName
Gender (nvarchar)	Gender
AddressLine1 (nvarchar)	AddressLine1
City (nvarchar)	City
CountryRegionName (nvarchar)	Country/Region
EmailAddress (nvarchar)	EmailAddress

3. Start the matching process and review the results when it is finished.
4. Export the results to the following Excel workbooks, specifying the **Most complete record** survivorship rule:

- **Matching Results**: D:\Labfiles\Lab09\Starter\Matches.xls
- **Survivorship Results**: D:\Labfiles\Lab09\Starter\Survivors.xls

▶ Task 2: Review and Apply Matching Results

1. Open D:\Labfiles\Lab09\Starter\Matches.xls in Excel.
2. Note that the matching process found a match with a score of 90 for the following customer records:
   - CustomerBusinessKey: 29261 (Robert Turner)
   - CustomerBusinessKey: 29484 (Rob Turner)
3. Open D:\Labfiles\Lab09\Starter\Survivors.xls in Excel.
4. Note that the survivors file contains all the records that should survive de-duplication based on the matches that were found. It contains the record for customer 29261 (Robert Turner), but not for 29484 (Rob Turner).
5. Open **D:\Labfiles\Lab09\Starter\Fix Duplicates.sql** in SQL Server Management Studio, connecting to the **MIA-SQL** instance of the database engine by using Windows authentication.
6. Review the Transact-SQL code and note that it performs the following tasks:
   - Updates the **InternetSales** table so that all sales currently associated with the duplicate customer record become associated with the surviving customer record.
   - Deletes the duplicate customer record.
7. Execute the SQL statement, and then close SQL Server Management Studio.

Results: After this exercise, you should have deduplicated data using a matching project and updated data in your database to reflect these changes.

Module Review and Takeaways

In this module, you have learned how Data Quality Service provides a knowledge-based solution for cleansing and matching data.

Review Question(s)

Question: Who is responsible for ensuring the consistency and accuracy of data in solutions you have implemented or managed?

Module 10

Master Data Services

Contents:

Module Overview

Organizations typically use different platforms to store different types of data. For example, sales data might be stored in an online transactional processing (OLTP) database, customer data in a dedicated customer relations management (CRM) system, and so on. Storing data across multiple, heterogeneous platforms can make it difficult to ensure that the data representing a single instance of a specific business entity is consistent and accurate across the enterprise.

Master Data Services provides a way for organizations to standardize and improve the quality, consistency, and reliability of the data that guides key business decisions. This module introduces Master Data Services and explains the benefits of using it.

Objectives

After completing this module, you will be able to:

- Describe the key concepts of Master Data Services.
- Implement a Master Data Services model.
- Use Master Data Services tools to manage master data.
- Use Master Data Services tools to create a master data hub.

Lesson 1
Introduction to Master Data Services

Master data management can represent a major challenge for organizations. Data representations of a single business entity, such as a specific individual customer, might be recorded in multiple locations in multiple formats. When you consider the number of different types of data a company might own, the potential scale of this problem can be huge. Master Data Services enables organizations to standardize data, which improves the consistency of their key business data, and ultimately, the quality of the decisions that they make.

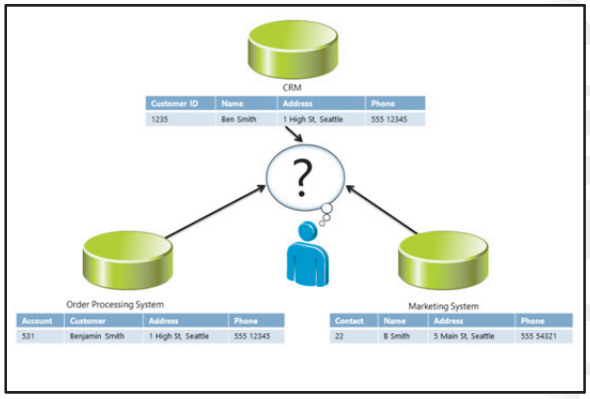
Lesson Objectives

After completing this lesson, you will be able to:

- Explain the need for master data management.
- Explain how Master Data Services helps you meet the challenges of master data management.
- Compare Master Data Services with Data Quality Services.
- Describe the components of Master Data Services.

The Need for Master Data Management

Data owned by an organization is one of its most valuable assets, and businesses rely on it for a variety of different reasons. For example, individuals in a company might use data to develop marketing strategies, plan new product lines, or identify areas where they can make efficiency savings. Report writers and data analysts interact directly with this data and decision-makers use the data to guide them when they make key choices about future business strategy. However, companies often struggle to maintain the quality, consistency, and accuracy of their data across the enterprise for a number of reasons, including:

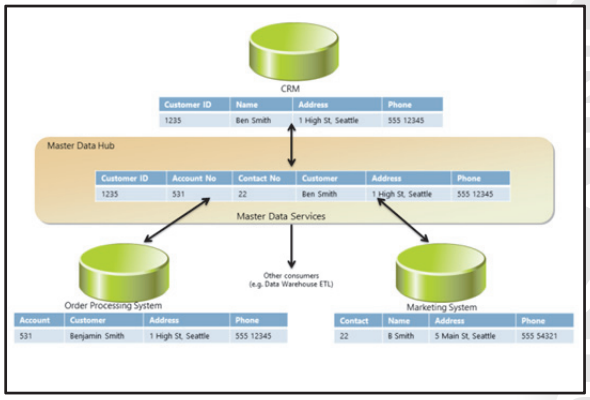


- **Decentralized data storage**. In modern, complex information ecosystems, data may be stored in multiple systems and formats, perhaps because of departmental differences or as a result of company mergers or acquisitions. This approach makes it difficult to identify where duplicate data exists, and if it does, how to identify a 'master' version.
- **Different methods of handling data changes**. Different applications might handle data changes, such as additions, updates, and deletions, by using different rules, and this can result in inconsistencies. For example, if one application records addresses without requiring a postal code but others do, the formats of stored addresses will be inconsistent.
- **Human error**. Errors in the insertion and updating of data can lead to inaccuracies. For example, if a user misspells a customer name, applications will accept the input as long as the format is correct, and the error will not be identified.
- **Latency and non-propagation of changes**. Changes to data in one system may not propagate to others where the same data is held, or it may do so with a time delay. Running reports against these diverse systems will yield different results.

Poorly managed data directly affects the quality of reporting and data analysis, and can ultimately result in inefficient procedures, missed opportunities, revenue loss, customer dissatisfaction, and increased time spent managing the data to try to solve these problems. It also makes it more difficult for organizations to comply with data regulation requirements. For these reasons, companies can realize major benefits from implementing a master data management system.

What Is Master Data Services?

Master Data Services is a master data management technology that enables organizations to handle the challenges of data management. Master Data Services can serve both as a system of entry for creating and updating master data, and a system of record for making authoritative data available to other applications. With Master Data Services, you can create a master data hub to consolidate and ensure consistency of key business entity data representations across the enterprise.



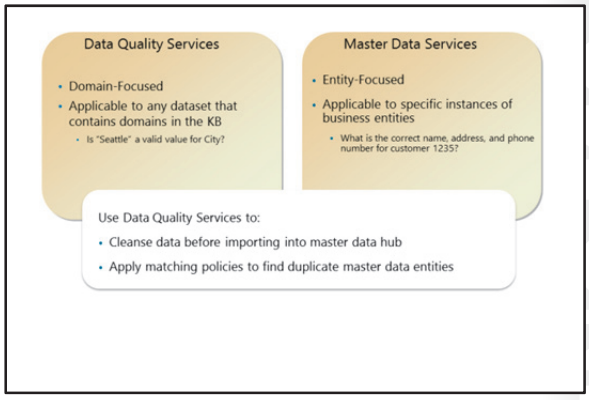
Master Data Services enables you to enforce data validation rules and track changes to master data through an audit trail that shows the time, date, and author of each change. This promotes accountability and helps organizations to comply with data regulation requirements.

Data Stewards

A data steward is an individual charged with managing master data. Usually a data steward is a business user with a detailed knowledge of the entities used in a particular area, and who is responsible for ensuring the integrity of data relating to those entities. Data stewards must ensure that each item of data is unambiguous, has a clear definition, and is used consistently across all systems. A data steward will typically use a master data management system, such as Master Data Services, to perform these tasks.

Note: Implementing a successful master data management initiative involves establishing extensive cooperation among the various stakeholders and the owners of the data, and this may not always be straightforward. For example, stakeholders may be reluctant to commit to the scheme because they see it as surrendering stewardship of their data, or there may be political issues that need to be overcome in an organization. It is therefore important that all parties fully understand the benefits of master data management and how these can be delivered by Master Data Services.

Master Data Services and Data Quality Services



On initial inspection, Master Data Services appears to address many of the same problems as Data Quality Services. Both technologies are designed to help users manage the quality and integrity of business data but there are some key differences. While Data Quality Services is focused on managing the integrity and consistency of individual domains or columns in a dataset, Master Data Services is designed to manage the integrity and consistency of specific entity instances.

For example, an organization might use Data Quality Services to ensure that all records that include address data use a consistent, approved set of valid values for **City**, **State**, and **Country** fields. This domain-based validation can apply to any record that contains address data, including customer, employee, and supplier records, and so on.

In contrast, the same organization might use Master Data Services to ensure the integrity of a specific customer named Ben Smith. Address data relating to that individual might be entered and maintained in multiple systems across the organization. Master Data Services ensures that there is definitive information about Ben Smith, enabling business users to be sure which of the multiple addresses for him that the organization has spread across various systems is the correct one. Note that, from a Data Quality Services perspective, all the addresses the organization has for Ben Smith might be *valid*, but with Master Data Services, users can be sure which one is *correct*.

In many scenarios, Master Data Services and Data Quality Services are used together. Examples of this include:

- Using Data Quality Services to cleanse data before loading into Master Data Services.
- Applying Master Data Services matching policies to identify duplicate master data records for business entities.

Components of Master Data Services

Master Data Services includes the following components:

- **Master Data Services database**. This database stores all the database objects that support Master Data Services. This database contains staging tables for processing imported data, views that enable client access to master data, and tables that store the master data itself. The database also supports additional functionality, including versioning, business rule validation, and email notification.
- **Master Data Manager Web application**. You use this application to perform the tasks associated with managing Master Data Services, such as creating models, entities, hierarchies, subscription views,

and business rules. You configure the Master Data Manager Web application by using the Explorer, Version Management, Integration Management, System Administration, and User and Group Permissions functional areas. Apart from the Explorer, all functional areas are restricted to Master Data Services administrators. Data stewards can use the Explorer functional area to manage the master data for which they have responsibility.

- **Master Data Services Configuration Manager**. You use this tool to create and manage the Master Data Services database and Web application, and to enable the Web service. You can also use it to create a Database Mail profile for Master Data Services to use.
- **Master Data Services Add-In for Microsoft Excel**. Business users and administrators can use this add-in to manage Master Data Services objects, and to work with master data in a Microsoft® Excel® workbook.

Considerations for Installing Master Data Services

Master Data Services is available only with the following editions of SQL Server 2014:

- SQL Server 2014 Business Intelligence.
- SQL Server 2014 Enterprise.
- SQL Server 2014 Developer.

You can install Master Data Services by using SQL Server Setup, and adding it as a shared feature. You should use Master Data Services Configuration Manager to complete the installation and create the Master Data Services database and Web application.

Lesson 2
Implementing a Master Data Services Model

At the center of every Master Data Services solution is a Master Data Services model. To create a master data management solution with Master Data Services, you must know how to create and manage a model. This lesson explains the key concepts you need to understand about Master Data Services models, and describes how to create and manage a model and the master data it contains.

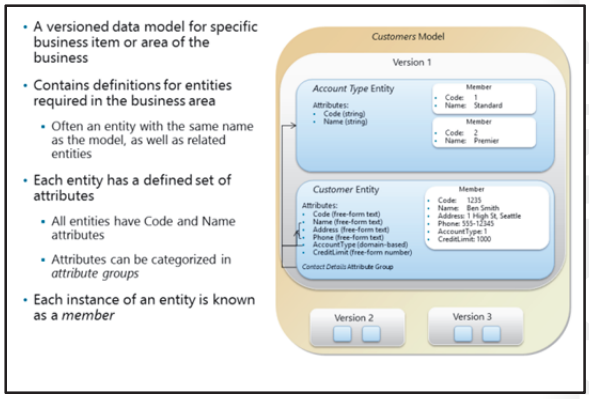
Lesson Objectives

After completing this lesson, you will be able to:

1. Describe the key features of a Master Data Services model.
2. Create a Master Data Services model.
3. Create entities and attributes in a Master Data Services model.
4. Add and edit entity members in a Master Data Services model.
5. Edit a Master Data Services model in Microsoft® Excel®.

What Is a Master Data Services Model?

A model is the highest level of organization in Master Data Services. It is a container for a related set of business entity definitions. You can create a model for each area of the business for which you want to manage data. For example, to create a definitive set of customer data, you might create a model named **Customers** that contains all customer-related data. The **Customers** model would include data and metadata about the customers themselves, and might also include related information such as customer account types and sales territory data. You might then create a second model named **Products** that contains all product data, and other models as required.



Versions

Master Data Services models are versioned, enabling you to maintain multiple versions of master data at the same time. This can be useful in scenarios where many business applications require a newer definition of a specific business entity, but some older applications cannot be upgraded to use the new model.

Entities and Attributes

An entity is a data definition for a specific type of item used in the business. For example, a **Customers** model might contain a **Customer** entity and an **Account Type** entity. An entity is analogous to a table in a relational database. In many cases, a model is created to primarily manage a single business entity, so an entity is created with the same name as the model. This scenario is so common that an option to create an identically-named entity when creating a model is available in Master Data Services.

Each entity has attributes that describe it. These attributes are analogous to columns in a database table. When you create an entity, Master Data Services automatically adds two attributes to that entity:

- **Code**. The **Code** attribute can contain only unique values. When you populate the entity, you must provide a **Code** value for each member. The value of a **Code** attribute will frequently be derived from the primary key column in a relational database table.
- **Name**. The **Name** attribute does not require a unique value, and you can leave this field blank for members if appropriate.

You cannot delete the **Code** and **Name** attributes.

In addition to the system generated attributes, you can also create one of three types of attribute that describe your data, using Master Data Services:

- **Free-form**. This type of attribute enables you to enter free-form values as text, numbers, dates, or links. You use free-form attributes for the text-based, numerical, and date and time data in your databases.
- **Domain-based**. This type of attribute accepts values only from other entities. You cannot directly enter values into domain-based attributes. You use domain-based attributes to ensure that the values for a particular attribute match **Code** values in an existing entity. For example, suppose you have an entity called **Product Category** that lists product categories by name and another entity called **Product Subcategory**. In the **Product Subcategory** entity, you can create a domain-based attribute called **Category** that references the **Product Category** entity.
- **File**. This type of attribute accepts files, such as documents or images. You can use file attributes to ensure that all files in an attribute have the same file extension.

Attribute Groups

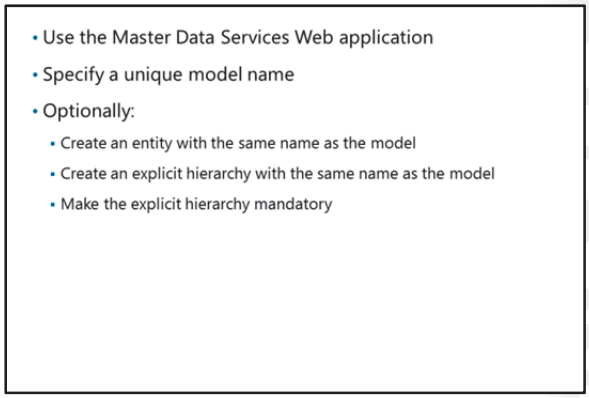
An attribute group is a named grouping of the attributes in an entity. Attribute groups are useful if an entity has a large number of attributes, which makes them difficult to view, or in scenarios where multiple applications will consume entity data from Master Data Services. However, some attributes are only relevant to particular applications. For example, a **Customer** entity might include attributes that are only used in a CRM application, and other attributes that are only used by an order processing system. By creating application-specific attribute groups, you can simplify the creation of data flows between the master data hub and the applications requiring master data.

Members

Members are individual instances of entities, and are analogous to records in a database table. Each instance of an entity is a member that represents a specific business object, such as an individual customer or product.

Creating a Model

To create a model, open the Master Data Manager Web application, and perform the following procedure:

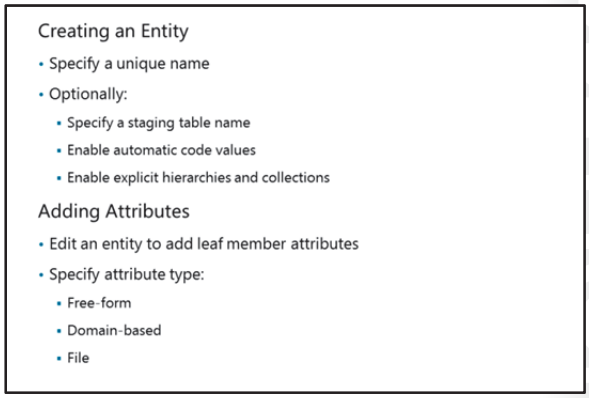


1. Click **System Administration**.
2. On the **Model View** page, on **Manage** menu, click **Models**.
3. On the **Model Maintenance** page, click **Add model**.
4. In the **Model name** box, type a unique name for the model.
5. Optionally:
   - Select **Create entity with same name as model** to create an entity with the same name as the model.
   - Select **Create explicit hierarchy with same name as model** to create an explicit hierarchy with the same name as the model. This option also enables the entity for collections.
   - Select **Mandatory hierarchy (all leaf members are included)** to make the explicit hierarchy mandatory.
6. Click **Save model**.

Note: Hierarchies and collections are discussed later in this module.

Creating Entities and Attributes

After you have created a model, you must add entities to represent the business objects for which you want to manage master data.



Creating an Entity

Use the following procedure to create an entity:

1. In Master Data Manager, click **System Administration**.
2. On the **Model View** page, on the **Manage** menu, click **Entities**.
3. On the **Entity Maintenance** page, in the **Model** list, select the model in which you want to create the entity and click **Add entity**.
4. In the **Entity name** box, type a name that is unique within the model.
5. Optionally, in the **Name for staging tables** box, type a name for the staging table (if you don't enter a name, the entity name is used by default).
6. Optionally, select the **Create Code values automatically** check box so that **Code** attribute values are generated automatically for members as they are added.

7. In the **Enable explicit hierarchies and collections** list, select **Yes** if you want to enable explicit hierarchies and collections for this entity, or **No** if you do not. If you select **Yes**, you can optionally select **Mandatory hierarchy (all leaf members are included)** to make the explicit hierarchy a mandatory one.
8. Click **Save entity**.

Adding Attributes to an Entity

After you have created an entity, it will contain the mandatory **Code** and **Name** attributes. You can then edit the entity to add more attributes by following this procedure:

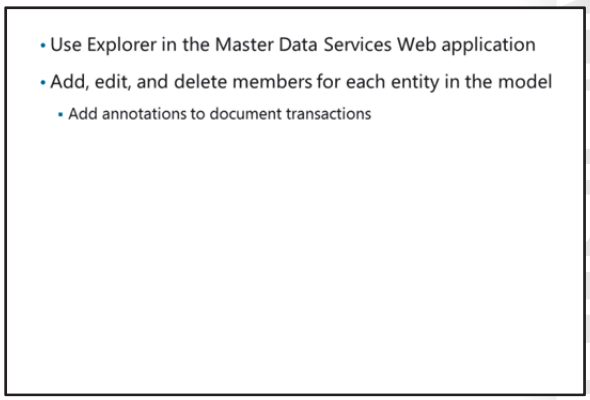
1. In Master Data Manager, click **System Administration**.
2. On the **Model View** page, on the **Manage** menu, click **Entities**.
3. On the **Entity Maintenance** page, in the **Model** list, select the model that contains the entity to which you want to add attributes.
4. Select the entity to which you want to add attributes, and click **Edit selected entity**.
5. On the **Edit Entity** page, in the **Leaf member attributes** pane, click **Add leaf attribute**.
6. On the **Add Attribute** page, select one of the following options:
   - **Free-form**. Use this option to enable users to enter attribute values.
   - **Domain-based**. Use this option to create an attribute that is used as a key to look up members in another, related entity.
   - **File**. Use this option to create an attribute that is represented by file, such as an image.
7. In the **Name** box, type a name for the attribute that is unique within the entity.
8. Depending on the option you chose previously, you can set additional display and data type settings for the attribute. For example, when adding a free-form attribute, you can specify a data type such as **Text**, **Number**, or **Date**. You can then set data type-specific constraints, such as a maximum length for a text attribute or a number of decimal places, for a numeric attribute.
9. Optionally, select **Enable change tracking** to track changes to groups of attributes.
10. Click **Save attribute**.
11. On the **Entity Maintenance** page, click **Save entity**.

Adding and Editing Members

After you have created entities and defined their attributes, you can add members to the model.

To use Master Data Manager to add a member:

1. On the Master Data Manager home page, in the **Model** list, select the model to which you want to add a member.
2. In the **Version** list, select the version of the model you want to work with.
3. Click **Explorer**.



4. On the **Entities** menu, click the name of the entity to which you want to add a member.
5. Click **Add member**.
6. In the **Details** pane, enter a value for each attribute.
7. Optionally, in the **Annotations** box, type a comment to document the addition of the member.
8. Click **OK**.

After you have added a member, you can edit its attributes by selecting it and modifying the attribute values in the **Details** pane.

Adding annotations helps document each change made to the data. You can view a history of all edits, and associated annotations by clicking the **View Transactions** button.

Demonstration: Creating a Master Data Services Model

In this demonstration, you will see how to create a Master Data Services model.

Demonstration Steps

Create a Model

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and log into the 20463C-MIA-SQL virtual machine as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**. Then, in the D:\Demofiles\Mod10 folder, right-click **Setup.cmd** and click **Run as administrator**. When prompted, click **Yes**.
2. Start Internet Explorer and browse to **http://localhost:81/MDS**.
3. On the Master Data Services home page, click **System Administration**.
4. On the **Model View** page, on the **Manage** menu, click **Models**.
5. On the **Model Maintenance** page, click **Add model**.
6. On the **Add Model** page, in the **Model name** box, type **Customers**, clear the **Create entity with same name as model** check box, and then click **Save model**.

Create an Entity

1. On the **Model Maintenance** page, on the **Manage** menu, click **Entities**.
2. On the **Entity Maintenance** page, in the **Model** list, select **Customers**, and then click **Add entity**.
3. On the **Add Entity** page, in the **Entity name** box, type **Customer**.
4. In the **Enable explicit hierarchies and collections** list, click **No**, and then click **Save entity**.

Create Attributes

1. On the **Entity Maintenance** page, in the **Entity** table, click **Customer**, and then click **Edit selected entity**.
2. On the **Edit Entity: Customer** page, in the **Leaf member attributes** area, click **Add leaf attribute**.
3. On the **Entity: Customer Add Attribute** page, ensure that the **Free-form** option is selected.
4. In the **Name** box, type **Address**, in the **Data type** list, select **Text**, in the **Length** box, type **400**, and then click **Save attribute**.
5. On the **Edit Entity: Customer** page, in the **Leaf member attributes** area, click **Add leaf attribute**.
6. On the **Entity: Customer Add Attribute** page, ensure that the **Free-form** option is selected.

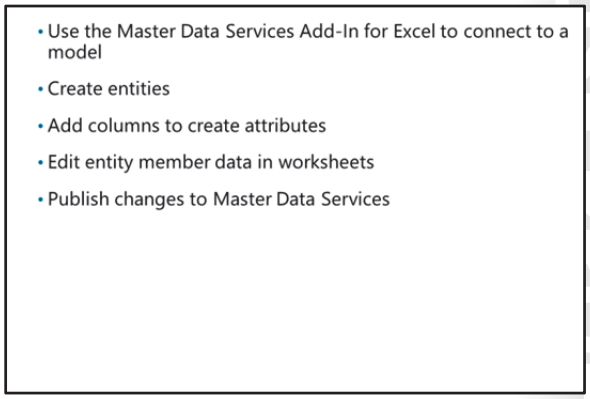
7. In the **Name** box, type **Phone**, in the **Data type** list, select **Text**, in the **Length** box, type **20**, and then click **Save attribute**.
8. On the **Edit Entity: Customer** page, click **Save Entity**, and then click the **Microsoft SQL Server 2014** logo to return to the home page.

Add and Edit Members

1. On the Master Data Services home page, click **Explorer**.
2. In the **Entities** menu, click **Customer**. If the **Entities** menu does not appear, click any other menu and then click the **Entities** menu again.
3. Click **Add Member**, and in the **Details** pane, enter the following data:
   - **Name**: Ben Smith
   - **Code**: 1235
   - **Address**: 1 High St, Seattle
   - **Phone**: 555 12345
   - **Annotations**: Initial data entry
4. Click **OK**, then click the entity row you have added. Then in the **Details** pane, edit the following fields:
   - **Phone**: 555 54321
   - **Annotations**: Changed phone number
5. Click **OK**, and then click **View Transactions** noting the list of transactions for this entity. Click each transaction and view the text in the **Annotations** tab before clicking **Close**.
6. Close Internet Explorer.

Editing a Model in Microsoft® Excel®

SQL Server 2014 Master Data Services supports the Master Data Services Add-in for Excel, which you can use to read and manage lists of Master Data Services data. The add-in is a free download for Excel 2007 or later that you can distribute to data stewards and Master Data Services administrators, enabling them to work with a familiar and easy-to-use interface. You can download the Master Data Services Add-In for Excel from the Microsoft download site, or users can install it directly from Master Data Manager.



The Master Data Services Add-in for Excel adds the **Master Data** tab to the Excel ribbon. You can use options on this tab to perform tasks, such as connecting to a Master Data Services server, applying business rules, creating new entities, and publishing changes back to the server. The add-in is security context aware, and will only allow users to view and change data for which they have the appropriate permissions.

To view Master Data Services data in Excel, you must first connect to a Master Data Services server. On the **Master Data** tab in Excel, the **Connect** option enables you to create a connection to Master Data Services. After connecting, you can use the Master Data Explorer to select a model and version to work with from those available on the server. You can then load data into an Excel worksheet from the entities listed in the Master Data Explorer, and filter that data so you only see the actual data you want to work with. After you have loaded the required data, you can browse and edit it just as you would any other data in Excel. You can create new entities, add columns to existing entities to define new attributes, and edit member data.

Most data editing operations are performed locally in the Excel worksheet, and changes are only propagated to the Master Data Services database when you explicitly publish the changes made in Excel. When you publish an entity, you can enter annotations to document the changes made.

Demonstration: Editing a Model in Excel

In this demonstration, you will see how to edit a master data model in Microsoft® Excel®.

Demonstration Steps

Connect to a Master Data Services Model in Excel

1. Ensure you have completed the previous demonstration in this module.
2. Start Microsoft® Excel® and create a new blank document.
3. On the **File** tab, click **Options**. Then in the **Excel Options** dialog box, on the **Add-Ins** tab, select **COM Add-ins** and click **Go**.
4. In the **COM Add-Ins** dialog box, if **Master Data Services Add-In for Excel** is not selected, select it. Then click **OK**.
5. On the **Master Data** tab of the ribbon, in the **Connect and Load** section, click the drop-down arrow under **Connect** and click **Manage Connections**.
6. In the **Manage Connections** dialog box, click **New**, and in the **Add New Connection** dialog box, enter the description **Demo MDS Server** and the MDS server address **http://localhost:81/mds**, and click **OK**. Then click **Close**.
7. On the **Master Data** tab of the ribbon, in the **Connect and Load** section, click the drop-down arrow under **Connect** and click **Demo MDS Server**.
8. In the **Master Data Explorer** pane, in the **Model** list, select **Customers**.
9. In the **Master Data Explorer** pane, click the **Customer** entity. Then on the ribbon, in the **Connect and Load** section, click **Refresh**. Note that the **Customer** entity, including the member you created in the previous demonstration, is downloaded into a new worksheet.

Add a Member

1. On the **Customer** worksheet, click cell **D4** (which should be an empty cell under Ben Smith).
2. Enter the following details in row 4:
   - **D4**: Andrew Sinclair
   - **E4**: 2600
   - **F4**: 2 Main St, Ontario
   - **G4**: 555 11111

3. Note that the row for the new member you have created has an orange background to indicate that the data has not been published to Master Data Services.
4. On the ribbon, in the **Publish and Validate** section, click **Publish**. Then in the **Publish and Annotate** dialog box, enter the annotation **Added new member**, and click **Publish**. Note that the data is published and the orange background is removed.

Add a Free-Form Attribute to an Entity

1. On the **Customer** worksheet, click cell **H2** (which should be an empty cell to the right of the **Phone** attribute header).
2. Type **CreditLimit** and press Enter. This adds a new attribute named **CreditLimit** (which is shown with a green background because it is not yet saved to the model).
3. Click cell **H2** to re-select it, and on the **Master Data** tab of the ribbon, in the **Build Model** section, click **Attribute Properties**. Then in the **Attribute Properties** dialog box, in the **Attribute type** drop-down list, click **Number**, note the default **Decimal places** value (2), and click **OK**. Note that the changes are uploaded to the data model and the cell background changes to blue.
4. In cell **H3**, enter 1000 as the **CreditLimit** value for the Ben Smith member, and in cell **H4** enter 500 as the **CreditLimit** value for the Andrew Sinclair member. Note that the cell background is orange to indicate that the data has not been published to Master Data Services.
5. On the ribbon, in the **Publish and Validate** section, click **Publish**. Then in the **Publish and Annotate** dialog box, enter the annotation **Set credit limit**, and click **Publish**. Note that the data is published and the orange background is removed.

Add a Domain-Based Attribute and Related Entity

1. On the **Customer** worksheet, click cell **I2** (which should be an empty cell to the right of the **CreditLimit** attribute header).
2. Type **AccountType** and press Enter. This adds a new attribute named **AccountType** (which is shown with a green background because it is not yet saved to the model).
3. In cell **I3**, enter 1 as the **AccountType** value for Ben Smith. Then in cell **I4**, enter 2 as the **AccountType** value for Andrew Sinclair.
4. Click cell **I2** to re-select it, and on the ribbon, in the **Build Model** section, click **Attribute Properties**. Then in the **Attribute Properties** dialog box, in the **Attribute type** drop-down list, click **Constrained list (Domain-based)**, in the **Populate the attribute with values from** list, ensure **the selected column** is selected, in the **New entity name** box, type **Account Type**, and click **OK**. Note that the **AccountType** column now contains the values 1{1} and 2{2}.
5. On the **Master Data** tab of the ribbon, in the **Connect and Load** section, click the drop-down arrow under **Connect** and click **Demo MDS Server**.
6. In the **Master Data Explorer** pane, in the **Model** list, select **Customers**.
7. In the **Master Data Explorer** pane, click the **Account Type** entity, which was created when you added a domain-based attribute to the **Customer** entity. Then on the ribbon, in the **Connect and Load** section, click **Refresh**. Note that the **Account Type** entity is downloaded into a new worksheet with two members.
8. Change the **Name** attribute for the existing members as follows:

Name	Code
Standard	1
Premier	2

9. On the ribbon, in the **Publish and Validate** section, click **Publish**. Then in the **Publish and Annotate** dialog box, enter the annotation **Changed account type names**, and click **Publish**.
10. Click the **Customer** worksheet tab, and on the ribbon, in the **Connect and Load** section, click **Refresh**. Note that the **AccountType** attribute values are updated to show the new **Name** values you specified for the members in the **Account Type** entity.
11. Click cell **I3**, and note that you can select the **AccountType** value for each member from a list that looks up the related name and code in the **Account Type** entity. Leave the selected **AccountType** for Ben Smith as 1 {Standard}.
12. Close Excel without saving the workbook.

Lesson 3
Managing Master Data

After you have defined a Master Data Services model and the entities you want to manage, you can use Master Data Services to organize, manage, and maintain entity members. This enables you to create master data solutions that meet the requirements of your organization's business applications and ensure the enterprise-wide integrity and consistency of your data.

Lesson Objectives

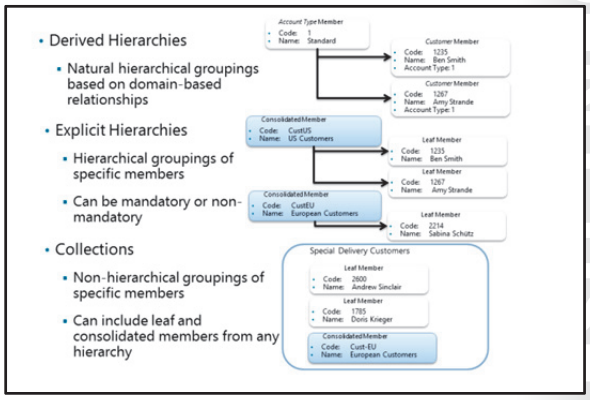
After completing this lesson, you will be able to:

- Describe how hierarchies and collections can be used to organize master data.
- Create derived hierarchies.
- Create explicit hierarchies.
- Create collections.
- Find duplicate members.
- Use business rules to validate members.

Hierarchies and Collections

Hierarchies and collections are two ways to group related data in a Master Data Services model.

Hierarchies enable you to group members to provide users with a structured view of data that is easier to browse. A hierarchy contains all the members from the entity or entities that you add to it, and each member can appear only once. Collections are flat groupings of members within an entity that have no hierarchical structure, but make it easier to find and manage members that are related in some way.



Derived Hierarchies

Derived hierarchies are based on the relationships that exist between entities. They use domain-based attributes that you create to infer parent-child relationships. For example, suppose that you have two entities, **Customer** and **Account Type**. The **Customer** entity contains a domain-based attribute named **AccountType** that references the **Code** attribute of the **Account Type** entity. You can use these attributes to create a derived hierarchy that groups customers by account type.

Explicit Hierarchies

Explicit hierarchies can contain members from a single entity, and do not use domain-based attributes to determine their structure. Instead, you must create *consolidated members* to define levels within the hierarchy, and then move the *leaf members* (members that represent instances of the entity) into the appropriate level. For example, you could create a consolidated member named **US Customers**, and group leaf members for customers with a US address under it. Unlike derived hierarchies, explicit hierarchies can be ragged, and so do not need a consistent number of levels. An explicit hierarchy contains all the members in an entity. If you make the hierarchy mandatory, then all entity leaf members

must be assigned to a consolidated member. In a non-mandatory hierarchy, any members that you do not group under a consolidated member are grouped together in the hierarchy as 'unused'.

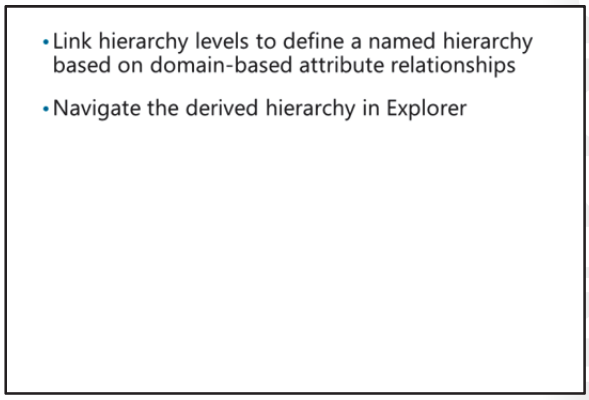
Collections

Collections enable you to combine leaf members and consolidated members from any existing explicit hierarchies into a group of related members. Collections provide flexibility and are efficient because they enable you to re-use existing hierarchies, removing the need to create a new one. For example, you could create a collection named **Special Delivery Customers** that contains consolidated members from an explicit hierarchy based on geographic locations, as well as some individual leaf members that don't belong to the consolidated members, but which also qualify for special delivery status.

Note: You can only create explicit hierarchies and collections for entities that have the **Enable explicit hierarchies and collections** option enabled.

Creating Derived Hierarchies

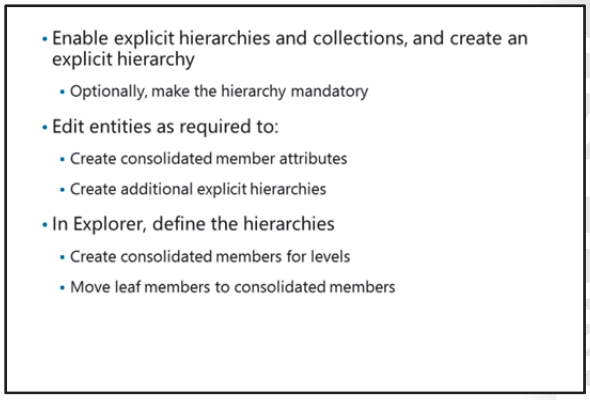
Use the following procedure to create a derived hierarchy:



1. In Master Data Manager, click **System Administration**.
2. On the **Model View** page, in the **Manage** menu, click **Derived Hierarchies**.
3. On the **Derived Hierarchy Maintenance** page, from the **Model** list, select the model in which you want to create a derived hierarchy.
4. Click **Add derived hierarchy**.
5. On the **Add Derived Hierarchy** page, in the **Derived hierarchy** name box, type a name for the hierarchy. Try to make the name meaningful, for example, **Customers by Account Type**.
6. Click **Save derived hierarchy**.
7. On the **Edit Derived Hierarchy** page, in the **Available Entities and Hierarchies** pane, click the entity that represents the top level of your hierarchy (for example, **Account Type**) and drag it to the **Current Levels** pane.
8. Drag the entity that represents the next level (for example, **Customer**) onto the level you created in the previous step. Note that there must be a relationship between a domain-based attribute in the entity you are dragging and the **Code** attribute in the parent entity.
9. Continue dragging entities until your hierarchy includes all the levels it requires.

Creating Explicit Hierarchies

To create an explicit hierarchy, you must enable explicit hierarchies and collections for the entity, and edit it to create any required consolidated member attributes. You can then define the consolidated members for the hierarchy levels and assign leaf members to them.



Enabling Explicit Hierarchies and Collections

Use the following procedure to enable explicit hierarchies and collections for an entity:

1. In Master Data Manager, click **System Administration**.
2. On the **Model View** page, in the **Manage** menu, click **Entities**.
3. On the **Entity Maintenance** page, in the **Model** list, select the model that contains the entity for which you want to create an explicit hierarchy.
4. Select the entity that you want to update, and then click **Edit selected entity**.
5. In the **Enable explicit hierarchies and collections** list, select **Yes**.
6. In the **Explicit hierarchy name** box, type a name for the explicit hierarchy you want to create. Try to make the name meaningful, for example, **Customers by Geographic Location**.
7. Optionally, clear the **Mandatory hierarchy** check box to create the hierarchy as non-mandatory.
8. Click **Save entity**.

Editing Entities

After you have enabled explicit hierarchies and collections, and created an explicit hierarchy, you can start defining consolidated members. However, by default, consolidated members only have **Code** and **Name** attributes, so you may want to edit the entity to add consolidated member attributes. Additionally, if you need to add another explicit hierarchy to the entity, you can edit it and click **Add explicit hierarchy**.

Defining Hierarchies

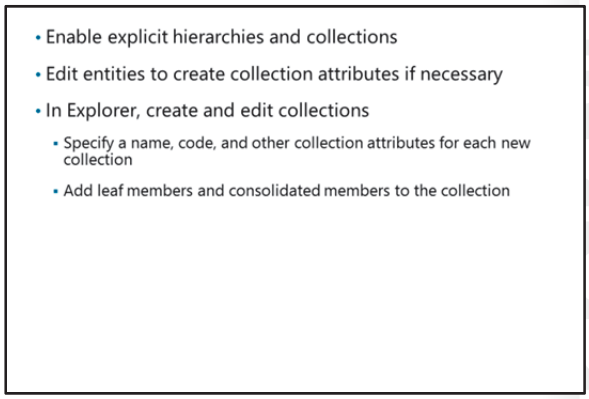
After you have created an explicit hierarchy, you must define the consolidated members that form the hierarchy levels and assign leaf members to them. Use the following procedure to define an explicit hierarchy:

1. On the Master Data Manager home page, in the **Model** list, select the model that contains the entity with the explicit hierarchy you want to edit.
2. In the **Version** list, select the version you want to work with.
3. Click **Explorer**.
4. In the **Hierarchies** menu, click the name of the hierarchy you want to define.
5. Above the grid, select either **Consolidated members** or **All consolidated members in hierarchy**.
6. Click **Add**.
7. In the **Details** pane, enter values for the consolidated member attributes. Optionally, in the **Annotations** box, type a comment to document the creation of the consolidated member.
8. Click **OK**.

9. After creating the consolidated members that define the hierarchy levels, you can move members to create the hierarchy structure. To place members under a consolidated member:
    a. In the hierarchy tree view, select the check box for each leaf or consolidated member you want to move.
    b. Click **Cut**.
    c. Select the consolidated member to which you want to assign the selected members.
    d. Click **Paste**.

Creating Collections

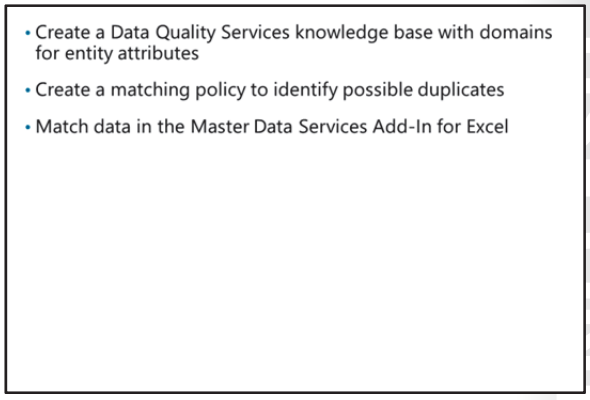
To create a collection, you must enable explicit hierarchies and collections for the entity as described in the previous topic. You can then use the following procedure to create a collection:



1. On the Master Data Manager home page, in the **Model** list, select the model that contains the entity for which you want to create a collection.
2. In the **Version** list, select the version you want to work with.
3. Click **Explorer**.
4. In the **Collections** area, click the entity for which you want to create a collection.
5. Click **Add collection**.
6. On the **Details** tab, in the **Name** box, type a name for the collection.
7. In the **Code** box, type a unique code for the collection.
8. Optionally, in the **Description** box, type a description for the collection.
9. Click **OK**.
10. On the **Collection Members** tab, click **Edit Members**.
11. To filter the list of available members, select from the list on the left.
12. Click each member you want to add and click **Add**.
13. Optionally, rearrange collection members by clicking **Up** or **Down**.

Finding Duplicate Members

When you create a model, the entities may have some initial members to help you structure the hierarchies and collections the model must support. However, as the model moves into production, you will start to load large volumes of member data, and it is important that data stewards can easily manage the integrity of this data.



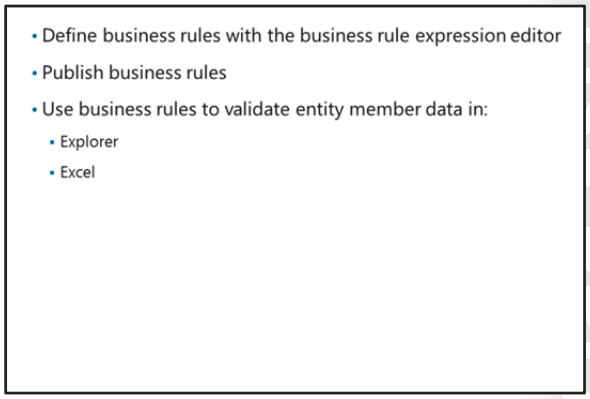
One common problem is the creation of duplicate members for the same business entity. To help avoid this problem, the Master Data Services Add-in for Excel includes a data matching feature that uses Data Quality Services matching policies to identify possible duplicate members.

To use the Master Data Services add-in for Excel to find duplicate members:

1. Create a Data Quality Services knowledge base that includes domains for the attributes on which you want to compare members.
2. Create a Data Quality Services matching policy that identifies potential duplicates based on domain value matches.
3. Open the entity in the Master Data Services Add-in for Excel, and click **Match Data**. Then specify the Data Quality Services server and knowledge base, and map the knowledge base domains to entity attributes.

Validating Members with Business Rules

You can use business rules to ensure that the members you add to your models are accurate and meet the criteria that you define. Business rules help to improve data quality, which increases the accuracy and reliability of the reports and data analyses that information workers produce. A business rule is an IF/THEN statement you create by using the **Business Rule Maintenance** page in the Master Data Manager Web application. To define the terms of a business rule, you use a drag-and-drop interface on the **Edit Business Rule** page.



Note: You must be a model administrator to create and publish business rules.

When you create a rule, you define a condition that gives a true or false result. For example, you can create conditions such as 'If a customer's credit limit is greater than 1,000.' You can use a range of value comparison operators to build conditions, including 'is equal to', 'starts with', 'is between', and 'contains'. When you validate data, Master Data Services compares the data values against these conditions. Any that match a condition are tested against the actions you define. For example, for the condition 'If a customer's credit limit is greater than 1,000', the action could be 'The account type must be Premier'.

After creating a business rule, you must publish it to make it active. You can then use the rule to validate new members when they are added to Master Data Services. Business rules do not prevent users who have update permission from making changes that violate the rules. Instead, the validation process reveals any data values that violate business rules, so that a data steward can identify problematic data and make the required changes. When validating data against business rules, you can choose to validate the whole version or just a subset of the data in a version, such as a single entity.

Note: You can use the AND operator and the OR operator to create rules with multiple conditions.

Business Rule Priority

You can set a value for each business rule that defines its priority. This enables you to control the order in which Master Data Services applies business rules. Rules with a low priority value run before rules that have higher priority values. For example, if a rule that checks the credit limit for customers has a priority of 10, it will run before rules with a priority greater than 10. By default, when you create a new rule, its priority value is higher than the previous rule by a value of 10, so the new rule has a lower priority than the previous rule and runs after it.

Priority is important when business rules depend on the outcome of other business rules. For example, suppose you create a rule that updates the value of the **CreditLimit** attribute of a **Customer** entity member. When you import a new customer, the rule sets the value to -1. A second rule checks the **CreditLimit** attribute for the value -1, changes the status of the member to 'is not valid,' and notifies the data steward that there is a new customer for which they need to specify a credit limit. By setting the priority of the second rule to a higher value than the first, you ensure that the rules run in a coordinated and logical order and the data steward can update credit limits for new customers sooner.

Demonstration: Creating and Applying Business Rules

In this demonstration, you will see how to create and apply business rules.

Demonstration Steps

Create Business Rules

1. Ensure you have completed the previous demonstrations in this module.
2. Start Internet Explorer and browse to **http://localhost:81/mds**.
3. On the Master Data Services home page, click **System Administration**, and then on the **Manage** menu, click **Business Rules**.
4. On the **Business Rule Maintenance** page, in the **Model** list, ensure that **Customers** is selected, in the **Entity** list, ensure that **Customer** is selected. In the **Member Type** list, select **Leaf**, and then in the **Attribute** list, ensure that **All** is selected.
5. Click **Add business rule** to create a new business rule.
6. Double-click in the **Name** column for the new rule, and then type **Check Credit** and press Enter.
7. Double-click in the **Description** column, type **Check that credit limit is valid**, and then press Enter. You will use this rule to ensure that all customer credit limits are greater than or equal to zero.
8. Click **Edit selected business rule**, and then in the **Version** list, ensure that VERSION_1 is selected.
9. Under **Components**, expand **Actions**, and then in the **Validation** list, drag **must be greater than or equal to** onto the **Actions** node in the THEN section of the expression editor.

10. In the **Attributes** list, drag **CreditLimit** onto **Select attribute** directly under **Edit Action**.
11. In the **Attribute value** box, ensure 0 is displayed, and then click the **Save item** button. Note that the text under **Actions** in the THEN section of the expression editor changes to read 'CreditLimit must be greater than or equal to 0.00'.
12. Click the green **Back** arrow button, and note that the rule's **Status** is listed as **Activation pending**.
13. Click **Add business rule**, double-click in the **Name** column, type **Check Premier Status** and then press Enter.
14. Double-click in the **Description** column, type **Check account type for high credit limits** and then press Enter. You will use this rule to check that customers with a credit limit value greater than 1,000 have a premier account type.
15. Click **Edit selected business rule**, and then in the **Version** list, ensure that VERSION_1 is selected.
16. Under **Components**, expand **Conditions**, and then drag **is greater than** onto the **Conditions** node in the IF section of the expression editor.
17. In the **Attributes** list, drag **CreditLimit** onto **Select attribute** directly under **Edit Action**, and then, in the **Attribute value** box, type **1000**. Then click the **Save item** button. Note that the text under **Conditions** in the IF section of the expression editor changes to read "CreditLimit is greater than 1000.00".
18. Under **Components**, expand **Actions**, and then in the **Validation** list, drag **must be equal to** onto the **Actions** node in the THEN section of the expression editor.
19. In the **Attributes** list, drag **AccountType** onto **Select attribute** directly under **Edit Action**, and then, in the **Attribute value** drop-down list, select **2**. Then click the **Save item** button. Note that the text under **Actions** in the THEN section of the expression editor changes to read "AccountType must be equal to 2".
20. Click the green **Back** arrow button, and note that the **Status** for both rules is listed as **Activation pending**.

Publish Business Rules

1. On the **Business Rule Maintenance** page, click **Publish business rules**, and then in the **Message from webpage** dialog box, click **OK**.
2. In the **Status** column, check that the value displayed for both rules is **Active**.
3. On the **Business Rule Maintenance** page, click the **Microsoft SQL Server 2014** logo to return to the home page.

Apply Business Rules in Explorer

1. On the Master Data Services home page, click **Explorer**.
2. In the **Entities** menu, click **Customer**. If the **Entities** menu does not appear, click any other menu and then click the **Entities** menu again.
3. Click **Apply Rules**, and note that a green tick is displayed next to all valid records.
4. Click the row for the Andrew Sinclair member, and then in the **Details** tab, change the **CreditLimit** value to **-200** and click **OK**. Note that a validation error message is displayed and that the green tick for this member record changes to a red exclamation mark.
5. Change the **CreditLimit** value for Andrew Sinclair back to **500**, noting that the validation message disappears and the red exclamation mark changes back to a green tick.
6. Close Internet Explorer.

Apply Business Rules in Excel

1. Start Microsoft® Excel® and create a new blank workbook.
2. On the **Master Data** tab of the ribbon, in the **Connect and Load** section, click the drop-down arrow under **Connect** and click **Demo MDS Server**.
3. In the **Master Data Explorer** pane, in the **Model** list, select **Customers**.
4. In the **Master Data Explorer** pane, click the **Customer** entity. Then on the ribbon, in the **Connect and Load** section, click **Refresh**. The **Customer** entity members are downloaded into a new worksheet.
5. In the ribbon, in the **Publish and Validate** section, click **Show Status**. This reveals columns that show the validation and input status for all records.
6. In the ribbon, in the **Publish and Validate** section, click **Apply Rules**. This refreshes the validation status for all rows. Currently validation is successful for all records.
7. In cell **H3**, change the **CreditLimit** value for Ben Smith to **1200**. Note that the **AccountType** value for this record is currently 1 {Standard}.
8. In the ribbon, in the **Publish and Validate** section, click **Publish**. Then in the **Publish and Annotate** dialog box, enter the annotation **Increased credit limit**, and click **Publish**.
9. Note that the validation status in cell **B3** is **Validation failed**, and that the cell is highlighted in red. Hold the mouse pointer over cell **B3** and note that a comment is displayed showing the validation error message.
10. Click cell **I3**, and then in the drop-down list, select **2 {Premier}** to change the **AccountType** value for Ben Smith.
11. In the ribbon, in the **Publish and Validate** section, click **Publish**. Then in the **Publish and Annotate** dialog box, enter the annotation **Upgraded account type**, and click **Publish**.
12. Note that validation has now succeeded for the Ben Smith member. Then close Excel without saving the workbook.

Lesson 4
Creating a Master Data Hub

Master data management provides a way to ensure data consistency for key business entities across the enterprise. In many cases, a master data hub is created in which data stewards can centrally manage data definitions for business entities used by multiple applications. This lesson describes the key features of a master data hub and teaches you how to create a solution in which data can flow into and out of a master data hub implemented in Master Data Services.

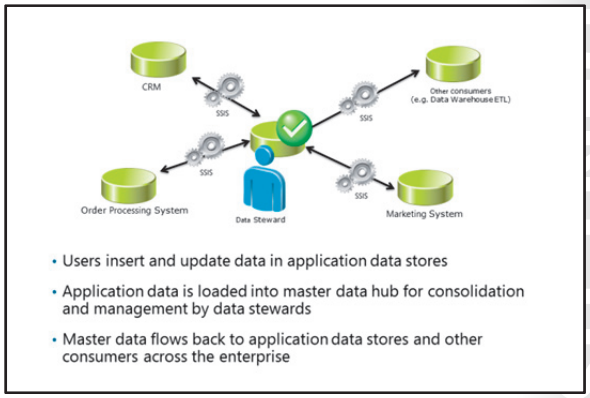
Lesson Objectives

After completing this lesson, you will be able to:

- Describe master data hub architecture.
- Describe the structure of Master Data Services staging tables.
- Import data into Master Data Services.
- Create subscription views so that applications can consume master data.

Master Data Hub Architecture

In some scenarios, data stewards can use Master Data Services as the sole data entry point for records that represent key data entities. This approach ensures that a single, consistent definition of the business entities exists and is used across the organization. However, in many organizations, data relating to the same business entity is entered and updated in multiple applications, and a solution that consolidates and harmonizes this data must be developed. In cases like this, you can use a master data hub as a central point where new and updated records from multiple applications are imported, managed, and then flow back out to the applications that use them.



For example, an organization might use the following applications to manage customer data:

- A CRM system.
- An order processing system.
- A marketing system.

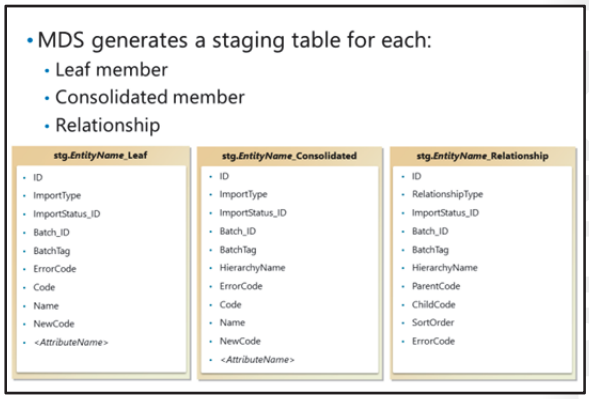
As customer records are entered or updated in each of these systems, an SSIS-based solution takes the modified records and imports them into the master data hub, where a data steward can validate and consolidate the data relating to customer entities. An SSIS-based solution can then extract the consolidated customer data from the master data hub and replicate it back to the source systems, ensuring that all systems have consistent definitions for customers. Each SSIS data flow may only transfer the attributes that are required by the individual application being synchronized. The complete set of customer attributes is managed centrally in the master data hub.

Additionally, other applications that require accurate customer data but do not need to modify it, can consume the data directly from the master data hub. For example, an ETL process for a data warehouse

might take customer data from the master data hub to ensure that the data warehouse is populated with consistent, accurate records for customers.

Master Data Services Staging Tables

To facilitate the import of new and updated data from business applications, Master Data Services generates staging tables for each entity defined in a model. You can load data into these staging tables by using an SSIS data flow, the Import and Export Data Wizard, Transact-SQL statements, the bulk copy program, or any other technique that can be used to insert data into an SQL Server database table.



Staging Tables for Leaf Members

Master Data Services creates a leaf member staging table for each entity you create. The table is created in the **stg** schema and by default is named in the format ***EntityName*_Leaf**. For example, a staging table for leaf members in a **Customer** entity would be named **stg.Customer_Leaf**. You can override the name of the staging table when you create an entity.

A leaf member staging table contains the following columns:

- **ID**. An automatically assigned identifier.
- **ImportType**. A numeric code that determines what to do when imported data matches an existing member in the model. For example, a value of 1 creates new members, but does not update existing members, and a value of 2 replaces existing members with matching staged members.
- **ImportStatus_ID**. The status of the import process. You initially set this to 0 to indicate that the data is staged and ready for import. During the import process, the value is automatically updated to 1 if the import succeeds or 2 if it fails.
- **Batch_ID**. A unique identifier for a batch of imported records. This value is automatically set when importing data through the Master Data Services Web service. Otherwise it is not required.
- **BatchTag**. A unique name that identifies a batch of imported records. This is used to group staged records instead of the **Batch_ID** column when not using the Master Data Services Web service.
- **ErrorCode**. Set by the import process if the import fails.
- **Code**. The unique code attribute for the member.
- **Name**. The name attribute for the member.
- **NewCode**. Used to change the Code attribute of a member.
- **<*AttributeName*>**. A column is created for each leaf attribute defined in the entity.

Staging Tables for Consolidated Members

Staging tables for consolidated members by default take the name format **stg.*EntityName*_Consolidated** (for example **stg.Customer_Consolidated**). Consolidated member staging tables contain the same columns as leaf member staging tables with an additional **HierarchyName** column you can use to indicate the explicit hierarchy to which the consolidated member should be imported. Additionally, the ***<AttributeName>*** columns are created to reflect the consolidated attributes defined in the entity instead of the leaf attributes.

Staging Tables for Relationships

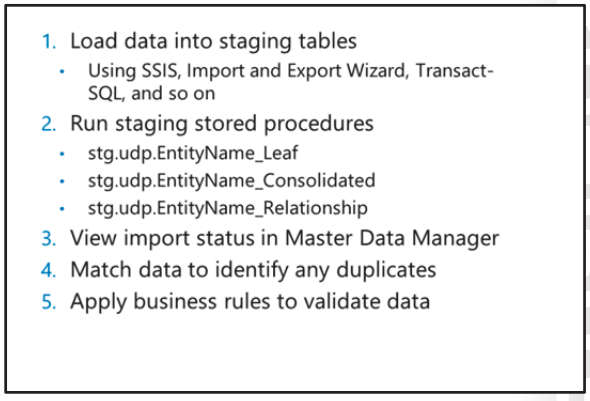
Relationship staging tables are used to change the location of members in an explicit hierarchy. By default, they take the name format **stg.*EntityName*_Relationship**, and they contain the following columns:

- **ID**. An automatically assigned identifier.
- **RelationshipType**. A numeric code that indicates the kind of relationship. Valid values for this column are 1 (parent) and 2 (sibling).
- **ImportStatus_ID**. The status of the import process. You initially set this to 0 to indicate that the data is staged and ready for import. During the import process, the value is automatically updated to 1 if the import succeeds or 2 if it fails.
- **Batch_ID**. A unique identifier for a batch of imported records. This value is automatically set when importing data through the Master Data Services Web service. Otherwise it is not required.
- **BatchTag**. A unique name that identifies a batch of imported records. This is used to group staged records instead of the **Batch_ID** column when not using the Master Data Services Web service.
- **HierarchyName**. The explicit hierarchy in which the relationship should be defined.
- **ParentCode**. The Code attribute of the member that will be the parent (for **RelationshipType** 1 imports) or sibling (for **RelationshipType** 2 imports) in the relationship.
- **ChildCode**. The Code attribute of the member that will be the child (for **RelationshipType** 1 imports) or sibling (for **RelationshipType** 2 imports) in the relationship.
- **SortOrder**. An optional integer that determines the sort order for sibling members under a parent.
- **ErrorCode**. Set by the import process if the import fails.

Staging and Importing Data

The process for importing data into Master Data Services consists of the following steps:

1. Load new or updated member records into staging tables. You can use any technique that inserts data into a table in a SQL Server database.
2. Run the appropriate staging stored procedures to process the staged records. Master Data Services generates a stored procedure for each staging table, with names in the following formats:
   - **stg.udp.*EntityName*_Leaf**



- **stg.udp.*EntityName*_Consolidated**
- **stg.udp.*EntityName*_Relationship**

When you execute the stored procedures, you must specify values for the **VersionName**, **LogFlag**, and **BatchTag** parameters to indicate the version of the model to be updated, whether or not to log transactions (*1*) or not (*0*), and the unique batch tag that identifies the staged records to be processed. Note that when called by the Master Data Services Web service, the **Batch_ID** parameter is used instead of the **BatchTag** parameter.

3. View the import status in Master Data Manager. The import process is asynchronous, so you can use the **Integration Management** area in Master Data Manager to view the start time, end time, and results of each batch that has been processed.
4. To identify and consolidate any duplicate records, data stewards should use the **Match Data** functionality in the Master Data Services Add-In for Excel after importing data.
5. Finally, a data steward should apply the business rules defined in the model for the imported members, to ensure that they are valid, before making them available to applications.

Demonstration: Importing Master Data

In this demonstration, you will see how to import data into a master data model.

Demonstration Steps

Use an SSIS Package to Import Master Data

1. Ensure you have completed the previous demonstrations in this module.
2. Start Visual Studio and open the **MDS Import.sln** solution in the D:\Demofiles\Mod10 folder.
3. In Solution Explorer, in the **SSIS Packages** folder, double-click the **Import Customers.dtsx** SSIS package to view its control flow.
4. On the control flow surface, double-click **Load Staging Tables** to view the data flow.
5. On the data flow surface, double-click **Customers Source**, and in the **Flat File Source Editor** dialog box, on the **Columns** tab, note the columns that will be imported from the source system. Then click **Cancel**.
6. On the data flow surface, double-click **Add MDS Columns**, and in the **Derived Column Transformation Editor** dialog box, note that the transformation generates additional columns named **ImportType** (with a value of 0), **ImportStatus_ID** (with a value of 0), and **BatchTag** (with a unique string value derived from the **ExecutionInstanceGUID** system variable). Then click **Cancel**.
7. On the data flow surface, double-click **Staging Table**, and in the **OLE DB Destination Editor** dialog box, on the **Connection Manager** tab, note that the data is loaded into the **[stg].[Customer_Leaf]** table in the **MDS** database, and on the **Mappings** tab, note the column mappings. Then click **Cancel**.
8. Click the **Control Flow** tab and on the control flow surface, double-click **Load Staged Members**. On the **General** tab, note that the **SqlStatement** property is set to execute the **stg.udp_Customer_Leaf** stored procedure with the values 'VERSION_1', 0, and a parameter, and on the **Parameter Mapping** tab, note that the **ExecutionInstanceGUID** system variable is mapped to the **@BatchTag** parameter. Then click **Cancel**.
9. On the **Debug** menu, click **Start Debugging**. Then when execution is complete, on the **Debug** menu, click **Stop Debugging** and close Visual Studio without saving any changes.

View Import Status

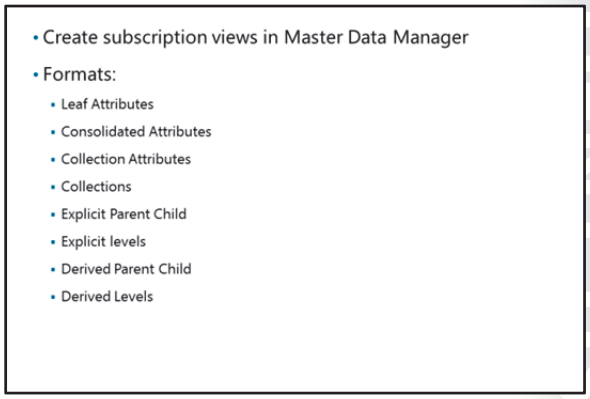
1. Start Internet Explorer and browse to **http://localhost:81/mds**.
2. On the Master Data Services home page, click **Integration Management**.
3. On the **Import Data** page, ensure that the **Customers** model is selected and note the batches listed. There should be a single batch for the data import you performed in the previous exercise. Note the **Started**, **Completed**, **Records**, **Status** and **Errors** values for this batch.
4. On the **Import Data** page, click the **Microsoft SQL Server 2014** logo to return to the home page.

Validate Imported Data

1. On the Master Data Services home page, click **Explorer**.
2. In the **Entities** menu, click **Customer**. Note that nine new member records have been imported, and that their validation status is indicated by a yellow question mark.
3. Click **Apply Rules**, note that the business rules in the model are applied to all records, and that the validation status for the new records changes to a green tick for records that have passed validation and an exclamation mark for records that have failed validation.
4. Close Internet Explorer.

Consuming Master Data with Subscription Views

A subscription view is a standard SQL Server view that enables applications to consume master data. Subscribing applications can use Transact-SQL SELECT statements to access data by using a view, just as they can for tables and views in a typical SQL Server database.



You can create views by using the **Export** page in the Master Data Manager Web application. You can use the standard Master Data Services view formats to create views displaying different types of data, including:

- Leaf attributes.
- Consolidated attributes.
- Collection attributes.
- Collections.
- Parent-Child members in an explicit hierarchy.
- Levels in an explicit hierarchy
- Parent-Child members in a derived hierarchy.
- Levels in a derived hierarchy.

If you modify a data model after creating views that are based on it, you must regenerate the view so that it remains synchronized with the data. You can identify views to be updated by looking at the **Changed** column for each one on the **Subscription Views** page in the Master Data Manager Web application. The column shows a value of **True** for any views that are no longer synchronized with the associated model.

Demonstration: Using Subscription Views

In this demonstration, you will see how to use subscription views to consume master data.

Demonstration Steps

Create a Subscription View

1. Ensure you have completed the previous demonstrations in this module.
2. In Internet Explorer, browse to **http://localhost:81/MDS**. Then, on the Master Data Services home page, click **Integration Management**.
3. Click **Create Views**, and on the **Subscription Views** page, click the **Add subscription view** button to add a subscription view.
4. On the **Subscription Views** page, in the **Create Subscription View** section, in the **Subscription view** name box, type **MasterCustomers**. In the **Model** list, select **Customers**, in the **Version** list, select **VERSION_1**, in the **Entity** list, select **Customer**, in the **Format** list select **Leaf members**, and then click **Save**.
5. Close Internet Explorer.

Query a Subscription View

1. Start SQL Server Management Studio. When prompted, connect to the **MIA-SQL** instance of SQL Server by using Windows authentication.
2. Click **New Query**, and then in the query editor enter the following Transact-SQL:

```
USE MDS
GO
SELECT * FROM mdm.MasterCustomers
```

3. Click **Execute** to run the query, and review the results, and then close SQL Server Management Studio without saving any items.

Lab: Implementing Master Data Services

Scenario

The data warehousing solution you are building for Adventure Works Cycles includes product data from various systems throughout the enterprise. You need to ensure that there is a single, consistent definition for each product.

Objectives

After completing this lab, you will be able to:

- Create a master data model in Master Data Services.
- Use the Master Data Services Add-in for Excel.
- Create and enforce business rules.
- Load data into a master data model.
- Consume master data.

Estimated Time: 60 Minutes

Virtual machine: 20463C-MIA-SQL

User name: ADVENTUREWORKS\Student

Password: Pa$$w0rd

Exercise 1: Creating a Master Data Services Model

Scenario

The ETL solution you are building for Adventure Works Cycles consumes product data from an application database. However, product data is created and updated in various systems throughout the enterprise, and you need to ensure that there is a single, consistent definition for each product.

To accomplish this, you plan to create a master data hub that includes complete definitions for **Product**, **Product Subcategory**, and **Product Category** entities.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Create a Model

3. Create Entities

4. Create Attributes

5. Add Members

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. Run Setup.cmd in the D:\Labfiles\Lab10\Starter folder as Administrator.

▶ Task 2: Create a Model

1. In the Master Data Manager Web application at **http://localhost:81/mds**, in the **System Administration** area, create a model named **Product Catalog**. Do not create an entity with the same name.

▶ Task 3: Create Entities

1. In the **Product Catalog** model, create the following entities. Do not enable explicit hierarchies and collections for these entities:
   - **Product**
   - **Product Subcategory**

▶ Task 4: Create Attributes

1. In the **Product** entity, create the following leaf attributes:
   - A domain-based attribute named **ProductSubcategoryKey** that references the **Product Subcategory** entity.
   - A free-form attribute named **Description** that can store a text value with a maximum length of 400.
   - A free-form attribute named **ListPrice** that can store a number with four decimal places and has the input mask **(####)**.

▶ Task 5: Add Members

1. In the **Explorer** area of Master Data Manager, view the **Product Subcategory** entity and add the following members:

Name	Code
Mountain Bikes	1
Chains	7
Gloves	20
Helmets	31

2. View the **Product** entity and add the following members:

Name	Code	ProductSubcategoryKey	Description	ListPrice
Mountain-100 Silver, 42	345	1	Top of the line competition mountain bike.	3399.99
Chain	559	7	Superior chain	20.24
Full-Finger Gloves, S	468	20	Synthetic gloves	37.99
Sport-100 Helmet, Red	214	31	Universal fit helmet	34.99

Results: After this exercise, you should have a Master Data Services model named **Product Catalog** that contains **Product** and **ProductSubcategory** entities. Each of these entities should contain four members.

Exercise 2: Using the Master Data Services Add-in for Excel

Scenario

To make it easier for data stewards to continue developing the master data model, you plan to use the Master Data Services Add-in for Excel.

The main tasks for this exercise are as follows:

1. Add Free-Form Attributes to an Entity

2. Create an Entity for a Domain-Based Attribute

▶ Task 1: Add Free-Form Attributes to an Entity

1. Start Microsoft® Excel®, create a new blank workbook, and enable the Master Data Services add-in.
2. Create a Master Data Services connection named **Local MDS Server** for the MDS server at **http://localhost:81/mds**.
3. Connect to the existing **Local MDS Server** and load the **Product** entity from the **Product Catalog** model.
4. Add the following attributes to the **Product** entity by entering the attribute name in the column heading cell (starting with the cell to the right of the existing **ListPrice** attribute heading), and then selecting the new heading cell, clicking **Attribute Properties** in the ribbon, and editing the properties of the attribute:

Attribute Name	Attribute Properties
StandardCost	Number (4 decimal places)
Color	Text (maximum length: 15)
SafetyStockLevel	Number (0 decimal places)
ReorderPoint	Number (0 decimal places)
Size	Text (maximum length: 50)
Weight	Number (4 decimal places)
DaysToManufacture	Number (0 decimal places)
ModelName	Text (maximum length: 500)

5. After you have added and configured the attribute headings, enter the following attribute values:
    - Sport-100 Helmet, Red
        - StandardCost: 13.0863
        - Color: Red
        - SafetyStockLevel: 3
        - ReorderPoint: 4
        - Size: U
        - Weight: 0.2000
        - DaysToManufacture: 5
        - ModelName: Sport-100
    - Mountain-100 Silver, 42
        - StandardCost: 1912.1544
        - Color: Silver
        - SafetyStockLevel: 100
        - ReorderPoint: 75
        - Size: L
        - Weight: 20.77
        - DaysToManufacture: 4
        - ModelName: Mountain-100
    - Full-Finger Gloves, S
        - StandardCost: 15.6709
        - Color: Black
        - SafetyStockLevel: 4
        - ReorderPoint: 3
        - Size: S
        - Weight: 0.2000
        - DaysToManufacture: 0
        - ModelName: Full-Finger Gloves
    - Chain
        - StandardCost: 8.9866
        - Color: Silver
        - SafetyStockLevel: 500
        - ReorderPoint: 375
        - Size: (leave blank)
        - Weight: (leave blank)
        - DaysToManufacture: 1

- ModelName: Chain

6. Publish the changes you have made to Master Data Services, entering an appropriate annotation when prompted.

▶ Task 2: Create an Entity for a Domain-Based Attribute

1. Connect to the **Local MDS Server** and load data from the **Product Subcategory** entity in the **Product Catalog** model.
2. Add a new attribute named **ProductCategoryKey** to the right of the attribute heading for the **Code** attribute. Do not configure the attribute properties yet.
3. Enter the following **ProductCategoryKey** values for the **Product Subcategory** members:
   - Mountain Bikes: 1
   - Gloves: 3
   - Helmets: 4
   - Chains: 2
4. Edit the attribute properties for the **ProductCategoryKey** attribute, and configure it as follows:
   - Attribute type: **Constrained list (Domain-based)**.
   - Populate the attribute with values from the selected column.
   - Create a new entity named **Product Category**.
5. Connect to the **Local MDS Server** connection and load data from the newly-created **Product Category** entity in the **Product Catalog** model. The **Name** and **Code** values for the entities in this entity have been copied from the distinct **ProductCategoryKey** attribute values in the **Product Subcategory** entity.
6. Change the **Name** value for the **Product Category** members as follows, and publish the changes:

Name	Code
Bikes	1
Components	2
Clothing	3
Accessories	4

7. On the **Product Subcategory** worksheet tab, refresh the entity data and verify that the product category names are now listed in the **ProductCategoryKey** attribute column.

Results: After this exercise, you should have a master data model that contains **Product**, **ProductSubcategory**, and **ProductCategory** entities that contain data entered in Excel.

Exercise 3: Enforcing Business Rules

Scenario

Users noticed inconsistencies in the product data, including missing list prices and invalid safety stock levels. You intend to create business rules that enable data stewards to identify inconsistent data.

The main tasks for this exercise are as follows:

1. Create a Rule for the ListPrice Attribute
2. Create a Rule for the SafetyStockLevel Attribute
3. Publish Business Rules and Validate Data

▶ Task 1: Create a Rule for the ListPrice Attribute

1. In the Master Data Manager Web application at **http://localhost:81/mds**, in the **System Administration** area, manage business rules and create a rule called **Validate Price** for the **Product** entity that checks values in the **ListPrice** column are greater than 0.
    - The business rule does not require an IF clause, only a THEN clause.
    - Use a **must be greater than** validation action that compares the **ListPrice** attribute to the value 0.

▶ Task 2: Create a Rule for the SafetyStockLevel Attribute

1. Create a rule called **Validate SafetyStockLevel** for the **Product** entity that checks the safety stock level is greater than reorder point for products taking a day or more to manufacture.
    - Create an IF condition with an **is greater than** value comparison that checks that the **DaysToManufacture** attribute is greater than **0**.
    - After creating the IF condition, save the item and add a THEN clause with a **must be greater than** validation action that checks that the **SafetyStockLevel** attribute is greater than the **ReorderPoint** attribute.
    - The completed rule should be based on the following expression:

```
IF DaysToManufacture is greater than 0
THEN SafetyStockLevel must be greater than ReorderPoint
```

▶ Task 3: Publish Business Rules and Validate Data

1. Publish the business rules.
2. In Master Data Manager, in the **Explorer** area, view the **Product** entity members and click **Apply Rules** to validate the members.
    - Note that the **Sport-100 Helmet, Red** member is invalid.
    - Fix the invalid product member by changing the **SafetyStockLevel** attribute value to **5**.
3. In Excel, connect to the **Local MDS Server** connection and load the **Product** entity from the **Product Catalog** model.
4. Display the **$ValidationStatus$** and **$InputStatus$** columns by clicking **Show Status** on the **Master Data** tab of the ribbon, and then click **Apply Rules** to verify that all members are valid.

5. Change the value in the **ListPrice** column for the **Chain** product to **0.00**, and then publish the changes.
   - Note that Excel highlights the invalid member.
   - Fix the member by changing the price back to **20.24** and publishing the changes.

Results: After this exercise, you should have a master data model that includes business rules to validate product data.

Exercise 4: Loading Data into a Model

Scenario

Now that the model is complete, you will populate it by using Transact-SQL scripts.

The main tasks for this exercise are as follows:

1. Load Data into the Model
2. Check the Status of the Data Load
3. Validate Imported Members

▶ Task 1: Load Data into the Model

1. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance using Windows authentication.
2. Open the **Import Products into MDS.sql** file in D:\Labfiles\Lab10\Starter and examine the Transact-SQL code it contains. The code performs the following tasks:
   - Generates a unique batch ID.
   - Inserts data into the **stg.Product_Category_Leaf** staging table from the **ProductCategory** table in the **Products** database, specifying an **ImportType** value of 2, an **ImportStatus_ID** value of 0, and a **BatchTag** value that is based on the batch ID generated previously.
   - Inserts data into the **stg.Product_Subcategory_Leaf** staging table from the **ProductSubcategory** table in the **Products** database, specifying an **ImportType** value of 2, an **ImportStatus**_ID value of 0, and a **BatchTag** value based on the batch ID generated previously.
   - Inserts data into the **stg.Product_Leaf** staging table from the **Product** table in the **Products** database, specifying an **ImportType** value of 2, an **ImportStatus**_ID value of 0, and a **BatchTag** value based on the batch ID generated previously.
   - Executes the **stg.udp_Product_Category_Leaf** stored procedure to start processing the staged **Product Category** members with the batch tag specified previously.
   - Executes the **stg.udp_Product_Subcategory_Leaf** stored procedure to start processing the staged **Product Subcategory** members with the batch tag specified previously.
   - Executes the **stg.udp_Product_Leaf** stored procedure to start processing the staged **Product** members with the batch tag specified previously.
3. Execute the script to stage the data and start the import process.

▶ Task 2: Check the Status of the Data Load

1. In the Master Data Manager Web application, in the **Integration Management** area, verify that the data load process is complete for all entities with no errors.

▶ Task 3: Validate Imported Members

1. In the Master Data Manager Web application, in the **Explorer** area, view the **Product** entity members and apply rules to validate the data.
2. After you have applied the business rules, filter the list of members on the following criteria so that only invalid records are shown:

Attribute	Operator	Criteria
[Validation Status]	Is equal to	Validation Failed

3. Review the invalid records (which have an invalid **ListPrice** attribute), and resolve them by changing the **ListPrice** attribute value to **1431.50**.
4. When you have resolved all the invalid members, clear the filter.

Results: After this exercise, you should have loaded members into the Master Data Services model.

Exercise 5: Consuming Master Data Services Data

Scenario

You need to make Master Data Services available to other applications. You will create subscription views and then modify an Integration Services package to use Master Data Services data.

The main tasks for this exercise are as follows:

1. Create Subscription Views
2. Query a Subscription View by Using Transact-SQL

▶ Task 1: Create Subscription Views

1. In the Master Data Manager Web application at **http://localhost:81/mds**, view the **Integration Management** area.
2. Use the values in the following table to create views:

Subscription View Name	Model	Version	Entity	Format
Products	Product Catalog	VERSION_1	Product	Leaf members
ProductCategories	Product Catalog	VERSION_1	Product Category	Leaf members
ProductSubcategories	Product Catalog	VERSION_1	Product Subcategory	Leaf members

▶ Task 2: Query a Subscription View by Using Transact-SQL

1. Use the Transact-SQL script in the Query **Subscription Views.sql** file in the D:\Labfiles\Lab10\Starter folder to query the subscription views, and verify that they return product data.

Results: After this exercise, you should have three subscription views that you can use to query product data from Master Data Services.

Module Review and Takeaways

In this module, you have learned how to use Master Data Services to create and manage a master data model for business entity definitions.

Review Question(s)

Question: In your experience, do you think that business users in a data steward capacity will be mostly comfortable using the web-based interface, the Excel add-in, or a combination of both tools to manage master data?

Module 11

Extending SQL Server Integration Services

Contents:

Module Overview

Microsoft® SQL Server® Integration Services (SSIS) provides a comprehensive collection of tasks, connection managers, data flow components, and other items you can use to build an extract, transform, and load (ETL) process for a data warehousing solution. However, there may be some cases where the specific requirements of your organization mean that you must extend SSIS to include some custom functionality.

This module describes the techniques you can use to extend SSIS. The module is not designed to be a comprehensive guide to developing custom SSIS solutions, but to provide an awareness of the fundamental steps required to use custom components and scripts in an ETL process, based on SSIS.

Objectives

After completing this module, you will be able to:

- Include custom scripts in an SSIS package.
- Describe how custom components can be used to extend SSIS.

Lesson 1
Using Scripts in SSIS

If the built-in SSIS components do not meet your requirements, you might be able to implement the functionality you need using a script.

This lesson describes the kinds of functionality you can implement with a script, and how you can use the control flow script task and data flow script component in a package.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the kinds of functionality you can implement with a script.
- Implement a script task in a package control flow.
- Implement a script component in a data flow.

Introduction to Scripting in SSIS

SSIS includes support for adding custom functionality to a package by creating scripts in Microsoft Visual Basic® or Microsoft Visual C#® code. You can add a script to a package and then configure its properties in the SSIS Designer before opening it in the Microsoft Visual Studio Tools for Applications (VSTA) development environment to edit it. When you include a script in a package, it is precompiled to optimize execution performance.

- Implement custom functionality with Visual Basic or Visual C# code
- Configure properties in the package designer, and develop code using the Visual Studio Tools for Applications (VSTA) environment
- Scripts are precompiled for optimal execution performance
- Two scripting scenarios
  - Use the *Script Task* to implement a custom control flow task
  - Use the *Script Component* to implement a custom data flow source, transformation, or destination

SSIS supports the following two scripting scenarios in a package:

- Use the *Script Task* to implement a custom control flow task.
- Use the *Script Component* to implement a custom data flow source, transformation, or destination.

Using the Control Flow Script Task

The script task provides a scriptable wrapper for the **Microsoft.SqlServer.Dts.Runtime.Task** base class, and enables you to create a custom control flow task without having to create a custom assembly. To use the script task, you must perform the following steps:

1. Add the script task to the control flow.

1. Add the task to the control flow
   - Configure properties and logging settings as required
2. Configure the task
   - Language
   - Entry point method
   - Access to variables
3. Implement the code
   - Use the **Dts** object to access package and runtime functionality

```
Public void Main()
{
  Dts.Variables["User::MyVar"].Value =
            Dts.Variables["System::StartTime"].Value.ToString();
  Dts.Log("Package started:" +
            (string)Dts.Variables["User::MyVar"].Value, 999, null);
  Dts.TaskResult = (int)ScriptResults.Success;
}
```

To create a scripted custom control flow task, add the script task to a control flow and configure its standard properties, such as **Name**, in the Properties pane. If you want to use the script task to write information to a log file, you must also enable logging for the task and specify the events to be logged. The script task supports the same events as other control flow tasks as well as a **ScriptTaskLogEntry** event, which must be enabled if you want to use the script to write log entries programmatically.

2. Configure the task.

   Use the Script Task Editor dialog box to configure the following settings for the task:

   - **ScriptLanguage** – the programming language in which you want to write the script. This can be Microsoft Visual Basic® 2010 or Microsoft Visual C#® 2010.
   - **EntryPoint** – the method in the script that should be called when the script is executed. By default, this is a method named Main.
   - **ReadOnlyVariables** and **ReadWriteVariables** – the package variables and parameters that you want to make accessible to the script. By default, the script task cannot access any package variables, so you must explicitly grant access to any variables you want to use in the script.

3. Implement the code.

   After you have configured the script task, you can click **Edit Script** to open the VSTA environment and implement the code for your script. You can use standard .NET programming structures and syntax in your code, adding references to additional assemblies if required. You can also make use of the **Dts** object, which provides a number of static properties and methods to help you access package and SSIS runtime functionality. For example, the following code uses the **Dts.Variables** property to access package variables, the **Dts.Log** method to write a custom log entry, and the **Dts.TaskResult** property to specify the control flow result for the script task:

```
Public void Main()
{
  Dts.Variables["User::MyVar"].Value = Dts.Variables["System::StartTime"].Value.ToString();
  Dts.Log("Package started:" + (string)Dts.Variables["User::MyVar"].Value, 999, null);
  Dts.TaskResult = (int)ScriptResults.Success;
}
```

 **Additional Reading:** For more information about *Using the Script Task to Extend a Package*, go to http://go.microsoft.com/fwlink/?LinkID=246743.

Demonstration: Implementing a Script Task

In this demonstration, you will see how to:

- Configure a Script Task.
- Implement a Script Task.

Demonstration Steps

Configure a Script Task

1. Ensure 20463C-MIA-DC and 20463C-MIA-SQL are started, and log onto 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. Start Visual Studio and open the **ScriptDemo.sln** solution in the D:\Demofiles\Mod11 folder.

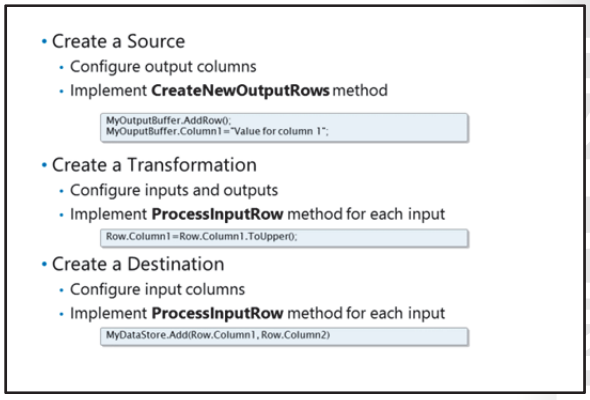
3. In Solution Explorer, double-click the **Package.dtsx** SSIS package to open it in the SSIS designer. Then view the control flow, and notice that it includes a script task.
4. On the **SSIS** menu, click **Logging**, then on the **Providers and Logs** tab, in the **Provider type** list, select **SSIS log provider for Windows Event Log** and click **Add**.
5. In the **Containers** tree, select the **Script Task** checkbox, and in the **Providers and Logs** tab select the **SSIS log provider for Windows Event Log** checkbox. Then on the **Details** tab, select the **ScriptTaskLogEntry** checkbox, and click **OK**.
6. On the control flow surface, double-click **Script Task**. Then, in the **Script Task Editor** dialog box, verify that the following settings are selected:
   - **ScriptLanguage**: Microsoft Visual C# 2012
   - **EntryPoint**: Main
   - **ReadOnlyVariables**: User::Results and System::StartTime
   - **ReadWriteVariables**: User::MyVar

Implement a Script Task

1. In the **Script Task Editor** dialog box, click **Edit Script**, and wait for the Visual Studio VstaProjects editor to open. If a message that the Visual C++ Language Manager Package did not load correctly is displayed, prompting you to continue to show this error, click **No**.
2. In the **VstaProjects – Microsoft Visual Studio** window, scroll through the **ScriptMain.cs** code until you can see the whole of the **public void Main()** method. Note that the code performs the following tasks:
   - Assigns the value of the **System::StartTime** variable to the **User::MyVar** variable.
   - Writes the value of the **User::MyVar** variable to the log.
   - Creates a string that contains the values in the **User::Results** variable (which is an array of strings), and displays it in a message box.
3. Close the **VstaProjects – Microsoft Visual Studio** window. Then in the **Script Task Editor** dialog box, click **Cancel**.
4. On the **Debug** menu, click **Start Debugging**, and observe the package as it executes. When the **Results** message box is displayed, click **OK**.
5. When package execution has completed, on the **Debug** menu, click **Stop Debugging**. Then minimize Visual Studio.
6. Right-click the **Start** button and click **Event Viewer**. Then in Event Viewer, expand the **Windows Logs** folder, click **Application**, select the first Information event with the source **SQLISPackage120**, and in the **General** tab, note that it contains the start time of the package, which was written by the script component.
7. Close Event Viewer.

Using the Data Flow Script Component

The script component uses classes in the **Microsoft.SqlServer.Dts.Pipeline**, **Microsoft.SqlServer.Dts.Pipeline.Wrapper**, and **Microsoft.SqlServer.Dts.Runtime.Wrapper** namespaces to enable you to implement a data flow component without the need to create a custom assembly.



Using the Script Component to Create a Source

To create a custom source, add the script component to a data flow and when prompted, select **Source**. Then, as a minimum, you must perform the following actions:

- Configure the output columns – a source must provide at least one output, and each output must contain at least one column. You must specify the name and data type for each column that your source will pass to the next component in the data flow.
- Implement the **CreateNewOutputRows** method – each output provides a buffer object that you can use to generate a new row and assign column values.

Using the Script Component to Create a Transformation

To create a custom transformation, add the script component to a data flow, and when prompted, select **Transformation**. Then, as a minimum, you must perform the following actions:

- Configure inputs and outputs – a transformation can have multiple inputs and outputs, and you must define the columns for each one as required.
- Implement the **ProcessInputRow** method for each input – the script component generates a **ProcessInputRow** method for each input you define. The name of the method is derived from the input name. For example, an input named **Input0** will result in a method named **Input0_ProcessInputRow**. The method is called for each row in the input, and you must add code to process the columns in the row. If your component has multiple outputs, direct them to the appropriate output.

Using the Script Component to Create a Destination

To create a custom destination, add the script component to a data flow, and when prompted, select **Destination**. Then, as a minimum, you must perform the following actions:

- Configure input columns – a destination must have at least one input, and each input must contain at least one column. You must specify the name and data type for each column that your destination will accept from the preceding component in the data flow.
- Implement the **ProcessInputRow** method for each input – just as for transformations, the script component generates a **ProcessInputRow** method for each input you define in a destination. The method is called for each row in the input, and you must add code to process a row, usually by adding it to a data store.

Additional Configuration Settings and Methods

In addition to the tasks described above, you can use the following configuration settings and methods to enhance the functionality of your scripted data flow component:

- Add variables – you can enable access to variables in a script component by setting the **ReadOnlyVariables** and **ReadWriteVariables** properties.
- Add a connection manager – you can add one or more connection managers to a script component, and use the connection manager in your code to access a database or other data store. For example, you could add a connection manager to a custom source, enabling you to retrieve the data for the output from a database.
- Override the **AcquireConnections** method – this method enables you to open a connection from a connection manager in preparation for execution.
- Override the **PreExecute** method – this method enables you to perform any preparation tasks for the component. For example, you could use this method in a custom source to retrieve a **SqlDataReader** object from a connection you opened in the **AcquireConnections** method.
- Override the **PostExecute** method – this method enables you to perform any clean-up tasks when execution is complete. For example, you could use this method to dispose of a **SqlDataReader** object you opened during the **PreExecute** method.
- Override the **ReleaseConnections** method – this method enables you to close any connections you opened in the **AcquireConnections** method.

 **Additional Reading:** For more information about *Extending the Data Flow with the Script Component*, go http://go.microsoft.com/fwlink/?LinkID=246744.

Demonstration: Using a Script Component in a Data Flow

In this demonstration, you will see how to:

- Implement a Source.
- Implement a Transformation.
- Implement a Destination.

Demonstration Steps

Implement a Source

1. Ensure you have completed the previous demonstration in this module.
2. Maximize Visual Studio, and view the control flow of the **Package.dtsx** SSIS package.
3. On the control flow surface, double-click **Data Flow Task** to view the data flow.
4. On the data flow surface, double-click **Script Source**.
5. In the **Script Transformation Editor** dialog box, on the **Inputs and Outputs** page, expand **MyOutput** and the **Output Columns** folder. Then, select **Column1** and **Column2** in turn and note the **DataType** property of each column.
6. In the **Script Transformation Editor** dialog box, on the **Script** page, click **Edit Script**, and wait for the Visual Studio VstaProjects editor to open. If a message that the Visual C++ Language Manager Package did not load correctly is displayed, prompting you to continue to show this error, click **No**.

7. In the **VstaProjects – Microsoft Visual Studio** window, view the script. Note that the **CreateNewOutputRows** method uses a loop to create six rows of data. Then close the **VstaProjects – Microsoft Visual Studio** window, and in the **Script Transformation Editor** dialog box, click **Cancel**.

Implement a Transformation

1. On the data flow surface, double-click **Script Transformation**.
2. In the **Script Transformation Editor** dialog box, on the **Inputs and Outputs** page, note that the component has a single input named **Input 0**, which consists of two columns named **Column1** and **Column2**. There is no output defined for the component, which means that it will pass the columns in the input buffer to the output buffer without adding any new columns.
3. In the **Script Transformation Editor** dialog box, on the **Input Columns** page, note that both **Column1** and **Column2** are selected and that the **Usage Type** for **Column1** is **ReadOnly**, while the **Usage Type** for **Column2** is **ReadWrite**. This configuration enables the script to make changes to **Column2** as it flows through the pipeline.
4. In the **Script Transformation Editor** dialog box, on the **Script** page, click **Edit Script**, and wait for the Visual Studio VstaProjects editor to open. If a message that the Visual C++ Language Manager Package did not load correctly is displayed, prompting you to continue to show this error, click **No**.
5. In the **VstaProjects – Microsoft Visual Studio** window, view the script. Note that the **Input0_ProcessInoutRow** method modifies **Column2** by making its value upper case. Then close the **VstaProjects – Microsoft Visual Studio** window, and in the **Script Transformation Editor** dialog box, click **Cancel**.

Implement a Destination

1. On the data flow surface, double-click **Script Destination**.
2. In the **Script Transformation Editor** dialog box, on the **Inputs and Outputs** page, note that the component has a single input named **Input 0**, which consists of two columns named **Column1** and **Column2**.
3. In the **Script Transformation Editor** dialog box, on the **Input Columns** page, note that both **Column1** and **Column2** are selected and that the **Usage Type** for both columns is **ReadOnly**. Since the component represents a destination, there is no need to modify the rows in the buffers.
4. In the **Script Transformation Editor** dialog box, on the **Script** page, note that the **ReadWriteVariables** property allows access to the **User::Results** variable, and then click **Edit Script** and wait for the Visual Studio VstaProjects editor to open. If a message that the Visual C++ Language Manager Package did not load correctly is displayed, prompting you to continue to show this error, click **No**.
5. In the **VstaProjects – Microsoft Visual Studio** window, view the script, and note the following details:
   - The **PreExecute** method initializes an array variable for six string elements.
   - The **Input0_ProcessInputRow** method adds the value of **Column2** to the next available empty element in the array.
   - The **PostExecute** method assigns the array variable in the script to the **User::Results** package variable.
6. Close the **VstaProjects – Microsoft Visual Studio** window, and in the **Script Transformation Editor** dialog box, click **Cancel**.

7. On the data flow surface, right-click the data flow path between **Script Source** and **Script Transformation**, and click **Enable Data Viewer**. Then repeat this step for the data flow path between **Script Transformation** and **Script Destination**.
8. On the **Debug** menu, click Start **Debugging**.
9. When the first data viewer window is displayed, view the rows in the pipeline and click the green **Continue** button. These rows were generated by the **Script Source** component.
10. When the second data viewer window is displayed, view the rows in the pipeline and click the green **Continue** button. Note that **Column2** was formatted as upper case by the **Script Transformation** component.
11. When the **Results** message box is displayed, note that it contains the **Column2** values that were passed to the **Script Destination** component. You may need to click the program icon on the task bar to bring the message box to the front of the open windows.
12. When package execution has completed, on the **Debug** menu, click **Stop Debugging**. Then close Visual Studio.

Lesson 2
Using Custom Components in SSIS

When the built-in functionality in SSIS does not fully meet your requirements, and you cannot achieve your goal by using a script component, you can obtain or create custom components, add them to the SSIS package designer, and use them in your packages. This lesson describes the types of functionality that you can implement with custom components, and how to create, install, and use them.

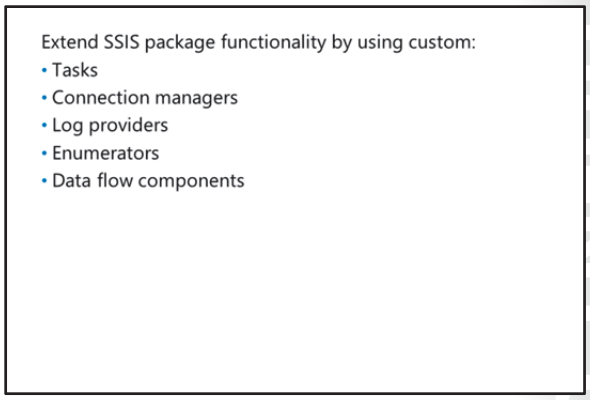
Lesson Objectives

After completing this lesson, you will be able to:

- Describe how custom components can be used to extend SSIS.
- Describe how to implement a custom component.
- Install and use a custom component.

Introduction to Custom Components

The control flow tasks, connection managers, data flow transformations, and other items in an SSIS package are all managed components. In other words, each item you can use when creating an SSIS package is implemented as a Microsoft .NET assembly.



SSIS has been designed as an extensible platform, enabling you to supplement the built-in components with custom assemblies that perform tasks that cannot otherwise be achieved. You can extend SSIS with the following kinds of custom components:

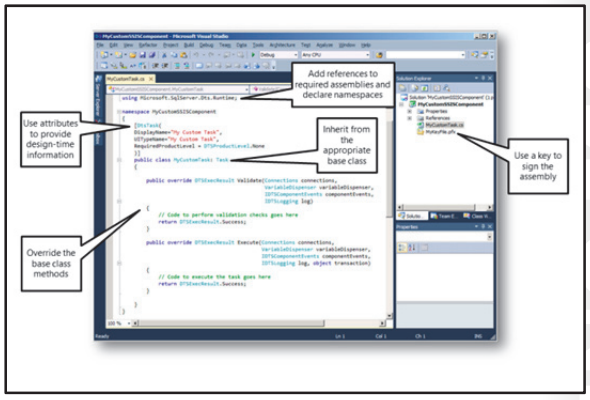
- Control flow tasks
- Connection managers
- Log providers
- Enumerators
- Data flow components, including sources, transformations, and destinations

You can develop your own custom SSIS components in a programming tool that targets the .NET Framework, such as Microsoft Visual Studio®. Alternatively, you can commission a professional software developer to create custom components for you, or obtain them from third-party vendors.

Implementing a Custom Component

Software developers can use the following process to implement a custom component for SSIS:

1. Create a class library project and add references to the relevant SSIS assemblies.



The SSIS assemblies provide access to SSIS task flow and pipeline runtime functionality, and are provided with the SQL Server Client SDK. This enables you to add a reference to them in your custom component project. The specific assemblies you need to reference depend on the type of custom component you are implementing, as described in the following list:

- **Microsoft.SqlServer.ManagedDTS.dll** – the managed runtime engine, used by managed control flow tasks, enumerators, log providers, and connection managers.
- **Microsoft.SqlServer.PipelineHost.dll** – the managed data flow engine, used by managed data flow sources, transformations, and destinations.
- **Microsoft.SqlServer.RuntimeWrapper.dll** – the primary interop assembly (PIA) for the native SSIS runtime engine.
- **Microsoft.SqlServer.PipelineWrapper.dll** – the PIA for the native SSIS data flow engine.

The base class assemblies include namespaces that contain the base classes your custom components must implement. For example, the Microsoft.SqlServer.ManagedDTS.dll includes the Microsoft.SqlServer.Dts.Runtime namespace, which contains the Task base class for control flow tasks.

2. Create a class that inherits from the appropriate SSIS base class, as described in the following table:

Component Type	Base Class
Task	Microsoft.SqlServer.Dts.Runtime.Task
Connection manager	Microsoft.SqlServer.Dts.Runtime.ConnectionManagerBase
Log provider	Microsoft.SqlServer.Dts.Runtime.LogProviderBase
Enumerator	Microsoft.SqlServer.Dts.Runtime.ForEachEnumerator
Data flow component	Microsoft.SqlServer.Dts.Pipeline.PipelineComponent

3. Add attributes to the class declaration to provide design-time information about the component that can be displayed in SQL Server Data Tools when using the component in an SSIS package.

   The class-specific attributes are listed in the following table:

Base Class	Attribute
Microsoft.SqlServer.Dts.Runtime.Task	DtsTaskAttribute
Microsoft.SqlServer.Dts.Runtime.ConnectionManagerBase	DtsConnectionAttribute
Microsoft.SqlServer.Dts.Runtime.LogProviderBase	DtsLogProviderAttribute
Microsoft.SqlServer.Dts.Runtime.ForEachEnumerator	DtsForEachEnumeratorAttribute
Microsoft.SqlServer.Dts.Pipeline.PipelineComponent	DtsPipelineComponentAttribute

4. Override the base class methods to implement the required custom functionality.

   Each base class includes methods that are called by the runtime engine when the component is used in an SSIS package. Some of the most important methods for each base class are listed in the following table:

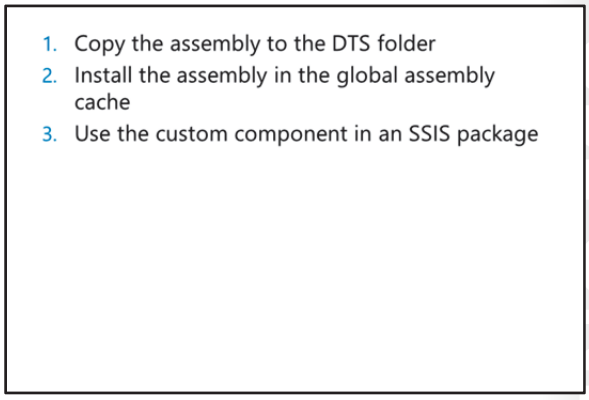
Base Class	Methods
Microsoft.SqlServer.Dts.Runtime.Task	Validate, Execute
Microsoft.SqlServer.Dts.Runtime.ConnectionManagerBase	AcquireConnection, ReleaseConnection
Microsoft.SqlServer.Dts.Runtime.LogProviderBase	OpenLog, Log, CloseLog
Microsoft.SqlServer.Dts.Runtime.ForEachEnumerator	GetEnumerator
Microsoft.SqlServer.Dts.Pipeline.PipelineComponent	PipeLineComponentProperties, PrimeOutput, ProcessInput

5. Sign the assembly to generate a strong name.

   Custom SSIS component assemblies are added to the global assembly cache (GAC), which requires that each assembly has a strong name to uniquely identify it. A strong name for an assembly is generated by signing the assembly with a cryptographic key when it is compiled.
   You can easily generate a key and sign an assembly on the **Signing** tab of the **Project Properties** dialog box in Visual Studio.

 **Reference Links:** For more information about *Extending Packages with Custom Objects*, go to http://go.microsoft.com/fwlink/?LinkID=246742.

Installing and Using a Custom Component



Most commercially-available custom SSIS components provide a setup program that installs the assemblies and enables you to use them in SQL Server Data Tools. However, as a business intelligence (BI) professional, you should be able to perform the necessary manual steps to install a custom component you have developed or had created for you by a software developer.

You can use the following procedure to install and use a custom SSIS component:

1. Copy the assembly file to the appropriate Data Transformation Services (DTS) subfolder.

   The SSIS Designer looks for components in a specific subfolder of the DTS folder for the SQL Server installation. By default, the DTS folder is located in C:\Program Files\Microsoft SQL Server\<*version*>\DTS. For 64-bit installations, this folder is used for 64-bit components, and 32-bit components are stored in subfolders of C:\Program Files (x86)\Microsoft SQL Server\<*version*>\DTS. The component type-specific subfolders are listed in the following table:

Component Type	Folder
Task	Tasks
Connection manager	Connections
Log provider	LogProviders
Enumerator	ForEachEnumerators
Data flow component	PipelineComponents

2. Install the assembly in the GAC.

   You can do this by using the gacutil.exe command line tool. The gacutil.exe tool is provided with Visual Studio and Windows® SDK. You can use the following syntax to install an assembly in the GAC:

```
gacutil /i assembly_name.dll
```

3. Use the custom component in an SSIS package.

   After you have deployed a custom component to the correct folder and installed it in the GAC, it will be available in SSIS Designer for use in a package. For example, if you have successfully deployed an assembly for a custom control flow task component, the task will be listed in the SSIS Toolbox pane and can be dragged to the control flow surface of a package.

Lab: Using Custom Scripts

Scenario

The senior database administrator has requested that the data warehouse ETL process logs the number of rows extracted from the staging database and loaded into the data warehouse in the Windows event log. You have decided to use a custom script to accomplish this.

Objectives

After completing this lab, you will be able to:

- Use a Script task.

Estimated Time: 30 minutes

Virtual machine: 20463C-MIA-SQL

User name: ADVENTUREWORKS\Student

Password: Pa$$w0rd

Exercise 1: Using a Script Task

Scenario

You have created an SSIS package that loads data from the staging database into the data warehouse, and writes the number of rows processed in each staging table to a package variable before truncating the staging tables. You want to add a script task that writes the row count variables to the Windows event log when the data warehouse load operation is complete.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Add a Script Task to a Control Flow
3. Enable Logging for the Script Task
4. Configure the Script Task
5. Test the Script

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. Run Setup.cmd in the D:\Labfiles\Lab11\Starter folder as Administrator.

▶ Task 2: Add a Script Task to a Control Flow

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab11\Starter folder.
2. Open the **Load DW.dtsx** package and review the control flow.
3. View the variables defined in the package. These are used by the **Get record counts and truncate Staging tables** SQL Command task to store the row counts for each staging table that was processed during the data warehouse load operation.
4. Add a Script Task to the control flow and name it **Log Rowcounts**. Then connect the success precedence arrow from the **Get record counts and truncate Staging tables** task to the **Log Rowcounts** task.

▶ Task 3: Enable Logging for the Script Task

1. Configure logging for the package as follows:
   - Add the **SSIS Log Provider for Windows Event Log** provider.
   - Enable logging for the Log **Rowcounts** script task and select the **SSIS Log Provider for Windows Event Log** provider.
   - Enable logging for the **ScriptTaskLogEntry** event of the **Log Rowcounts** task.

▶ Task 4: Configure the Script Task

1. Configure the **Log Rowcounts** script task with the following settings:
   - **ScriptLanguage**: Microsoft Visual C# 2012
   - **EntryPoint**: Main
   - **ReadOnlyVariables**: User::CustomerCount, User::EmployeeCount, User:InternetSalesCount, User::PaymentCount, User::ResellerCount, and User::ResellerSalesCount
   - **ReadWriteVariables**: None
2. Edit the script, and replace the comment **//TODO: Add your code here** with the following code (above the existing **Dts.TaskResult = (int)ScriptResults.Success** statement). You can copy and paste this code from **Script.txt** in the D:\Labfiles\Lab11\Starter folder:

```
String logEntry = "Data Warehouse records loaded (";
logEntry += Dts.Variables["User::CustomerCount"].Value.ToString() + " customers, ";
logEntry += Dts.Variables["User::ResellerCount"].Value.ToString() + " resellers, ";
logEntry += Dts.Variables["User::EmployeeCount"].Value.ToString() + " employees, ";
logEntry += Dts.Variables["User::PaymentCount"].Value.ToString() + " payments, ";
logEntry += Dts.Variables["User::InternetSalesCount"].Value.ToString() + " Internet sales, and ";
logEntry += Dts.Variables["User::ResellerSalesCount"].Value.ToString() + " reseller sales) ";
Dts.Log(logEntry, 999, null);
```

▶ Task 5: Test the Script

1. Start debugging the **Load DW** package and observe the control flow tab as it executes, noting that each package executed opens in a new window. The entire load process can take a few minutes.
2. When execution is complete, stop debugging and close Visual Studio.
3. View the Windows Application log in the Event Viewer administrative tool and verify that the most recent information event with a source of **SQLISPackage120** contains the row counts for each staged table.

Results: After this exercise, you should have an SSIS package that uses a script task to log row counts.

Module Review and Takeaways

In this module, you have learned how to extend SSIS with custom script tasks and custom components.

Review Question(s)

Question: What might you consider when deciding whether to implement a custom process as a script or a custom component?

Module 12

Deploying and Configuring SSIS Packages

Contents:

Module Overview

Microsoft® SQL Server® Integration Services (SSIS) provides tools that make it easy to deploy packages to another computer. The deployment tools also manage any dependencies, such as configurations and files that the package needs. In this module, you will learn how to use these tools to install packages and their dependencies on a target computer.

Objectives

After completing this module, you will be able to:

- Describe considerations for SSIS deployment.
- Deploy SSIS projects.
- Plan SSIS package execution.

Lesson 1
Overview of SSIS Deployment

SQL Server 2014 supports two deployment models for SSIS packages, and choosing the right one can greatly simplify the maintenance and operation of your ETL solution. This lesson discusses the two deployment models and compares their key features.

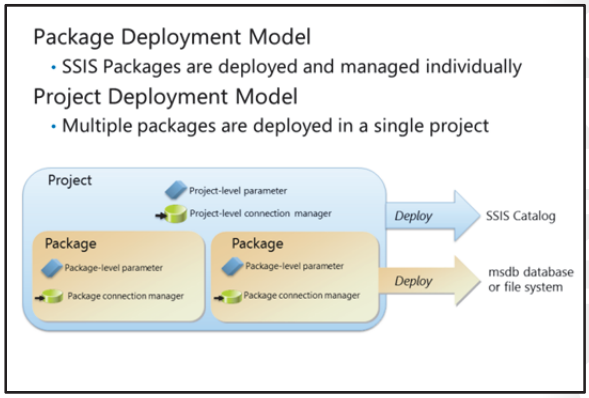
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the SSIS deployment model infrastructure.
- Explain the package deployment model.
- Describe the project deployment model.
- Compare the two deployment models.

SSIS Deployment Models

In Microsoft® SQL Server® 2014 Integration Services there are two deployment models, one for packages another for projects.



The Package Deployment Model

The package deployment model is used in previous releases of SSIS. When using the package deployment, SSIS packages in a solution are deployed and managed individually. You can deploy packages to the **msdb** database on an instance of SQL Server, or to the file system. The package deployment model provides support and continued development of existing SSIS packages from previous versions of SQL Server.

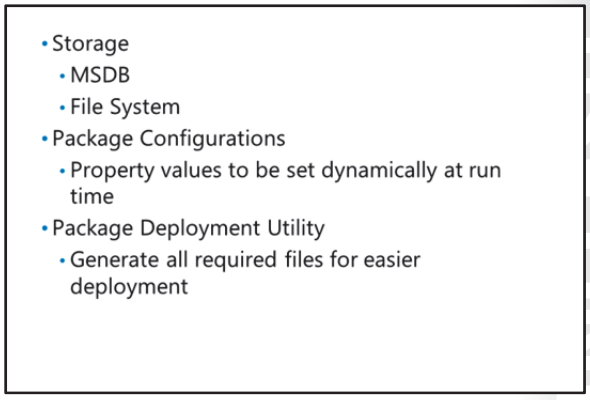
The Project Deployment Model

The project deployment model was first introduced in SQL Server 2012 and enables you to deploy a project that contains multiple packages as a single unit. You can then manage connection managers and parameters that are shared by all packages in the project. When you use the project deployment model, you must create an SSIS catalog on an instance of SQL Server 2012 or later and deploy your project to there.

Package Deployment Model

The package deployment model is available to provide backwards compatibility. Other than this topic, this module focuses on the project deployment model.

By default, new SSIS projects are configured to use the project deployment model. To use the package deployment instead, you must click **Convert to Package Deployment Model** on the **Project** menu.



Storage

With the package deployment model, packages can be deployed to an instance of SQL Server or to the file system.

When you deploy packages to an instance of Integrations Services, you must decide if you will store the packages in SQL Server or in the file system.

You can deploy packages to SQL Server. SQL Server stores Integration Services packages in the **msdb** system database, and package dependencies in a file folder. When you deploy a package to SQL Server, you must specify the SQL Server instance and type of authentication.

When you deploy a package to the file system (as a .dtsx file), Integration Services accesses the package files and their dependencies from the folder where you deployed the packages. Alternatively, you can deploy packages to the SSIS package store. The SSIS package stores are directories related to the SSIS installation.

Package Configurations

A package configuration is a set of property/value pairs that is added to a completed package, enabling it to dynamically access property values at run time.

A package can retrieve property settings from a number of different source types, such as an XML file or Microsoft® SQL Server® database. You can update settings on any executable (including the package object), connection manager, log provider, or variable within a package.

When Integration Services runs the package, it extracts the property value from a source that is based on the configuration type. By using package configurations in projects you intend to deploy by using the package deployment model, you can define values at run time. Package configurations make it easier to move packages from development to the production environment.

To create a package configuration, open the package, and on the SSIS menu, click **Package Configurations**. You can then select where you want to save the configuration settings (for example, an XML file) and choose the properties to be included in the configuration.

Package Deployment Utilities

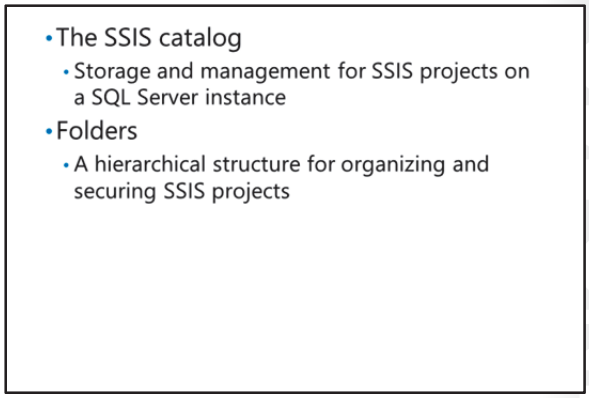
You can simplify package deployment and any configuration files or other resources used by creating a package deployment utility. To do this, first configure the package as required, including any package configurations, and then in the properties dialog box for the project, on the Deployment page, set the **CreateDeploymentUtility** property to **True**. When you build the project, a folder containing the packages, all the required files, and a manifest is generated.

Additional Reading: For more information about *Managing and Deploying SQL Server Integration Services*, go to http://go.microsoft.com/fwlink/?LinkID=246746.

Project Deployment Model

The project deployment model is the preferred model for SSIS projects in SQL Server 2014. There are many features only available with the project deployment model, making deployment, security, and execution more straightforward.

The SSIS catalog is a storage repository for SSIS projects. The catalog contains all folders, projects, packages, environments, and parameters for projects using the project deployment model.



The SSIS Catalog

The catalog must be created before first use. Initially you must enable CLR integration on the server, and then you can right-click **Integration Services** in SQL Server Management Studio and create the catalog. You can edit the properties in SQL Server Management Studio to define how data is encrypted. You can also define whether logs are cleared periodically, and if so how often. The maximum time allowed to perform project validation can be defined, too.

The catalog is stored as the **SSISDB** database in SQL Server. Although it is possible to perform some administration on this database, and it is useful for listing catalog-specific views and stored procedures, most administration is performed on the **SSISDB** node inside the **Integration Services** node.

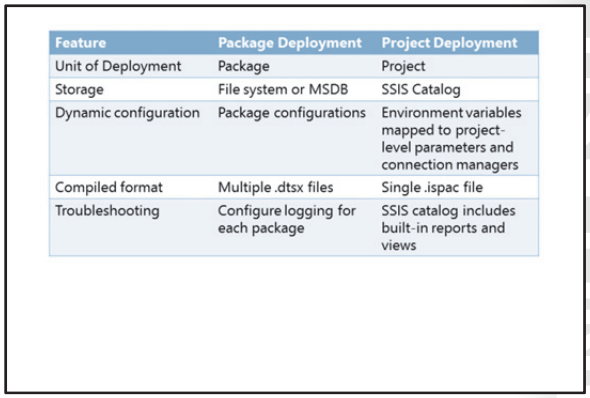
When you deploy a project with the same name as an existing one, it will be stored as a new version. You can set catalog properties to allow old versions to be cleaned up and define how many versions should be kept.

Folders

Folders are used to both organize and secure projects. Putting projects in folders makes it more straightforward to navigate to a specific project. However, folders are also securable objects so you can assign someone administrator rights to projects in specific folders without giving them any rights to projects situated outside.

Deployment Model Comparison

The following table lists the key differences between the package deployment and project deployment models:



Feature	Package Deployment	Project Deployment
Unit of Deployment	Package	Project
Storage	File system or MSDB	SSIS Catalog
Dynamic configuration	Package configurations	Environment variables mapped to project-level parameters and connection managers
Compiled format	Multiple .dtsx files	Single .ispac file
Troubleshooting	Configure logging for each package	SSIS catalog includes built-in reports and views

Feature	Package Deployment Model	Project Deployment Model
Unit of deployment	Individual packages	Project
Storage	Packages and all associated files can be copied to the file system of a local or remote computer. They can also be deployed to the MSDB database of an instance of SQL Server.	A project is deployed to the SSIS catalog, or a folder within the catalog, of an instance of SQL Server.
Dynamic configuration	Property values are stored in individual package configurations and assigned at run time.	Environment variables in the SSIS catalog are mapped to project-level parameters and connection managers.
Compiled format	Packages and associated resources are each stored as single files in the file system. The entire project might comprise of many files.	The entire project is compiled as a single file (with an .ispac extension).
Troubleshooting	To log events, you must add a log provider to the package and configure logging for each one individually.	Events are automatically logged and saved to the catalog. These events can then be displayed with views such as **catalog.executions** and **catalog.event_messages**.

Lesson 2
Deploying SSIS Projects

In this lesson you will learn how to deploy and manage an SSIS project.

Lesson Objectives

After completing this lesson, you will be able to:

- Create an SSIS catalog.
- Use environments and variables to set dynamic values at run time.
- Deploy an SSIS project.
- View package execution information.

Creating an SSIS Catalog

The first step in deploying an SSIS project is to create an SSIS catalog. However, an SSIS catalog can only be created on an SQL Server instance in which the SQL CLR feature is enabled, so if you have not enabled the SQL CLR, this will be done automatically when you create the SSIS catalog.

- Pre-requisites
  - SQL Server 2012 or later
  - SQL CLR enabled
- Creating a catalog
  - Use SQL Server Management Studio
  - One SSIS catalog per SQL Server instance
- Catalog Security
  - Folder Security
  - Object Security
  - Catalog Encryption
  - Sensitive Parameters

Creating an SSIS Catalog

To create an SSIS catalog, use SQL Server Management Studio to connect to the SQL Server instance on which you want to deploy SSIS projects, and in Object Explorer, right-click **Integration Services Catalogs**, and click **Create Catalog**. You must then provide a secure password that is used to create the database master key for the new catalog.

Catalog Security

Only members of the **ssis_admin** or **sysadmin** roles can manage the catalog. All security in the catalog uses Windows authentication and SQL Server authentication is not supported.

To avoid having to give all SSIS administrators full administration rights, you can grant individuals or roles the MANAGE_OBJECT_PERMISSIONS permission on any folders that you wish them to administer.

If you right-click a folder, project, or environment, and click **Properties**, you can access the permissions on that object. This will allow you to grant or deny permissions on actions such as **Read** or **Modify**, depending on the object type.

You can also use Transact-SQL to manage object permissions using stored procedures such as **catalog.grant_permission** and views like **catalog.effective_object_permisions**.

Additional Reading: For more information about *Stored Procedures in the Integration Services Catalog*, go to http://go.microsoft.com/fwlink/?LinkID=246747.

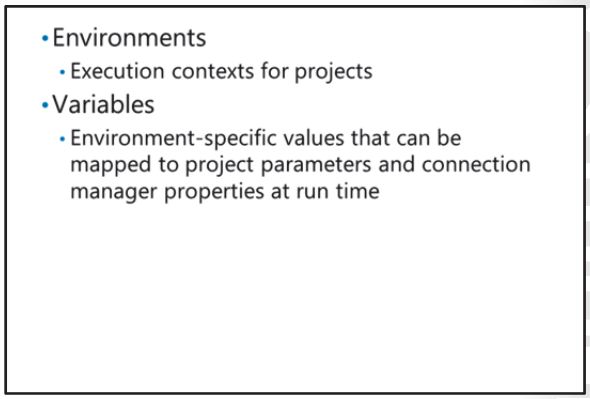
When you create a catalog, a database master key is automatically created. Using the catalog properties, you can define the encryption type from DES, TRIPLE_DES, TRIPLE_DES_3KEY, DESX, AES_128, AES_192, or AES_256 (the default).

Parameters have a **sensitive** property. If a parameter stores sensitive data, you should set the **sensitive** property to TRUE. This causes the parameter value to be encrypted and, if the parameter value is queried directly with Transact-SQL, NULL is returned.

Additional Reading: For more information about SSIS security, go to http://go.microsoft.com/fwlink/?LinkID=246748.

Environments and Variables

Often, you need to change specific properties or values used in a package dynamically at run time. To accomplish this, SSIS catalog supports multiple environments for each project, and the creation of variables within those environments that can be mapped to project parameters and connection managers.



Environments

An environment is an execution context for an SSIS project. You can create multiple environments, and then reference them from a project you have deployed in the SSIS catalog. A project can have multiple environment references, but each time a package is executed, it can use only one. This enables you to switch very quickly from one environment to another and change all associated environment settings from a single property. For example, you could create **Test** and **Production** environments and add references to them both in a deployed project.

Variables

Environments are used to organize and group environment variables that you can map to project-level parameters or properties of project-level connection managers. For example, in the **Test** and **Production** environments, you could create a variable named **DBServer** with the value TEST_SERVER in the **Test** environment and PROD_SERVER in the **Production** environment. You can then map this variable to a project-level parameter or to the **ServerName** property of a connection manager in a deployed project. When you execute a package in the project, the value used for the parameter or **ServerName** property is determined by the selected environment.

Deploying an SSIS Project

After you have built your SSIS project, there are two methods that you can use to deploy it to your catalog. You can deploy it from Visual Studio, or you can deploy it from SQL Server Management Studio.

If you deploy your project to the same location with the same name, it will overwrite the original. Based on the properties of your catalog it might create an additional version of the project and keep a copy of the original.

Deploying an SSIS Project in Visual Studio

After building your project in Visual Studio, you can deploy it by using the Integration Services Deployment Wizard. This wizard will guide you through the process of deployment.

The deployment source is the project that you are currently deploying. The deployment destination is the location where you want to deploy the package, including the server and folder.

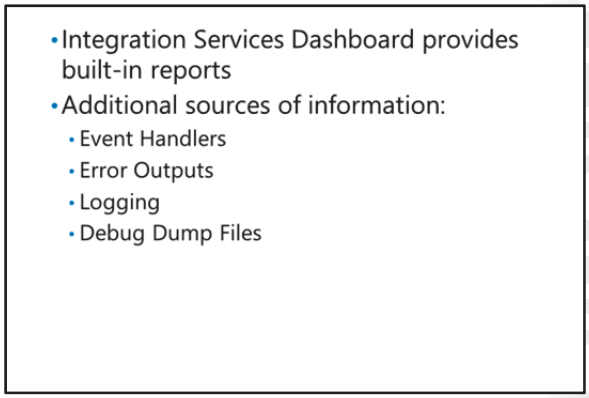
Deploying an SSIS Project in SQL Server Management Studio

In SQL Server Management Studio, you can navigate to the folder where you wish to deploy the package. Then right-click the **Projects** folder within the folder you wish to deploy to and click **Deploy Project**. This will also run the Integration Services Deployment Wizard. The only difference with this method is that you have to find the project deployment file. This will be in the folder created for the Visual Studio project in the **bin** folder, and then in the **Development** folder. It will have an **.ispac** extension.

You can also deploy an existing package within the catalog, or a catalog on another server, to make a copy of a package. Simply choose the existing package as the source in the Integration Services Deployment Wizard.

Viewing Project Execution Information

SSIS includes a number of tools to troubleshoot packages after they have been deployed. During deployment you can use various techniques, such as breakpoints and data viewers, which are not available once you have deployed the package. However, there are other tools that allow you to perform troubleshooting. The Integration Services Dashboard should be the starting point for most debugging as it provides every level of detail for all packages, but there are other tools that can perform specific tasks.



Integration Services Dashboard

The Integration Services Dashboard provides details for each package that has run on the server. If you right-click the **SSISDB** database in the **Integration Services** node in SQL Server Management Studio and point to Reports, you can click Integration Services Dashboard. From here, you can get an overview of activity in the last 24 hours and open reports for All Executions, All Validations, All Operations, and Connections. Each of these reports allows you to drill down further to get specific details for events and view performance data for the package executions.

 **Additional Reading:** For more information about *Troubleshooting Reports for Package Execution*, go to http://go.microsoft.com/fwlink/?LinkID=246749.

Event Handlers

You can create an event handler for the **OnError** event. This can be used for many tasks, including sending an email message to an administrator, or logging system information to a file. Event handlers have control flow and data flows which are identical to those in a typical package, so you can perform almost unlimited tasks when an error has occurred.

 **Additional Reading:** For more information about *Creating Package Event Handlers*, go to http://go.microsoft.com/fwlink/?LinkID=246750.

Error Outputs

Most data flow components have outputs for success and failure. The error output can be used to direct rows that contain errors to a different destination. Initially the error output adds columns containing the error code. While you could manually look up the error code, you can add additional information, such as the error description, using the **Script** component.

Error outputs are for a specific component whereas event handlers are for the whole package.

 **Additional Reading:** For more information about *Enhancing an Error Output with the Script Component*, go to http://go.microsoft.com/fwlink/?LinkID=246751.

Logging

You can enable logging and view package execution information in an SQL Server table or a file.

Log providers implement logging in packages, containers, and tasks. You can also specify which information you require in the log. To enable logging in the package, open it in Visual Studio, and on the SSIS menu, click **Logging**. You can now define how the log information is stored, as well as which events and information are logged.

 **Additional Reading:** For more information about *How to Enable Logging in a Package*, go to http://go.microsoft.com/fwlink/?LinkID=246752.

Debug Dump Files

If you execute a package with **dtexec**, you can specify that a debug dump file is created. This creates a .tmp file that contains items such as environment information and recent messages. The file is located in <drive>:\Program Files\Microsoft SQL Server\110\Shared\ErrorDumps. You can specify whether to create the debug dump files on an error, a warning, or on information.

 **Additional Reading:** For more information consult *The dtexec Utility Reference*, at http://go.microsoft.com/fwlink/?LinkID=246753.

Demonstration: Deploying an SSIS Project

In this demonstration you will see how to:

- Configure the SSIS Environment.
- Deploy an SSIS Project.
- Create Environments and Variables.
- Run an SSIS Package.
- View Execution Information.

Demonstration Steps

Configure the SSIS Environment

1. Ensure 20463C-MIA-DC and 20463C-MIA-SQL are started, and log onto 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Demofiles\Mod12 folder, run **Setup.cmd** as Administrator.
3. Start SQL Server Management Studio and connect to the **localhost** database engine using Windows authentication.
4. In Object Explorer, right-click **Integration Services Catalogs**, and click **Create Catalog**.
5. In the **Create Catalog** dialog box, note that you must enable CLR integration when creating an SSIS catalog, and that you can also choose to enable automatic execution of the stored procedures used by the catalog when SQL Server starts. Then enter and confirm the password **Pa$$w0rd**, and click **OK**.
6. In Object Explorer, expand **Integration Services Catalogs**, and then right-click the **SSISDB** node that has been created, and click **Create Folder**. Then in the **Create Folder** dialog box, enter the folder name **Demo**, and click **OK**.
7. Expand the **SSIDB** node to see the folder, and then minimize SQL Server Management Studio.

Deploy an SSIS Project

1. Start Visual Studio, and open the **DeploymentDemo.sln** solution in the D:\Demofiles\Mod12 folder. This project contains the following two packages:
    - **Extract Login List.dtsx** – a package that uses a data flow to extract a list of logons from the **master** database and save them in a text file.
    - **Extract DB List.dtsx** – a package that extracts a list of databases from the **master** database and saves them in a text file.

Both packages use a project-level connection manager to connect to the **master** database, and a project-level parameter to determine the folder where the text files containing the extracted data should be saved.

2. On the **Build** menu, click **Build Solution**.
3. When the build has succeeded, on the **Project** menu, click **Deploy**.
4. In the **Introduction** page of the Integration Services Deployment Wizard dialog box, click **Next**.

5. In the **Select Destination** page, enter **localhost** in the **Server name** box and in the **Path** box, browse to the **SSIDB\Demo** folder you created earlier. Then click **Next**.
6. On the **Review** page, click **Deploy**. Then, when deployment has completed, click **Close** and close Visual Studio.

Create Environments and Variables

1. In SQL Server Management Studio, expand the **Demo** folder you created earlier, and expand its **Projects** folder. Note that the **DeploymentDemo** project has been deployed.
2. Right-click the **Environments** folder, and click **Create Environment**. Then in the **Create Environment** dialog box, enter the environment name **Test**, and click **OK**.
3. Repeat the previous step to create a second environment named **Production**.
4. Expand the **Environments** folder to see the environments you have created, and then right-click the **Production** environment, and click **Properties**.
5. In the **Environment Properties** dialog box, on the **Variables** tab, add a variable with the following settings:
   - **Name**: DBServer
   - **Type**: String
   - **Description**: Server
   - **Value**: MIA-SQL
   - **Sensitive**: No
6. Add a second variable with the following settings (making sure to include the trailing "\" in the value), and then click **OK**:
   - **Name**: FolderPath
   - **Type**: String
   - **Description**: Folder
   - **Value**: D:\Demofiles\Mod12\Production\
   - **Sensitive**: No
7. Right-click the **Test** environment and click **Properties**.
8. In the **Environment Properties** dialog box, on the **Variables** tab, add a variable with the following settings:
   - **Name**: DBServer
   - **Type**: String
   - **Description**: Server
   - **Value**: localhost
   - **Sensitive**: No

9. Add a second variable with the following settings (making sure to include the trailing "\" in the value), and then click **OK**:
   - **Name**: FolderPath
   - **Type**: String
   - **Description**: Folder
   - **Value**: D:\Demofiles\Mod12\Test\
   - **Sensitive**: No
10. In the **Projects** folder, right-click **DeploymentDemo** and click **Configure**.
11. In the **Configure – DeploymentDemo** dialog box, on the **References** page, click **Add** and add the **Production** environment. Then click **Add** again and add the **Test** environment.
12. In the **Configure – DeploymentDemo** dialog box, on the **Parameters** page, in the **Scope** list, select **DeploymentDemo**.
13. On the **Parameters** tab, click the ellipses (**...**) button for the **OutputFolder** parameter, and in the **Set Parameter Value** dialog box, select **Use environment variable** and click **FolderPath** in the list of variables, and click **OK**.
14. In the **Configure – DeploymentDemo** dialog box, on the **Connection Managers** tab, click the ellipses button (**...**) for the **ServerName** property, and in the **Set Parameter Value** dialog box, select **Use environment variable**, click **DBServer** in the list of variables, and click **OK**.
15. In the **Configure – DeploymentDemo** dialog box, click **OK**.

Run an SSIS Package

1. Expand the **DeploymentDemo** package and its **Packages** folder, and then right-click **Extract DB List.dtsx** and click **Execute**.
2. In the **Execute Package** dialog box, select the **Environment** checkbox, and in the drop-down list, select **.\Test**. Then view the **Parameters** and **Connection Managers** tabs and note that the **FolderPath** and **DBServer** environment variables are used for the **OutputFolder** parameter and **ServerName** property.
3. Click **OK** to run the package. Click **No** when prompted to open the **Overview Report**.
4. Right-click **Extract Login List.dtsx** and click **Execute**.
5. In the **Execute Package** dialog box, select the **Environment** checkbox and in the drop-down list, select **.\Production**. Then view the **Parameters** and **Connection Managers** tabs and note that the **FolderPath** and **DBServer** environment variables are used for the **OutputFolder** parameter and **ServerName** property.
6. Click **OK** to run the package. Click **No** when prompted to open the **Overview Report**.
7. View the contents of the D:\Demofiles\Mod12\Test folder and note that it contains a file named **DBs.csv** that was produced by the **Extract DB List.dtsx** package when it was executed in the **Test** environment.
8. View the contents of the D:\Demofiles\Mod12\Production folder and note that it contains a file named **Logins.csv** that was produced by the **Extract Login List.dtsx** package when it was executed in the **Production** environment.

View Execution Information

1. In SQL Server, in Object Explorer, under **Integration Services Catalogs**, right-click **SSISDB**, point to **Reports**, point to **Standard Reports**, and click **Integration Services Dashboard**.
2. In the **Packages Detailed Information (Past 24 Hours)** list, notice that the two most recent package executions succeeded, and then click the **Overview** link for the first package in the list.
3. In the **Overview** report, in the **Execution Information** table, note the environment that was used, and in the **Parameters Used** table, note the values you configured with environment variables.
4. At the top of the report, click the **Navigate Backward** button to return to the **Integration Services Dashboard** report.
5. In the **Packages Detailed Information (Past 24 Hours)** list, click the **Overview** link for the second package in the list.
6. In the **Overview** report, in the **Execution Information** table, note the environment that was used, and in the **Parameters Used** table, note the values you configured with environment variables.
7. Close SQL Server Management Studio.

Lesson 3
Planning SSIS Package Execution

There are a number of factors you should consider before performing an SSIS deployment. Along with the deployment model, security and variables, all mentioned in previous lessons, you should also consider factors such as scheduling, dependencies, notifications, and which actions take place in SSIS packages and which ones occur in SQL Server Agent jobs.

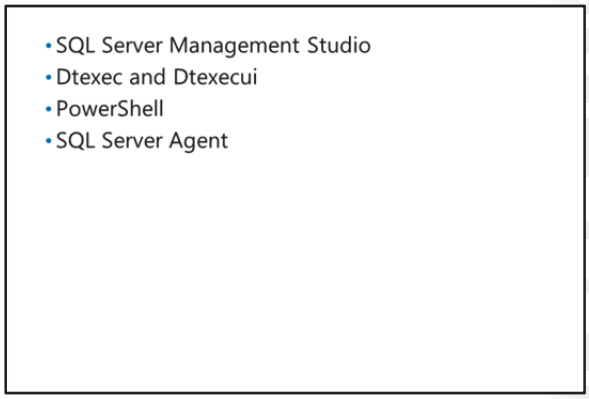
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the methods for running SSIS packages.
- Describe considerations for package scheduling.
- Explain the security context options when running a package.
- Describe the choices when deciding where notifications are handled.
- Explain the options when handling SSIS logging.
- Describe the options for combining SSIS tasks with SQL Server Agent job tasks.
- Explain how to implement an SQL Server Agent job.

Options for Running SSIS packages

After deploying your package, there are several methods you can employ to execute it.



SQL Server Management Studio

After the package is deployed to SQL Server, you can run it from the context menu. You can supply values for the environment and any parameter, modify settings for the connection managers, and specify logging options.

Dtexec and Dtexecui

Dtexec provides a command line interface to run SSIS packages. You can supply all necessary values for the environment, parameters, connection strings and logging. **Dtexecui** provides a graphical interface to run **dtexec** and has the same functionality.

Additional Reading: For more information consult *the dtexec Utility Reference*, at http://go.microsoft.com/fwlink/?LinkID=246753.

Windows PowerShell

SQL Server 2012 introduces Windows PowerShell® cmdlets, enabling you to manage, monitor and execute SSIS packages. This is a very powerful environment for managing most aspects of your server and this functionality enables you to integrate SSIS with other Windows PowerShell tasks.

Additional Reading: For more information about *SSIS and PowerShell in SQL Server*, go to http://go.microsoft.com/fwlink/?LinkID=246754.

SQL Server Agent

SQL Server Agent automates tasks in SQL Server. This is particularly useful for SSIS packages as they are often run on a repeating schedule to periodically load data. You can specify the schedule for the job, the environment, parameters, connection strings and logging.

Additional Reading: For more information about *SQL Server Agent Jobs for Packages*, go to http://go.microsoft.com/fwlink/?LinkID=246755.

Scheduling the ETL Process

When scheduling package execution with SQL Server Agent or another automation tool, there are some considerations you should take to plan an ETL strategy.

- Data Acquisition Windows
- Package Execution Order
- Execution Dependencies

Data Acquisition Windows

In a data warehouse scenario, data sources that are periodically used to extract data for your data warehouse are typically live transactional systems. These are often some of the most business critical systems in your organization. For example, the orders system for an Internet-based bicycle shop holds the most important data for the data warehouse, but is also the most critical one to keep running at optimum performance levels. If the orders system is down, the business cannot operate.

It is crucial, therefore, to ascertain the best time to extract data from the live systems. For a weekly data load it might be at the weekend, for a daily load it may be at night, but it is important to check the systems for real-world statistics. Even truly international systems have quiet and busy periods, caused either by customer distribution, or because the population of the world is not uniformly distributed.

If data acquisition is problematic at any time, it might be possible to run queries against mirrored systems that are used to provide high availability. Mirrored systems can provide access to live data at any time with no effect on the live transactional systems.

Package Execution Order

Ensure that packages execute in the correct order and that one step is completed before the next is started. This is typically the case when data is loaded to a staging database. The data load to staging must complete, followed by data transformation and cleansing, and then a data load to the data warehouse. You must either create a system where one step in the ETL process triggers the next one, or create schedules with enough time delay for the previous step to complete.

Package execution order sounds very straightforward. However, a completely linear system, with one package starting the next and so on, might be very slow, as no packages are run in parallel. Running packages in parallel either requires careful scheduling or more complex logic to test whether all prerequisite packages have run before the next stage starts.

Execution Dependencies

Some packages require others to already have run due to data dependencies. For example, a **Salesperson** table has a foreign key relationship with a **Store** table. You have recently opened a new store with new employees. If the employees are loaded before the store, they will break the foreign key constraint and the data load will fail. However, if the store is loaded before the employees, then the data load is successful.

Configuring Execution Context

Using SQL Server Agent, you can run a job with a different security context to your own. For example, a user has read-only access to the production databases, staging databases, and data warehouse, but also needs to be able to periodically load data from production to data warehouse via staging. In the SQL Server Agent job, this can be performed by setting the **Run as** property on the job step to an account that has the required permissions on all three systems. The user does not require the special privileges on the system, just permission to run the SQL Server Agent job.

- Create a credential in the SQL Server instance
- Create a SQL Agent proxy that maps to the credential
  - Activate it for the SQL Server Integration Services Package job subsystem
- Configure package execution job steps to run as the proxy

There are three fixed database roles that apply to the SQL Server Agent – **SQLAgentUserRole, SQLAgentReaderRole**, and **SQLAgentOperatorRole. SQLAgentUserRole** and **SQLAgentReaderRole** members can only execute jobs that they have created, but **SQLAgentOperatorRole** members can execute, but not edit, a job created by another user. You can add users to these roles in the MSDB database.

To use the **Run as** property of a job step, you must first create a credential in the **Security** node of SQL Server Management Studio that maps to an account with the permissions required for the job step. You can then create a proxy in SQL Server Agent that uses this credential. The job step can be configured to use this proxy and the user who needs to execute the job is put in the appropriate database role.

Where to Handle Notifications

You can send notifications from SSIS packages using the **SendMail** task and other notifications from SQL Server Agent jobs using operators. Be careful not to send too many notifications as the operator can become overloaded and not notice an important email message amongst many duplicates.

- SSIS Package
- SQL Server Agent Job

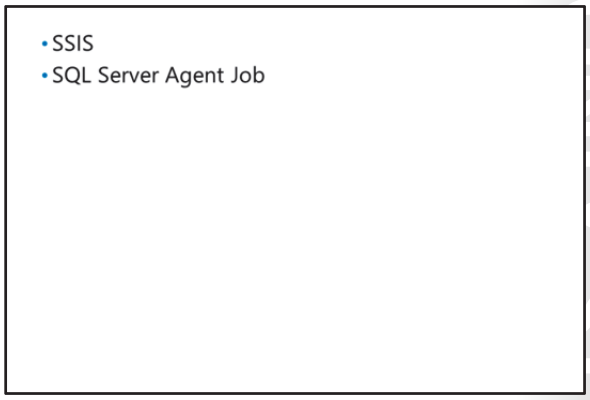
You should consider how the SSIS package is used and the steps that the SQL Server Agent job performs. If the SSIS package is executed as an SQL Server Agent job, manually from SQL Server Management Studio, and from a Windows PowerShell® script, you should consider putting notifications in the SSIS package to ensure that notifications are sent regardless of the execution method. If the SSIS package is always executed as part of

an SQL Server Agent job that has additional steps apart from the SSIS package, then you should consider putting notifications in the SQL Server Agent job.

In many situations, the decision is not clear cut and you should balance the advantages and disadvantages of putting notifications either in SQL Server Agent jobs or in SSIS packages.

Where to Handle Logging

Logging is less intrusive than email notifications and, therefore, is less of a problem to implement logging in both SQL Server Agent and SSIS packages. For SSIS, you should consider using the Integration Services Dashboard if that provides sufficient detail. The Integration Services Dashboard is automatic and displays an overview of packages that have executed, allowing you to drill down into particular items to display the detail.



In SQL Server Agent, logging is enabled by default and will log any errors or warnings. You can specify the size and maximum age of the job history log in the properties of SQL Server Agent.

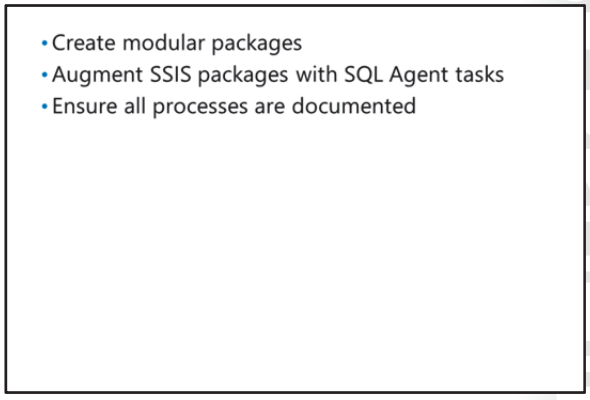
Additional Reading: For more information about the *SQL Server Agent Error Log*, go to http://go.microsoft.com/fwlink/?LinkID=246756.

In a typical scenario, you would examine the SQL Server Agent log and, if you find there was an error in an SSIS package execution, you would investigate the Integration Services Dashboard.

Combining SSIS Tasks with Other Tasks

SSIS is a powerful tool and you could perform every task you need from a single SSIS package and schedule this as a single step in a SQL Server Agent job. Typically, however, this is not the best approach.

If you take the policy that SSIS packages perform one defined task, you can make modular packages that run multiple smaller ones. In this way, the work that you have done to create one, perhaps complex, SSIS package, can easily be reused.

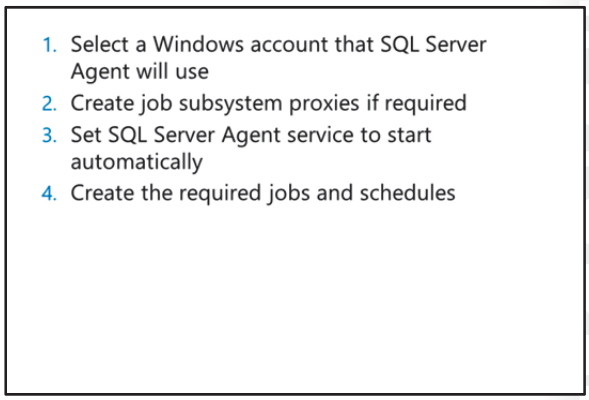


Using this approach, you could continue to use SQL Server Agent to execute a single SSIS package, albeit one that runs several child packages. In this scenario, you might want to add another minor step to a package, perhaps executing a Transact-SQL query—you could add this as a new task in your SSIS package. To avoid changing any existing ones, you could create a new package to perform this task, and then create a new parent package to run the existing ones, along with your new package, containing the Transact-SQL task. In this scenario, it would be simpler to add a step to the SQL Server Agent job to run the Transact-SQL query.

One factor to consider if you have actions performed in both SQL Server Agent jobs and SSIS packages is documentation. You must ensure that everyone involved in developing the ETL solution is aware of where the actions are performed and, therefore, where they need to be monitored and maintained.

Implementing SSIS Agent Jobs and Schedules

To use SQL Server Agent to execute jobs, you should perform a number of steps:



1. Select or create a Windows account that the SQL Server Agent will use and ensure that it has sufficient permissions to perform the assigned job steps. The Windows domain account you specify must have the following:
   - Permission to log on as a service (SeServiceLogonRight).
   - The following permissions to support SQL Server Agent proxies:
     - Permission to bypass traverse checking (SeChangeNotifyPrivilege).
     - Permission to replace a process-level token (SeAssignPrimaryTokenPrivilege).
     - Permission to adjust memory quotas for a process (SeIncreaseQuotaPrivilege).
     - Permission to log on using the batch logon type (SeBatchLogonRight).
2. If necessary, create Windows accounts for SQL Server Agent proxies, add credentials for them to the SQL Server instance, and create proxies for the appropriate job subsystems.
3. Set SQL Server Agent service to start automatically. By default, the SQL Server Agent is configured to start manually and, if there is a server restart, jobs will cease to run.
4. Create jobs for each package execution you want to automate, and specify a schedule on which the job should run.

Lab: Deploying and Configuring SSIS Packages

Scenario

You have completed the SSIS project containing the packages required for the data warehouse ETL process. Now you must deploy the project to an SSIS catalog, configure environments for dynamic configuration, and schedule automatic execution of packages.

Objectives

After completing this lab, you will be able to:

- Create an SSIS catalog.
- Deploy an SSIS project.
- Create environments.
- Run an SSIS package.
- Schedule SSIS package execution.

Estimated Time: 45 minutes

Virtual machine: 20463C-MIA-SQL

User name: ADVENTUREWORKS\Student

Password: Pa$$w0rd

Exercise 1: Creating an SSIS Catalog

Scenario

You want to deploy your SSIS packages as an SSIS project. Initially, you will configure the SSIS environment.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Create the SSIS Catalog and a Folder

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. Run **Setup.cmd** in the D:\Labfiles\Lab12\Starter folder as Administrator.

▶ Task 2: Create the SSIS Catalog and a Folder

1. Use SQL Server Management Studio to create an SSIS catalog with a password of **Pa$$w0rd** in the **localhost** instance of SQL Server.
2. In the catalog, create a folder named **DW ETL** with the description **Folder for the Adventure Works ETL SSIS Project**.

Results: After this exercise, you should have enabled CLR, creating the SSIS catalog, and a folder.

Exercise 2: Deploying an SSIS Project

Scenario

You now want to deploy an existing SSIS project, created in Visual Studio, to the new folder from the previous exercise.

The main tasks for this exercise are as follows:

1. Deploy an SSIS Project

▶ Task 1: Deploy an SSIS Project

1. Start Visual Studio, and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab12\Starter folder.
2. Build the solution.
3. Deploy the **AdventureWorksETL** project to the **DW ETL** folder in the **localhost** server.
4. Close Visual Studio.

Results: After this exercise, you should have deployed an SSIS project to a folder in your SSIS database.

Exercise 3: Creating Environments for an SSIS Solution

Scenario

You have parameters and connection managers in an SSIS package. You would like to create test and production environments enabling you to quickly pass the correct values to these parameters when switching from test to production.

The main tasks for this exercise are as follows:

1. Create Environments
2. Create Variables
3. Map Environment Variables

▶ Task 1: Create Environments

1. In SQL Server Management Studio, create an SSIS environment named **Test** in the **DW ETL** folder.
2. Create a second environment named **Production**.

▶ Task 2: Create Variables

1. Add the following variables to the **Production** environment:
   - A string variable named **StgServer** with the value **MIA-SQL.**
   - A string variable named **FolderPath** with the value **D:\Accounts**.
2. Add the following variables to the **Test** environment:
   - A string variable named **StgServer** with the value **localhost**.
   - A string variable named **FolderPath** with the value **D:\TestAccts**.

▶ Task 3: Map Environment Variables

1. Configure the **AdventureWorksETL** package in the SSIS Catalog to use the **Production** and **Test** environments.

2. Map the **AccountsFolder** project-level parameter to the **FolderPath** environment variable.
3. Map the **ServerName** property for the **localhost Staging ADO NET** and **localhost.Staging** connection managers to the **StgServer** environment variable.

Results: After this exercise, you should have configured your project to use environments to pass values to package parameters.

Exercise 4: Running an SSIS Package in SQL Server Management Studio

Scenario

Now you have deployed the SSIS project and defined execution environments, you can run the packages in SQL Server Management Studio.

The main tasks for this exercise are as follows:

1. Run a Package

▶ Task 1: Run a Package

1. Use SQL Server Management Studio to run the **Extract Payment Data.dtsx** package using the **Test** environment.
2. When the package has been executed, consider the overview report and verify that it used the environment variable values you specified for the **Test** environment.

Results: After this exercise, you should have performed tests to verify that a value was passed from the environment to the package.

Exercise 5: Scheduling SSIS Packages with SQL Server Agent

Scenario

You've tested the packages by running them interactively. Now you must schedule them to be executed automatically.

The main tasks for this exercise are as follows:

1. Create a SQL Server Agent job
2. View the Integration Services Dashboard

▶ Task 1: Create a SQL Server Agent job

1. In SQL Server Management Studio, create a new SQL Server Agent job named **Extract Reseller Data**.
2. Add a step named **Run Extract Reseller Data Package** to the job, and configure it to run the **Extract Reseller Data.dtsx** SQL Server Integration Services package in the **AdventureWorksETL** project you deployed earlier.
3. Configure the job step to use the **Production** environment.
4. Create a schedule for the job to run once, two minutes from the current time.
5. Wait for two minutes, and then in the SQL Server Agent Job Activity Monitor, view job activity and verify that the **Extract Reseller Data** job has completed successfully.

▶ Task 2: View the Integration Services Dashboard

1. View the Integration Services Dashboard and verify that the **Extract Reseller Data.dtsx** package was successfully executed.
2. View the overview report for the **Extract Reseller Data.dtsx** package execution and verify the parameter values that were used.
3. View the messages for the **Extract Reseller Data.dtsx** package execution, and note the diagnostic information that was logged.
4. When you have finished, close SQL Server Management Studio.

Results: After this exercise, you should have created an SQL Server Agent job that automatically runs your SSIS package.

Module Review and Takeaways

In this module, you have learned how to deploy and run SSIS packages.

Review Question(s)

Question: What are some of the advantages of the project deployment model?

Question: What are the advantages of environments?

Question: Where would you handle notifications?

Module 13

Consuming Data in a Data Warehouse

Contents:

Module Overview

A data warehouse is the foundation for a business intelligence (BI) solution, enabling business users, information workers, and data analysts to make faster, better decisions.

This module introduces BI, describing the components of Microsoft® SQL Server® that you can use to create a BI solution, and the client tools with which users can create reports and analyze data.

Objectives

After completing this module, you will be able to:

- Describe BI and common BI scenarios.
- Describe how a data warehouse can be used in enterprise BI scenarios.
- Describe how a data warehouse can be used in self-service BI scenarios.

Lesson 1
Introduction to Business Intelligence

BI technologies enable you to create decision support systems that enable the individual in an organization to work more effectively. However, a BI solution is only as good as the data that it processes. Therefore, using a data warehouse as the platform for your BI solution makes a lot of sense.

This lesson explains the benefits of using a data warehouse as a platform for BI, and introduces common reporting and data analysis scenarios.

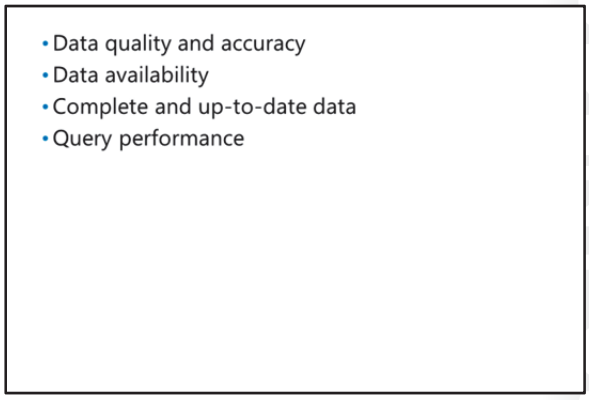
Lesson Objectives

After completing this lesson, you will be able to:

- Explain the benefits of using a data warehouse as a platform for BI.
- Describe data analysis scenarios and technologies.
- Describe reporting scenarios and technologies.

The Data Warehouse as a Platform for Business Intelligence

The primary purpose of creating a data warehouse is to provide a platform enabling BI workers to access historical data so that they can perform reporting and data analysis. A well-designed data warehouse will include mechanisms that ensure the quality and accuracy of data, as described in previous modules, and will be optimized to provide the best performance for BI applications. Using a data warehouse as a data source for BI provides many benefits, including:

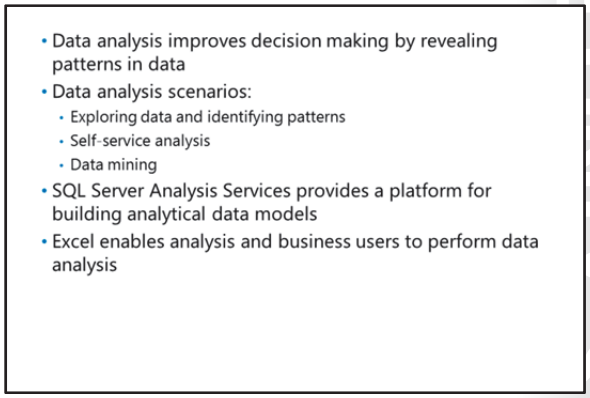


- **Data quality and accuracy**. Effective reporting and data analysis rely on the availability of data that is complete, accurate, consistent, and which does not contain duplicate data, or any other data that might compromise the accuracy of the reports and data analyses that information workers create. Using Master Data Services and Data Quality Services ensures that the data in the data warehouse meets the requirements of BI applications, enabling executives, managers, and other information workers to make better strategic and operational business decisions.
- **Data availability**. The data used by information workers must also be easily available, so that individuals do not have to search for information. Centralizing data in a data warehouse makes it much easier for them to find the data they require.
- **Complete and up-to-date data**. To support reporting and data analysis, the data in the data warehouse needs to be complete and up to date. SQL Server Integration Services provides comprehensive extract, transform, and load (ETL) capabilities, enabling you to populate your data warehouse and maintain it by using periodic or incremental updates.
- **Query performance**. BI queries can be highly demanding, requiring a significant amount of processing power. A dedicated data warehouse that is optimized for BI queries, such as a Fast Track Data Warehouse system or Parallel Data Warehouse system, can handle the processing load that BI queries even when many users are simultaneously accessing the database.

SQL Server provides a complete platform for data warehousing, and supports reporting tools such as SQL Server Reporting Services, Report Builder, and Power View, and data analysis tools like SQL Server Analysis Services, Microsoft® Excel®, and PerformancePoint Services in SharePoint Server.

Data Analysis

Data analysis involves exploring data to find interesting and useful information that can help companies to identify patterns, opportunities, inefficiencies, and so on. Data in the data warehouse can be used to perform different types of analysis, such as identifying sales patterns over the previous quarter, or predicting future performance based on past data.



Key data analysis scenarios include:

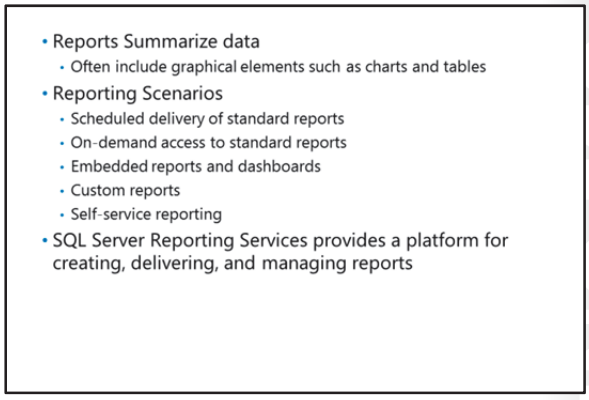
- **Exploring data to identify patterns**. Data analysts use various tools to explore data to try and identify useful information, such as the best performing sales regions or correlations between sales figures and other factors. Business decision-makers can use this information to guide strategic planning decisions.
- **Self-service analysis and data mash-ups**. Business users might perform self-service analysis when they need to combine data from the data warehouse and other sources without support from the IT department.
- **Data mining**. Analysts use data mining to discover difficult-to-identify patterns and correlations between data. This information is useful because it can help to predict the success of future activities such as targeted advertising campaigns.

Data analysis tools combine the data and metadata from data warehouses or other stores to create models that make this data available as useable information. For example, your data warehouse might include a table that contains millions of rows of sales data, a second one for product information, and other tables with details about dates and geographical location. You could create an analysis solution that loads data into a data model, such as a multidimensional cube or a tabular data model. This then calculates and stores aggregations of key data to improve query response times, making all this available to a range of client tools.

SQL Server Analysis Services is the SQL Server component that BI developers can use to create data models, enabling users to access data to perform analysis. Excel can help analysts and business users to create PivotTables and other data visualizations from enterprise data models. They can also create personal data models that combine data from the data warehouse with information from other sources.

Reporting

Reporting involves producing documents that summarize specific data. Reports typically include graphical elements such as charts and tables that help to make the information easy to understand. There are several common scenarios that report developers and administrators are likely to encounter, such as:



- **Scheduled delivery of standard reports**. BI developers create a set of standard reports that business users receive by email message or delivery to a file share or Microsoft® SharePoint® site on a regular basis.
- **On-demand access to standard reports**. Business users consume reports on-demand by browsing to an online report repository.
- **Embedded reports and dashboards**. Reports are integrated into an application or portal to provide contextualized business information at a glance.
- **Request to IT for custom reports**. Business users request specific reports from the IT department, and a BI developer creates these reports to order.
- **Self-service reporting**. Business users can use authoring tools to create their own reports from data models that have been published by the IT department.

SQL Server Reporting Services is a component of SQL Server that provides a platform for building and managing reports in an enterprise. Reporting Services provides comprehensive functionality that supports all these scenarios.

Lesson 2
Enterprise Business Intelligence

Business executives, departmental managers, team leaders, and many other individuals in an organization, rely on reports and analytical dashboards to provide clear and easy-to-interpret summaries of data to help them make decisions that directly affect the business and its employees.

In many organizations, an enterprise BI solution is designed, implemented, and managed by the IT department to support reporting and analytical requirements. This BI solution is often based on the foundation provided by an enterprise data warehouse.

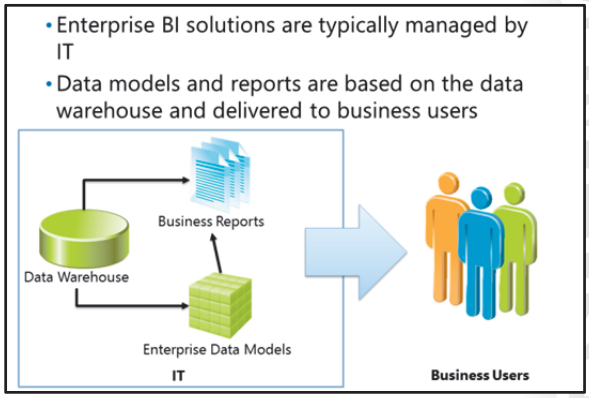
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the key features of an enterprise BI solution.
- Describe how SQL Server Analysis Services can support enterprise data models.
- Describe how Reporting Services can support enterprise reporting.
- Describe how Microsoft® SharePoint® Server can be used to support enterprise BI.

Introduction to Enterprise BI

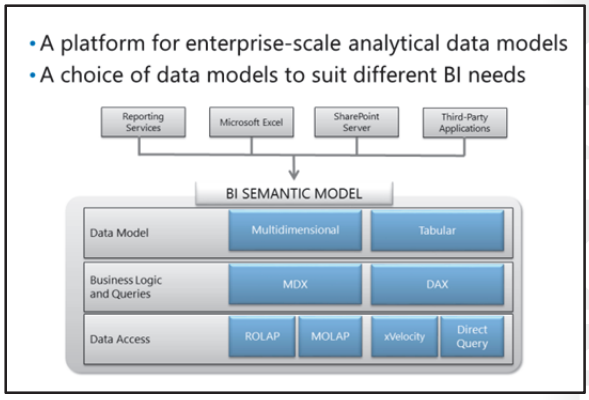
Many organizations seek to use BI to inform business decision-making and communicate key information to executives, managers, information workers, customers, and other stakeholders. Often, the BI solution used to meet these requirements is managed by IT department technical specialists, who create and maintain the data models and reports necessary to support the organization's specific needs.



Data warehouses and marts are at the core of an IT-managed enterprise BI solution, providing a central source of validated business data for reports, dashboards, and analyses. BI specialists create analytical data models that aggregate the data in the data warehouse, defining hierarchies and key performance indicators (KPIs) to support well-defined business requirements. These hierarchies and KPIs can be viewed in dashboards (for example, in SharePoint Server) or browsed interactively (for example, in Microsoft® Excel®). Additionally, business reports are created from the data models, and often directly from the data warehouse. These reports can be viewed interactively in a web browser, and are sometimes distributed automatically by email at regularly scheduled intervals.

Additional Reading: To learn about implementing an enterprise BI solution, you should attend course 20466C, *Implementing Data Models and Reports with Microsoft SQL Server.*

SQL Server Analysis Services

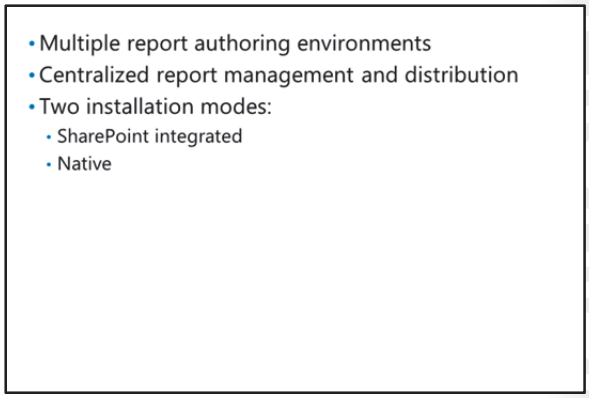


You create a data model to enable users to analyze business data, without having to understand the complexities of the underlying database, and to enable optimal performance for BI queries. Data models expose data in a format that users can interact with more easily. SQL Server Analysis Services enables you to create enterprise-scale data models to support BI applications including Reporting Services, Microsoft® Excel®, PerformancePoint Services in SharePoint Server, and other third-party tools. Analysis Services in SQL Server 2014 enables you to create two different kinds of data model, each with its own features and performance characteristics:

- **Multidimensional data models**. These have been used in all versions of SQL Server Analysis Services, up to and including SQL Server 2014 Analysis Services, and are familiar to experienced BI specialists. Multidimensional data models expose data through dimensions and cubes. They use the Multidimensional Expressions (MDX) language to implement business logic, and can provide access to data through relational online analytical processing (ROLAP) storage or multidimensional online analytical processing (MOLAP) storage.
- **Tabular**. Tabular data models were first introduced in SQL Server 2012 Analysis Services. They are constructed from tables and relationships, making them easier to work with for database professionals with no multidimensional data modeling experience. Tabular data models use the Data Analysis Expressions (DAX) language to implement business logic, and can provide access to data by using the in-memory xVelocity engine. They also use DirectQuery mode to access data in the underlying data source, which is often a data warehouse.

SQL Server Analysis Services supports a concept known as the BI semantic model (BISM), which abstracts the specific implementation details of the data model, enabling client tools to use standard protocols and interfaces to access it, regardless of the specific model type.

SQL Server Reporting Services



SQL Server Reporting Services is a development environment for professional report developers and information workers who need to occasionally create reports. You can use Reporting Services to create interactive, visually-interesting reports that can include charts, maps, data bars, and spark lines, and which support parameters, enabling users to set the context for a report.

SQL Server Reporting Services is available in most editions of SQL Server, including SQL Server Express with Advanced Services, though not all features are supported in every edition.

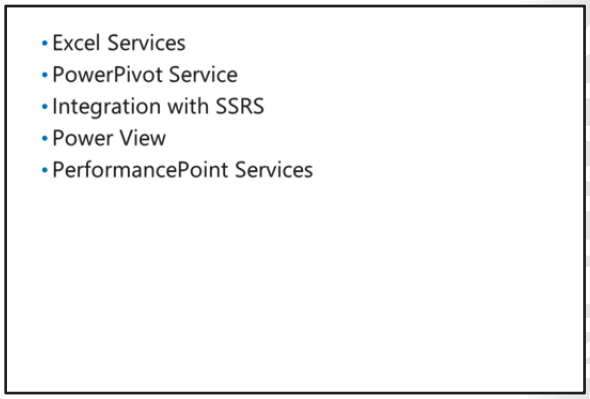
When you install SQL Server Reporting Services, you must choose between two possible deployment modes:

- **SharePoint integrated mode**. In this mode, the report server is installed as a service application in a SharePoint Server farm, and users manage and view reports in a SharePoint site.
- **Native mode**. In this mode, Reporting Services provides a management and report viewing user interface (UI) called Report Manager, which is implemented as a stand-alone web application.

In addition to enabling users to create reports, Reporting Services provides a management environment that enables you to control the complete report life cycle. Users can publish and subscribe to reports, which can be delivered in a variety of ways. Administrators can define email alerts that inform users of changes to the reports they are interested in, and define permissions to ensure that reports can only be viewed and edited by selected individuals.

SharePoint Server

Microsoft® SharePoint® Server provides enterprise information sharing services through collaborative websites, and is often the platform by which enterprise BI is delivered to business users.



SharePoint Server provides the following BI capabilities:

- **Excel Services**. Users can view and interact with Excel workbooks that are shared in a SharePoint document library through a web browser. This includes workbooks that use data connections to query data in a data warehouse or Analysis Services data model.
- **PowerPivot Services**. Users can share and interact with Excel workbooks that contain a PowerPivot tabular data model. This enables business users to create and share their own analytical data models.
- **Integration with SSRS**. You can deliver and manage reports and data alerts through SharePoint document libraries instead of the native Report Manager interface provided with SSRS.
- **Power View**. Power View is an interactive data visualization technology through which users can graphically explore a data model in a web browser.
- **PerformancePoint Services**. This enables BI developers to create dashboards and scorecards that deliver KPIs and reports through a SharePoint site.

Lesson 3
Self-Service BI and Big Data

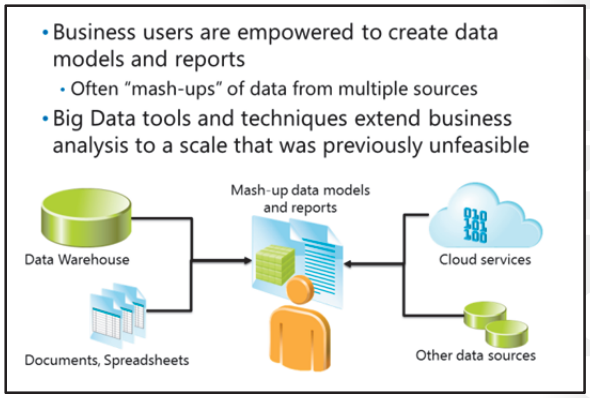
One of the emerging trends in corporate business intelligence is a move towards empowerment of users to perform self-service analysis and reporting, often combining corporate data with external data. Additionally, the growth in volume and sources of data has led to a new generation of "Big Data" tools. These enable organizations to extend their enterprise BI solution to include massive amounts of data that previously would have been difficult and expensive to process and analyze.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe key features of self-service BI and Big Data.
- Describe how Microsoft® Excel® empowers users to perform self-service BI.
- Describe how Report Builder and Power View enable self-service reporting.
- Describe services that help organizations extend an enterprise BI solution to include Big Data.

Introduction to Self-Service BI and Big Data

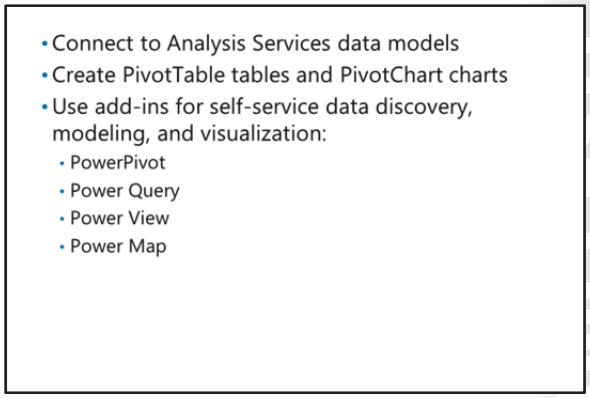


As business users have become more technologically proficient, and computer systems have evolved to support fast, ubiquitous access to large volumes of data, a trend towards self-service BI has emerged. Self-service BI enables users to create their own data models, reports, and mash-ups that combine data from managed corporate data sources (such as an enterprise data warehouse) with data from elsewhere, often external. By empowering users to build their own BI visualizations, IT professionals free themselves to focus on maintaining business-critical systems and implementing new solutions. Business users are not forced to wait for IT to have available resources to support fast-moving BI requirements.

Some of the external data sources that business users want to include in mash-up analyses can contain massive volumes of data in a variety of formats. To make analysis of these sources possible, organizations are increasingly using specialist Big Data technologies to process data and generate the datasets that users need for their analysis. In some cases, technically sophisticated business users and specialist data analysts might use these Big Data tools directly. In other situations, the IT department might implement a Big Data processing solution, from which business users can obtain the resulting datasets.

Additional Reading: To learn how to support self-service BI and Big Data solutions, you should attend course 20467C, *Designing Self-Service BI and Big Data Solutions*.

Using Excel as a Self-Service BI Tool



Excel is a well-established BI tool that data analysts and information workers use to create workbook-based applications to support business activities. You can use Excel to connect to Analysis Services data models, and explore the data that they contain, by creating PivotTable tables and PivotChart charts. PivotTable tables enable you to display data as a table, a matrix, or as a combination of the two. In a PivotTable table, you can expand and collapse data, color code values to highlight them, and use a range of formatting options to render the data as you require. You can use PivotChart charts to add graphical elements to a data analysis report, such as a column or pie chart.

PowerPivot

The PowerPivot for Excel add-in extends the capabilities of Excel as a data analysis tool by enabling you to create a tabular data model that stores data in the workbook. This approach enables you to perform complex calculations offline, and with excellent performance, even for very large data sets. PowerPivot for Excel adds the PowerPivot ribbon to the Excel interface, from which you can launch the PowerPivot window to create database connections, manage tables, and create DAX measures and calculated columns. In addition to creating PivotTable tables and PivotChart charts, you can add slicers, which enable you to group and sort data with just a few clicks. The key features of PowerPivot for Excel include:

- Fast response times when performing complex analyses, even for very large datasets.
- Minimal memory footprint due to the VertiPaq storage engine.
- Advanced features such as hierarchies, perspectives, and relationship management.
- The ability to create custom measures and calculated columns by using DAX.
- The ability to scale up the workbook to a departmental solution by using it as a template for a new SQL Server Analysis Services Tabular project.

SharePoint Server provides a central, managed portal through which analysts can securely share and collaborate on their PowerPivot for Excel workbooks. Sharing PowerPivot workbooks in SharePoint sites enables users to locate the workbooks they need, and reduces the effort wasted in creating duplicates because they did not realize that a similar workbook already existed. PowerPivot for SharePoint helps administrators to establish governance over workbooks, which might be stored on the computers of many different users throughout an organization. It also makes it easier to provision appropriate resources to optimize application performance.

Power Query

Power Query is an add-in for Excel that enables users to find and query data in a wide range of sources. Power Query is can be used as a stand-alone add-in in Microsoft® Excel®, and adds the ability to share queries and manage corporate data sources when used with a Power BI for Microsoft® Office 365™ subscription. With Power Query, business users can:

- Find and import data from external sources.
- Search public data.
- Combine and shape data from multiple sources.
- Filter, sort, and group data.

The results of a query can be displayed in an Excel worksheet or loaded into a workbook data model, and refined using PowerPivot.

Power View

Power View is a data visualization application that enables you to interactively explore data and create reports with a strong visual impact. You can create Power View reports by choosing from a range of charts and tables, which are called visualizations. These include bar charts, column charts, line charts, scatter charts, cards, tiles, tables, and matrices, which you can use to display data in many ways. You can include images and animations to enhance the appearance of a report and make it easier to understand, adding interactive elements that enable you to update a report with a single click. For example, you can create tiled images that represent different categories of products. When you click on one of these images, the report updates to display the data for that category.

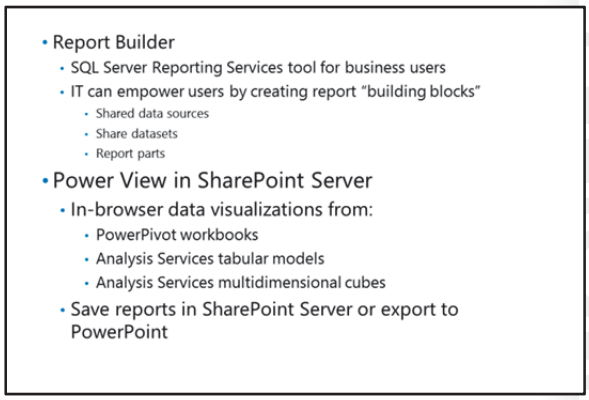
Power View reports are very easy to create, and the UI is simple and intuitive. When you select a field or measure to add to a visualization, the application automatically uses it in the chart in the most appropriate way. For example, when you select a field, Power View might add it to the X axis, use it as a measure value, or use it as a chart series. This behavior speeds up the time it takes to build meaningful reports, and makes it easier for inexperienced users to get started with data exploration. Modifying reports is also very easy, as you can drag visualizations to move and resize them, and replace fields and measures in existing visualizations with just two or three clicks. Power View will automatically display the recalculated values based on the modifications that you make.

Power Map

Power Map is a Microsoft® Excel® add-in that enables users to render data values with a geographical dimension on a map and use animation techniques to explore the data visually. For example, you could use Power Map to compare data aggregations such as sales volume by country, average property value by postal code, or population by city. Additionally, you can display the geographical data aggregations over time, enabling you to see how values changed or grew during a specific period. Users can view the Power Map visualizations directly in the Excel workbook, or they can be exported as a video.

Report Builder and Power View in SharePoint Server

In addition to creating personal or shared data models that support data analysis in Excel, business users often want to create reports and data visualizations that they can print or distribute. To support this requirement, SQL Server Reporting Services provides Report Builder, a report creation tool specifically designed for business users, and Power View in SharePoint server, an interactive data visualization tool.



Report Builder

While the Report Designer interface in the SQL Server Data Tools for Business Intelligence Microsoft® Visual Studio® add-in provides a comprehensive report authoring environment for BI specialists, it includes many features that are relevant only to professional developers. Business users who are more familiar with productivity software packages, such as Microsoft® Office® applications, can find the Visual Studio interface distracting or even intimidating.

Report Builder is a report authoring tool that combines many Report Designer features with the familiar ribbon-based interface of Microsoft® Office®. It is designed specifically for business users, and is the

primary report authoring tool for self-service reporting scenarios. With Report Builder, business users can create and edit reports as easily as they can work with other kinds of business document, publishing their reports to Report Manager or Microsoft® SharePoint® document libraries.

Report Builder supports the same report features as Report Designer, including:

- Rich formatting.
- Tablix data regions for table, matrix, and list reports.
- Groups, aggregations, and drill-down interactions.
- Charts and images.
- Parameters.

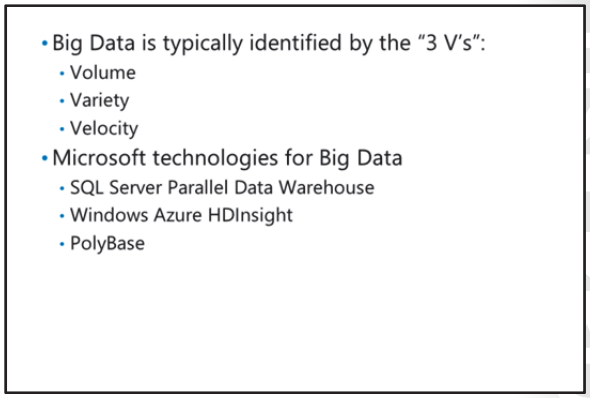
To help users create their own reports, BI professionals can define shared data sources, shared datasets, and report parts that business users can use as building blocks when creating their own reports.

Power View in SharePoint Server

When SQL Server Reporting Services is installed in SharePoint integrated mode, you can enable the Power View SharePoint application so users can create dynamic data visualizations directly in the web browser. Power View reports are very easy to create, and the UI is simple and intuitive, making it easier for inexperienced users to get started with data exploration. Data sources for Power View in SharePoint Server include PowerPivot workbooks, tabular Analysis Services databases, and multidimensional Analysis Services cubes. These enable business users to create self-service data visualizations from their own mash-up data models and managed enterprise data models. Power View visualizations can be saved in SharePoint server as reports that other users can view and extend, or as PowerPoint presentations that can be used to bring live data visualizations to a meeting.

Big Data

Organizations, individuals, services, and devices generate an ever-increasing volume of data at an ever- increasing rate. The growth of trends like social media, the use of digital devices for photography and video capture, and the use of profile data to personalize user experiences and content, has led to a massive expansion of data processing requirements for business organizations and Internet services.



What is Big Data?

The term "Big Data" is used to describe data too large or complex to manage and process in a traditional relational database or data warehouse. While database systems such as Microsoft® SQL Server® 2014 are designed to handle terabytes of data that can be normalized into a relational schema, many organizations find themselves needing to process petabytes of data in multiple, non-relational formats.

Big Data is typified by the so-called "three Vs" definition, in which a data processing problem is defined as a Big Data scenario if the data meets one or more of the following classifications:

- **Volume**. A huge volume of data must be processed, typically hundreds of terabytes or more.
- **Variety**. The data is unstructured, or consists of a mix of structured and unstructured data in many formats.
- **Velocity**. New data is generated at frequent intervals, often as a constant stream of data values. These values can be captured in real time using a technology such as Microsoft® SQL Server® StreamInsight, and then analyzed.

Some examples of typical Big Data problems include:

- Analyzing web server logs for high-traffic web sites.
- Extracting data from social media streams to enable sentiment analysis.
- Processing high volumes of data generated by sensors or devices to detect anomalies.

Microsoft Technologies for Big Data

Microsoft has responded to the demand for Big Data solutions by creating two products that support the storage and analysis of large volumes of data. These products are SQL Server Parallel Data Warehouse, and Windows Azure™ HDInsight.

SQL Server Parallel Data Warehouse

Microsoft® SQL Server® Parallel Data Warehouse is an edition of SQL Server that is only available as a preinstalled and configured solution in enterprise data warehouse appliances from Microsoft and its hardware partners. Parallel Data Warehouse is designed specifically for extremely large-scale data warehouses needing to store and query hundreds of terabytes of data.

A Parallel Data Warehouse appliance consists of a server that acts as the control node, and multiple servers that operate as compute and storage nodes. Each compute node has its own dedicated processors and memory, and is associated with a dedicated storage node. A high-performance network connects the nodes, and dual fiber channels link up the compute and storage nodes. The control node intercepts incoming queries, divides each query into multiple smaller operations, and then passes these on to the compute nodes to process. Each compute node returns the results back to the control node. The control node integrates the data to create a result set, which it then returns to the client.

Control nodes are housed in a control rack. There are three other types of nodes that share this rack with the control node:

- Management nodes, through which administrators manage the appliance.
- Landing Zone nodes, which act as staging areas for data that you load into the data warehouse by using an ETL tool.
- Backup nodes, which back up the data warehouse.

Compute and storage nodes are housed in a separate data rack. To scale the application, you can add more racks as required. Hardware components are duplicated, including control and compute nodes, to provide redundancy.

SQL Server Parallel Data Warehouse is a suitable Big Data solution for large organizations that need to create on-premises enterprise data warehouses for ongoing data analysis and reporting.

Windows Azure™ HDInsight

HDInsight is a cloud-based distribution of Hadoop, and an open source platform for processing Big Data by using a Map/Reduce technique in which the data processing is distributed across multiple nodes in a cluster. The HDFS storage for HDInsight is based on Windows Azure™ blob storage. You can provision and decommission multi-node HDInsight clusters on an as-needed basis to perform Map/Reduce processing of data files in the associated Windows Azure™ Storage containers.

HDInsight supports the core HDFS and Map/Reduce capabilities of Hadoop, as well as related technologies including Hive, Pig, HCatalog, and Oozie. Additionally, an HDInsight SDK for Microsoft® .NET is available that enables developers to create Big Data processing applications that leverage HDInsight and other Windows Azure™ services.

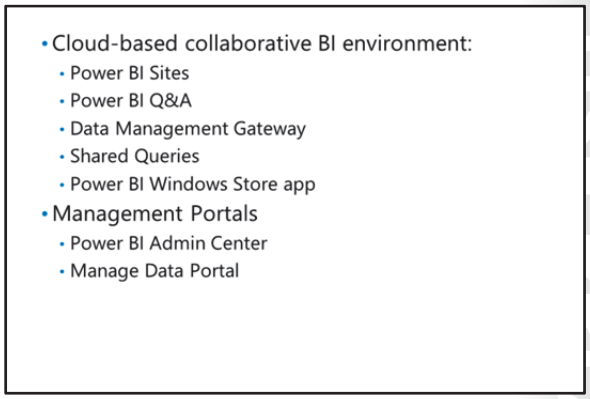
HDInsight is a suitable Big Data solution for organizations needing to process large volumes of data as part of a specific, time-limited initiative or on an occasional basis, paying only for the processing services actually used.

PolyBase

When an organization needs to use both SQL Server Parallel Data Warehouse and Windows Azure™ HDInsight, they can use the PolyBase capabilities of Parallel Data Warehouse to create an integrated Big Data processing solution. PolyBase enables you to define tables in SQL Server Parallel Data Warehouse that reference data in HDInsight, so you can query both SQL Server tables and data in HDFS using Transact-SQL syntax. Additionally, PolyBase supports a hybrid storage technique in which frequently accessed (or "hot") data can be stored in SQL Server tables, while less used (or "cold") data can reside in HDFS. PolyBase intelligently uses the appropriate query engine (SQL Server or HDInsight Map/Reduce) to service client queries, so that client applications do not need to be aware of the actual storage location of the data.

Microsoft® Office 365® Power BI

Power BI for Office 365 builds on the self-service features provided by PowerPivot, Power Query, Power View, and Power Map to create a collaborative environment where business users can share data visualizations and queries. Shared data can also be explored here using natural language query semantics.



Key elements of Power BI for Office 365 include:

- **Power BI Sites**. SharePoint Online sites where users can share workbooks and Power View visualizations.
- **Power BI Q&A**. A SharePoint site where users can create visualizations from PowerPivot workbooks using natural language queries.
- **Data Management Gateway**. A service application that enables organizations to make on-premises data sources available to Power BI cloud services.
- **Shared Queries**. Power Query is used to define queries that individuals can then share with other Office 365 Power BI users in their organization.
- **The Power BI Windows Store app**. An app for Windows that enables users to view reports on mobile devices.

Power BI for Office 365 provides the following two web-based portals for managing the service:

- **The Power BI Admin Center**. Administrators can use this portal to define data sources, register data management gateways, manage user roles, and monitor service health statistics.
- **The Manage Data Portal**. Business users can use this portal to manage their own shared queries and data sources, and view usage analytics for the queries they have shared.

Lab: Using a Data Warehouse

Scenario

In this course, you have created a complete data warehousing solution for the Adventure Works Cycles company. Now you can explore the kinds of BI applications that can be built on the data warehouse.

This lab provides a high-level introduction to some of the ways in which the data warehouse you have built can provide business users with the information they need to perform effectively. You will explore an enterprise BI solution that consists of an SSAS data model and SSRS reports, and use Excel to perform self-service BI analysis. The underlying data for all these BI activities is provided by the data warehouse you have built in this course.

Objectives

After completing this lab, you will be able to:

- View data from a data warehouse in an enterprise BI solution.
- Use a data warehouse in a self-service BI solution.

Estimated Time: 45 minutes

Virtual machine: 20463C-MIA-SQL

User name: ADVENTUREWORKS\Student

Password: Pa$$w0rd

Exercise 1: Exploring an Enterprise BI Solution

Scenario

You have created a data warehouse, and now want to deploy a proof of concept for an enterprise BI solution.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. View an Enterprise Data Model

3. View Enterprise Reports

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. Run **Setup.cmd** in the D:\Labfiles\Lab13\Starter folder as Administrator.

▶ Task 2: View an Enterprise Data Model

1. Start Visual Studio and open the **AW Enterprise BI.sln** solution in the D:\Labfiles\Lab13\Starter folder. If you are prompted to specify a workspace database, specify **localhost\SQL2**.
2. View the **Model.bim** data model in the **AWDataModel** project, noting that it contains tables of data from the data warehouse.
3. View the diagram view for the model, noting the relationships that are defined between the tables.
4. View the measures in the **Reseller Sales** table, noting that a KPI is defined on the **Reseller Margin** measure. The KPI compares the **Reseller Margin** measure to a target of 0.5, indicating that Adventure Works Cycles seeks to achieve a 50 percent margin on reseller sales.

5. View the project properties for the **AWDataModel** project, noting that the data model is configured to be deployed as a cube named **AdventureWorks** in a database called **AWOLAP**.
6. Deploy the **AWDataModel** project, specifying the impersonation credentials **ADVENTUREWORKS\ServiceAcct** with the password **Pa$$w0rd** if prompted.
7. Start Excel and create a blank workbook, and on the Data tab, get external data from the **AdventureWorks** in a database named **AWOLAP** and import it into a PivotTable report.
8. In the PivotTable report, view the **Internet Sales** and **Reseller Sales** measures and KPIs.
9. Add the **Calendar Date** hierarchy from the **Order Date** table and the **Location** hierarchy from the **Geography** table. Then expand the years and locations to drill down into the detailed results for specific time periods and locations.
10. When you have finished exploring the data, close Excel without saving the workbook.

▶ Task 3: View Enterprise Reports

1. In Visual Studio, in the **AW Enterprise BI.sln** solution, note the following shared data sources in the **AWReports** project:
   - **AWDataWarehouse**.rds: This data source connects to the **AWDataWarehouse** SQL Server database.
   - **AWOLAP.rds**: This data source connects to the **AWOLAP** Analysis Services database.
2. View the **Dashboard.rdl** report, and note that it contains visualizations for various business metrics. The report uses a hidden parameter named **Year** to filter the data so that only the data for the latest year is shown.
3. View the **Monthly Sales Report.rdl** report, and note that this report provides details of sales performance for a specific month, which is determined by two parameters named **Year** and **Month**.
4. Deploy the **AWReports** project and close Visual Studio.
5. Use Internet Explorer® to browse to the Adventure Works Portal SharePoint Server site at **http://mia-sql/sites/adventureworks**. Then view the **Dashboard** report in the **Reports** document library.
6. View the **Monthly Sales Report** in the **Reports** document library, and export it to Microsoft® Word®.

Results: After this exercise, you should have deployed an Analysis Services database and some Reporting Services reports in SharePoint Server.

Exercise 2: Exploring a Self-Service BI Solution

Scenario

You have seen how the data warehouse can be used in an enterprise BI solution, and you now want to explore how the data warehouse can support a self-service BI solution. Initially you need to create a report showing the top 10 products sold through the Internet. Then you want to combine some gasoline price data from the Internet with data in the data warehouse to create a custom data model that will enable you to visually compare sales in different regions based on prices.

The main tasks for this exercise are as follows:

1. Use Report Builder to Create a Report
2. Use Excel to Create a Data Model
3. Use Excel to Visualize Data

▶ Task 1: Use Report Builder to Create a Report

1. In Internet Explorer®, in the **Reports** page of the Adventure Works Portal SharePoint Server site, click the **Files** tab on the ribbon, and in the **New Document** drop-down list, click **Report Builder Report**. Then download and run the application.
2. In Report Builder, use the Table or Matrix Wizard to create a new report.
   - The report should include a new dataset based on the existing **AWDataWarehouse.rsds** shared data source, which is in the **Reports\Data Sources** folder in the SharePoint Server site.
   - The dataset should show the top 10 selling products based on the sum of the **OrderQuantity** column in the **FactInternetSales** table grouped by the **ProductName** column from the **DimProduct** table. You can use the following Transact-SQL code for the query:

     ```
     SELECT TOP 10 DimProduct.ProductName, SUM(FactInternetSales.OrderQuantity) AS SalesCount
     FROM DimProduct INNER JOIN FactInternetSales
     ON DimProduct.ProductKey = FactInternetSales.ProductKey
     GROUP BY DimProduct.ProductName
     ORDER BY SalesCount DESC;
     ```

   - Both fields in the dataset should be displayed in the **Values** area of the report.
3. When the report has been created, add the title **Top 10 Internet Sellers**, and format the report to your satisfaction. You can preview it by clicking **Run**.
4. Save the report as **Top 10 Sellers.rdl** in the **Reports** folder in the SharePoint Server site.
5. Close Report Builder and use Internet Explorer to view the **Top 10 Sellers** report.

▶ Task 2: Use Excel to Create a Data Model

1. Create a new, blank Excel workbook and save it as **Analysis.xslx** in the D:\Labfiles\Lab13\Start folder.
2. Enable the following COM add-ins:
   - **Microsoft Office PowerPivot for Excel 2013**
   - **Microsoft Power Query for Excel**
3. Use PowerPivot to manage the workbook data model and import the following tables from the **AWDataWarehouse** database on the MIA-SQL instance of SQL Server, specifying the friendly names indicated:
   - DimCustomer (Customer)
   - DimGeography (Geography)

- FactInternetSales (Internet Sales)

4. View the design of the data model and verify that the relationships between the tables have been automatically detected.
5. In Excel, use Power Query to import the data from **Sheet1** in the **GasPrices.xlsx** workbook in the D:\Labfiles\Lab13\Starter folder.
6. Edit the query created by Power Query to shape the data as follows:
   - Filter the **Country/Region** column so that only rows for **Australia**, **France**, **Germany**, **United Kingdom**, and **United States** are included.
   - Rename the **US$/US gallon (95 RON)** column to **Price per Gallon**.
   - Change the data type of the **Price per Gallon** column to **Number**.
   - Name the query **Gas Prices**.
   - Do not load the query results to the worksheet, but instead load them to the data model.
7. Use PowerPivot to modify the data model and create a relationship that joins the **Gas Prices** table to the **Geography** table based on the **Country/Region** and **CountryRegionName** columns.
8. Create a PivotTable that shows the **Country/Territory** and **Price per Gallon** fields from the **Gas Prices** table with the **SalesAmount** field from the **Internet Sales** table. Then save the workbook.

▶ Task 3: Use Excel to Visualize Data

1. In Excel, in the **Analysis.xlsx** workbook, enable the **Power View** COM add-in.
2. Insert a Power View report with the title **Sale By Geography**.
3. In the top half of the Power View report, create a stacked column chart that shows the sum of the **SalesAmount** field in the **Internet Sales** table by the **CountryRegionName** field in the **Geography** table.
4. In the bottom half of the Power View report, create a clustered bar chart that shows the **Price per Gallon** and average **SalesAmount** for each **Country/Region** in the **Gas Prices** table.
5. Click bars in the bar chart and note that the relationship between the tables causes the rest of the chart to be filtered based on the country/region that you clicked.
6. Save the workbook and close Excel.

Results: After this exercise, you should have a report named Top Sellers and an Excel workbook called Analysis.xslx.

Module Review and Takeaways

In this module, you have learned how enterprise and self-service BI solutions can use a data warehouse.

Review Question(s)

Question: What are some of the issues you need to consider when designing a data warehouse that must support self-service BI?

Course Evaluation

- Your evaluation of this course will help Microsoft understand the quality of your learning experience.
- Please work with your training provider to access the course evaluation form.
- Microsoft will keep your answers to this survey private and confidential and will use your responses to improve your future learning experience. Your open and honest feedback is valuable and appreciated.

Your evaluation of this course will help Microsoft understand the quality of your learning experience.

Please work with your training provider to access the course evaluation form.

Microsoft will keep your answers to this survey private and confidential and will use your responses to improve your future learning experience. Your open and honest feedback is valuable and appreciated.

Module 1: Introduction to Data Warehousing

Lab: Exploring a Data Warehousing Solution

Exercise 1: Exploring Data Sources

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Labfiles\Lab01\Starter folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. Click **Yes** when prompted to confirm that you want to run the command file, and then wait for the script to finish.

▶ Task 2: View the Solution Architecture

1. In the D:\Labfiles\Lab01\Starter folder, double-click **Adventure Works DW Solution.jpg** to open it in Paint.
2. Examine the diagram, and note that it shows several data sources on the left, which provide the source data for the data warehouse. You will examine these data sources in the remainder of this exercise.

Note: In addition to the data sources that you will examine in this lab, the diagram includes a Microsoft® SQL Server® Master Data Services model for product data and a SQL Server Data Quality Services task to cleanse data as it is staged. These elements form part of the complete solution for the lab scenario in this course, but they are not present in this lab.

3. Minimize Paint. You will return to it in the next exercise.

▶ Task 3: View the Internet Sales Data Source

1. Start SQL Server Management Studio, and when prompted, connect to the **MIA-SQL** database engine instance using Windows authentication.
2. Open the **View Internet Sales.sql** query file in the D:\Labfiles\Lab01\Starter folder.
3. Click **Execute** to run the query. When the query completes, review the results and note that this data source contains data about customers and the orders they have placed through the Adventure Works e-commerce website.
4. Keep SQL Server Management Studio open for the next task.

▶ Task 4: View the Reseller Sales Data Source

1. In SQL Server Management Studio, open the **View Reseller Sales.sql** query file in the D:\Labfiles\Lab01\Starter folder.
2. Click **Execute** to run the query. When the query completes, review the results and note that this data source contains data about resellers and the orders they have placed through Adventure Works reseller account managers.
3. Keep SQL Server Management Studio open for the next task.

▶ Task 5: View the Products Data Source

1. In SQL Server Management Studio, open the **View Products.sql** query file in the D:\Labfiles\Lab01\Starter folder.

2. Click **Execute** to run the query. When the query completes, review the results and note that this source contains data about products that Adventure Works sells, organized into categories and subcategories.
3. Keep SQL Server Management Studio open for the next task.

▶ Task 6: View the Human Resources Data Source

1. In SQL Server Management Studio, open the **View Employees.sql** query file the D:\Labfiles\Lab01\Starter folder.
2. Click **Execute** to run the query. When the query completes, review the results and note that this source contains data about employees, including the sales representatives associated with reseller sales.
3. Minimize SQL Server Management Studio. You will return to it later in this exercise.

▶ Task 7: View the Accounts Data Source

1. View the contents of the D:\Accounts folder that contains several comma-delimited text files.
2. Start Excel, and use it to open the **Payments – AU.csv** file in the D:\Accounts folder.
3. Review the file contents and note that it contains data about reseller payments processed by the Adventure Works accounting system. Each file in the Accounts folder relates to payments made by resellers in a specific country.
4. Close Excel without saving any changes.

▶ Task 8: View the Staging Database

1. Restore SQL Server Management Studio.
2. In Object Explorer, expand **Databases**, expand **Staging**, and then expand **Tables**.
3. Right-click the **dbo.Customers** table, and then click **Select Top 1000 Rows**. When the query completes, note that this table is empty.
4. Repeat the previous step to verify that the **dbo.EmployeeDeletes**, **dbo.EmployeeInserts**, **dbo.EmployeeUpdates**, **dbo.InternetSales**, **dbo.Payments**, **dbo.Resellers**, and **dbo.ResellerSales** tables are also empty.

Note: The **dbo.ExtractLog** table contains data that is used to track data extractions from the **Internet Sales** and **Reseller Sales** data sources.

5. Minimize SQL Server Management Studio. You will return to it in the next exercise.

Results: After this exercise, you should have viewed data in the InternetSales, ResellerSales, Human Resources, and Products SQL Server databases, viewed payments data in comma-delimited files, and viewed an empty staging database.

Exercise 2: Exploring an ETL Process

▶ Task 1: View the Solution Architecture

1. Maximize Paint and view the solution architecture diagram.
2. Note that the ETL solution consists of two main phases: a process to extract data from the various data sources and load it into a staging database, and another to load the data in the staging database into the data warehouse. In this exercise, you will observe these ETL processes as they run.
3. Minimize Paint. You will return to it in the next exercise.

▶ Task 2: Run the ETL Staging Process

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab01\Starter folder.
2. If the Solution Explorer pane is not visible, on the **View** menu, click **Solution Explorer**. If necessary, click the pin icon to freeze it in position.
3. In Solution Explorer, in the **SSIS Packages** folder, double-click **Stage Data.dtsx** to open it. Note that the staging process consists of five tasks:
   - Stage Internet Sales
   - Stage Reseller Sales
   - Stage Payments
   - Stage Employees
   - Notify Completion
4. On the **Debug** menu, click **Start Debugging**, and then observe that the staging process runs a SQL Server Integration Services package for each task.
5. When the message **Staging process complete** is displayed, click **OK**, and then on the **Debug** menu, click **Stop Debugging**. Note that the message box may be hidden by the Visual Studio window. Look for a new icon on the taskbar, and then click it to bring the message box to the front.
6. Minimize Visual Studio. You will return to it later in this exercise.

▶ Task 3: View the Staged Data

1. Restore SQL Server Management Studio.
2. If necessary, in Object Explorer, expand **Databases**, expand **Staging**, and then expand **Tables**.
3. Right-click the **dbo.Customers** table, and then click **Select Top 1000 Rows**. When the query completes, note that this table now contains data that the ETL process has extracted from the data source.
4. Repeat the previous step to verify that the **dbo.EmployeeInserts**, **dbo.InternetSales**, **dbo.Payments**, **dbo.Resellers**, and **dbo.ResellerSales** tables also contain staged data.
5. Minimize SQL Server Management Studio. You will return to it later in this exercise.

▶ Task 4: Run the ETL Data Warehouse Load Process

1. Restore Visual Studio.
2. In Solution Explorer, in the SSIS Packages folder, double-click **Load DW.dtsx** to open it. Note that the data warehouse loading process consists of a sequence of tasks to load various dimensions and facts, followed by a task to determine the number of records loaded from each staging table before truncating the staging tables, and a task to log the row counts.

3. On the **Debug** menu, click **Start Debugging**, and then observe that the data warehouse loading process runs an SSIS package for the dimension or fact table to be loaded. The process may take several minutes to complete.
4. When the message **Data warehouse load complete** is displayed, click **OK**, and then on the **Debug** menu, click **Stop Debugging**. Note that the message box may be hidden by the Visual Studio window. Look for a new icon on the taskbar, and then click it to bring the message box to the front.
5. Close Visual Studio.

Results: After this exercise, you should have viewed and run the SQL Server Integration Services packages that perform the ETL process for the Adventure Works data warehousing solution.

Exercise 3: Exploring a Data Warehouse

▶ Task 1: View the Solution Architecture

1. Maximize Paint and view the solution architecture diagram.
2. Note that the data warehouse provides a central data source for business reporting and analysis.
3. Close Paint without saving any changes.

▶ Task 2: Query the Data Warehouse

1. Restore SQL Server Management Studio, and open the **Query DW.sql** query file in the D:\Labfiles\Lab01\Starter folder.
2. Click **Execute** to run the query. When the query completes, review the results and note that the query uses the data warehouse to retrieve:
   - Total sales for each country by fiscal year.
   - Total units sold for each product category by calendar year.
3. Close SQL Server Management Studio without saving any changes.

Results: After this exercise, you should have successfully retrieved business information from the data warehouse.

Module 2: Planning Data Warehouse Infrastructure

Lab: Planning Data Warehouse Infrastructure

Exercise 1: Planning Data Warehouse Hardware

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Labfiles\Lab02\Starter folder, right-click **Setup.cmd** and then click **Run as administrator**.
3. Click **Yes** when prompted to confirm that you want to run the command file, and then wait for the script to finish.

▶ Task 2: Measure Maximum Consumption Rate

1. Start SQL Server Management Studio and connect to the MIA-SQL database engine using Windows authentication.
2. In SQL Server Management Studio, open the **Create BenchmarkDB.sql** query file in the D:\Labfiles\Lab02\Starter folder.
3. Click **Execute**, and wait for query execution to complete.
4. In SQL Server Management Studio, open the **Measure MCR.sql** query file in the D:\Labfiles\Lab02\Starter folder.
5. Click **Execute**, and wait for query execution to complete.
6. In the results pane, click the **Messages** tab.
7. Add the **logical reads** value for the two queries together, and then divide the result by two to find the average.
8. Add the **CPU time** value for the two queries together, and then divide the result by two to find the average. Divide the result by 100 to convert it to seconds.
9. Calculate MCR by using the following formula:

```
(average logical reads / average CPU time) * 8 / 1024
```

10. Calculate the number of cores required to support a workload with an average query size of 500 MB, 10 concurrent users, and a target response time of 20 seconds:

```
((500 / MCR) * 10) / 20
```

11. Close SQL Server Management Studio without saving any files.

▶ Task 3: Estimate Server Hardware Requirements

1. In the D:\Labfiles\Lab02\Starter folder, double-click **DW Hardware Spec.xlsx** to open it in Microsoft Excel®.
2. In cell **F1**, use the following formula to calculate the number of cores required for the given workload figures:

```
=((B6/C3)*B7)/B8
```

3. Based on the results of the preceding formula, recommend the number and type of processors to include in the data warehouse server.
4. Calculate the volume of fact data in gigabytes (estimated fact rows multiplied by bytes per row, divided by 100,000), and add 50 GB for indexes and dimensions. Then divide the result by three to allow for a 3:1 compression ratio. The resulting figure is the required data storage.
5. Add 50 GB each for log space, TempDB storage, and staging data to calculate the total data volume.
6. Assuming an annual data growth of 150 GB, calculate the required storage capacity in three years.
7. Based on the data volume and CPU requirements, suggest a suitable amount of memory for the server.
8. In the D:\Labfiles\Lab02\Starter folder, double-click **Storage Options.docx** and review the available options for storage hardware. Then, based on the storage requirements you have calculated, select a suitable option for the data warehouse.
9. Record your recommendations in DW Hardware Spec.xlsx, and then close Excel and Word, saving your changes.

Results: After this exercise, you should have a completed worksheet that specifies the required hardware for your data warehouse server.

Module 3: Designing and Implementing a Data Warehouse

Lab: Implementing a Data Warehouse

Exercise 1: Implement a Star Schema

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Labfiles\Lab03\Starter folder, right-click **Setup.cmd** and then click **Run as administrator**.
3. Click **Yes** when prompted to confirm that you want to run the command file, and then wait for the script to finish.

▶ Task 2: View a Data Warehouse Schema

1. Start SQL Server Management Studio and connect to the **MIA-SQL** instance of the SQL Server database engine using Windows authentication.
2. In Object Explorer, expand **Databases**, expand **AWDataWarehouse**, and then expand **Tables**. Note that the database contains four tables.
3. Right-click the **Database Diagrams** folder, and then click **New Database Diagram**. If you are prompted to create the required support objects, click **Yes**.
4. In the **Add Table** dialog box, click each table while holding down the **Ctrl** key to select them all, click **Add**, and then click **Close**.
5. In the database diagram, click each table while holding down the **Ctrl** key to select them all.
6. On the toolbar, in the **Table View** drop-down list, click **Standard**.
7. Arrange the tables and adjust the zoom level so you can see the entire database schema, and then examine the tables, noting the columns that they contain.
8. Note that the **FactResellerSales** table contains foreign key columns that relate it to the **DimReseller**, **DimEmployee**, and **DimProduct** tables. It also contains some numerical measures that can be aggregated to provide useful business information, such as the total sales amount per reseller or the total quantity of units sold per product.
9. On the **File** menu, click **Save Diagram_0**, enter the name **AWDataWarehouse Schema**, and then click **OK**.

▶ Task 3: Create a Dimension Table

1. In Microsoft SQL Server Management Studio, open the **DimCustomer.sql** query file in the D:\Labfiles\Lab03\Starter folder.
2. Review the Transact-SQL code, noting that it creates a table named **DimCustomer** in the **AWDataWarehouse** database.
3. Click **Execute** to create the table.

▶ Task 4: Create a Fact Table

1. In Microsoft SQL Server Management Studio, open the **FactInternetSales.sql** query file in the D:\Labfiles\Lab03\Starter folder.

2. Review the Transact-SQL code, noting that it creates a table named **FactInternetSales** in the **AWDataWarehouse** database, and that this table is related to the **DimCustomer** and **DimProduct** tables.
3. Click **Execute** to create the table.

▶ Task 5: View the Revised Data Warehouse Schema

1. In SQL Server Management Studio, view the database diagram window that you opened previously.
2. On the **Database Diagram** menu, click **Add Table**.
3. In the **Add Table** dialog box, click **Refresh**, select the **DimCustomer** and **FactInternetSales** tables, click **Add**, and then click **Close**.
4. Select each of the new tables and then on the toolbar, in the **Table View** drop-down list, click **Standard**.
5. Arrange the tables and adjust the zoom level so that you can see the entire database schema.
6. On the **File** menu, click **Save AWDataWarehouse Schema**.
7. Keep SQL Server Management Studio open. You will return to it in the next exercise.

Results: After this exercise, you should have a database diagram in the **AWDataWarehouse** database that shows a star schema consisting of two fact tables (**FactResellerSales** and **FactInternetSales**) and four dimension tables (**DimReseller**, **DimEmployee**, **DimProduct**, and **DimCustomer**).

Exercise 2: Implementing a Snowflake Schema

▶ Task 1: Create Dimension Tables That Form a Hierarchy

1. In Microsoft SQL Server Management Studio, open the **DimProductCategory.sql** query file in the D:\Labfiles\Lab03\Starter folder.
2. Review the Transact-SQL code, noting that it:
   - Creates a table named **DimProductCategory**.
   - Creates a table named **DimProductSubcategory** that has a foreign-key relationship to the **DimProductCategory** table.
   - Drops the **ProductSubcategoryName** and **ProductCategoryName** columns from the **DimProduct** table.
   - Adds a **ProductSubcategoryKey** column to the **DimProduct** table that has a foreign-key relationship to the **DimProductSubcategory** table.
3. Click **Execute** to create the tables.

▶ Task 2: Create a Shared Dimension table

1. In Microsoft SQL Server Management Studio, open the **DimGeography.sql** query file in the D:\Labfiles\Lab03\Starter folder.
2. Review the Transact-SQL code, noting that it:
   - Creates a table named **DimGeography**.
   - Drops the **City**, **StateProvinceName**, **CountryRegionCode**, **CountryRegionName**, and **PostalCode** columns from the **DimReseller** table.

- Adds a **GeographyKey** column to the **DimReseller** table that has a foreign-key relationship to the **DimGeography** table.
- Drops the **City**, **StateProvinceName**, **CountryRegionCode**, **CountryRegionName**, and **PostalCode** columns from the **DimCustomer** table.
- Adds a **GeographyKey** column to the **DimCustomer** table that has a foreign-key relationship to the **DimGeography** table.

3. Click **Execute** to create the table.

▶ Task 3: View the Data Warehouse Schema

1. In SQL Server Management Studio, view the database diagram window that you opened previously.
2. Select the **DimProduct**, **DimReseller**, and **DimCustomer** tables, and then press **Delete** to remove them from the diagram (this does not drop the tables from the database).
3. On the **Database Diagram** menu, click **Add Table**.
4. In the **Add Table** dialog box, click **Refresh**, select the **DimCustomer**, **DimGeography**, **DimProduct**, **DimProductCategory**, **DimProductSubcategory**, and **DimReseller** tables, click **Add**, and then click **Close**.
5. Select each of the new tables and then on the toolbar, in the **Table View** drop-down list, click **Standard**.
6. Arrange the tables and adjust the zoom level so that you can see the entire database schema.
7. On the **File** menu, click **Save AWDataWarehouse Schema**.
8. Keep SQL Server Management Studio open. You will return to it in the next exercise.

Results: After this exercise, you should have a database diagram in the **AWDataWarehouse** database showing a snowflake schema that contains a dimension consisting of a **DimProduct**, **DimProductSubcategory**, and **DimProductCategory** hierarchy of tables, as well as a **DimGeography** dimension table that is referenced by the **DimCustomer** and **DimReseller** dimension tables.

Exercise 3: Implementing a Time Dimension Table

▶ Task 1: Create a Time Dimension Table

1. In Microsoft® SQL Server® Management Studio, open the **DimDate.sql** query file in the D:\Labfiles\Lab03\Starter folder.
2. Review the Transact-SQL code, noting that it:
   - Creates a table named **DimDate**.
   - Adds **OrderDateKey** and **ShipDateKey** columns to the **FactInternetSales** and **FactResellerSales** tables that have foreign-key relationships to the **DimDate** table.
   - Creates indexes on the **OrderDateKey** and **ShipDateKey** foreign-key fields in the **FactInternetSales** and **FactResellerSales** tables.
3. Click **Execute** to create the table.

▶ Task 2: View the Database Schema

1. In SQL Server Management Studio, view the database diagram window that you opened previously.
2. Select the **FactResellerSales** and **FactInternetSales** tables, and then press **Delete** to remove them from the diagram (this does not drop the tables from the database).
3. On the **Database Diagram** menu, click **Add Table**.
4. In the **Add Table** dialog box, click **Refresh**, select the **DimDate**, **FactInternetSales**, and **FactResellerSales** tables, click **Add**, and then click **Close**.
5. Select each of the new tables and then on the toolbar, in the **Table View** drop-down list, click **Standard**.
6. Arrange the tables and adjust the zoom level so you can see the entire database schema.
7. On the **File** menu, click **Save AWDataWarehouse Schema**.

▶ Task 3: Populate the Time Dimension Table

1. In Microsoft SQL Server Management Studio, open the **GenerateDates.sql** query file in the D:\Labfiles\Lab03\Starter folder.
2. Review the Transact-SQL code, noting that it:
   - Declares a variable named **@StartDate** with the value 1/1/2000, and a variable named **@EndDate** with the value of the current date.
   - Performs a loop to insert appropriate values for each date between **@StartDate** and **@EndDate** into the **DimDate** table.
3. Click **Execute** to populate the table.
4. When the script has completed, in Object Explorer, right-click the **Tables** folder for the **AWDataWarehouse** database, and then click **Refresh**.
5. Right-click the **DimDate** table, and then click **Select Top 1000 Rows**.
6. View the data in the table, noting that it contains appropriate values for each date.
7. Close SQL Server Management Studio, saving changes if you are prompted.

Results: After this exercise, you should have a database that contains a **DimDate** dimension table that is populated with date values from January 1, 2000, to the current date.

Module 4: Creating an ETL Solution with SSIS

Lab: Implementing Data Flow in an SSIS Package

Exercise 1: Exploring Source Data

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Labfiles\Lab04\Starter folder, right-click **Setup.cmd** and then click **Run as administrator**.
3. Click **Yes** when prompted to confirm that you want to run the command file, and then wait for the script to finish.

▶ Task 2: Extract and View Sample Source Data

1. On the Start screen, type **Import and Export** and then start the **SQL Server 2014 Import and Export Data (64-bit)** app.
2. On the **Welcome to SQL Server Import and Export Wizard** page, click **Next**.
3. On the **Choose a Data Source** page, specify the following options, and then click **Next**:
   - **Data source**: SQL Server Native Client 11.0
   - **Server name**: localhost
   - **Authentication**: Use Windows Authentication
   - **Database**: InternetSales
4. On the **Choose a Destination** page, set the following options, and then click **Next**:
   - **Destination**: Flat File Destination
   - **File name**: D:\Labfiles\Lab04\Starter\Top 1000 Customers.csv
   - **Locale**: English (United States)
   - **Unicode**: Unselected
   - **Code page**: 1252 (ANSI – Latin 1)
   - **Format**: Delimited
   - **Text qualifier**: " (a quotation mark)
   - **Column names in the first data row**: Selected
5. On the **Specify Table Copy or Query** page, select **Write a query to specify the data to transfer**, and then click **Next**.
6. On the **Provide a Source Query** page, enter the following Transact-SQL code, and then click **Next**:

```
SELECT TOP 1000 * FROM Customers
```

7. On the **Configure Flat File Destination** page, select the following options, and then click **Next**:
   - **Source query**: [Query]
   - **Row delimiter**: {CR}{LF}
   - **Column delimiter**: Comma {,}
8. On the **Save and Run Package** page, select only **Run immediately**, and then click **Next**.
9. On the **Complete the Wizard** page, click **Finish**.
10. When the data extraction has completed successfully, click **Close**.
11. Double click to open file **Top 1000 Customers.csv** in the D:\Labfiles\Lab04\Starter folder.
12. Examine the data, noting the columns that exist in the **Customers** table and the range of data values they contain, and then close Excel without saving any changes.

▶ Task 3: Profile Source Data

1. Start Visual Studio, and on the **File** menu, point to **New**, and then click **Project**.
2. In the **New Project** dialog box, select the following values, and then click **OK**:
   - **Project Template**: Integration Services Project
   - **Name**: Explore Internet Sales
   - **Location**: D:\Labfiles\Lab04\Starter
   - **Create directory for solution**: Selected
   - **Solution name**: Explore Internet Sales
3. If the **Getting Started (SSIS)** window is displayed, close it.
4. In the Solution Explorer pane, right-click **Connection Managers**, and then click **New Connection Manager**.
5. In the **Add SSIS Connection Manager** dialog box, click **ADO.NET**, and then click **Add**.
6. In the **Configure ADO.NET Connection Manager** dialog box, click **New**.
7. In the **Connection Manager** dialog box, enter the following values, and then click **OK**:
   - **Server name**: localhost
   - **Log on to the server**: Use Windows Authentication
   - **Select or enter a database name**: InternetSales
8. In the **Configure ADO.NET Connection Manager** dialog box, verify that a data connection named **localhost.InternetSales** is listed, and then click **OK**.
9. If the SSIS Toolbox pane is not visible, on the **SSIS** menu, click **SSIS Toolbox**. Then, in the SSIS Toolbox pane, in the **Common** section, double-click **Data Profiling Task** to add it to the Control Flow surface. Alternatively, you can drag the task icon to the Control Flow surface.
10. Double-click the **Data Profiling Task** icon on the Control Flow surface to open its editor.
11. In the **Data Profiling Task Editor** dialog box, on the **General** tab, in the **Destination** property value drop-down list, click **<New File connection...>**.
12. In the **File Connection Manager Editor** dialog box, in the **Usage type** drop-down list, click **Create file**.
13. In the **File** box, type **D: \ETL\Internet Sales Data Profile.xml**, and then click **OK**.

14. In the **Data Profiling Task Editor** dialog box, on the **Profile Requests** tab, in the **Profile Type** drop-down list, select **Column Statistics Profile Request**, and then click the **RequestID** column.
15. In the **Request Properties** pane, set the following property values. Do not click **OK** when finished:
    - **ConnectionManager**: localhost.InternetSales
    - **TableOrView**: [dbo].[SalesOrderHeader]
    - **Column**: OrderDate
16. In the **Data Profiling Task Editor** dialog box, on the **Profile Requests** tab, in the **Profile Type** drop-down list for the empty row under the profile you just added, select **Column Length Distribution Profile Request**, and then click the **RequestID** column.
17. In the **Request Properties** pane, set the following property values. Do not click OK when finished:
    - **ConnectionManager**: localhost.InternetSales
    - **TableOrView**: [dbo].[Customers]
    - **Column**: AddressLine1
    - **IgnoreLeadingSpaces**: False
    - **IgnoreTrailingSpaces**: True
18. In the **Data Profiling Task Editor** dialog box, on the **Profile Requests** tab, in the **Profile Type** drop-down list for the empty row under the profile you just added, select **Column Null Ratio Profile Request**, and then click the **RequestID** column.
19. In the **Request Properties** pane, set the following property values. Do not click **OK** when finished:
    - **ConnectionManager**: localhost.InternetSales
    - **TableOrView**: [dbo].[Customers]
    - **Column**: AddressLine2
20. In the **Data Profiling Task Editor** dialog box, on the **Profile Requests** tab, in the **Profile Type** drop-down list for the empty row under the profile you just added, select **Value Inclusion Profile Request**, and then click the **RequestID** column.
21. In the **Request Properties** pane, set the following property values:
    - **ConnectionManager**: localhost.InternetSales
    - **SubsetTableOrView**: [dbo].[SalesOrderHeader]
    - **SupersetTableOrView**: [dbo].[PaymentTypes]
    - **InclusionColumns**:
      - **Subset side Columns**: PaymentType
      - **Superset side Columns**: PaymentTypeKey
    - **InclusionThresholdSetting**: None
    - **SupersetColumnsKeyThresholdSetting**: None
    - **MaxNumberOfViolations**: 100
22. In the **Data Profiling Task Editor** dialog box, click **OK**.
23. On the **Debug** menu, click **Start Debugging**.

24. When the Data Profiling task has completed, with the package still running, double-click the **Data Profiling** task, and then click **Open Profile Viewer**.
25. Maximize the **Data Profile Viewer** window, and under the **[dbo].[SalesOrderHeader]** table, click **Column Statistics Profiles**. Then review the minimum and maximum values for the **OrderDate** column.
26. Under the **[dbo].[Customers]** table, click **Column Length Distribution Profiles** and click the **AddressLine1** column to view the statistics. Click the bar chart for any of the column lengths, and then click the **Drill Down** button to view the source data that matches the selected column length.
27. Under the **[dbo].[Customers]** table, click **Column Null Ratio Profiles** and view the null statistics for the **AddressLine2** column. Select the **AddressLine2** column, and then click the **Drill Down** button to view the source data.
28. Under the **[dbo].[SalesOrderHeader]** table, click **Inclusion Profiles** and review the inclusion statistics for the **PaymentType** column. Select the inclusion violation for the payment type value of **0**, and then click the **Drill Down** button to view the source data.
29. Close the **Data Profile Viewer** window, and then in the **Data Profiling Task Editor** dialog box, click **Cancel**.
30. On the **Debug** menu, click **Stop Debugging**.
31. Close Visual Studio, saving your changes if you are prompted.

Results: After this exercise, you should have a comma-separated text file that contains a sample of customer data, and a data profile report that shows statistics for data in the **InternetSales** database.

Exercise 2: Transferring Data by Using a Data Flow Task

▶ Task 1: Examine an Existing Data Flow

1. Start Visual Studio and open the AdventureWorksETL.sln solution in the D:\Labfiles\Lab04\Starter\Ex2 folder.
2. In the Solution Explorer pane, expand **SSIS Packages** and double-click **Extract Reseller Data.dtsx** to open it in the designer.
3. On the Control Flow surface, note that the package contains two Data Flow tasks.
4. Double-click the **Extract Resellers Data Flow** task to view it on the **Data Flow** tab and note that it contains a data source named **Resellers** and a data destination named **Staging DB**.
5. Double-click the **Resellers** data source, note the following details, and then click **Cancel**:
   - On the **Connection Manager** page, the data source is configured to use an OLE DB connection manager named **localhost.ResellerSales** and extracts data from the **[dbo].[Resellers]** table.
   - On the **Columns** tab, the data source is configured to retrieve every column from the **Resellers** table, and the output columns that the task generates have the same names as the source columns.

6. Double-click the **Staging DB** data destination, note the following details, and then click **Cancel**:
   - On the **Connection Manager** page, the data destination is configured to use an OLE DB connection manager named **localhost.Staging** and to use a **Table or view - fast load** mode to insert data into the **[dbo].[Resellers]** table.
   - On the **Mappings** tab, the data destination is configured to map the input columns (which are the output columns from the **Resellers** data source) to columns in the destination table. The order of the columns in the destination is different from the column order in the source, and the source **ResellerKey** column is mapped to the **ResellerBusinessKey** destination column.
7. Right-click anywhere on the **Data Flow** surface, click **Execute Task**, and then observe the task as it runs, noting how many rows are transferred.
8. On the **Debug** menu, click **Stop Debugging**.

▶ Task 2: Create a Data Flow task

1. In Solution Explorer, right-click **SSIS Packages**, and then click **New SSIS Package**.
2. Right-click **Package1.dtsx**, click **Rename**, and then change the name of the package to **Extract Internet Sales Data.dtsx**.
3. With the **Extract Internet Sales Data.dtsx** package open, and the Control Flow surface visible, in the SSIS Toolbox pane, double-click **Data Flow Task**, and then drag the new Data Flow task to the center of the Control Flow surface.
4. Right-click **Data Flow Task** on the Control Flow surface, click **Rename**, and then change the name of the task to **Extract Customers**.
5. Double-click the **Extract Customers** Data Flow task to view the Data Flow surface.

▶ Task 3: Add a Data Source to a Data Flow

1. In Solution Explorer, right-click **Connection Managers**, and then click **New Connection Manager**.
2. In the **Add SSIS Connection Manager** dialog box, click **OLEDB**, and then click **Add**.
3. In the **Configure OLE DB Connection Manager** dialog box, click **New**.
4. In the **Connection Manager** dialog box, enter the following values, and then click **OK**:
   - **Server name**: localhost
   - **Log on to the server**: Use Windows Authentication
   - **Select or enter a database name**: InternetSales

Note: When you create a connection manager, it is named automatically based on the server and database name, for example, **localhost.InternetSales**. If you have previously created a connection manager for the same database, a name such as **localhost.InternetSales1** may be generated.

5. In the **Configure OLE DB Connection Manager** dialog box, verify that a new data connection is listed, and then click **OK**.
6. In the SSIS Toolbox pane, in the **Favorites** section, double-click **Source Assistant**.
7. In the **Source Assistant - Add New Source** dialog box, in the **Select source type** list, select **SQL Server**, in the **Select connection manager** list, select the connection manager for the **localhost.InternetSales1** database that you created previously, and then click **OK**.
8. Drag the new **OLE DB Source** data source to the center of the Data Flow surface, right-click it, click **Rename**, and then change the name of the data source to **Customers**.
9. Double-click the **Customers** source, set the following configuration values, and then click **OK**:

- On the **Connection Manager** page, ensure that the OLE DB connection manager for the **localhost.InternetSales** database is selected, ensure that the **Table or view** data access mode is selected, and then in the **Name of the table or the view** drop-down list, click **[dbo].[Customers]**.
- On the **Columns** tab, ensure that every column from the **Customers** table is selected, and that the output columns have the same names as the source columns.

▶ Task 4: Add a Data Destination to a Data Flow

1. In the SSIS Toolbox pane, in the **Favorites** section, double-click **Destination Assistant**.
2. In the **Destination Assistant - Add New Destination** dialog box, in the **Select destination type** list, click **SQL Server**. In the **Select connection manager** list, click **localhost.Staging**, and then click **OK**.
3. Drag the new **OLE DB Destination** data destination below the **Customers** data source, right-click it, click **Rename**, and then change the name of the data destination to **Staging DB**.
4. On the Data Flow surface, click the **Customers** source, and then drag the blue arrow from the **Customers** data source to the **Staging DB** destination.
5. Double-click the **Staging DB** destination, set the following configuration values, and then click **OK**:
   - On the **Connection Manager** page, ensure that the **localhost.Staging** OLE DB connection manager is selected, ensure that the **Table or view – fast load** data access mode is selected. In the **Name of the table or the view** drop-down list, click **[dbo].[Customers]**, and then click **Keep nulls**.
   - On the **Mappings** tab, drag the **CustomerKey** column from the list of available input columns to the **CustomerBusinessKey** column in the list of available destination columns. Then verify that all other input columns are mapped to destination columns of the same name.

▶ Task 5: Test the Data Flow Task

1. Right-click anywhere on the Data Flow surface, click **Execute Task**, and then observe the task as it runs, noting how many rows are transferred.
2. On the **Debug** menu, click **Stop Debugging**.
3. Close Visual Studio.

Results: After this exercise, you should have an SSIS package that contains a single Data Flow task, which extracts customer records from the **InternetSales** database and inserts them into the Staging database.

Exercise 3: Using Transformations in a Data Flow

▶ Task 1: Examine an Existing Data Flow

1. Start Visual Studio and open the AdventureWorksETL.sln solution in the D:\Labfiles\Lab04\Starter\Ex3 folder.
2. In Solution Explorer, expand **SSIS Packages** and double-click **Extract Reseller Data.dtsx**.
3. On the **Data Flow** tab, in the **Data Flow Task** drop-down list, click **Extract Reseller Sales**. Note that this data flow includes a data source, two transformations, and two destinations.
4. Double-click the **Reseller Sales** data source, and in the **OLE DB Source Editor** dialog box, note the following details, and then click **Cancel**:

- On the **Connection Manager** page, the data source is configured to use an OLE DB connection manager named **localhost.ResellerSales** and to extract data using a Transact-SQL command that queries the **SalesOrderHeader**, **SalesOrderDetail**, and **PaymentTypes** tables. The query includes an ISNULL function to check for a payment type. If none is specified, the value "other" is used.
- On the **Columns** tab, the data source is configured to retrieve several columns including **ProductKey**, **OrderQuantity**, and **UnitPrice**.

5. Double-click the **Calculate Sales Amount** transformation, and in the **Derived Column Transformation Editor** dialog box, note the following details, and then click **Cancel**:
   - The transformation creates a derived column named **SalesAmount**.
   - The derived column is added as a new column to the data flow.
   - The column value is calculated by multiplying the **UnitPrice** column value by the **OrderQuantity** column value.
6. Double-click the **Lookup Product Details** transformation, and in the **Lookup Transformation Editor** dialog box, note the following details, and then click **Cancel**:
   - On the **General** tab, the Lookup transformation is configured to use full cache mode and an OLE DB connection manager, and to redirect rows with no matching entries.
   - On the **Connection** tab, the Lookup transformation is configured to return the results of a Transact-SQL query using the **localhost.Products** OLE DB connection manager. The query returns a table that contains product data from the **Products** database.
   - On the **Columns** tab, the Lookup transformation is configured to match rows in the data flow with products data based on the **ProductKey** column value, and add all the other columns in the products data as new columns in the data flow.
7. Right-click the arrow between the **Lookup Product Details** transformation and the **Staging DB** destination, and then click **Properties**. Note that this arrow represents the lookup match output, so rows where a matching product record was found will follow this data flow path.
8. Right-click the arrow between the **Lookup Product Details** transformation and the **Orphaned Sales** destination, and then click **Properties**. Note that this arrow represents the lookup no match output, so rows where no matching product record was found will follow this data flow path.
9. Double-click the **Staging DB** data destination, note the following details, and then click **Cancel**:
   - On the **Connection Manager** page, the data destination is configured to use an OLE DB connection manager named **localhost.Staging** and to use a **Table or view - fast load** access mode to insert data into the **[dbo].[ResellersSales]** table.
   - On the **Mappings** tab, the order of the columns in the destination is different from the column order in the source, and the **ProductKey** and **ResellerKey** source columns are mapped to the **ProductBusinessKey** and **ResellerBusinessKey** destination columns.
10. Double-click the **Orphaned Sales** destination, note that it uses a flat file connection manager named **Orphaned Reseller Sales**, and then click **Cancel**.
11. In the Connection Managers pane at the bottom of the design surface, double-click the **Orphaned Reseller Sales** connection manager, note that the rows for sales with no matching product record are redirected to the **Orphaned Reseller Sales.csv** text file in the D:\ETL folder, and then click **Cancel**.
12. Right-click anywhere on the Data Flow surface, click **Execute Task**, and then observe the task as it runs, noting how many rows are transferred. There should be no orphaned sales records.
13. On the **Debug** menu, click **Stop Debugging**.

14. Keep Visual Studio open for the next task.

▶ Task 2: Create a Data Flow Task

1. In the Solution Explorer pane, double-click **Extract Internet Sales Data.dtsx**.
2. View the **Control Flow** tab, and then in the SSIS Toolbox pane, in the **Favorites** section, double-click **Data Flow Task**.
3. Drag the new **Data Flow** task under the existing **Extract Customers** task.
4. Right-click the new **Data Flow** task, click **Rename**, and then change the name to **Extract Internet Sales**.
5. Click the **Extract Customers** Data Flow task, and then drag the arrow from the **Extract Customers** task to the **Extract Internet Sales** task.
6. Double-click the **Extract Internet Sales** task to view the Data Flow surface.

▶ Task 3: Add a Data Source to a Data Flow

1. In the SSIS Toolbox pane, in the **Favorites** section, double-click **Source Assistant**.
2. In the **Source Assistant - Add New Source** dialog box, in the **Select source type** list, select **SQL Server**. In the **Select connection manager** list, select **localhost.InternetSales**, and then click **OK**.
3. Drag the new **OLE DB Source** data source to the center of the Data Flow surface, right-click it, click **Rename**, and then change the name of the data source to **Internet Sales**.
4. Double-click the **Internet Sales** source, set the following configuration values, and then click **OK**:
   - On the **Connection Manager** page, ensure that the **localhost.InternetSales** OLE DB connection manager is selected, in the **Data access mode** list, click **SQL command**, click **Browse**, and then import the **InternetSales.sql** query file from the D:\Labfiles\Lab04\Starter\Ex3 folder.
   - On the **Columns** tab, ensure that every column from the query is selected, and that the output columns have the same names as the source columns.

▶ Task 4: Add a Derived Column transformation to a data flow

1. In the SSIS Toolbox pane, in the **Common** section, double-click **Derived Column**.
2. Drag the new **Derived Column** transformation below the existing **Internet Sales** data source, right-click it, click **Rename**, and then change the name of the transformation to **Calculate Sales Amount**.
3. On the Data Flow surface, click the **Internet Sales** source, and then drag the blue arrow from the **Internet Sales** data source to the **Calculate Sales Amount** transformation.
4. Double-click the **Calculate Sales Amount** transformation, in the **Derived Column Transformation Editor** dialog box, perform the following steps, and then click **OK**:
   - In the **Derived Column Name** column, type **SalesAmount**.
   - In the **Derived Column** column, ensure that **<add as new column>** is selected.
   - Expand the **Columns** folder, and then drag the **UnitPrice** column to the **Expression** box.

- Type *****, and then drag the **OrderQuantity** column to the Expression box so that the expression looks like the following example:

```
[UnitPrice] * [OrderQuantity]
```

- Ensure that the **Data Type** column contains the value **numeric [DT_NUMERIC]**, the **Precision** column contains the value **25**, and the Scale column contains the value **4**.

▶ Task 5: Add a Lookup Transformation to a Data Flow

1. In the SSIS Toolbox pane, in the **Common** section, double-click **Lookup**.
2. Drag the new **Lookup** transformation below the existing **Calculate Sales Amount** transformation, right-click it, click **Rename**, and then change the name of the transformation to **Lookup Product Details**.
3. On the Data Flow surface, click the **Calculate Sales Amount** transformation, and then drag the blue arrow from the **Calculate Sales Amount** transformation to the **Lookup Product Details** transformation.
4. Double-click the **Lookup Product Details** transformation, and in the **Lookup Transformation Editor** dialog box, perform the following steps and then click **OK**:
   - In the **General** column, under **Cache mode**, ensure that **Full cache** is selected, and under **Connection type**, ensure that **OLE DB connection manager** is selected. In the **Specify how to handle rows with no matching entries** list, click **Redirect rows to no match output**.
   - On the **Connection** tab, in the **OLE DB connection manager** list, select **localhost.Products** and click **OK**. Then select **Use results of an SQL query**, click **Browse**, and import the **Products.sql** query file from the D:\Labfiles\Lab04\Starter\Ex3 folder.
   - On the **Columns** tab, drag **ProductKey** from the **Available Input Columns** list to **ProductKey** in the **Available Lookup Columns** list.
   - In the **Available Lookup Columns** list, select the check box next to the **Name** column heading to select all columns, and then clear the check box for the **ProductKey** column.
5. In the SSIS Toolbox pane, in the **Other Destinations** section, double-click **Flat File Destination**.
6. Drag the new flat file transformation to the right of the existing **Lookup Product Details** transformation, right-click it, click **Rename**, and then change the name of the transformation to **Orphaned Sales**.
7. On the Data Flow surface, click the **Lookup Product Details** transformation, and then drag the blue arrow from the **Lookup Product Details** transformation to the **Orphaned Sales** destination.
8. In the **Input Output Selection** dialog box, in the **Output** list, click **Lookup No Match Output**, and then click OK.
9. Double-click the **Orphaned Sales** destination, and then in the **Flat File Destination Editor** dialog box, next to the **Flat File connection manager** drop-down list, click **New**.
10. In the **Flat File Format** dialog box, click **Delimited**, and then click OK.
11. In the **Flat File Connection Manager Editor** dialog box, change the text in the **Connection manager name** box to **Orphaned Internet Sales**.
12. On the **General** tab, set the **File name** value to **D:\ETL\Orphaned Internet Sales.csv**, select **Column names in the first data row**, and then click **OK**.

13. In the **Flat File Destination Editor** dialog box, ensure that **Overwrite data in the file** is selected, and then click the **Mappings** tab. Verify that all input columns are mapped to destination columns with the same name, and then click **OK**.

▶ Task 6: Add a Data Destination to a Data Flow

1. In the SSIS Toolbox pane, in the **Favorites** section, double-click **Destination Assistant**.
2. In the **Destination Assistant - Add New Destination** dialog box, in the **Select destination type** list, click **SQL Server**. In the **Select connection manager** list, click **localhost.Staging**, and then click **OK**.
3. Drag the new **OLE DB Destination** data destination below the **Lookup Product Details** transformation, right-click it, click **Rename**, and then change the name of the data destination to **Staging DB**.
4. On the Data Flow surface, click the **Lookup Product Details** transformation, and then drag the blue arrow from the **Lookup Product Details** transformation to the **Staging DB** destination. Note that the **Lookup Match Output** is automatically selected.
5. Double-click the **Staging DB** destination, set the following configuration values, and then click OK:
   - On the **Connection Manager** page, ensure that the **localhost.Staging** OLE DB connection manager is selected, and ensure that the **Table or view – fast load** data access mode is selected. In the **Name of the table or the view** drop-down list, click **[dbo].[InternetSales]**, and select **Keep nulls**.
   - On the **Mappings** tab, drag the following **ProductKey** columns from the list of available input columns to the **ProductBusinessKey** corresponding columns in the list of available destination columns:

Available Input Columns	Available Destination Columns
ProductKey	ProductBusinessKey
CustomerKey	CustomerBusinessKey
ProductSubcategoryKey	ProductSubcategoryBusinessKey
ProductCategoryKey	ProductCategoryBusinessKey

   - Verify that all other input columns are mapped to destination columns of the same name.

▶ Task 7: Test the Data Flow task

1. Right-click anywhere on the Data Flow surface, click **Execute Task**, and then observe the task as it runs, noting how many rows are transferred. There should be no orphaned sales records.
2. On the **Debug** menu, click **Stop Debugging**.
3. Close Visual Studio.

Results: After this exercise, you should have a package that contains a Data Flow task including Derived Column and Lookup transformations.

Module 5: Implementing Control Flow in an SSIS Package

Lab A: Implementing Control Flow in an SSIS Package

Exercise 1: Using Tasks and Precedence in a Control Flow

▶ Task 1: Prepare the Lab Environment

1. Ensure the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Labfiles\Lab05A\Starter folder, right-click **Setup.cmd** and click **Run as administrator**.
3. Click **Yes** when prompted to confirm you want to run the command file, and wait for the script to finish.

▶ Task 2: View a Control Flow

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab05A\Starter\Ex1 folder.
2. In Solution Explorer, in the **SSIS Packages** folder, double-click **Extract Reseller Data.dtsx**. Then view the control flow for the **Extract Reseller Data** package and note that it includes two **Send Mail** tasks – one that runs when either the **Extract Resellers** or **Extract Reseller Sales** tasks fail, and one that runs when the **Extract Reseller Sales** task succeeds.
3. Double-click the red dotted arrow connecting the **Extract Resellers** task to the **Send Failure Notification** task. In the **Precedence Constraint Editor**, in the **Multiple Constraints** section, note that **Logical OR. One constraint must evaluate to True** is selected so that the **Send Failure Notification** task runs if either of the data flow tasks connected should fail. Then click **Cancel**.
4. Double-click the **Send Failure Notification** task to view its settings. On the **Mail** tab, note that the task uses an SMTP connection manager named **Local SMTP Server** to send a high-priority email message with the subject **Data Extraction Notification** and the message "The reseller data extraction process failed" to Student@adventureworks.msft. Then click **Cancel**.
5. Double-click the **Send Success Notification** task to view its settings. On the **Mail** tab, note that the task uses an SMTP connection manager named **Local SMTP Server** to send a high-priority email message with the subject **Data Extraction Notification** and the message "The reseller data was successfully extracted" to Student@adventureworks.msft. Then click **Cancel**.
6. In the **Connection Managers** pane, double-click **Local SMTP Server** to view its settings, and note that it connects to the **localhost** SMTP server. Then click **Cancel**.
7. On the **Debug** menu, click **Start Debugging**, and observe the control flow as the task executes. Then, when the task has completed, on the **Debug** menu, click **Stop Debugging**.
8. View the contents of the C:\inetpub\mailroot\Drop folder and note the email messages that have been received by the local SMTP server.
9. Double-click the most recent message to open it with Outlook, read the email message, and then close Outlook.

▶ Task 3: Add Tasks to a Control Flow

1. In Visual Studio, in Solution Explorer, in the **SSIS Packages** folder, double-click **Extract Internet Sales Data.dtsx**. Then view the control flow for the **Extract Internet Sales Data** package.
2. In the SSIS Toolbox, in the **Common** section, double-click **Send Mail Task**. Then position the new **Send Mail** task below and to the right of the **Extract Internet Sales** task.
3. Double-click the **Send Mail** task on the control flow surface, to view its settings, and in the **Send Mail Task Editor** dialog box, on the **General** tab, set the **Name** property to **Send Success Notification**.
4. In the **Send Mail Task Editor** dialog box, on the **Mail** tab, in the **SmtpConnection** drop-down list, click **<New connection>**. Then in the **SMTP Connection Manager Editor** dialog box, enter the following settings and click **OK**:
   - **Name**: Local SMTP Server
   - **SMTP Server**: localhost
5. In the **Send Mail Task Editor** dialog box, on the **Mail** tab, enter the following settings, and then click **OK**:
   a. **From**: ETL@adventureworks.msft
   b. **To**: Student@adventureworks.msft
   c. **Subject**: Data Extraction Notification
   d. **MessageSourceType**: Direct Input
   e. **MessageSource**: The Internet Sales data was successfully extracted
   f. **Priority**: Normal
6. On the Control Flow surface, click the **Extract Internet Sales** task, and then drag the green arrow from the **Extract Internet Sales** task to the **Send Success Notification** task.
7. In the SSIS Toolbox, in the **Common** section, double-click **Send Mail Task**. Then position the new **Send Mail** task below and to the left of the **Extract Internet Sales** task.
8. Double-click the **Send Mail** task on the control flow surface, to view its settings, and in the **Send Mail Task Editor** dialog box, on the **General** tab, set the **Name** property to **Send Failure Notification**.
9. In the **Send Mail Task Editor** dialog box, on the **Mail** tab, enter the following settings, and then click **OK**:
   - **SmtpConnection**: Local SMTP Server
   - **From**: ETL@adventureworks.msft
   - **To**: Student@adventureworks.msft
   - **Subject**: Data Extraction Notification
   - **MessageSourceType**: Direct Input
   - **MessageSource**: The Internet Sales data extraction process failed
   - **Priority**: High
10. On the Control Flow surface, click the **Extract Customers** task, and then drag the green arrow from the **Extract Customers** task to the **Send Failure Notification** task. Then right-click the arrow and click **Failure**.

11. On the Control Flow surface, click the **Extract Internet Sales** task, and then drag the green arrow from the **Extract Internet Sales** task to the **Send Failure Notification** task. Then right-click the arrow and click **Failure**.
12. Double-click the red arrow connecting the **Extract Customers** task to the **Send Failure Notification** task. In the **Precedence Constraint Editor**, in the **Multiple Constraints** section, select **Logical OR. One constraint must evaluate to True**. Then click **OK**.

▶ Task 4: Test the Control Flow

1. In Visual Studio, on the Control Flow surface for the **Extract Internet Sales Data** package, click the **Extract Customers** task, and press **F4**. Then in the **Properties** pane, set the **ForceExecutionResult** property to **Failure**.
2. On the **Debug** menu, click **Start Debugging**, and observe the control flow as the task executes, noting that the **Extract Customer** task fails. Then, when the task has completed, on the **Debug** menu, click **Stop Debugging**.
3. View the contents of the C:\inetpub\mailroot\Drop folder and note the email messages that have been received by the local SMTP server.
4. Double-click the most recent message to open it with Outlook, and read the email message, noting that it contains a failure message. Then close the email message.
5. In Visual Studio, on the Control Flow surface for the **Extract Internet Sales Data** package, click the **Extract Customers** task. Then in the **Properties** pane, set the **ForceExecutionResult** property to **None**.
6. On the **Debug** menu, click **Start Debugging**, and observe the control flow as the task executes, noting that the **Extract Customer** task succeeds. Then, when the task has completed, on the **Debug** menu, click **Stop Debugging**.
7. View the contents of the C:\inetpub\mailroot\Drop folder and note the email messages that have been received by the local SMTP server.
8. Double-click the most recent message to open it with Outlook, and read the email message, noting that it contains a success message. Then close the email message.
9. Close Visual Studio, saving your changes if you are prompted.

Results: After this exercise, you should have a control flow that sends an email message if the **Extract Internet Sales** task succeeds, or sends an email message if either the **Extract Customers** or **Extract Internet Sales** tasks fail.

Exercise 2: Using Variables and Parameters

▶ Task 1: View a Control Flow

1. View the contents of the D:\Accounts folder and note the files it contains. In this exercise, you will modify an existing package to create a dynamic reference to one of these files.
2. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab05A\Starter\Ex2 folder.
3. In Solution Explorer, in the **SSIS Packages** folder, double-click **Extract Payment Data.dtsx**. Then view the control flow for the **Extract Payment Data** package, and note that it contains a single data flow task named **Extract Payments**.

4. Double-click the **Extract Payments** task to view it in the Data Flow tab, and note that it contains a flat file source named **Payments File**, and an OLE DB destination named **Staging DB.**
5. Double-click the **Payments File** source and note that it uses a connection manager named **Payments File**. Then click **Cancel**.
6. In the **Connection Managers** pane, double-click **Payments File**, and note that it references the Payments.csv file in the D:\Labfiles\Lab05A\Starter\Ex2 folder. Then click **Cancel**.
7. On the **Debug** menu, click **Start Debugging** and observe the data flow while the package runs. When the package has completed, on the **Debug** menu, click **Stop Debugging**.
8. On the **Execution Results** tab, find the following line in the package execution log:

```
[Payments File [2]] Information: The processing of the file
"D:\Labfiles\Lab05A\Starter\Ex2\Payments.csv" has started
```

9. Click the **Data Flow** tab to return to the data flow design surface.

▶ Task 2: Create a Variable

1. In Visual Studio, with **Extract Payments Data.dtsx** package open, on the **View** menu, click **Other Windows**, and then click **Variables**.
2. In the **Variables** pane, click the **Add Variable** button, and create a variable with the following properties:
   - **Name**: fName
   - **Scope**: Extract Payments Data
   - **Data type**: String
   - **Value**: Payments - US.csv

Note that the value includes a space on either side of the "-" character.

▶ Task 3: Create a Parameter

1. In Visual Studio, in Solution Explorer, double-click **Project.params**.
2. In the **Project.params [Design]** window, click the **Add Parameter** button, and add a parameter with the following properties:
   - **Name**: AccountsFolderPath
   - **Data** type: String
   - **Value**: D:\Accounts\
   - **Sensitive**: False
   - **Required**: True
   - **Description**: Path to accounts files

Note: Be sure to include the trailing "\" in the **Value** property.

3. On the **File** menu, click **Save All**, and then close the **Project.params [Design]** window.

▶ Task 4: Use a Variable and a Parameter in an Expression

1. On the **Data Flow** design surface for the **Extract Payments Data.dtsx** package, in the **Connection Managers** pane, click **Payments File**. Then press **F4** to view the **Properties** pane.
2. In the **Properties** pane, in the **Expressions** property box, click the ellipsis (**...**) button. Then in the **Property Expressions Editor** dialog box, in the **Property** box, select **ConnectionString** and in the **Expression** box, click the ellipsis (**...**) button.
3. In the **Expression Builder** dialog box, expand the **Variables and Parameters** folder, and drag the **$Project::AccountsFolderPath** parameter to the **Expression** box.
4. In the **Expression** box, type a plus (**+**) symbol after the **$Project::AccountsFolderPath** parameter.
5. Drag the **User::fName** variable to the **Expression** box to create the following expression:

```
@[$Project::AccountsFolderPath]+ @[User::fName]
```

6. In the **Expression Builder** dialog box, click **Evaluate Expression** and verify that the expression produces the result **D:\Accounts\Payments - US.csv**. Then click **OK** to close the **Expression Builder** dialog box, and in the **Property Expressions Editor** dialog box, click **OK**.
7. On the **Debug** menu, click **Start Debugging** and observe the data flow while the package runs. When the package has completed, on the **Debug** menu, click **Stop Debugging**.
8. On the **Execution Results** tab, find the following line in the package execution log, noting that the default values for the **fName** variable and **AccountsFolderPath** parameter were used:

```
[Payments File [2]] Information: The processing of the file "D:\Accounts\Payments – US.csv" has started
```

9. Close the Visual Studio, saving the changes if you are prompted.

Results: After this exercise, you should have a package that loads data from a text file based on a parameter that specifies the folder path where the file is stored, and a variable that specifies the file name.

Exercise 3: Using Containers

▶ Task 1: Add a Sequence Container to a Control Flow

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab05A\Starter\Ex3 folder.
2. In Solution Explorer, in the **SSIS Packages** folder, double-click **Extract Internet Sales Data.dtsx**. Then view the control flow for the **Extract Internet Sales Data** package. You created the control flow for this package in Exercise 1.
3. Right-click the red dotted arrow connecting the **Extract Customers** task to the **Send Failure Notification** task, and click **Delete**. Repeat this step to delete the red dotted line connecting the **Extract Internet Sales** task to the **Send Failure Notification** task, and the green arrow connecting the **Extract Internet Sales** task to the **Send Success** notification task.
4. Drag a **Sequence Container** from the **Containers** section of the SSIS Toolbox to the control flow surface. Right-click the new sequence container, click **Rename**, and change the container name to **Extract Customer Sales Data**.
5. Click the **Extract Customers** task, hold the **Ctrl** key, and click the **Extract Internet Sales** task, then drag both tasks into the **Extract Customer Sales Data** sequence container.

6. Click the **Extract Customer Sales Data** sequence container, and then drag the green arrow from the **Extract Customer Sales Data** sequence container to the **Send Success Notification** task.
7. Click the **Extract Customer Sales Data** sequence container, and then drag the green arrow from the **Extract Customer Sales Data** sequence container to the **Send Failure Notification** task. Right-click the green arrow connecting the **Extract Customer Sales Data** sequence container to the **Send Failure Notification** task, and click **Failure**.
8. On the **Debug** menu, click **Start Debugging**, and observe the control flow as the package executes. Then, when package execution is complete, on the **Debug** menu, click **Stop Debugging**.

▶ Task 2: Add a Foreach Loop Container to a Control Flow

1. In Visual Studio, in Solution Explorer, in the **SSIS Packages** folder, double-click **Extract Payment Data.dtsx**. Then view the control flow for the **Extract Payment Data** package, and note that it contains a single data flow task named **Extract Payments**. This is the same data flow task you updated in the previous exercise.
2. In the SSIS Toolbox, in the **Containers** section, double-click **Foreach Loop Container**. Then on the control flow surface, click the **Extract Payments** task and drag it into the **Foreach Loop Container**.
3. Double-click the title area at the top of the **Foreach Loop Container** to view the **Foreach Loop Editor** dialog box.
4. In the **Foreach Loop Editor** dialog box, on the **Collection** tab, in the Enumerator list, select the **Foreach File Enumerator**. In the **Expressions** box, click the ellipsis (**…**) button.
5. In the **Property Expressions Editor** dialog box, in the **Property** list, select **Directory** and in the **Expression** box click the ellipsis (**…**) button.
6. In the **Expression Builder** dialog box, expand the **Variables and Parameters** folder and drag the $**Project::AccountsFolderPath** parameter to the **Expression** box. Click **OK** to close the Expression Builder, and click **OK** again to close the Property Expression Editor.
7. In the **Foreach Loop Editor** dialog box, on the **Collection** tab, in the **Retrieve file name** section, select **Name and extension**.
8. In the **Foreach Loop Editor** dialog box, on the **Variable Mappings** tab, in the **Variable** list, select **User::fName** and in the **Index** column ensure that **0** is specified. Then click **OK**.
9. On the **Debug** menu, click **Start Debugging** and observe the data flow while the package runs. When the package has completed, on the **Debug** menu, click **Stop Debugging**.
10. On the **Execution Results** tab, scroll through the package execution log, noting that the data flow was executed once for each file in the D:\Accounts folder. The following files should have been processed:
    - Payments – AU.csv
    - Payments – CA.csv
    - Payments – DE.csv
    - Payments – FR.csv
    - Payments – GB.csv
    - Payments – US.csv
11. Close Visual Studio, saving your changes if prompted.

Results: After this exercise, you should have one package that encapsulates two data flow tasks in a sequence container, and another that uses a Foreach Loop to iterate through the files in a folder specified in a parameter and uses a data flow task to load their contents into a database.

Lab B: Using Transactions and Checkpoints

Exercise 1: Using Transactions

▶ Task 1: Prepare the Lab Environment

1. Ensure the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Labfiles\Lab05B\Starter folder, right-click **Setup.cmd** and click **Run as administrator**.
3. Click **Yes** when prompted to confirm you want to run the command file, and wait for the script to finish.

▶ Task 2: View the Data in the Database

1. Start SQL Server Management Studio, and when prompted, connect to the **localhost** database engine using Windows authentication.
2. In Object Explorer, expand **Databases**, **Staging**, and **Tables**.
3. Right-click **dbo.Customers** and click **Select Top 1000 Rows**. Note that the table is empty.
4. Right-click **dbo.InternetSales** and click **Select Top 1000 Rows**. Note that the table is also empty.
5. Minimize SQL Server Management Studio.

▶ Task 3: Run a Package to Extract Data

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab05B\Starter\Ex1 folder.
2. In Solution Explorer, in the **SSIS Packages** folder, double-click **Extract Internet Sales Data.dtsx**. Then view the control flow for the **Extract Internet Sales Data** package.
3. On the **Debug** menu, click **Start Debugging**, and observe the control flow as the package is executed, noting that the **Extract Customers** task succeeds, but the **Extract Internet Sales** task fails. Then, when package execution has completed, on the **Debug** menu, click **Stop Debugging**.
4. Maximize SQL Server Management Studio and re-execute the queries you created earlier to view the top 1,000 rows in the **dbo.Customers** and **dbo.InternetSales** tables. Verify that the **dbo.InternetSales** table is still empty but the **dbo.Customers** table now contains customer records.
5. In SQL Server Management Studio, click **New Query**. Then in the new query window, enter the following Transact-SQL and click **Execute**:

```
TRUNCATE TABLE Staging.dbo.Customers;
```

6. Close the query tab containing the TRUNCATE TABLE statement without saving it, and minimize SQL Server Management Studio.

▶ Task 4: Implement a Transaction

1. In SQL Server Data Tools, on the control flow surface for the **Extract Internet Sales Data.dtsx** package, click the **Extract Customer Sales Data** sequence container and press **F4** to view the **Properties** pane.
2. In the **Properties** pane, set the **TransactionOption** property of the **Extract Customer Sales Data** sequence container to **Required**.
3. Click the **Extract Customers** task, and in the **Properties** pane, ensure that the **TransactionOption** property value is set to **Supported**, and set the **FailParentOnFailure** property to **True**.

4. Repeat the previous step for the **Extract Internet Sales** task.

▶ Task 5: Observe Transaction Behavior

1. In Visual Studio, on the **Debug** menu, click **Start Debugging**, and observe the control flow as the package is executed, noting that once again the **Extract Customers** task succeeds, but the **Extract Internet Sales** task fails. Then, when package execution has completed, on the **Debug** menu, click **Stop Debugging**.
2. Maximize SQL Server Management Studio and re-execute the queries you created earlier to view the top 1,000 rows in the **dbo.Customers** and **dbo.InternetSales** tables, and verify that both tables are empty. Then minimize SQL Server Management Studio.
3. In Visual Studio, on the control flow surface for the **Extract Internet Sales Data.dtsx** package, double-click the **Extract Internet Sales** task to view it in the Data Flow tab.
4. Double-click the **Calculate Sales Amount** transformation and modify the **Expression** value to remove the text "/ (OrderQuantity % OrderQuantity)". When the expression matches the following code, click **OK**:

```
UnitPrice * OrderQuantity
```

5. Click the **Control Flow** tab, and on the **Debug** menu, click **Start Debugging**. Observe the control flow as the package is executed, noting that both the **Extract Customers** and **Extract Internet Sales** tasks succeed. Then, when package execution has completed, on the **Debug** menu, click **Stop Debugging**.
6. Maximize SQL Server Management Studio and re-execute the queries you created earlier to view the top 1,000 rows in the **dbo.Customers** and **dbo.InternetSales** tables, and verify that both tables now contain data. Minimize SQL Server Management Studio as you will use it again in the next exercise.
7. Close Visual Studio, saving changes if prompted.

Results: After this exercise, you should have a package that uses a transaction to ensure that all data flow tasks succeed or fail as an atomic unit of work.

Exercise 2: Using Checkpoints

▶ Task 1: View the Data in the Database

1. Maximize SQL Server Management Studio, and in Object Explorer, ensure that **Databases**, **Staging**, and **Tables** are expanded for the **localhost** database engine instance.
2. Right-click **dbo.Resellers** and click **Select Top 1000 Rows**. Note that the table is empty.
3. Right-click **dbo.ResellerSales** and click **Select Top 1000 Rows**. Note that this table is also empty.
4. Minimize SQL Server Management Studio.

▶ Task 2: Run a Package to Extract Data

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab05B\Starter\Ex2 folder.
2. In Solution Explorer, in the **SSIS Packages** folder, double-click **Extract Reseller Data.dtsx**. Then view the control flow for the **Extract Internet Sales Data** package.

3. On the **Debug** menu, click **Start Debugging**, and observe the control flow as the package is executed, noting that the **Extract Resellers** task succeeds, but the **Extract Reseller Sales** task fails. Then, when package execution has completed, on the **Debug** menu, click **Stop Debugging**.
4. Maximize SQL Server Management Studio and re-execute the queries you created earlier to view the top 1,000 rows in the **dbo.Resellers** and **dbo.ResellerSales** tables. Verify that the **dbo.ResellerSales** table is still empty but the **dbo.Resellers** table now contains records.
5. In SQL Server Management Studio, click **New Query**. In the new query window, enter the following Transact-SQL code and click **Execute**:

```
TRUNCATE TABLE Staging.dbo.Resellers;
```

6. Close the query tab containing the TRUNCATE TABLE statement without saving it, and minimize SQL Server Management Studio.

▶ Task 3: Implement Checkpoints

1. In Visual Studio, click any empty area on the control flow surface for the **Extract Reseller Data.dtsx** package, and press **F4** to view the **Properties** pane.
2. In the **Properties** pane, set the following properties of the **Extract Reseller Data** package:
   - **CheckpointFileName**: D:\ETL\CheckPoint.chk
   - **CheckpointUsage**: IfExists
   - **SaveCheckpoints**: True
3. Click the **Extract Resellers** task, and in the **Properties** pane, set the **FailPackageOnFailure** property to **True**.
4. Repeat the previous step for the **Extract Reseller Sales** task.

▶ Task 4: Observe Checkpoint Behavior

1. View the contents of the D:\ETL folder and verify that no file named CheckPoint.chk exists.
2. In Visual Studio, on the **Debug** menu, click **Start Debugging**. Observe the control flow as the **Extract Reseller Sales Data.dtsx** package is executed, noting that once again the **Extract Resellers** task succeeds, but the **Extract Reseller Sales** task fails. Then, when package execution has completed, on the **Debug** menu, click Stop **Debugging**.
3. View the contents of the D:\ETL folder and verify that a file named CheckPoint.chk has been created.
4. Maximize SQL Server Management Studio and re-execute the queries you created earlier to view the top 1,000 rows in the **dbo.Resellers** and **dbo.ResellerSales** tables, and verify that the **dbo.ResellerSales** table is still empty, but the **dbo.Resellers** table now contains reseller records.
5. In Visual Studio, on the control flow surface for the **Extract Reseller Data.dtsx** package, double-click the **Extract Reseller Sales** task to view it in the Data Flow tab.
6. Double-click the **Calculate Sales Amount** transformation and modify the **Expression** value to remove the text "/ (OrderQuantity % OrderQuantity)". When the expression matches the following code, click **OK**:

```
UnitPrice * OrderQuantity
```

7. Click the **Control Flow** tab, and on the **Debug** menu, click **Start Debugging**. Observe the control flow as the package is executed, noting that the **Extract Resellers** task is not re-executed, and package execution starts with the **Extract Reseller Sales** task, which failed on the last attempt. When package execution has completed, on the **Debug** menu, click **Stop Debugging**.

8. View the contents of the D:\ETL folder and verify that the CheckPoint.chk file has been deleted.
9. Maximize SQL Server Management Studio and re-execute the queries you created earlier to view the top 1,000 rows in the **dbo.Resellers** and **dbo.ResellerSales** tables. Verify that both now contain data, then close SQL Server Management Studio.
10. Close Visual Studio, saving changes if prompted.

Results: After this exercise, you should have a package that uses checkpoints to enable execution to be restarted at the point of failure on the previous execution.

Module 6: Debugging and Troubleshooting SSIS Packages

Lab: Debugging and Troubleshooting an SSIS Package

Exercise 1: Debugging an SSIS Package

▶ Task 1: Prepare the Lab Environment

1. Ensure the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Labfiles\Lab06\Starter folder, right-click **Setup.cmd** and click **Run as administrator**.
3. Click **Yes** when prompted to confirm you want to run the command file, and wait for the script to finish.

▶ Task 2: Run an SSIS Package

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab06\Starter\Ex1 folder.
2. In Solution Explorer, double-click the **Extract Payment Data.dtsx** SSIS package to open it in the designer. Note that this package includes a control flow that iterates through files in a folder, and loads payments data from each file into a staging database.
3. On the **Debug** menu, click **Start Debugging** and observe the package as it executes, noting that it fails. When execution has completed, on the **Debug** menu, click **Stop Debugging**.

▶ Task 3: Add a Breakpoint

1. On the control flow surface for the **Extract Payment Data.dtsx** package, right-click **Foreach Loop Container** and click **Edit Breakpoints**.
2. In the **Set Breakpoints – Foreach Loop Container** dialog box, select **Enabled** for the **Break at the beginning of every iteration of the loop** break condition. Then click **OK**.

▶ Task 4: Add a Data Viewer

1. On the control flow surface for the **Extract Payment Data.dtsx** package, double-click the **Extract Payments** data flow task to view the data flow design surface.
2. Double-click the data flow path between the **Payments File** source and the **Staging DB** destination to open its editor.
3. In the **Data Flow Path Editor** dialog box, on the **Data Viewer** tab, select **Enable data viewer** and note that all available columns are included in the **Displayed columns** list. Then click **OK**.

▶ Task 5: View Breakpoints

1. Click the **Control Flow** tab to view the control flow design surface for the **Extract Payment Data.dtsx** package.
2. On the **Debug** menu, click **Windows**, and then click **Breakpoints** to open the **Breakpoints** pane.
3. View the breakpoints, noting that they include the one you added to the Foreach Loop container, and the data viewer you defined in the data flow.

▶ Task 6: Observe Variables while Debugging

1. On the **Debug** menu, click **Start Debugging** and note that execution stops at the first breakpoint.
2. On the **Debug** menu, click **Windows**, and then click **Locals** to view the **Locals** pane.
3. In the **Locals** pane, expand **Variables**, find the **User::fName** variable, and note its value (which should be **Payments – AU.csv**). Then right-click the **User::fName** variable and click **Add Watch**.
4. Note that the **User::fName** variable has been added to the **Watch 1** pane, which is now shown. Then, click the **Locals** tab to view the **Locals** pane again, and add a watch for the **$Project::AccountsFolderPath** parameter.
5. Ensure that the **Watch 1** pane is visible and that you can see the values for the **User::fName** variable and the **$Project::AccountsFolderPath** parameter, and then on the **Debug** menu, click **Continue**.
6. Note that execution stops at the next breakpoint, which is the data viewer you added to the data flow. View the data in the data viewer pane, noting that this represents the contents of the Payments – AU.csv file. Then drag the data viewer pane and position it so you can see all the following user interface elements:
   - The **Foreach Loop Container** on the control flow surface.
   - The variables in the **Watch 1** pane.
   - The data viewer pane.
7. In the data viewer pane, click the **Continue** button (a green arrow). Note that execution continues until the next breakpoint, and that the **Extract Payments** task completed successfully for the first loop iteration. In the **Watch 1** window, note that the value of the **User::fName** variable has changed to **Payments – CA.csv**, but that the contents of the data viewer pane still reflect the Payments – AU.csv file because the data flow for the current loop iteration has not yet started.
8. On the **Debug** menu, click **Continue**, and note that execution continues to the next breakpoint. The data viewer pane now shows the contents of the Payments – CA.csv file.
9. In the data viewer pane, click the **Continue** button. Note that execution continues until the next breakpoint, and that the **Extract Payments** task completed successfully for the second loop iteration. In the **Watch 1** window, note that the value of the **User::fName** variable has changed to **Payments – DE.csv**, indicating that it is the next file to be processed by the **Extract Payments** task.
10. On the **Debug** menu, click **Continue**, and note that execution continues to the next breakpoint. The data viewer pane now shows the contents of the Payments – DE.csv file.
11. In the data viewer pane, click the **Continue** button. Note that execution continues until the next breakpoint, and that the **Extract Payments** task completed successfully for the Payments – DE.csv file. In the **Watch 1** window, note that the value of the **User::fName** variable has changed to **Payments – EU.csv**, indicating that it is the next file to be processed by the **Extract Payments** task.
12. On the **Debug** menu, click **Continue**, and note that execution continues to the next breakpoint. The data viewer pane now shows the contents of the Payments – EU.csv file (which contains a mixture of country codes including DE, FR, and GB).
13. In the data viewer pane, click the **Continue** button. Note that execution continues until the next breakpoint, and that the **Extract Payments** task failed for the Payments – EU.csv file.
14. In the data viewer pane, note that the contents of the Payments – EU.csv file are still shown. Then click **Copy Data** to copy the data to the clipboard.
15. On the **Debug** menu, click **Stop Debugging**.
16. Close Visual Studio.

▶ Task 7: View Data Copied from a Data Viewer

1. Start Excel, and create a new blank workbook.
2. Ensure cell A1 is selected, and on the **Home** tab of the ribbon, click **Paste**.
3. Widen the columns as necessary to see the data from the Payments – EU.csv file that the SSIS package failed to load to the staging database. Note that it contains the following issues:
   - Payment 4074 has an invalid payment date.
   - Payment 4102 has an invalid payment amount.
   - Payment 4124 has an invalid payment date.
4. Close Excel without saving the workbook.

Results: After this exercise, you should have observed the variable values and data flows for each iteration of the loop in the Extract Payment Data.dtsx package. You should also have identified the file that caused the data flow to fail and examined its contents to find the data errors that triggered the failure.

Exercise 2: Logging SSIS Package Execution

▶ Task 1: Configure SSIS Logs

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab06\Starter\Ex2 folder.
2. In Solution Explorer, double-click the **Extract Payment Data.dtsx** SSIS package to open it in the designer.
3. On the **SSIS** menu, click **Logging**. If an error message is displayed, click **OK**.
4. In the **Configure SSIS Logs: Extract Payment Data** dialog box, in the **Containers** tree, select **Extract Payment Data**.
5. On the **Providers and Logs** tab, in the **Provider type** drop-down list, select **SSIS log provider for Windows Event Log**. Then click **Add**.
6. In the **Select the logs to use for the container** list, select **SSIS log provider for Windows Event Log**.
7. On the **Details** tab, select the **OnError** and **OnTaskFailed** events. Then click **OK**.

▶ Task 2: View Logged Events

1. On the **Debug** menu, click **Start Debugging**, and observe the execution of the package, noting that it fails. When execution is complete, on the **Debug** menu, click **Stop Debugging**.
2. On the **SSIS** menu, click **Log Events** and view the events that were logged during the package execution (if there are none, re-run the package). Then close the **Log Events** tab and close Visual Studio.
3. On the Start screen, type **Event** and start the **Event Viewer** app. Then in Event Viewer, expand **Windows Logs** and click **Application**.
4. In Event Viewer, click each event with the source **SQLISPackage120** and view the event information in the pane at the bottom of the Event Viewer window. Then close Event Viewer.

Results: After this exercise, you should have a package that logs event details to the Windows Event Log.

Exercise 3: Implementing an Event Handler

▶ Task 1: Create an Event Handler

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab06\Starter\Ex3 folder.
2. In Solution Explorer, double-click the **Extract Payment Data.dtsx** SSIS package to open it in the designer.
3. Click the **Event Handlers** tab of the **Extract Payment Data.dtsx** package.
4. In the **Executable** list, ensure **Extract Payment Data** is selected, and in the **Event handler** list, ensure **OnError** is selected. Then click the **Click here to create an 'OnError' event handler for executable 'Extract Payment Data'** link.

▶ Task 2: Add a File System Task

1. In the SSIS Toolbox, double-click **File System Task** to add a file system task to the event handler design surface for the **OnError** event handler. Then on the design surface, right-click **File System Task**, click **Rename**, and change the name to **Copy Failed File**.
2. Double-click the **Copy Failed File** data flow task to view its task editor, and on the **General** tab, ensure the **Copy File** operation is selected.
3. In the **File System Task Editor** dialog box, on the **General** tab, in the **SourceConnection** drop-down list, select **Payments File**.
4. In the **File System Task Editor** dialog box, on the **General** tab, in the **DestinationConnection** drop-down list, select **<New connection>**. Then in the **File Connection Manager Editor** dialog box, in the **Usage type** drop-down list, select **Create file**. In the **File** box, type **D:\ETL\FailedPayments.csv**, and click **OK** to close the **File Connection Manager Editor** dialog box.
5. In the **File System Task Editor** dialog box, on the **General** tab, in the **OverwriteDestination** drop-down list, select **True**. Then click **OK** to close the **File System Task Editor** dialog box.
6. In the **Connection Managers** area at the bottom of the design surface, click **FailedPayments.csv** and press **F4**, and in the **Properties** pane, in the **Expressions** property value, click the ellipsis (**...**) button.
7. In the **Property Expressions Editor** dialog box, in the **Property** drop-down list, select **ConnectionString**, and in the **Expression** box, click the ellipsis (**...**) button to open the **Expression Builder**.
8. In the **Expression Builder** dialog box, in the **Expression** text area, enter the following expression. Then click **OK**:

```
"D:\\ETL\\" + @[System::ExecutionInstanceGUID] + @[User::fName]
```

9. In the **Property Expressions Editor** dialog box, click **OK**.

▶ Task 3: Add a Send Mail Task

1. In the SSIS Toolbox, double-click **Send Mail Task** to add a send mail task to the event handler design surface. Then on the design surface, right-click **Send Mail Task**, click **Rename**, and change the name to **Send Notification**.
2. Move **Send Notification** below **Copy Failed File**, and then click **Copy Failed File,** and drag the green precedence connection from **Copy Failed File** to **Send Notification**. Right-click the precedence connection and click **Completion**.
3. Double-click **Send Notification** to open its task editor.

4. In the **Send Mail Task Editor** dialog box, on the **Mail** tab, set the following properties:
   - **SmtpConnection**: Local SMTP Server
   - **From**: etl@adventureworks.msft
   - **To**: student@adventureworks.msft
   - **Subject**: An error occurred
   - **Priority**: High
5. In the **Send Mail Task Editor** dialog box, on the **Expressions** tab, click the ellipsis (**...**) button. Then in the **Property Expressions Editor** dialog box, in the **Property** drop-down list, select **MessageSource,** and in the **Expression** box, click the ellipsis (**...**) button to open the **Expression Builder**.
6. In the **Expression Builder** dialog box, in the **Expression** text area, enter the following expression. Then click **OK**:

```
@[User::fName] + " failed to load. " + @[System::ErrorDescription]
```

7. In the **Property Expressions Editor** dialog box, click **OK**, and then in the **Send Mail Task Editor**, click **OK**.

▶ Task 4: Test the Event Handler

1. Click the **Control Flow** tab.
2. On the **Debug** menu, click **Start Debugging**, and observe the execution of the package, noting that the package fails. When execution is complete, click the **Event Handlers** tab to verify that the event handler has been executed, and then on the **Debug** menu, click **Stop Debugging** and close Visual Studio. Save your changes if you are prompted.
3. View the contents of the D:\ETL folder and note that a file with a name similar to {1234ABCD-1234-ABCD-1234-ABCD1234}Payments - EU.csv has been created. Open this file in Excel, view the contents, and then close Excel without saving any changes.
4. View the contents of the C:\inetpub\mailroot\Drop folder, and note that several email messages were sent at the same time. Double-click the most recent file to open it in Outlook. Read the email message, and then close Outlook.

Results: After this exercise, you should have a package that includes an event handler for the **OnError** event. The event handler should create a copy of files that contain invalid data and send an email message.

Exercise 4: Handling Errors in a Data Flow

▶ Task 1: Redirect Data Flow Errors

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab06\Starter\Ex4 folder.
2. In Solution Explorer, double-click the **Extract Payment Data.dtsx** SSIS package to open it in the designer.
3. On the control flow surface, double-click the **Extract Payments** task to view its data flow.
4. Double-click the **Staging DB** destination to view its editor, and on the **Error Output** tab, in the **Error** column for **OLE DB Destination Input**, select **Redirect row**. Then click **OK**.

5. In the SSIS Toolbox, in the **Other Destinations** section, double-click **Flat File Destination**. Then on the design surface, right-click **Flat File Destination**, click **Rename**, and change the name to **Invalid Rows**.
6. Drag **Invalid Rows** to the right of **Staging DB**, and then connect the red data flow path from **Staging DB** to **Invalid Rows**. In the **Configure Error Output** dialog box, ensure that **Redirect row** is selected in the **Error** column, and click **OK**.
7. Double-click **Invalid Rows**, and in the **Flat File Destination Editor** dialog box, next to the **Flat File connection** manager drop-down list, click **New**.
8. In the **Flat File Format** dialog box, ensure **Delimited** is selected and click **OK**. Then in the **Flat File Connection Manager Editor** dialog box, change the following properties and click **OK**:
   - **Connection manager name**: Invalid Payment Records
   - **File name**: D:\ETL\InvalidPaymentsLog.csv
9. In the **Flat File Destination Editor** dialog box, on the **Connection Manager** tab, clear the **Overwrite data** in the file checkbox. Then on the **Mappings** tab, note that the input columns include the columns from the payments file and two additional columns for the error code and error column, and then click **OK**.

▶ Task 2: View Invalid Data Flow Rows

1. Click the **Control Flow** tab.
2. On the **Debug** menu, click **Start Debugging**, and observe the execution of the package, noting that it succeeds. When execution is complete, on the **Debug** menu, click **Stop Debugging** and close Visual Studio.
3. View the contents of the D:\ETL folder and note that a file named InvalidPaymentsLog.csv has been created.
4. Start Excel and open InvalidPaymentsLog.csv to view its contents. Then close Excel without saving any changes.

Results: After this exercise, you should have a package that includes a data flow where rows containing errors are redirected to a text file.

Module 7: Implementing a Data Extraction Solution

Lab: Extracting Modified Data

Exercise 1: Using a Datetime Column to Incrementally Extract Data

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Labfiles\Lab07\Starter folder, right-click **Setup.cmd** and click **Run as administrator**.
3. Click **Yes** when prompted to confirm you want to run the command file, and wait for the script to finish.

▶ Task 2: View Extraction Data

1. Start SQL Server Management Studio and connect to the **MIA-SQL** instance of the database engine by using Windows authentication.
2. In Object Explorer, expand **Databases**, **Staging**, and **Tables**.
3. Right-click **dbo.ExtractLog** and click **Select Top 1000 Rows**.
4. View the data in the **ExtractLog** table, noting the values in the **LastExtract** column, which indicate the date and time of the last extract operations for the **InternetSales** and **ResellerSales** databases. This is initially set to January 1st 1900.
5. In Object Explorer, under **Databases**, expand **InternetSales** and **Tables**.
6. Right-click **dbo.SalesOrderHeader** and click **Select Top 1000 Rows**. Then note that the **LastModified** column indicates the date and time that the sales record was last modified.
7. Minimize SQL Server Management Studio.

▶ Task 3: Examine an Incremental Data Extraction

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab07\Starter\Ex1 folder.
2. In Solution Explorer, under **SSIS Packages**, double-click **Extract Reseller Data.dtsx**.
3. On the **SSIS** menu, click **Variables** and then in the **Variables** pane, note that the following variables have been defined with a data type of **DateTime**:
   - CurrentTime
   - ResellerSalesLastExtract
4. On the **Control Flow** tab of the design surface, double-click the **Get Current Time** task, and note that it uses the GETDATE() function to assign the current data and time to the **CurrentTime** variable. Then click **Cancel** to close the **Expression Builder** dialog box.
5. Double-click the **Get Last Extract Time** task to open the **Execute SQL Task Editor** dialog box, and note the following configuration settings. Then click **Cancel**:
   - On the **General** tab, the task is configured to return a single row from the **Staging** database by executing the following Transact-SQL statement:

```
SELECT MAX(LastExtract) LastExtract
FROM ExtractLog
WHERE DataSource = 'ResellerSales'
```

5. Double-click the **Get Current Time** task, and in the **Expression Builder** dialog box, in the **Expression box**, specify the following expression and then click **OK**:

```
@[User::CurrentTime] = GETDATE()
```

6. In the SSIS Toolbox, drag an **Execute SQL** task to the control flow surface and drop it inside the **Extract Customer Sales Data** sequence container, immediately below the **Get Current Time** task. Then right-click the Execute SQL task, click **Rename**, and change the name to **Get Last Extract Time**.
7. Double-click the **Get Last Extract Time** task and in the **Execute SQL Task Editor** dialog box, configure the following settings. Then click **OK**:
   - On the **General** tab, in the **Connection** drop-down list, select **localhost.Staging**.
   - On the **General** tab, in the **SQLStatement** box, click the ellipsis (**...**) button and then in the **Enter SQL Query** dialog box, enter the following Transact-SQL query and click **OK**:

```
SELECT MAX(LastExtract) LastExtract
FROM ExtractLog
WHERE DataSource = 'InternetSales'
```

   - On the **General** tab, in the **ResultSet** drop-down list, select **Single row**.
   - On the **Result Set** tab, click **Add**, and then in the **Result Name** column, change **NewResultName** to **LastExtract**, and in the **Variable Name** drop-down list, select **User::InternetSalesLastExtract**.
8. On the control flow surface, click the **Get Current Time** task and drag its green precedence constraint to the **Get Last Extract Time** task. Then click the **Get Last Extract Time** task and drag its green precedence constraint to the **Extract Customers** task.

▶ Task 5: Modify a Data Source to Filter Data

1. On the control flow surface, double-click the **Extract Internet Sales** task to display its data flow.
2. On the data flow surface, double-click the **Internet Sales** source, and in the **OLE DB Source Editor** dialog box, review the existing SQL command text used to extract sales data. Then add the following parameterized WHERE clause to the SQL Command text:

```
WHERE LastModified > ?
AND LastModified <= ?
```

3. Click the **Parameters** button, and in the **Set Query Parameters** dialog box, specify the following parameter mappings with a **Param Direction** of **Input**, and click **OK**:
   - **Parameter0**: User::InternetSalesLastExtract
   - **Parameter1**: User:CurrentTime
4. In the **OLE DB Source Editor** dialog box, click **OK**.

▶ Task 6: Add a Task to Update the Extraction Log

1. Click the **Control Flow** tab, and then in the SSIS Toolbox, drag an **Execute SQL Task** to the **Extract Customer Sales Data** sequence under the **Extract Internet Sales** task on the control flow surface.
2. Right-click **Execute SQL Task** and click **Rename**. Then change the name to **Update Last Extract Time**.
3. Double-click **Update Last Extract Time** and configure the following settings. Then click **OK**:
   - On the **General** tab, in the **Connection** drop-down list, select **localhost.Staging**.

- On the **General** tab, in the **SQLStatement** box, click the ellipsis (**...**) button and then in the **Enter SQL Query** dialog box, enter the following Transact-SQL query and click **OK**:

```
UPDATE ExtractLog
SET LastExtract = ?
WHERE DataSource = 'InternetSales'
```

- On the **Parameter Mapping** tab, click **Add** and create the following parameter mapping:
  - **Variable Name**: User::CurrentTime
  - **Direction**: Input
  - **Data Type**: DATE
  - **Parameter Name**: 0
  - **Parameter Size**: -1

4. On the control flow surface, click the **Extract Internet Sales** task and then drag its green precedence constraint to the **Update Last Extract Time** task.

▶ Task 7: Test the Package

1. Click the **Data Flow** tab and view the **Extract Internet Sales** data flow.
2. On the **Debug** menu, click **Start Debugging** and observe the package as it executes, noting the number of rows transferred.
3. When execution is complete, on the **Debug** menu, click **Stop Debugging**.
4. Maximize SQL Server Management Studio, and in Object Explorer, in the **Staging** database, right-click the **dbo.ExtractLog** table and click **Select Top 1000 Rows**.
5. View the data in the **ExtractLog** table, noting the value in the **LastExtract** column for the **InternetSales** database has been updated to the date and time when you ran the package.
6. Right-click the **dbo.InternetSales** table and click **Select Top 1000 Rows**. The sales records in this table were extracted from the **InternetSales** database, where the **SalesOrderHeader** table had a **LastModified** column value between the previous **LastExtract** value, and the date and time when the package was executed.
7. Minimize SQL Server Management Studio.
8. In Visual Studio, with the **Extract Internet Sales** data flow displayed in the designer, on the **Debug** menu, click **Start Debugging** to execute the package again, noting that no rows are transferred during this execution
9. When execution is complete, on the **Debug** menu, click **Stop Debugging**. Then close Visual Studio.

Results: After this exercise, you should have an SSIS package that uses the high water mark technique to extract only records that have been modified since the previous extraction.

Exercise 2: Using Change Data Capture

▶ Task 1: Enable Change Data Capture

1. Maximize SQL Server Management Studio, and open the **Enable CDC.sql** file in the D:\Labfiles\Lab07\Starter\Ex2 folder.
2. Examine the Transact-SQL code in this script, noting that it enables CDC in the **InternetSales** database, and for the Customers table. Then click **Execute** to run the script. Two jobs should be started.
3. Open the **Test CDC.sql** file in the D:\Labfiles\Lab07\Starter\Ex2 folder, and examine the query, noting that it contains statements to perform the following tasks:
   - Retrieve data changes between 1/1/1900 and the current date by using a CDC function.
   - Modify the data in the **Customers** table.
   - Retrieve data changes between 1/1/1900 and the current date again.
4. Select the code under the comment **Select all changed customer records between 1/1/1900 and today** and click **Execute**. Note that no records are returned because there have been no changes in the database since Change Data Capture was enabled.
5. Select the two UPDATE statements under the comment **Make a change to all customers (to create CDC records)** and click **Execute**. This statement modifies the data in the **Customers** table by reversing the **FirstName** value and then reversing it back to its original value.
6. Select the code under the comment **Now see the net changes** and click Execute. Note that the query returns all records in the **Customers** table, because they have all been changed within the specified time period.

▶ Task 2: Create a Stored Procedure to Retrieve Modified Rows

1. In SQL Server Management Studio, open the **Create SP.sql** file in the D:\Labfiles\Lab07\Starter\Ex2 folder.
2. Examine the Transact-SQL code in the query file, and note that it creates a stored procedure with **StartDate** and **EndDate** parameters. The stored procedure performs the following tasks:
   - Retrieves the log sequence numbers for the dates specified in the **StartDate** and **EndDate** parameters.
   - If neither of the log sequence numbers is null, then at least one transaction has occurred in the database within the specified time period. The stored procedure uses a CDC function to return all records that have changed in the **Customers** table.
   - If no transactions have taken place in the specified time period, the stored procedure returns an empty rowset.
3. Click **Execute** to run the Transact-SQL code and create the stored procedure.
4. Click **New Query**, and type the following Transact-SQL in the new query window. Then click **Execute** to test the stored procedure:

```
USE InternetSales
GO
EXEC GetChangedCustomers '1/1/1900', '1/1/2099';
GO
```

▶ Task 3: Use the Stored Procedure in a Data Flow

1. In SQL Server Management Studio, open the **Reset Staging.sql** file in the D:\Labfiles\Lab07\Starter\Ex2 folder.
2. Click **Execute** to reset the staging database.
3. Minimize SQL Server Management Studio.
4. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab07\Starter\Ex2 folder.
5. In Solution Explorer, under **SSIS Packages**, double-click **Extract Internet Sales Data.dtsx**.
6. On the control flow surface, double-click the **Extract Customers** task.
7. On the data flow surface, double-click the **Customers** source.
8. In the **OLE DB Source Editor** dialog box, make the following changes to the configuration of the **Customers** source. Then click **OK**:
   - In the **Data access mode** drop-down list, select **SQL Command**.
   - In the **SQL command** text box, type the following Transact-SQL statement:

     ```
     EXEC GetChangedCustomers ?, ?
     ```

   - Click the **Parameters** button, and in **the Set Query Parameters** dialog box, create the following parameter mappings with a **Param direction** of **Input**, and then click then **OK**.
     - **@StartDate**: User::InternetSalesLastExtract
     - **@EndDate**: User::CurrentTime

▶ Task 4: Test the Package

1. With the **Extract Customers** data flow displayed in the designer, on the **Debug** menu, click **Start Debugging** and observe the package as it executes, noting the number of rows transferred.
2. When execution is complete, on the **Debug** menu, click **Stop Debugging**.
3. Maximize SQL Server Management Studio, and in Object Explorer, in the **Staging** database, right-click the **dbo.ExtractLog** table and click **Select Top 1000 Rows**.
4. View the data in the **ExtractLog** table, noting the value in the **LastExtract** column for the **InternetSales** database has been updated to the date and time when you ran the package.
5. Right-click the **dbo.Customers** table and click **Select Top 1000 Rows**. The customer records in this table were extracted from the **InternetSales** database, where no row has been changed between the previous **LastExtract** value, and the date and time when the package was executed.
6. Minimize SQL Server Management Studio.
7. In Visual Studio, with the **Extract Customers** data flow displayed in the designer, on the **Debug** menu, click **Start Debugging** to execute the package again, noting that no rows are transferred during this execution.
8. When execution is complete, on the **Debug** menu, click **Stop Debugging**. Then close Visual Studio.

Results: After this exercise, you should have a database in which Change Data Capture has been enabled, and an SSIS package that uses a stored procedure to extract modified rows based on changes monitored by Change Data Capture.

Exercise 3: Using the CDC Control Task

▶ Task 1: Enable Change Data Capture

1. Maximize SQL Server Management Studio, and open the **Enable CDC.sql** file in the D:\Labfiles\Lab07\Starter\Ex3 folder.
2. Examine the query, noting that it enables CDC in the **HumanResources** database, and for the **Employee** table. Then click **Execute** to run the query. Two jobs should be started.

▶ Task 2: View Staging Tables

1. In SQL Server Management Studio, in Object Explorer, under the **Tables** folder for the **Staging** database, right-click the **dbo.EmployeeDeletes** table and click **Select Top 1000 Rows**. Note that the table is empty.
2. Repeat the previous step for the **dbo.EmployeeInserts** and **dbo.EmployeeUpdates** tables.
3. Minimize SQL Server Management Studio.

▶ Task 3: Create Connection Managers for CDC Components

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab07\Starter\Ex3 folder.
2. In Solution Explorer, right-click the **Connection Managers** folder and click **New Connection Manager**. Then in the **Add SSIS Connection Manager** dialog box, click the **ADO.NET** connection manager type, and click **Add**.
3. In the **Configure ADO.NET Connection Manager** dialog box, click **New**. Then in the **Connection Manager** dialog box, in the **Server name** box type **localhost**, ensure **Use Windows authentication** is selected, and in the **Select or enter a database name** drop-down list, select **HumanResources**.
4. Click **OK** to close the **Connection Manager** dialog box, and click **OK** again to close the **Configure ADO.NET Connection Manager** dialog box.
5. In Solution Explorer, right-click the **localhost.HumanResources.conmgr** connection manager and click **Rename**. Then rename the connection manager to **localhost.HumanResources.ADO.NET.conmgr**.
6. In Solution Explorer, right-click the **Connection Managers** folder and click **New Connection Manager**. Then in the **Add SSIS Connection Manager** dialog box, click the **ADO.NET** connection manager type and click **Add**.
7. In the **Configure ADO.NET Connection Manager** dialog box, click **New**. Then in the **Connection Manager** dialog box, in the **Server name** box type **localhost**, ensure **Use Windows authentication** is selected, and in the **Select or enter a database** name drop-down list, select **Staging**.
8. Click **OK** to close the **Connection Manager** dialog box, and click **OK** again to close the **Configure ADO.NET Connection Manager** dialog box.
9. In Solution Explorer, right-click the **localhost.Staging 1.conmgr connection** manager and click **Rename**. Then rename the connection manager to **localhost.Staging.ADO.NET.conmgr**.

▶ Task 4: Create a Package for Initial Data Extraction

1. In Solution Explorer, right-click the **SSIS Packages** folder and click **New SSIS Package**.
2. When the new package is created, in Solution Explorer, right-click **Package1.dtsx** and click **Rename**. Then rename the package to **Extract Initial Employee Data.dtsx**.
3. In the SSIS Toolbox, in the **Other Tasks** section, drag a **CDC Control Task** to the control flow surface of the **Extract Initial Employee Data.dtsx** package.

4. On the control flow surface, right-click **CDC Control Task** and click **Rename**. Then rename it to **Mark Initial Load Start**.
5. Double-click the **Mark Initial Load Start** task, and in the **CDC Control Task Editor** dialog box, set the following properties. Then click **OK**:
   - **SQL Server CDC database ADO.NET connection manager**: localhost HumanResources ADO NET.
   - **CDC control operation**: Mark initial load start.
   - **Variable containing the CDC state**: Click **New** and then in the **Add New Variable** dialog box, click **OK** to create a variable named **CDC_State** in the **Extract Initial Employee Data** container.
   - **Automatically store state in a database table**: Selected.
   - **Connection manager for the database where the state is stored**: localhost Staging ADO NET.
   - **Table to use for storing state**: Click **New**, and in the **Create New State Table** dialog box, click **Run** to create a table named **[dbo].[cdc_states]** in the **Staging** database.
   - **State name**: CDC_State.
6. In the SSIS Toolbox, in the **Favorites** section, drag a **Data Flow Task** to the control flow surface of the **Extract Initial Employee Data.dtsx** package, under the **Mark Initial Load Start** task.
7. On the control flow surface, right-click **Data Flow Task** and click **Rename**. Then rename it to **Extract Initial Employee Data**. Then drag a green precedence constraint from the **Mark Initial Load Start** task to the **Extract Initial Employee Data** task.
8. Double-click the **Extract Initial Employee Data** task to view its data flow surface.
9. In the SSIS Toolbox, in the **Other Sources** section, drag an **ADO NET Source** to the data flow surface. Then on the data flow surface, right-click **ADO NET Source**, click **Rename**, and rename it to **Employees**.
10. Double-click **Employees** and in the **ADO.NET Source Editor** dialog box, set the following properties. Then click **OK**:
    - **ADO.NET connection manager**: localhost HumanResources ADO NET.
    - **Data access mode**: Table or view.
    - **Name of the table or view**: dbo"."Employee".
11. In the SSIS Toolbox, in the **Other Destinations** section, drag an **ADO NET Destination** to the data flow surface, below the **Employees** data source. Then on the data flow surface, right-click **ADO NET Destination**, click **Rename**, and rename it to **Employee Inserts**.
12. Click the **Employees** source and drag the blue data flow path connection to the **Employee Inserts** destination.
13. Double-click **Employee Inserts** and in the **ADO.NET Destination Editor** dialog box, set the following properties. Then click **OK**:
    - **Connection manager**: localhost Staging ADO NET.
    - **Use a table or view**: "dbo"."EmployeeInserts".
    - **Mappings**: On the **Mappings** tab, ensure all available input columns are mapped to destination columns of the same name.
14. Click the **Control Flow** tab, and in the SSIS Toolbox, in the **Other Tasks** section, drag a **CDC Control Task** to the control flow surface, below the **Extract Initial Employee Data** task.

15. On the control flow surface, right-click **CDC Control Task** and click **Rename**. Then rename it to **Mark Initial Load End**. Drag a "success" precedence constraint from the **Extract Initial Employee Data** task to the **Mark Initial Load End** task.
16. Double-click the **Mark Initial Load End** task, and in the **CDC Control Task Editor** dialog box, set the following properties. Then click **OK**:
    - **SQL Server CDC database ADO.NET connection manager**: localhost HumanResources ADO NET.
    - **CDC control operation**: Mark initial load end.
    - **Variable containing the CDC state**: User:: CDC_State.
    - **Automatically store state in a database table**: Selected.
    - **Connection manager for the database where the state is stored**: localhost Staging ADO NET.
    - **Table to use for storing state**: [dbo].[cdc_states].
    - **State name**: CDC_State.
17. On the **File** menu, click **Save All**.

▶ Task 5: Test Initial Extraction

1. In Visual Studio, ensure that the control flow for the **Extract Initial Employee Data.dtsx** package is open, and on the **Debug** menu, click **Start Debugging**. Then, when package execution is complete, on the **Debug** menu, click **Stop Debugging**.
2. Minimize Visual Studio and maximize SQL Server Management Studio.
3. In Object Explorer, right-click the **dbo.EmployeeInserts** table in the **Staging** database and click **Select Top 1000 Rows**. Note that the table now contains employee records.
4. In Object Explorer, under the **Staging** database, right-click the **Tables** folder and click **Refresh**. Note that a new table named **dbo.cdc_states** has been created in the staging database.
5. Right-click the **dbo.cdc_states** table and click **Select Top 1000 Rows**. Note that the table contains an encoded string that indicates the CDC state.
6. Minimize SQL Server Management Studio.

▶ Task 6: Create a Package for Incremental Data Extraction

1. Maximize Visual Studio, and then in Solution Explorer, right-click the **SSIS Packages** folder and click **New SSIS Package**.
2. When the new package is created, in Solution Explorer, right-click **Package1.dtsx** and click **Rename**. Then rename the package to **Extract Changed Employee Data.dtsx**.
3. In the SSIS Toolbox, in the **Other Tasks** section, drag a **CDC Control Task** to the control flow surface of the package.
4. On the control flow surface, right-click **CDC Control Task** and click **Rename**. Then rename it to **Get Processing Range**.
5. Double-click the **Get Processing Range** task, and in the **CDC Control Task Editor** dialog box, set the following properties. Then click **OK**:
    - **SQL Server CDC database ADO.NET connection manager**: localhost HumanResources ADO NET.
    - **CDC control operation**: Get processing range.

- **Variable containing the CDC state**: Click **New** and then click **OK** to create a variable named **CDC_State** in the **Extract Changed Employee Data** container.
- **Automatically store state in a database table**: Selected.
- **Connection manager for the database where the state is stored**: localhost Staging ADO NET.
- **Table to use for storing state**: [dbo].[cdc_states].
- **State name**: CDC_State.

6. In the SSIS Toolbox, drag a **Data Flow Task** to the control flow surface, and drop it under the **Get Processing Range** task.
7. On the control flow surface, right-click **Data Flow Task** and click **Rename**. Then rename it to **Extract Changed Employee Data**.
8. Click **Get Processing Range** and drag its green precedence constraint to the **Extract Changed Employee Data** task.
9. Double-click the **Extract Changed Employee Data** task to view its data flow surface.
10. In the SSIS Toolbox, in the **Other Sources** section, drag a **CDC Source** to the data flow surface of the **Extract Changed Employee Data** task. Then on the data flow surface, right-click **CDC Source**, click **Rename**, and rename it to **Employee Changes**.
11. Double-click **Employee Changes**, and in the **CDC Source** dialog box, set the following properties. Then click **OK**:
    - **ADO.NET connection manager**: localhost HumanResources ADO NET.
    - **CDC enabled table**: [dbo].[Employee].
    - **Capture instance**: dbo_Employee.
    - **CDC processing mode**: Net.
    - **Variable containing the CDC state**: User::CDC_State.
12. In the SSIS Toolbox, in the **Other Transforms** section, drag a **CDC Splitter** to the data flow surface, below the **Employee Changes** data source. Then click **Employee Changes** and drag the blue data flow path connection to the **CDC Splitter** transformation.
13. In the SSIS Toolbox, in the **Other Destinations** section, drag an **ADO NET Destination** to the data flow surface, below the **CDC Splitter** transformation. Then on the data flow surface, right-click **ADO NET Destination**, click **Rename**, and rename it to **Employee Inserts**.
14. Click the **CDC Splitter** transformation and drag the blue data flow path connection to the **Employee Inserts** destination. In the **Input Output Selection** dialog box, select the **InsertOutput** output and click **OK**.
15. Double-click **Employee Inserts** and in the **ADO.NET Destination Editor** dialog box, set the following properties. Then click **OK**:
    - **Connection manager**: localhost Staging ADO NET.
    - **Use a table or view**: "dbo"."EmployeeInserts".
    - **Mappings**: On the **Mappings** tab, verify that all available input columns other than **_$start_lsn**, **_$operation**, and **_$update_mask** are mapped to destination columns of the same name.
16. In the SSIS Toolbox, in the **Other Destinations** section, drag an **ADO NET Destination** to the data flow surface, directly below and to the left of the **CDC Splitter** transformation. Then on the data flow surface, right-click **ADO NET Destination**, click **Rename**, and rename it to **Employee Updates**.

17. Click the **CDC Splitter** transformation and drag the blue data flow path connection to the **Employee Updates** destination. In the **Input Output Selection** dialog box, select the **UpdateOutput** output and click **OK**.
18. Double-click **Employee Updates** and in the **ADO.NET Destination Editor** dialog box, set the following properties. Then click **OK**:
    - **Connection manager**: localhost Staging ADO NET.
    - **Use a table or view**: "dbo"."EmployeeUpdates"
    - **Mappings**: On the **Mappings** tab, verify that all available input columns other than **_$start_lsn**, **_$operation**, and **_$update_mask** are mapped to destination columns of the same name.
19. In the SSIS Toolbox, in the **Other Destinations** section, drag an **ADO NET Destination** to the data flow surface, below and to the right of the **CDC Splitter** transformation. Then on the data flow surface, right-click **ADO NET Destination**, click **Rename**, and rename it to **Employee Deletes**.
20. Click the **CDC Splitter** transformation and drag the blue data flow path connection to the **Employee Deletes** destination. The **DeleteOutput** output should be selected automatically.
21. Double-click **Employee Deletes** and in the **ADO.NET Destination Editor** dialog box, set the following properties. Then click **OK**:
    - **Connection manager**: localhost Staging ADO NET.
    - **Use a table or view**: "dbo"."EmployeeDeletes".
    - **Mappings**: On the **Mappings** tab, verify that all available input columns other than **_$start_lsn**, **_$operation**, and **_$update_mask** are mapped to destination columns of the same name.
22. Click the **Control Flow** tab, and in the SSIS Toolbox, in the **Other Tasks** section, drag a **CDC Control Task** to the control flow surface, below the **Extract Changed Employee Data** task.
23. On the control flow surface, right-click **CDC Control Task** and click **Rename**. Then rename it to **Mark Processed Range**.
24. Click **Extract Changed Employee Data** and drag its green precedence constraint to the **Mark Processed Range** task.
25. Double-click the **Mark Processed Range** task, and in the **CDC Control Task Editor** dialog box, set the following properties. Then click **OK**:
    - **SQL Server CDC database ADO.NET connection manager**: localhost HumanResources ADO NET.
    - **CDC control operation**: Mark processed range.
    - **Variable containing the CDC state**: User:: CDC_State.
    - **Automatically store state in a database table**: Selected.
    - **Connection manager for the database where the state is stored**: localhost Staging ADO NET.
    - **Table to use for storing state**: [dbo].[cdc_states]
    - **State name**: CDC_State.
26. On the **File** menu, click **Save All**.

▶ Task 7: Test Incremental Extraction

1. In Visual Studio, ensure that the control flow for the **Extract Changed Employee Data.dtsx** package is open, and on the **Debug** menu, click **Start Debugging**.
2. When package execution is complete, double-click the **Extract Changed Employee Data** task to verify that no rows were extracted (because no changes have been made to the source data since the initial extraction). Then, on the **Debug** menu, click **Stop Debugging**.
3. Maximize SQL Server Management Studio, and open the **Change Employees.sql** file in the D:\Labfiles\Lab07\Starter\Ex3 folder.
4. Review the Transact-SQL code and note that it truncates the **dbo.EmployeeInserts**, **dbo.EmployeeUpdates**, and **dbo.EmployeeDeletes** tables in the **Staging** database, and then makes the following changes to the **dbo.Employee** table in the **HumanResources** database:
   - Inserts a new employee record.
   - Updates employee 281 to change the **Title** column value.
   - Deletes employee 273.
5. Click **Execute** to run the Transact-SQL code, and then minimize SQL Server Management Studio and maximize Visual Studio.
6. In Visual Studio, ensure that the data flow for the **Extract Changed Employee Data** task is open, and on the **Debug** menu, click **Start Debugging**.
7. When package execution is complete, double-click the **Extract Changed Employee Data** task to verify that three rows were extracted and split into one insert, one update, and one delete. Then, on the **Debug** menu, click **Stop Debugging**.

Note: If no rows were transferred, stop debugging, wait for a few seconds, and then repeat the previous two steps.

8. Close Visual Studio and maximize SQL Server Management Studio.
9. In Object Explorer, under the **Tables** folder for the **Staging** database, right-click the **dbo.EmployeeInserts** table and click **Select Top 1000 Rows**. Note that the table contains the row that was inserted.
10. Repeat the previous step for the **dbo.EmployeeUpdates** and **dbo.EmployeeDeletes** tables, and verify that they contain the updated and deleted records respectively.
11. Minimize SQL Server Management Studio.

Results: After this exercise, you should have a **HumanResources** database in which Change Data Capture has been enabled, and an SSIS package that uses the CDC Control to extract the initial set of employee records. You should also have an SSIS package that uses the CDC Control and CDC data flow components to extract modified employee records based on changes recorded by Change Data Capture.

Exercise 4: Using Change Tracking

▶ Task 1: Enable Change Tracking

1. Maximize SQL Server Management Studio, and open the **Enable CT.sql** file in the D:\Labfiles\Lab07\Starter\Ex4 folder.
2. Examine the query, noting that it enables Change Tracking in the **ResellerSales** database, and for the **Resellers** table. Then click **Execute** to run the query.
3. Open the **Test CT.sql** file in the D:\Labfiles\Lab07\Starter\Ex4 folder, and note that it contains statements to perform the following tasks:
    - Get the current change tracking version number.
    - Retrieve all data from the **Resellers** table.
    - Store the current version number as the previously-retrieved version.
    - Update the **Resellers** table.
    - Get the new current version number.
    - Get all changes between the previous and current versions.
    - Store the current version number as the previously-retrieved version.
    - Update the **Resellers** table again.
    - Get the new current version number.
    - Get all changes between the previous and current versions.
4. Click **Execute** and view the results. Note that:
    - The first resultset shows all reseller records.
    - The second resultset indicates that the previously-retrieved version was numbered 0, and the current version is numbered 1.
    - The third resultset indicates that the previously-retrieved version was numbered 1, and the current version is numbered 2.

▶ Task 2: Create a Stored Procedure to Retrieve Modified Rows

1. In SQL Server Management Studio, open the **Create SP.sql** file in the D:\Labfiles\Lab07\Starter\Ex4 folder.
2. Examine the Transact-SQL code in the query file, and note that it enables snapshot isolation and creates a stored procedure with a single parameter named **LastVersion**. The stored procedure performs the following tasks:
    - Sets the isolation level to snapshot.
    - Retrieves the current change tracking version number.
    - If the **LastVersion** parameter is -1, the stored procedure assumes that no previous versions have been retrieved, and returns all records from the **Resellers** table.
    - If the **LastVersion** parameter is not -1, the stored procedure retrieves all changes between **LastVersion** and the current version.
    - The stored procedure updates the **LastVersion** parameter to the current version, so the calling application can store the last version retrieved for next time.
    - Sets the isolation level back to read "committed".

3. Click **Execute** to run the Transact-SQL code and create the stored procedure.
4. Click **New Query**, and type the following Transact-SQL in the new query window. Then click **Execute** to test the stored procedure:

```
USE ResellerSales
GO
DECLARE @p BigInt = -1;
EXEC GetChangedResellers @p OUTPUT;
SELECT @p LastVersionRetrieved;
EXEC GetChangedResellers @p OUTPUT;
```

▶ Task 3: Modify a Data Flow to use the Stored Procedure

1. In SQL Server Management Studio, open the **Reset Staging.sql** file in the D:\Labfiles\Lab07\Starter\Ex4 folder.
2. Click **Execute** to reset the staging database.
3. Minimize SQL Server Management Studio.
4. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab07\Starter\Ex4 folder.
5. In Solution Explorer, under **SSIS Packages**, double-click **Extract Reseller Data.dtsx**.
6. If the Variables pane is not visible, on the **SSIS** menu, click **Variables**. Then, in the Variables pane, click the **Add Variable** button and add a variable with the following settings:
   - **Name**: PreviousVersion
   - **Data Type**: Decimal
   - **Value**: 0
7. In the SSIS Toolbox, drag an **Execute SQL Task** to the control flow surface, above the **Extract Resellers** task. Then right-click the expression task, click Rename, and change the name to **Get Previously Extracted Version**.
8. Double-click the **Get Previously Extracted Version** task and in the **Execute SQL Task Editor** dialog box, configure the following settings. Then click **OK**:
   - On the **General** tab, in the **ResultSet** drop-down list, select **Single row**.
   - On the **General** tab, in the **Connection** drop-down list, select **localhost.Staging**.
   - On the **General** tab, in the **SQLStatement** box, click the ellipsis (**...**) button and then in the **Enter SQL Query**, dialog box, enter the following Transact-SQL query and click **OK**:

     ```
     SELECT MAX(LastVersion) LastVersion
     FROM ExtractLog
     WHERE DataSource = 'ResellerSales'
     ```

   - On the **Result Set** tab, click **Add**, and then in the **Result Name** column, change **NewResultName** to **LastVersion**, and in the **Variable Name** drop-down list, select **User::PreviousVersion**.
9. On the control flow surface, right-click the green precedence constraint between **Get Last Extract Time** and **Extract Resellers**, and click **Delete**.
10. Click the **Get Last Extract Time** task and drag its green precedence constraint to the **Get Previously Extracted Version** task. Then click the **Get Previously Extracted Version** task and drag its green precedence constraint to the **Extract Resellers** task.

11. In the SSIS Toolbox, drag an **Execute SQL Task** under the **Update Last Extract Time** task on the control flow surface.
12. Right-click **Execute SQL Task** and click **Rename**. Then change the name to **Update Previous Version**.
13. Double-click **Update Previous Version** and configure the following settings. Then click **OK**:
    - On the **General** tab, in the **Connection** drop-down list, select **localhost.Staging**.
    - On the **General** tab, in the **SQLStatement** box, click the ellipsis (**...**) button and then in the **Enter SQL Query**, dialog box, enter the following Transact-SQL query and click **OK**:

      ```
      UPDATE ExtractLog
      SET LastVersion = ?
      WHERE DataSource = 'ResellerSales'
      ```

    - On the **Parameter Mapping** tab, click **Add** and create the following parameter mapping:
      - **Variable Name**: User::PreviousVersion
      - **Direction**: Input
      - **Data Type**: LARGE_INTEGER
      - **Parameter Name**: 0
      - **Parameter Size**: -1
14. On the control flow surface, right-click the green precedence constraint between **Update Last Extract Time** and **Send Success Notification**, and click **Delete**. Then click the **Update Last Extract Time** task and drag its green precedence constraint to the **Update Previous Version** task. Click the **Update Previous Version** task and drag its green precedence constraint to the **Send Success Notification** task.
15. On the control flow surface, double-click the **Extract Resellers task**.
16. On the data flow surface, double-click the **Resellers** source.
17. In the **OLE DB Source Editor** dialog box, make the following changes to the configuration of the **Customers** source. Then click **OK**.
    - In the **Data access mode** drop-down list, select **SQL Command**.
    - In the **SQL command** text box, type the following Transact-SQL statement:

      ```
      EXEC GetChangedResellers ? OUTPUT
      ```

    - Click the **Parameters** button, and in the **Set Query Parameters** dialog box, create the following parameter mappings, and then click **OK**:
      - **Parameters**: @LastVersion
      - **Variables**: User::PreviousVersion
      - **Param direction**: InputOutput

▶ Task 4: Test the Package

1. In Visual Studio, with the **Extract Resellers** data flow displayed in the designer, on the **Debug** menu, click **Start Debugging** and observe the package as it executes, noting the number of rows transferred.
2. When execution is complete, on the **Debug** menu, click **Stop Debugging**.

3. Maximize SQL Server Management Studio, and in Object Explorer, in the **Staging** database, right-click the **dbo.ExtractLog** table and click **Select Top 1000 Rows**.
4. View the data in the **ExtractLog** table, noting the value in the **LastVersion** column for the **ResellerSales** database has been updated to the latest version retrieved from the source database.
5. Right-click the **dbo.Resellers** table and click **Select Top 1000 Rows**. The customer records in this table were extracted from the **ResellerSales** database, where no row has been changed between the previous **LastVersion** value, and the current version.
6. Close SQL Server Management Studio without saving any changes.
7. In Visual Studio, with the **Extract Resellers** data flow displayed in the designer, on the **Debug** menu, click **Start Debugging** to execute the package again, noting that no rows are transferred during this execution.
8. When execution is complete, on the **Debug** menu, click **Stop Debugging**. Then close Visual Studio.

Results: After this exercise, you should have a database in which Change Tracking has been enabled, and an SSIS package that uses a stored procedure to extract modified rows based on changes recorded by Change Tracking.

Module 8: Loading Data into a Data Warehouse

Lab: Loading a Data Warehouse

Exercise 1: Loading Data from CDC Output Tables

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Labfiles\Lab08\Starter folder, right-click **Setup.cmd** and click **Run as administrator**.
3. When prompted, click **Yes** to confirm you want to run the command file, and wait for the script to finish.

▶ Task 2: Create a Data flow for Inserts

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab08\Starter\Ex1 folder.
2. In Solution Explorer, in the **Connection Managers** folder, note that a connection manager for the **AWDataWarehouse** database has been created.
3. In Solution Explorer, right-click the **SSIS Packages** folder and click **New SSIS Package**.
4. When the new package is created, in Solution Explorer, right-click **Package1.dtsx** and click **Rename**. Then rename the package to **Load Employee Data.dtsx**.
5. In the SSIS Toolbox, in the **Favorites** section, drag a **Data Flow Task** to the control flow surface of the package.
6. On the control flow surface, right-click **Data Flow Task** and click **Rename**. Then rename it to **Insert Employees**.
7. Double-click **Insert Employees** to view its data flow surface.
8. In the SSIS Toolbox, drag a **Source Assistant** to the data flow surface. Then in the **Add New Source** dialog box, in the **Select source type** list, select **SQL Server**. In the **Select connection managers** list, select **localhost.Staging**, and click **OK**.
9. Right-click **OLE DB Source**, click **Rename**, and change the name to **Staged Employee** Inserts.
10. Double-click the **Staged Employee Inserts** source, and in the **OLE DB Source Editor** dialog box, configure the following settings. Then click **OK**:
    - **OLE DB connection manager**: localhost.Staging.
    - **Data access mode**: Table or view.
    - **Name of the table or view**: [dbo].[EmployeeInserts].
11. In the SSIS Toolbox, drag a **Destination Assistant** to the data flow surface below the **Staged Employee Inserts** source. Then in the **Add New Destination** dialog box, in the **Select destination type** list, select **SQL Server**. In the **Select connection managers** list, select **localhost.AWDataWarehouse**, and click **OK**.
12. Right-click **OLE DB Destination**, click **Rename**, and change the name to **New Employees**.
13. Click the **Staged Employee Inserts** source and drag the blue data flow arrow to the **New Employees** destination.

14. Double-click the **New Employees** destination, and in the **OLE DB Source Editor** dialog box, configure the following settings. Then click **OK**:
    - **OLE DB connection manager**: localhost.AWDataWarehouse.
    - **Data access mode**: Table or view – fast load.
    - **Name of the table or view**: [dbo].[DimEmployee].
    - **Mappings**: On the **Mappings** tab, drag the **EmployeeID** input column to the **EmployeeAlternateKey** destination column, and verify that all other input columns are mapped to destination columns of the same name and that the **EmployeeKey** and **Deleted** destination columns are not mapped.
15. On the **File** menu, click **Save All**.

▶ Task 3: Create a Data Flow for Updates

1. In Visual Studio, click the **Control Flow** tab of the **Load Employee Data.dtsx** package.
2. In the SSIS Toolbox, in the **Favorites** section, drag a **Data Flow Task** to the control flow surface, directly below the **Insert Employees** data flow task.
3. On the control flow surface, right-click **Data Flow Task** and click **Rename**. Then rename it to **Update Employees**.
4. Click the **Insert Employees** data flow task, and then drag its green precedence constraint to the **Update Employees** data flow task.
5. Double-click **Update Employees** to view its data flow surface.
6. In the SSIS Toolbox, drag a **Source Assistant** to the data flow surface. Then in the **Add New Source** dialog box, in the **Select source types** list, select **SQL Server**. In the **Select connection managers** list, select **localhost.Staging**, and click **OK**.
7. Right-click **OLE DB Source**, click **Rename**, and change the name to **Staged Employee Updates**.
8. Double-click the **Staged Employee Updates** source, and in the **OLE DB Source Editor** dialog box, configure the following settings. Then click **OK**:
    - **OLE DB connection manager**: localhost.Staging.
    - **Data access mode**: Table or view.
    - **Name of the table or view**: [dbo].[EmployeeUpdates].
9. In the SSIS Toolbox, drag an **OLE DB Command** to the data flow surface below the **Staged Employee Updates** source. Then right-click **OLE DB Command**, click **Rename**, and change the name to **Update Existing Employees**.
10. Click the **Staged Employee Updates** data source, and then drag its blue data flow path to the **Update Existing Employees** transformation.
11. Double-click the **Update Existing Employees** transformation. Then in the **Advanced Editor** for **Update Existing Employees** dialog box, configure the following settings and click **OK**:
    - On the **Connection Managers** tab, in the **Connection Manager** drop-down list, select **localhost.AWDataWarehouse**.

- On the **Component Properties** tab, set the **SqlCommand** property to the following Transact-SQL statement:

```
UPDATE dbo.DimEmployee
SET    FirstName = ?, LastName = ?, EmailAddress = ?, Title = ?, HireDate = ?
WHERE EmployeeAlternateKey = ?
```

- On the **Column Mappings** tab, create the following mappings:
  - **FirstName**: Param_0
  - **LastName**: Param_1
  - **EmailAddress**: Param_2
  - **Title**: Param_3
  - **HireDate**: Param_4
  - **EmployeeID**: Param_5

12. On the **File** menu, click **Save All**.

▶ Task 4: Create a Data Flow for Deletes

1. In Visual Studio, click the **Control Flow** tab of the **Load Employee Data.dtsx** package.
2. In the SSIS Toolbox, in the **Favorites** section, drag a **Data Flow Task** to the control flow surface, directly below the **Update Employees** data flow task.
3. On the control flow surface, right-click **Data Flow Task** and click **Rename**. Then rename it to **Delete Employees**.
4. Click the **Update Employees** data flow task and then drag its green precedence constraint to the **Delete Employees** data flow task.
5. Double-click **Delete Employees** to view its data flow surface.
6. In the SSIS Toolbox, drag a **Source Assistant** to the data flow surface. Then in the **Add New Source** dialog box, in the **Select source types** list, select **SQL Server**. In the **Select connection managers** list, select **localhost.Staging**, and click **OK**.
7. Right-click **OLE DB Source**, click **Rename**, and change the name to **Staged Employee Deletes**.
8. Double-click the **Staged Employee Deletes** source, and in the **OLE DB Source Editor** dialog box, configure the following settings. Then click **OK**:
   - **OLE DB connection manager**: localhost.Staging.
   - **Data access mode**: Table or view.
   - **Name of the table or view**: [dbo].[EmployeeDeletes].
9. In the SSIS Toolbox, drag an **OLE DB Command** to the data flow surface below the **Staged Employee Deletes** source. Then right-click **OLE DB Command**, click **Rename**, and change the name to **Delete Existing Employees**.
10. Click the **Staged Employee Deletes** data source, and then drag its blue data flow path to the **Delete Existing Employees** transformation.
11. Double-click the **Delete Existing Employees** transformation. Then in the **Advanced Editor for Delete Existing Employees** dialog box, configure the following settings and click OK:
    - On the **Connection Managers** tab, in the **Connection Manager** drop-down list, select **localhost.AWDataWarehouse**.

- On the **Component Properties** tab, set the **SqlCommand** property to the following Transact-SQL statement:

```
UPDATE dbo.DimEmployee
SET Deleted = 1
WHERE EmployeeAlternateKey = ?
```

- On the **Column Mappings** tab, map the **EmployeeID** column to **Param_0**.

12. On the **File** menu, click **Save All**.

▶ Task 5: Test the Package

1. In Visual Studio, click the **Control Flow** tab of the **Load Employee Data.dtsx** package.
2. On the **Debug** menu, click **Start Debugging** and observe the package as it executes.
3. When execution is complete, double-click the **Insert Employees** data flow task and review the number of rows inserted. Then in the **Data Flow Task** drop-down list at the top of the data flow surface designer, select **Update Employees** and **Delete Employees** in turn, noting the number of rows processed by these tasks.
4. On the **Debug** menu, click **Stop Debugging**.
5. Close Visual Studio, saving the changes if prompted.

Results: After this exercise, you should have an SSIS package that uses data flows to apply inserts, updates, and logical deletes in the data warehouse, based on staging tables extracted by the CDC Control task and data flow components.

Exercise 2: Using a Lookup Transformation to Insert or Update Dimension Data

▶ Task 1: View Data Flows

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab08\Starter\Ex2 folder.
2. In Solution Explorer, under the **SSIS Packages** folder, double-click **Load Products Data.dtsx**.
3. On the control flow surface, in the **Load Product Dimension Hierarchy** sequence container, double-click the **Load Product Category Dimension** task to view its data flow.
4. Examine the data flow for the **Load Product Category Dimension** task, and note the following features:
   - The **Staged Product Category Data** source extracts product category data from the **InternetSales** and **ResellerSales** tables in the **Staging** database.
   - The **Lookup Existing Product Categories** task retrieves the **ProductCategoryKey** value for product categories that exist in the **DimProductCategory** table in the **AWDataWarehouse** database by matching the product category business key in the staging database to the product category alternative key in the data warehouse.
   - The **Lookup No Match Output** data flow path from the **Lookup Existing Product Categories** task connects to the **New Product Categories** destination, and the **Lookup Match Output** data flow path connects to the **Update Existing Product Categories** task.
   - The **New Product Categories** destination loads new product category records into the **DimProductCategory** table.

- The **Update Existing Product Categories** task executes a Transact-SQL statement to update the **ProductCategoryName** column in the **DimProductCategory** table for an existing row based on the **ProductCategoryKey**.

5. In the **Data Flow Task** drop-down list, select **Load Product Subcategory Dimension**, and note that this data flow inserts or updates product subcategory dimension data using a similar approach to the **Load Product Category Dimension** data flow. Additionally, there is a lookup task to retrieve the **ProductCategoryKey** in **AWDataWarehouse** for the parent category, which should have already been loaded.

▶ Task 2: Create a Data Flow

1. In Visual Studio, in the **Load Products Data.dtsx** package designer, click the **Control Flow** tab. Then in the SSIS Toolbox, drag a **Data Flow Task** to the control flow surface, under the **Load Product Subcategory Dimension** task in the **Load Product Dimension Hierarchy** sequence container.
2. Right-click the data flow task you added, click **Rename**, and change the name to **Load Product Dimension**. Then click the **Load Product Subcategory Dimension** task, and drag the green precedence constraint to the **Load Product Dimension** task.
3. Double-click the **Load Product Dimension** task to view the data flow surface.
4. In the SSIS Toolbox, drag a **Source Assistant** to the data flow surface. Then in the **Add New Source** dialog box, in the **Select source type** list, select **SQL Server**. In the **Connection Managers** list, select **localhost.Staging**, and click **OK**.
5. Right-click **OLE DB Source**, click **Rename**, and change the name to **Staged Product Data**.
6. Double-click the **Staged Product Data** source, and in the **OLE DB Source Editor** dialog box, in the **Data access mode** drop-down list, select **SQL Command**.
7. In the **OLE DB Source Editor** dialog box, in the **SQL command** text box, type the following Transact-SQL statement. Then click **OK**:

```
SELECT DISTINCT ProductSubcategoryBusinessKey, ProductBusinessKey, ProductName,
StandardCost, Color, ListPrice, Size, Weight, Description
FROM dbo.InternetSales
UNION
SELECT DISTINCT ProductSubcategoryBusinessKey, ProductBusinessKey, ProductName,
StandardCost, Color, ListPrice, Size, Weight, Description
FROM dbo.ResellerSales
```

▶ Task 3: Add a Lookup Transformation for Parent Keys

1. In the SSIS Toolbox, drag a Lookup transformation to the data flow surface below the **Staged Product Data** source. Then right-click Lookup, click **Rename**, and change the name to **Lookup Parent Subcategory**.
2. Click the **Staged Product Data** source and drag the blue data flow arrow to the **Lookup Parent Subcategory** transformation.
3. Double-click the **Lookup Parent Subcategory** transformation, and then in the **Lookup Transformation Editor** dialog box, configure the following settings and click **OK**:
   - On the **General** tab, in the **Specify how to handle rows with no matching entries** drop-down list, ensure **Fail component** is selected.
   - On the **Connection** tab, in the **OLE DB connection manager** drop-down list, select **localhost.AWDataWarehouse**.
   - On the **Connection** tab, in the **Use a table or a view** drop-down list, select **[dbo].[DimProductSubcategory]**.

- On the **Columns** tab, drag the **ProductSubcategoryBusinessKey** column in the **Available Input Columns** list to the **ProductSubcategoryAlternateKey** column in the **Available lookup columns** list. Then, in the **Available lookups column** list, select the check box for the **ProductSubcategoryKey** column.
- On the **Columns** tab, ensure that the **ProductSubcategoryKey** column is added as a new lookup column with the output alias **ProductSubcategoryKey**, and then click **OK**.

▶ Task 4: Add a Lookup Transformation for Product Records

1. In the SSIS Toolbox, drag a **Lookup** transformation to the data flow surface below the **Lookup Parent Subcategory** transformation. Then right-click **Lookup**, click **Rename**, and change the name to **Lookup Existing Products**.
2. Click the **Lookup Parent Subcategory** transformation and drag the blue data flow arrow to the **Lookup Existing Products** transformation. Then, in the **Input Output Selection** dialog box, in the **Output** drop-down list, select **Lookup Match Output** and click **OK**.
3. Double-click the **Lookup Existing Products** transformation, and then in the **Lookup Transformation Editor** dialog box, configure the following settings and click **OK**:
   - On the **General** tab, in the **Specify how to handle rows with no matching entries** drop-down list, select **Redirect rows to no match output**.
   - On the **Connection** tab, in the **OLE DB connection manager** drop-down list, select **localhost.AWDataWarehouse**.
   - On the **Connection** tab, in the **Use a table or a view** drop-down list, select **[dbo].[DimProduct]**.
   - On the **Columns** tab, drag the **ProductBusinessKey** column in the **Available Input Columns** list to the **ProductAlternateKey** column in the **Available lookup columns** list. Then in the **Available lookups** column list, select the check box for the **ProductKey** column.
   - On the **Columns** tab, ensure that the **ProductKey** column is added as a new column with the output alias **ProductKey**, and then click **OK**.

▶ Task 5: Add a Destination for New Products

1. In the SSIS Toolbox, drag a **Destination Assistant** to the data flow surface below and to the right of the **Lookup Existing Products** transformation. Then in the **Add New Destination** dialog box, in the **Types** list, select **SQL Server**, in the **Connection Managers** list, select **localhost.AWDataWarehouse**, and click **OK**.
2. Right-click **OLE DB Destination**, click **Rename**, and change the name to **New Products**.
3. Click the **Lookup Existing Products** transformation and drag the blue data flow arrow to the **New Products** destination. Then in the **Input Output Selection** dialog box, in the **Output** drop-down list, select **Lookup No Match Output** and click **OK**.
4. Double-click the **New Products** destination. Then in the **OLE DB Destination Editor** dialog box, in the **Name of the table or the view** drop-down list, select [**dbo].[DimProduct]**.

5. On the **Mappings** tab, create the following mappings. Then click **OK**.
   - <ignore>: ProductKey
   - ProductBusinessKey: ProductAlternateKey
   - ProductName: ProductName
   - StandardCost: StandardCost
   - Color: Color
   - ListPrice: ListPrice
   - Size: Size
   - Weight: Weight
   - Description: Description
   - ProductSubcategoryKey: ProductSubcategoryKey

▶ Task 6: Add an OLE DB Command for Updated Product Records

1. In the SSIS Toolbox, drag an **OLE DB Command** to the data flow surface below and to the left of the **Lookup Existing Products** transformation.
2. Right-click **OLE DB Command**, click **Rename**, and change the name to **Update Existing Products**.
3. Click the **Lookup Existing Products** transformation and drag the blue data flow arrow to the **Update Existing Products** transformation. The **Lookup Match Output** is automatically selected.
4. Double-click the **Update Existing Products** transformation. Then in the **Advanced Editor for Update Existing Products** dialog box, configure the following settings and click **OK**:
   - On the **Connection Managers** tab, in the **Connection Manager** drop-down list, select **localhost.AWDataWarehouse**.
   - On the **Component Properties** tab, set the **SqlCommand** property to the following Transact-SQL statement:

```
UPDATE dbo.DimProduct
SET     ProductName = ?, StandardCost = ?, Color = ?, ListPrice = ?, Size = ?,
Weight = ?, Description = ?, ProductSubcategoryKey = ?
WHERE ProductKey = ?
```

   - On the **Column Mappings** tab, create the following mappings:
     - ProductName: Param_0
     - StandardCost: Param_1
     - Color: Param_2
     - ListPrice: Param_3
     - Size: Param_4
     - Weight: Param_5
     - Description: Param_6
     - ProductSubcategoryKey: Param_7
     - ProductKey: Param_8

▶ Task 7: Test the Package

1. With the **Load Product Dimension** data flow displayed in the designer, on the **Debug** menu, click **Start Debugging** and observe the package as it executes, noting that all rows flow to the **New Products** destination (because the data warehouse contained no existing product records). When execution is complete, on the **Debug** menu, click **Stop Debugging**.
2. On the **Debug** menu, click **Start Debugging** and observe the package as it executes again, noting that this time, all rows flow to the **Update Existing Products** transformation. This is because all staged product records were loaded to the data warehouse during the previous execution, so they all match existing records. When execution is complete, on the **Debug** menu, click **Stop Debugging**.
3. Close Visual Studio, saving your changes if prompted.

Results: After this exercise, you should have an SSIS package that uses a Lookup transformation to determine whether product records already exist, and updates them or inserts them as required.

Exercise 3: Implementing a Slowly Changing Dimension

▶ Task 1: Execute a Package to Load a Non-Changing Dimension

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab08\Starter\Ex3 folder.
2. In Solution Explorer, under the **SSIS Packages** folder, double-click **Load Geography Data.dtsx**.
3. Review the control flow and data flow defined in the package. This package includes a simple data flow to load staged geography data into the data warehouse. Only new rows are loaded, and rows that match existing data are discarded.
4. On the **Debug** menu, click **Start Debugging**, and observe the package execution as it loads geography data into the data warehouse.
5. When package execution has completed, on the **Debug** menu, click **Stop Debugging**.

▶ Task 2: Observe a Data Flow for a Slowly Changing Dimension

1. In Solution Explorer, under the **SSIS Packages** folder, double-click **Load Reseller Data.dtsx**.
2. On the control flow surface, double-click the **Load Reseller Dimension** task to view its data flow.
3. Examine the data flow for the **Load Reseller Dimension** task, and note the following features:
   - The **Staged Reseller Data** source extracts data from the **Resellers** table in the **Staging** database.
   - The **Lookup Geography Key** transformation looks up the geography key for the reseller in the **DimGeography** table in the **AWDataWarehouse** database.
   - The **Reseller SCD** is a slowly changing dimension transformation that has generated the remaining transformations and destinations. You can double-click the Reseller SCD transformation to view the wizard used to configure the slowly changing dimension, and then click **Cancel** to avoid making any unintentional changes.
   - The **Reseller SCD** transformation maps the **ResellerBusinessKey** input column to the **ResellerAlternateKey** dimension column and uses it as a business key to identify existing records.
   - The **Reseller SCD** transformation treats **AddressLine1**, **AddressLine2**, **BusinessType**, **GeographyKey**, and **NumberEmployees** as historical attributes, **Phone** and **ResellerName** as changing attributes, and **YearOpened** as a fixed attribute.

4. On the **Debug** menu, click **Start Debugging** and observe the data flow as it executes.
5. When execution is complete, on the **Debug** menu, click **Stop Debugging**.

▶ Task 3: Implement a Slowly Changing Dimension Transformation

1. In Solution Explorer, under the **SSIS Packages** folder, double-click **Load Customer Data.dtsx**.
2. On the control flow surface, double-click the **Load Customer Dimension** task to view its data flow. Note that a source to extract customer data from the **Staging** database and a **Lookup** transformation that retrieves a **GeographyKey** value from the **AWDataWarehouse** database have already been added to the data flow.
3. In the SSIS Toolbox pane, drag a **Slowly Changing Dimension** transformation to the data flow surface, below the **Lookup Geography Key** transformation. Then right-click **Slowly Changing Dimension** click **Rename**, and change the name to **Customer SCD**.
4. Click the **Lookup Geography Key** transformation and drag the blue data flow arrow to the **Customer SCD** transformation. Then, in the **Input Output Selection** dialog box, in the **Output** drop-down list, select **Lookup Match Output** and click **OK**.
5. Double-click the **Customer SCD** transformation to start the Slowly Changing Dimension wizard, and on the **Welcome to the Slowly Changing Dimension Wizard** page, click **Next**.
6. On the **Select a Dimension Table and Keys** page, in the **Connection manager** drop-down list, select **localhost.AWDataWarehouse**, and in the **Table or view** drop-down list, select **[dbo].[DimCustomer]**. Then specify the following column mappings and click **Next**:

Input Columns	Dimension Columns	Key Type
AddressLine1	AddressLine1	Not a key column
AddressLine2	AddressLine2	Not a key column
BirthDate	BirthDate	Not a key column
CommuteDistance	CommuteDistance	Not a key column
	CurrentRecord	
CustomerBusinessKey	CustomerAlternateKey	Business key
EmailAddress	EmailAddress	Not a key column
FirstName	FirstName	Not a key column
Gender	Gender	Not a key column
GeographyKey	GeographyKey	Not a key column
HouseOwnerFlag	HouseOwnerFlag	Not a key column
LastName	LastName	Not a key column

Input Columns	Dimension Columns	Key Type
MaritalStatus	MaritalStatus	Not a key column
MiddleName	MiddleName	Not a key column
NumberCarsOwned	NumberCarsOwned	Not a key column
Occupation	Occupation	Not a key column
Phone	Phone	Not a key column
Suffix	Suffix	Not a key column
Title	Title	Not a key column

7. On the **Slowly Changing Dimension Columns** page, specify the following change types and click **Next**:

Dimension Columns	Change Type
AddressLine1	Historical attribute
AddressLine2	Historical attribute
BirthDate	Changing attribute
CommuteDistance	Historical attribute
EmailAddress	Changing attribute
FirstName	Changing attribute
Gender	Historical attribute
GeographyKey	Historical attribute
HouseOwnerFlag	Historical attribute
LastName	Changing attribute
MaritalStatus	Historical attribute
MiddleName	Changing attribute
NumberCarsOwned	Historical attribute

Dimension Columns	Change Type
Occupation	Historical attribute
Phone	Changing attribute
Suffix	Changing attribute
Title	Changing attribute

8. On the **Fixed and Changing Attribute Options** page, leave both options clear and click **Next**.
9. On the **Historical Attribute Options** page, select **Use a single column to show current and expired records**. Then, in the **Column to indicate current record** drop-down list, select **CurrentRecord**. In the **Value when current** drop-down list, select **True**, and in the **Expiration value** drop-down list, select **False**. Then click **Next**.
10. On the **Inferred Dimension Members** page, uncheck the **Enable inferred member support** option and click **Next**.
11. On the **Finish the Slowly Changing Dimension Wizard** page, click **Finish**. Note that a number of transformations and a destination are created.

▶ Task 4: Test the Package

1. With the **Load Customer Dimension** data flow displayed in the designer, on the **Debug** menu, click **Start Debugging** and observe the package as it executes, noting that all rows pass through the **New Output** data flow path.
2. When execution is complete, on the **Debug** menu, click **Stop Debugging**. Then on the **Debug** menu, click **Start Debugging** and observe the package as it executes again, noting that no rows pass through the **New Output** data flow path, because they already exist and no changes have been made. When execution is complete, on the **Debug** menu, click **Stop Debugging**.
3. Start SQL Server Management Studio and connect to the **localhost** database engine instance by using Windows authentication. Then open the **Update Customers.sql** file in the D:\Labfiles\Lab08\Starter\Ex3 folder.
4. Examine the script and note that it updates two records in the staging database, changing one customer's phone number and another's marital status. Then click **Execute**.
5. When the query has completed, close SQL Server Management Studio.
6. In Visual Studio, on the **Debug** menu, click **Start Debugging** and observe the package as it executes, noting that one rows passes through the **Historical Attribute Inserts Output** data flow path, and another passes through the **Changing Attributes Updates Output**.
7. When execution is complete, on the **Debug** menu, click **Stop Debugging**. Then close Visual Studio, saving your changes if prompted.

Results: After this exercise, you should have an SSIS package that uses a Slowly Changing Dimension transformation to load data into a dimension table.

Exercise 4: Using the MERGE Statement

▶ Task 1: Examine a Control Flow that uses the MERGE Statement

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab08\Starter\Ex4 folder.
2. In Solution Explorer, under the **SSIS Packages** folder, double-click **Load Reseller Sales Data.dtsx**.
3. On the control flow surface, double-click the **Merge Reseller Sales** task, and in the **Execute SQL Task Editor** dialog box, note the following configuration settings. Then click **Cancel**:
   - The task uses the **localhost.Staging** connection manager to connect to the **Staging** database.
   - The task executes a Transact-SQL MERGE statement that retrieves reseller sales and related dimension keys from the **Staging** and **AWDataWarehouse** databases. It then matches these records with the **FactResellerSales** table based on the **SalesOrderNumber** and **SalesOrderLineNumber** columns, updates rows that match, and inserts new records for rows that do not.
4. On the **Debug** menu, click **Start Debugging** and observe the package as it executes.
5. When execution is complete, on the **Debug** menu, click **Stop Debugging**.

▶ Task 2: Create a Package that Uses the MERGE Statement

1. In Solution Explorer, right-click the **SSIS Packages** folder and click **New SSIS Package**. Then right-click **Package1.dtsx**, click **Rename**, and change the name to **Load Internet Sales Data.dtsx**.
2. In the SSIS Toolbox pane, drag an **Execute SQL Task** to the control flow surface. Then right-click **Execute SQL Task**, click **Rename**, and change the name to **Merge Internet Sales Data**.
3. Double-click **Merge Internet Sales Data**. Then in the **Execute SQL Task Editor** dialog box, configure the following settings and click **OK**:
   - In the **Connection** drop-down list, select **localhost.Staging**.
   - Click **Browse**, and then open D:\Labfiles\Lab08\Starter\Ex4\Merge Internet Sales.sql.

▶ Task 3: Test the Package

1. Save the project, and then on the **Debug** menu, click **Start Debugging** and observe the package as it executes.
2. When execution is complete, on the **Debug** menu, click **Stop Debugging**. Then, close Visual Studio.

Results: After this exercise, you should have an SSIS package that uses an Execute SQL task to execute a MERGE statement that inserts or updates data in a fact table.

Module 9: Enforcing Data Quality

Lab A: Cleansing Data

Exercise 1: Creating a DQS Knowledge Base

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Labfiles\Lab09\Starter folder, right-click **Setup.cmd** and click **Run as administrator**.
3. Click **Yes** when prompted to confirm you want to run the command file, and wait for the script to finish.

▶ Task 2: View Existing Data

1. In the D:\Labfiles\Lab09\Starter folder, double-click **Sample Customer Data.xls** to open it in Excel.
2. In Excel, on the **CountryRegion** worksheet, note that the data shows the number of customers by country name and country code. Also note that the data contains the following quality issues:
   - The list of country names includes both **America** and **United States**. For the purposes of Adventure Works sales, these represent the same sales territory.
   - The list of country names includes both **Great Britain** and **United Kingdom**. For the purposes of Adventure Works sales, these represent the same sales territory.
   - The list of country codes includes both **GB** and **UK**. For the purposes of Adventure Works sales, these represent the same sales territory.
3. In Excel, on the **StateProvince** worksheet, note that the data shows the number of customers by country and state. In addition to the concerns previously identified on the **CountryRegion** worksheet, note that the data contains the following quality issues:
   - The states **Oregon** and **California** exist in both **America** and **United States**.
   - The states **Wales** and **England** exist in both **Great Britain** and **United Kingdom**.
   - **Australia** includes a state named **New South Wales** and a state named **NSW**, which represent the same one.
   - **United States** includes **Washington** and **WA**, which represent the same state, as do **California** and **CA**.
4. In Excel, on the **Gender** worksheet, note that two customers have a gender code of **W**. Valid values for the gender code are **F** and **M**.
5. On the **Sample Customer Data** worksheet, apply column filters to explore the data further and view the source records for the anomalous data.
6. Close Excel without saving any changes to the workbook.

▶ Task 3: Create a Knowledge Base

1. On the task bar, click **SQL Server 2014 Data Quality Client**.
2. When prompted, enter the server name **MIA-SQL**, and click **Connect**.

3. In SQL Server Data Quality Services, in the **Knowledge Base Management** section, click **New Knowledge Base**. Then enter the following details and click **Next**:
   - **Name**: Customer KB.
   - **Description**: Customer data knowledge base.
   - **Create Knowledge Base From**: Existing Knowledge Base.
   - **Select Knowledge Base**: DQS Data.
   - **Select Activity**: Domain Management.
4. In the **Domain** list, click **Country/Region**. Then click the **Domain Values** tab and note the country values that are defined in this knowledge base. Leading domain values are shown in bold, with synonym values indented below.
5. In the **Domain** list, click **Country/Region (two-letter leading)**, and on the **Domain Values** tab note the country code values that are defined in this knowledge base. This defines the same values as the **Country/Region** domain, but with the two-character country code designated as the leading value.
6. In the **Domain** list, click **US - State**, and on the **Domain Values** tab, note the state values that are defined in this knowledge base. Note that state codes (such as CA and OR) are valid, but are shown as synonyms for leading state name values (such as California and Oregon). State values that are commonly entered in error are shown with a red cross and a valid value to which they are automatically corrected.
7. With the **US - State** domain selected, on the **Doman Properties** tab, change **Domain Name** to **State**. Later in this lab, you will use this domain for states and provinces in countries other than the United States.
8. Click the **Create a domain** button, and in the **Create Domain** dialog box, enter the following details, and then click **OK**:
   - **Domain Name**: Gender
   - **Description**: Male or female
   - **Data Type**: String
   - **Use Leading Values**: Selected
   - **Normalize String**: Selected
   - **Format Output to**: Upper Case
   - **Language**: English
   - **Enable Speller**: Selected
   - **Disable Syntax Error Algorithms**: Not selected
9. In the **Domain** list, click **Gender**. Then click the **Domain Values** tab and note that the value **DQS_NULL** (which represents a null value) has already been defined as a valid value for this domain.
10. On the **Domain Values** tab, click the **Add new domain value** button, and then enter the value **F** and press Enter.
11. Repeat the previous step to add the values **M, Female**, and **Male**.
12. Click the value **F**, and then hold the **Ctrl** key and click the value **Female**. Then, with both values selected, click the **Set selected domain values as synonyms** button. Depending on the screen resolution, this may be in a drop-down list at the end of the toolbar above the table. Note that **F** becomes the leading value, with **Female** indented below it.

13. Repeat the previous step to set **M** and **Male** as synonyms with **M** as the leading value.
14. Click **Finish**. When prompted to publish the knowledge base, click **No**.

▶ Task 4: Perform Knowledge Discovery

1. In SQL Server Data Quality Services, under **Recent knowledge base**, click **Customer KB** and then click **Knowledge Discovery**.
2. On the **Map** page, in the **Data Source** drop-down list, select **Excel File**, and then click **Browse** and open the **Sample Customer Data.xls** file in the D:\Labfiles\Lab09\Starter folder.
3. After the file has loaded, in the **Worksheet** drop-down list, select **'Sample Customer Data$'** and ensure that **Use first row as header** is selected. Then, in the **Mappings** table, select the following mappings, then click **Next**.

Source Column	Domain
CountryRegionCode (String)	Country/Region (two-letter leading)
CountryRegionName (String)	Country/Region
StateProvinceName (String)	State
Gender (String)	Gender

4. On the **Discover** page, click **Start**. Then wait for the discovery analysis process to finish.
5. When discovery analysis is complete, click **Next**.
6. On the **Manage Domain Values** page, in the **Domain** list, click **State** and view the new values that have been discovered.
7. In the list of values, click **New South Wales**, press Ctrl, click **NSW**, and then click **Set selected domain values as synonyms**. Note that **NSW** now has a **Correct to** value of **New South Wales**.
8. On the **Manage Domain Values** page, in the **Domain** list, click **Country/Region (two-letter leading)** and view the new values that have been discovered.
9. In the **Type** column for the value **UK**, click the symbol for **Error**, and in the **Correct To** column, type **GB**. Then clear the **Show Only New** checkbox and note that **UK** now appears as an error under the **GB** leading value.
10. On the **Manage Domain Values** page, in the **Domain** list, click **Gender** and view the new values that have been discovered.
11. In the **Type** column for the value **W**, click the symbol for **Invalid**, and in the **Correct To** column, type **F**. Then clear the **Show Only New** checkbox and note that **W** now appears as an invalid value under the **F** leading value.
12. On the **Manage Domain Values** page, in the **Domain** list, click **Country/Region** and view the new values that have been discovered. Then clear the **Show Only New** checkbox to show all values for this domain.
13. In the list of country values, find the **United States** leading value and click to select it. Find the new **America** value, press Ctrl, and then click **Set selected domain values as synonyms**. Note that, under the **United States** leading value, **America** is indented as a valid synonym value.

14. Repeat the previous step to designate **Great Britain** as a valid synonym value for the **United Kingdom** leading value.
15. Click **Finish**. When prompted to publish the knowledge base, click **Publish**, and when notified that publication was successful, click **OK**.
16. Keep SQL Server Data Quality Services open for the next exercise.

Results: After this exercise, you should have created a knowledge base and performed knowledge discovery.

Exercise 2: Using a DQS Project to Cleanse Data

▶ Task 1: Create a Data Quality Project

1. In SQL Server Data Quality Services, in the **Data Quality Projects** section, click **New Data Quality Project**. Then enter the following details and click **Next**:
   - **Name**: Cleanse Customer Data
   - **Description**: Apply Customer KB to customer data
   - **Use knowledge base**: Customer KB
   - **Select Activity**: Cleansing
2. On the **Map** page, in the **Data Source** list select **SQL Server**, in the **Database** list select **InternetSales**, and in the **Table/View** list select **Customers**. Then in the **Mappings** table, select the following mappings, and click **Next**:

Source Column	Domain
CountryRegionCode (nvarchar)	Country/Region (two-letter leading)
CountryRegionName (nvarchar)	Country/Region
StateProvinceName (nvarchar)	State
Gender (nvarchar)	Gender

3. On the **Cleanse** page, click **Start**. Then wait for the cleansing process to finish.
4. When cleansing is complete, review the source statistics in the **Profiler** pane, and click **Next**.
5. On the **Manage and View Results** page, select the **Country/Region** domain, and on the **Suggested** tab, note that DQS has found the value **Astralia**, which is likely to be a typographical error, and suggested it be corrected to **Australia**.
6. Click the **Approve** option to accept the suggested correction, and then click the **Corrected** tab to see all the corrections that have been made for the **Country/Region** domain.
7. In the list of domains, click **Country/Region (two-letter leading)** and on the **Corrected** tab, note the corrections that have been made for this domain.
8. In the list of domains, click **Gender** and on the **Corrected** tab, note the corrections that have been made for this domain.

9. In the list of domains, click **State** and on the **New** tab, note the new values that have been found for this domain.
10. Click the **Approve** option for each new value, and then click the **Corrected** tab to see all the corrections that have been made for the **State** domain.
11. Click **Next**.
12. On the **Export** page, view the output data, which contains:
    - An **Output** column for each column in the source that was not mapped to a domain, containing the original source value.
    - A **Source** column for each domain in the knowledge base containing the original source value.
    - An **Output** column for each domain containing the output value.
    - A **Reason** column for each domain containing the reason for the output value.
    - A **Confidence** column for each domain indicating the confidence level (0 to 1) for any corrections to the original value.
    - A **Status** column for each domain indicating the status of the output value.
13. In the **Export Cleansing Results** section, in the **Destination Type** list, select **Excel File**. Then in the **Excel file name** box, enter **D:\Labfiles\Lab09\Starter\CleansedCustomers.xls**.
14. Ensure that **Data and Cleansing Info** is selected, and click **Export**.
15. When you are notified that the file download is complete, click **Close**, and then click **Finish,** and close SQL Server Data Quality Services.
16. In the D:\Labfiles\Lab09\Starter folder, double-click **CleansedCustomers.xls** to open it with Excel. Then view the output from the cleansing process, and close Excel.

Results: After this exercise, you should have used a DQS project to cleanse data and export it as an Excel workbook.

Exercise 3: Using DQS in an SSIS Package

▶ Task 1: Add a DQS Cleansing Transformation to a Data Flow

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab09\Starter folder.
2. In Solution Explorer, in the **SSIS Packages** folder, double-click **Extract Internet Sales Data.dtsx**. Then, on the **SSIS** menu, click **SSIS Toolbox** to display the toolbox.
3. On the control flow surface, double-click the **Extract Customers** task to view its data flow.
4. In the SSIS Toolbox pane, in the **Other Transforms** section, double-click **DQS Cleansing**. Then, on the data flow surface, right-click **DQS Cleansing**, click **Rename**, and change the name to **Cleanse Customer Data**.
5. Right-click the data flow path between **Customers** and **Staging DB**, and click **Delete**. Then reposition **Cleanse Customer Data** below **Customers**, click **Customers**, and drag the data flow arrow to **Cleanse Customer Data**.

6. Double-click **Cleanse Customer Data**, and in the **DQS Cleansing Transformation Editor** dialog box, configure the following settings. Then click **OK**:
   - On the **Connection Manager** tab, next to the **Data quality connection manager** drop-down list, click **New**. Then in the **DQS Cleansing Connection Manager** dialog box, in the **Server Name** box, type **MIA-SQL**, and click **OK**.
   - On the **Connection Manager** tab, in the **Data Quality Knowledge Base** drop-down list, select **Customer KB**.
   - On the **Mapping** tab, check the checkboxes for the **Gender**, **StateProvinceName**, **CountryRegionCode**, and **CountryRegionName** input columns. Then create the following mappings with the default source, output, and status alias values:

Input Column	Domain
Gender	Gender
StateProvinceName	State
CountryRegionCode	Country/Region (two-letter leading)
CountryRegionName	Country/Region

   - On the **Advanced** tab, ensure that **Standardize output** is selected.
7. Click **Cleanse Customer Data**, and drag the data flow arrow to **Staging DB**.
8. Double-click **Staging DB**, and in the **OLE DB Destination Editor** dialog box, on the **Mappings** tab, change the current column mappings for the following destination columns. Then click **OK**:

Input Column	Destination Column
Gender_Output	Gender
StateProvinceName_Output	StateProvinceName
CountryRegionCode_Output	CountryRegionCode
CountryRegionName_Output	CountryRegionName

▶ Task 2: Test the Package

1. With the **Extract Customers** data flow displayed in the designer, on the **Debug** menu, click **Start Debugging** and observe the package as it executes, noting the number of rows processed by the **Cleanse Customer Data** transformation. Execution may take several minutes.
2. When execution is complete, on the **Debug** menu, click **Stop Debugging**. Then close Visual Studio, saving the solution files if prompted.

Results: After this exercise, you should have created and tested an SSIS package that cleanses data.

Lab B: Deduplicating Data

Exercise 1: Creating a Matching Policy

▶ Task 1: Prepare the Lab Environment

1. Complete the previous lab in this module.
2. In the D:\Labfiles\Lab09\Starter folder, right-click **LabB.cmd** and click **Run as administrator**.
3. Click **Yes** when prompted to confirm you want to run the command file, and wait for the script to finish.

▶ Task 2: Create a Matching Policy

1. On the task bar, click **SQL Server 2014 Data Quality Client**.
2. When prompted, enter the server name **MIA-SQL**, and click **Connect**.
3. In the **Knowledge Base Management** section, under **Recent knowledge base**, click **Customer KB**, and then click **Matching Policy**.
4. On the **Map** page, in the **Data Source** drop-down list, select **Excel File**. Then in the **Excel File** box, click **Browse** and select **Sample Staged Data.xls** in the D:\Labfiles\Lab09\Starter folder.
5. In the **Worksheet** drop-down list, ensure that **Sheet1$** and **Use first row as header** are selected.
6. In the **Mappings** table, in the first **Source Column** row, select **FirstName (String)**. Click **Create a domain**, and create a domain using the following properties, and click **OK**:
   - **Domain Name**: FirstName
   - **Description**: First Name
   - **Data Type**: String
   - **Use Leading Values**: Selected
   - **Normalize String**: Selected
   - **Format Output to**: None
   - **Language**: English
   - **Enable Speller**: Selected
   - **Disable Syntax Error Algorithms**: Not selected
7. In the **Mappings** table, in the second **Source Column** row, select **LastName (String)**. Then click the **Create a domain** button, create a domain using the following properties, and click **OK**:
   - **Domain Name**: LastName
   - **Description**: Last Name
   - **Data Type**: String
   - **Use Leading Values**: Selected
   - **Normalize String**: Selected
   - **Format Output to**: None

- **Language**: English
- **Enable Speller**: Selected
- **Disable Syntax Error Algorithms**: Not selected

8. In the **Mappings** table, in the third **Source Column** row, select **AddressLine1 (String)**. Then click the **Create a domain** button, create a domain using the following properties, and click **OK**:
   - **Domain Name**: AddressLine1
   - **Description**: First line of address
   - **Data Type**: String
   - **Use Leading Values**: Selected
   - **Normalize String**: Selected
   - **Format Output to**: None
   - **Language**: English
   - **Enable Speller**: Selected
   - **Disable Syntax Error Algorithms**: Not selected
9. In the **Mappings** table, in the fourth **Source Column** row, select **City (String)**. Then click the **Create a domain** button, create a domain using the following properties, and click **OK**:
   - **Domain Name**: City
   - **Description**: City
   - **Data Type**: String
   - **Use Leading Values**: Selected
   - **Normalize String**: Selected
   - **Format Output to**: None
   - **Language**: English
   - **Enable Speller**: Selected
   - **Disable Syntax Error Algorithms**: Not selected
10. In the **Mappings** table, in the fifth **Source Column** row, select **EmailAddress (String)**. Then click the **Create a domain** button, create a domain using the following properties, and click OK:
    - **Domain Name**: EmailAddress
    - **Description**: Email address
    - **Data Type**: String
    - **Use Leading Values**: Selected
    - **Normalize String**: Selected
    - **Format Output to**: None
    - **Language**: English
    - **Enable Speller**: Selected
    - **Disable Syntax Error Algorithms**: Not selected

11. Use the **Add a column mapping** button to create the following mappings for existing domains. Then click **Next**:

Source Column	Domain
Gender (String)	Gender
StateProvinceName (String)	State
CountryRegionCode (String)	Country/Region (two-letter leading)
CountryRegionName (String)	Country/Region

12. On the **Matching Policy** page, click the **Create a matching rule** button. Then create a matching rule with the following details:
    - **Rule name**: Is Same Customer
    - **Description**: Checks for duplicate customer records
    - **Min. matching score**: 80

Domain	Similarity	Weight	Prerequisite
Country/Region	Exact		Selected
Gender	Exact	10	Unselected
City	Exact	20	Unselected
EmailAddress	Exact	30	Unselected
FirstName	Similar	10	Unselected
LastName	Similar	10	Unselected
AddressLine1	Similar	20	Unselected

13. Click **Next**, and on the **Matching Results** page, click **Start**. Then wait for the matching process to finish.
14. When the matching process has finished, review the matches found by DQS, noting that there are duplicate records for three customers.
15. Click **Finish**, and when prompted to publish the knowledge base, click **Publish**. When notified that the knowledge base has been published successfully, click **OK**.

Results: After this exercise, you should have created a matching policy and published the knowledge base.

Exercise 2: Using a DQS Project to Match Data

▶ Task 1: Create a Data Quality Project for Matching Data

1. In SQL Server Data Quality Services, in the **Data Quality Projects** section, click **New data quality project**. Then enter the following details and click **Next**.
   - **Name**: Deduplicate Customers
   - **Description**: Identify customer matches
   - **Use knowledge base**: Customer KB
   - **Select Activity**: Matching
2. On the **Map** page in the **Data Source** list select **SQL Server**, in the **Database** list select **Staging**, and in the **Table/View** list, select **Customers**. Then in the **Mappings** table, add the following mappings, and click **Next**:

Source Column	Domain
FirstName (nvarchar)	FirstName
LastName (nvarchar)	LastName
Gender (nvarchar)	Gender
AddressLine1 (nvarchar)	AddressLine1
City (nvarchar)	City
CountryRegionName (nvarchar)	Country/Region
EmailAddress (nvarchar)	EmailAddress

3. On the **Matching** page, click **Start** then wait for the matching process to finish.
4. When the matching process has completed, review the matches that have been found and then click **Next**.
5. On the **Export** page, in the **Destination Type** drop-down list, select **Excel File**. Then specify the following content to export settings:
   - **Matching Results**: D:\Labfiles\Lab09\Starter\Matches.xls
   - **Survivorship Results**: D:\Labfiles\Lab09\Starter\Survivors.xls
6. In the **Survivorship Rule** options, select **Most complete record**, and click **Export**.
7. When the file download is complete, in the **Matching Export** dialog box, click **Close**, and then click **Finish** and close SQL Server Data Quality Services.

▶ Task 2: Review and Apply Matching Results

1. In the D:\Labfiles\Lab09\Starter folder, double-click **Matches.xls** to open the file in Excel.
2. Note that the matching process found a match with a score of 90 for the following customer records:
   - CustomerBusinessKey: 29484 (Rob Turner)
   - CustomerBusinessKey: 29261 (Robert Turner)
3. In the D:\Labfiles\Lab09\Starter folder, double-click **Survivors.xls** to open the file in Excel.
4. Note that the survivors file contains all the records that should survive deduplication based on the matches found. It contains the record for customer 29261 (Robert Turner), but not for 29484 (Rob Turner).
5. Close all instances of Excel without saving any changes.
6. In the D:\Labfiles\Lab09\Starter folder, double-click **Fix Duplicates.sql** to open the file in SQL Server Management Studio. When prompted, connect to the **MIA-SQL** instance of the database engine by using Windows authentication.
7. Review the Transact-SQL code and note that it performs the following tasks:
   - Updates the **InternetSales** table so that all sales currently associated with the duplicate customer record become associated with the surviving customer record.
   - Deletes the duplicate customer record.
8. Click **Execute**.
9. When the query has completed, close SQL Server Management Studio without saving any changes.

Results: After this exercise, you should have deduplicated data using a matching project and updated data in your database to reflect these changes.

Module 10: Master Data Services

Lab: Implementing Master Data Services

Exercise 1: Creating a Master Data Services Model

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Labfiles\Lab10\Starter folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. Click **Yes** when prompted to confirm that you want to run the command file, and then wait for the script to finish.

▶ Task 2: Create a Model

1. Start Internet Explorer, and navigate to **http://localhost:81/MDS**.
2. On the Master Data Services home page, click **System Administration**.
3. On the **Model View** page, on the **Manage** menu, click **Models**.
4. On the **Model Maintenance** page, click **Add model**.
5. On the **Add Model** page, in the **Model name** box, type **Product Catalog**, clear the **Create entity with same name as model** check box, and then click **Save model**.

▶ Task 3: Create Entities

1. On the **Model Maintenance** page, on the **Manage** menu, click **Entities**.
2. On the **Entity Maintenance** page, in the **Model** drop-down list, select **Product Catalog**, and then click **Add entity**.
3. On the **Add Entity** page, in the **Entity name** box, type **Product**.
4. In the **Enable explicit hierarchies and collections** drop-down list, click **No**, and then click **Save entity**.
5. On the **Entity Maintenance** page, in the **Model** drop-down list, ensure that **Product Catalog** is selected, and then click **Add entity**.
6. On the **Add Entity** page, in the **Entity** name box, type **Product Subcategory**.
7. In the **Enable explicit hierarchies and collections** drop-down list, click **No**, and then click **Save entity**.

▶ Task 4: Create Attributes

1. On the **Entity Maintenance** page, in the **Entity** table, click **Product**, and then click **Edit selected entity**.
2. On the **Edit Entity: Product** page, in the **Leaf member attributes** area, click **Add leaf attribute**.
3. On the **Entity: Product Add Attribute** page, select the **Domain-based** option.
4. In the **Name** box, type **ProductSubcategoryKey**, in the **Entity** list, select **Product Subcategory**, and then click **Save attribute**.
5. On the **Edit Entity: Product** page, in the **Leaf member attributes** area, click **Add leaf attribute**.
6. On the **Entity: Product Add Attribute** page, ensure that the **Free-form** option is selected.

7. In the **Name** box, type **Description**, in the **Length** box, type **400**, and then **click Save attribute**.
8. On the **Edit Entity: Product** page, in the **Leaf member attributes** area, click **Add leaf attribute**.
9. On the **Entity: Product Add Attribute** page, ensure that the **Free-form** option is selected.
10. In the **Name** box, type **ListPrice**, in the **Data type** list, select **Number**, in the **Decimals** box, type **4**, in the **Input mask** list, select **(####)**, and then click **Save attribute**.
11. On the **Edit Entity: Product** page, click **Save Entity**.

▶ Task 5: Add Members

1. In Internet Explorer, click the **Microsoft SQL Server 2014** logo to return to the Master Data Manager home page.
2. Click **Explorer**.
3. On the **Entities** menu, click **Product Subcategory**. If the **Entities** menu does not appear, click any other menu and then click the **Entities** menu again.
4. Click **Add Member**, and in the **Details** pane, enter the following attribute values:

- **Name**: Mountain Bikes
- **Code**: 1

5. In the **Annotations** box, type **New subcategory**. Then click **OK**.
6. Repeat the previous two steps to add the following **Product Subcategory** members:

Name	Code
Chains	7
Gloves	20
Helmets	31

7. On the **Entities** menu, click **Product**. If the **Entities** menu does not appear, click any other menu and then click the **Entities** menu again.
8. Click **Add Member**, and in the **Details** pane, enter the following attribute values:
    - **Name:** Mountain-100 Silver, 42
    - **Code:** 345
    - **ProductSubcategoryKey:** 1
    - **Description:** Top of the line competition mountain bike
    - **ListPrice:** 3399.99
9. In the **Annotations** box, type **New product**. Then click **OK**.

10. Repeat the previous two steps to add the following **Product** members:

Name	Code	ProductSubcategoryKey	Description	ListPrice
Chain	559	7	Superior chain	20.24
Full-Finger Gloves, S	468	20	Synthetic gloves	37.99
Sport-100 Helmet, Red	214	31	Universal fit helmet	34.99

11. Click the **Microsoft SQL Server 2014** logo to return to the Master Data Manager home page. Then close Internet Explorer.

Results: After this exercise, you should have a Master Data Services model named **Product Catalog** that contains **Product** and **ProductSubcategory** entities. Each of these entities should contain four members.

Exercise 2: Using the Master Data Services Add-in for Excel

▶ Task 1: Add Free-Form Attributes to an Entity

1. Start Microsoft® Excel® and create a new blank document.
2. On the **File** tab, click **Options**. Then in the **Excel Options** dialog box, on the **Add-Ins** tab, select **COM Add-ins** and click **Go**.
3. In the **COM Add-Ins** dialog box, if **Master Data Services Add-In for Excel** is not selected, select it. Then click **OK**.
4. In Excel, on the ribbon, click the **Master Data** tab.
5. On the ribbon **Master Data** tab, in the **Connect and Load** section, click the drop-down arrow under **Connect** and click **Manage Connections**.
6. In the **Manage Connections** dialog box, click **New**, and in the **Add New Connection** dialog box, enter the description **Local MDS Server** and the MDS server address **http://localhost:81/mds**, and click **OK**. Then click **Close**.
7. On the ribbon, in the **Connect and Load** area, click **Connect**, and then click **Local MDS Server**.
8. In the **Master Data Explorer** pane, in the **Model** list, select **Product Catalog**, in the list of entities, click **Product**, and then on the ribbon, in the **Connect and Load** area, click **Refresh**. The **Product** members you created in the previous exercise are displayed in a worksheet named **Product**.
9. In cell **I2** (to the right of the **ListPrice** attribute header), enter **StandardCost**. Then click cell **I2** (in which you just entered **StandardCost**), and on the ribbon, in the **Build Model** area, click **Attribute Properties**.
10. In the **Attribute Properties** dialog box, in the **Attribute type** list, select **Number**, in the **Decimal places** box, type **4**, and then click **OK**. This creates a new free-form attribute named **StandardCost** and publishes it to the model.

11. Repeat the previous two steps to create the following free-form attributes:

Cell	Attribute Name	Attribute Properties
J2	Color	Text (maximum length: 15)
K2	SafetyStockLevel	Number (0 decimal places)
L2	ReorderPoint	Number (0 decimal places)
M2	Size	Text (maximum length: 50)
N2	Weight	Number (4 decimal places)
O2	DaysToManufacture	Number (0 decimal places)
P2	ModelName	Text (maximum length: 500)

12. In cell **I3** (the **StandardCost** value for the Sport-100 Helmet, Red product), enter **13.0863**.
13. In cell **J3** (the **Color** value for the Sport-100 Helmet, Red product), enter **Red**.
14. In cell **K3** (the **SafetyStockLevel** value for the Sport-100 Helmet, Red product), enter **3**.
15. In cell **L3** (the **ReorderPoint** value for the Sport-100 Helmet, Red product), enter **4**.
16. In cell **M3** (the **Size** value for the Sport-100 Helmet, Red product), enter **U**.
17. In cell **N3** (the **Weight** value for the Sport-100 Helmet, Red product), enter **0.2000**.
18. In cell **O3** (the **DaysToManufacture** value for the Sport-100 Helmet, Red product), enter **5**.
19. In cell **P3** (the **ModelName** value for the Sport-100 Helmet, Red product), enter **Sport-100**.
20. Enter the following attribute values for the remaining products:
    - Mountain-100 Silver, 42
        - StandardCost: 1912.1544
        - Color: Silver
        - SafetyStockLevel: 100
        - ReorderPoint: 75
        - Size: L
        - Weight: 20.77
        - DaysToManufacture: 4
        - ModelName: Mountain-100
    - Full-Finger Gloves, S
        - StandardCost: 15.6709
        - Color: Black
        - SafetyStockLevel: 4
        - ReorderPoint: 3

- Size: S
- Weight: 0.2000
- DaysToManufacture: 0
- ModelName: Full-Finger Gloves

- Chain
  - StandardCost: 8.9866
  - Color: Silver
  - SafetyStockLevel: 500
  - ReorderPoint: 375
  - Size: (leave blank)
  - Weight: (leave blank)
  - DaysToManufacture: 1
  - ModelName: Chain

21. On the ribbon, in the **Publish and Validate** area, click **Publish**. Then, in the **Publish and Annotate** dialog box, enter the annotation **Added product details**, and click **Publish**.

▶ Task 2: Create an Entity for a Domain-Based Attribute

1. On the ribbon, in the **Connect and Load** area, click **Connect**, and then click **Local MDS Server**.
2. In the **Master Data Explorer** pane, in the **Model** list, select **Product Catalog**, in the list of entities, click **Product Subcategory**, and then on the ribbon, in the **Connect and Load** area, click **Refresh**. The **Product Subcategory** members you created in the previous exercise are displayed in a worksheet named **Product Subcategory**.
3. In cell **F2** (to the right of the **Code** attribute header), enter **ProductCategoryKey**.
4. In cell **F3**, enter the value **1** as the **ProductCategoryKey** for the Mountain Bikes subcategory.
5. Enter the following **ProductCategoryKey** values for the remaining subcategories:
   - Gloves: 3
   - Helmets: 4
   - Chains: 2
6. Click cell **F2** (the header for the **ProductCategoryKey** attribute), and on the ribbon, in the **Build Model** area, click **Attribute Properties**.
7. In the **Attribute Properties** dialog box, in the **Attribute type** list, select **Constrained list (Domain-based)**. In the **Populate the attribute with values from** list, ensure that **the selected column** is selected, in the **New entity name** box, type **Product Category**, and then click **OK**.
8. On the ribbon, in the **Connect and Load** area, click **Connect**, and then click **Local MDS Server**.
9. In the **Master Data Explorer** pane, in the **Model** list, select **Product Catalog**, in the list of entities, click **Product Category**, and then on the ribbon, in the **Connect and Load** area, click **Refresh**. The newly-created **Product Category** entity members are displayed in a worksheet named **Product Category**.
10. In cell **D3** (the **Name** value for the first member, which currently contains the value 1), enter **Bikes**.

11. Enter the following **Name** values for the remaining categories:
    - **Components** (2)
    - **Clothing** (3)
    - **Accessories**: 4
12. On the ribbon, in the **Publish and Validate** area, click **Publish**. Then, in the **Publish and Annotate** dialog box, enter the annotation **Added product categories**, and click **Publish**.
13. Click the tab for the **Product Subcategory** worksheet, and on the ribbon, in the **Connect and Load** area, click **Refresh**. Note that the **Product Category** member names are displayed in the **ProductCategoryKey** column.
14. Minimize Excel. You will return to it in a later exercise.

Results: After this exercise, you should have a master data model that contains **Product**, **ProductSubcategory**, and **ProductCategory** entities that contain data entered in Excel.

Exercise 3: Enforcing Business Rules

▶ Task 1: Create a Rule for the ListPrice Attribute

1. Start Explorer, and navigate to **http://localhost:81/MDS**.
2. On the Master Data Services home page, click **System Administration**. Then on the **Manage** menu click **Business Rules**.
3. On the **Business Rule Maintenance** page, in the **Model** list, ensure that **Product Catalog** is selected, in the **Entity** list, ensure that **Product** is selected, in the **Member Type** list, select **Leaf**, and then in the **Attribute** list, ensure **All** is selected.
4. Click **Add business rule**, double-click the **Name** column, type **Validate Price**, and then press Enter.
5. Double-click in **Description** column, type **Check that prices are non-zero**, and then press Enter.
6. Click **Edit selected business rule**, and then in the **Version** list, ensure that **VERSION_1** is selected.
7. Under **Components**, expand **Actions**, and then in the **Validation** list, drag **must be greater than** onto the **Actions** node in the THEN section of the expression editor.
8. In the **Attributes** list, drag **ListPrice** onto **Select attribute** under **Edit Action**.
9. In the **Attribute value** box, ensure **0** is displayed, and then click **Save item**. In **Actions**, under THEN, the entry should now read 'ListPrice must be greater than 0.0000'.
10. On the **Edit Business Rule: Validate Price** page, click **Back**.

▶ Task 2: Create a Rule for the SafetyStockLevel Attribute

1. On the **Business Rule Maintenance** page, click **Add business rule**, double-click in the **Name** column, type **Validate SafetyStockLevel**, and then press Enter.
2. Double-click in the **Description** column, type **Check that safety stock level is greater than reorder point for products that take a day or more to manufacture**, and then press Enter.
3. Click **Edit selected business rule**, and then in the **Version** list, ensure that **VERSION_1** is selected.
4. Under **Components**, expand **Conditions**, and then in the **Value** comparison list, drag **is greater than** onto the **Conditions** node in the IF section of the expression editor.

5. In the **Attributes** list, drag **DaysToManufacture** onto **Select attribute** under **Edit Condition**.
6. In the **Attribute value** box, ensure **0** is displayed, and then click **Save item**.
7. Under **Components**, expand **Actions**, and in the **Validation** list, drag **must be greater than** onto the **Actions** node under THEN.
8. In the **Attributes** list, drag **SafetyStockLevel** onto **Select attribute**, directly under **Edit Action**. Then in the **Edit Action** section, under **must be greater than**, select **Attribute** and drag the **ReorderPoint** attribute from the **Attributes** list to **Select attribute**.
9. Click **Save item**. The IF section of the expression should read 'DaysToManufacture is greater than 0', and the THEN section should read 'SafetyStockLevel must be greater than ReorderPoint'.
10. On the **Edit Business Rule: Validate SafetyStockLevel** page, click **Back**.

▶ Task 3: Publish Business Rules and Validate Data

1. On the **Business Rule Maintenance** page, click **Publish business rules**, and then in the **Message from webpage** dialog box, click **OK**.
2. In the **Status** column, check that the value displayed for both rules is **Active**.
3. Click the **Microsoft® SQL Server® 2014** logo to return to the home page, and then click **Explorer**.
4. In the **Entities** menu, click **Product** to ensure that you are viewing the **Product** entity members. If the **Entities** menu does not appear, click any other menu and then click the **Entities** menu again.
5. Click **Apply Rules**, and note that the **Sport-100 Helmet, Red** member has a red exclamation mark, indicating that a business rule has been violated.
6. Click the **Sport-100 Helmet, Red** member, and at the bottom of the **Details** tab, note the validation error.
7. In the details tab, change the **SafetyStockLevel** attribute for the Sport-100 Helmet, Red member to **5** and click **OK**. Note that the validation error and red exclamation mark disappear.
8. Close Internet Explorer.
9. Maximize Excel, and click the **Product** worksheet tab. Then on the **Master Data** tab of the ribbon, in the **Connect and Load** area, click **Refresh**.
10. On the ribbon, on the **Master Data** tab, in the **Publish and Validate** area, click **Show Status** to display the **$ValidationStatus$** and **$InputStatus$** columns.
11. In the **Publish and Validate** area, click **Apply Rules**. Note that validation succeeded for all members.
12. Change the value in the **ListPrice** column for the **Chain** product to **0.00**.
13. On the ribbon, in the **Publish and Validate** section, click **Publish** and in the **Publish and Annotate** dialog box, click **Publish**. Note that validation failed for this member.
14. Move the mouse pointer over the **$ValidationStatus$** entry for the **Chain** member (which should currently say **Validation Failed**), and note the comment that is displayed.
15. Change the value in the **ListPrice** column for the **Chain** product back to **20.24**.
16. On the ribbon, in the **Publish and Validate** section, click **Publish,** and in the **Publish and Annotate** dialog box, click **Publish**. Note that validation succeeds this time.
17. Close Excel without saving the workbook.

Results: After this exercise, you should have a master data model that includes business rules to validate product data.

Exercise 4: Loading Data into a Model

▶ Task 1: Load Data into the Model

1. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance using Windows authentication.
2. Open the **Import Products into MDS.sql** file in the D:\Labfiles\Lab10\Starter folder.
3. Review the Transact-SQL code in this script, noting that it performs the following tasks:
   - Generates a unique batch ID.
   - Inserts data into the **stg.Product_Category_Leaf** staging table from the **ProductCategory** table in the **Products** database, specifying an **ImportType** value of 2, an **ImportStatus_ID** value of 0, and a **BatchTag** value based on the batch ID generated previously.
   - Inserts data into the **stg.Product_Subcategory_Leaf** staging table from the **ProductSubcategory** table in the **Products** database, specifying an **ImportType** value of 2, an **ImportStatus**_ID value of 0, and a **BatchTag** value based on the batch ID generated previously.
   - Inserts data into the **stg.Product_Leaf** staging table from the **Product** table in the **Products** database, specifying an **ImportType** value of 2, an **ImportStatus**_ID value of 0, and a **BatchTag** value based on the batch ID generated previously.
   - Executes the **stg.udp_Product_Category_Leaf** stored procedure to start processing the staged **Product Category** members with the batch tag specified previously.
   - Executes the **stg.udp_Product_Subcategory_Leaf** stored procedure to start processing the staged **Product Subcategory** members with the batch tag specified previously.
   - Executes the **stg.udp_Product_Leaf** stored procedure to start processing the staged **Product** members with the batch tag specified previously.
4. Click **Execute** and note the numbers of rows affected by the statements.
5. Close SQL Server Management Studio without saving any changes.

▶ Task 2: Check the Status of the Data Load

1. Start Internet Explorer, and navigate to **http://localhost:81/MDS**.
2. On the Master Data Services home page, click **Integration Management**, and then click **Import Data**.
3. Review the information on the **Import Data** page. In **Status** column, check that the value displayed is **Completed**, and in the **Errors** column, check that the value displayed is **0**.

▶ Task 3: Validate Imported Members

1. Click the **Microsoft SQL Server 2014** logo to return to the home page. Then click **Explorer**.
2. On the **Entities** menu, click **Product** to view the **Product** members. If the **Entities** menu does not appear, click any other menu and then click the **Entities** menu again.
3. Click **Apply Rules** to apply business rules to the imported members.
4. Click **Filter**, and then click **Add Criteria**.
5. In the **Attribute** column, select **[Validation Status]**, in the **Operator** column, ensure that **Is equal to** is selected, and in the **Criteria** column, select **Validation Failed**. Then click **Apply**. The list is filtered to show only invalid members.

6. Click each member and review the validation errors at the bottom of the **Details** pane. Some members have failed because they have no **ListPrice** value. Resolve this problem by setting the **ListPrice** of each invalid member to **1431.50**.
7. Click **Filter**, and then click **Clear Filter**.
8. Click the **Microsoft SQL Server 2014** logo to return to the home page.
9. Keep Internet Explorer open. You will return to it in the next exercise.

Results: After this exercise, you should have loaded members into the Master Data Services model.

Exercise 5: Consuming Master Data Services Data

▶ Task 1: Create Subscription Views

1. In Internet Explorer, on the Master Data Manager home page, click **Integration Management**. Then click **Create Views**.
2. On the **Subscription Views** page, click **Add subscription view**.
3. On the **Subscription Views** page, in the **Create Subscription View** section, in the **Subscription view name** box, type **Products**.
4. In the **Model** list, select **Product Catalog**.
5. In the **Version** list, select **VERSION_1**.
6. In the **Entity** list, select **Product**.
7. In the **Format** list, select **Leaf members**.
8. Click **Save**.
9. Repeat steps 2 to 8 to create the subscription views named **ProductSubcategories** and **ProductCategories** for the **Product Subcategory** and **Product Category** leaf members respectively.
10. Close internet Explorer.

▶ Task 2: Query a Subscription View by Using Transact-SQL

1. In the D:\Labfiles\Lab10\Starter folder, double-click **Query Subscription Views.sql**. When prompted, connect to the **localhost** database instance by using Windows authentication.
2. Review the Transact-SQL Statement, click **Execute**, and then review the results. Then close SQL Server Management Studio without saving any files.

Results: After this exercise, you should have three subscription views that you can use to query product data from Master Data Services.

Module 11: Extending SQL Server Integration Services

Lab: Using Custom Scripts

Exercise 1: Using a Script Task

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to 20463C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Labfiles\Lab11\Starter folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. Click **Yes** when prompted to confirm that you want to run the command file, and then wait for the script to finish.

▶ Task 2: Add a Script Task to a Control Flow

1. Start Visual Studio and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab11\Starter folder.
2. In Solution Explorer, double-click the **Load DW.dtsx** SSIS package to open it in the designer.
3. Review the control flow, noting that it runs multiple packages to load data from the staging database into the data warehouse. Then it uses an Execute SQL task named **Get record counts and truncate Staging tables** to determine the number of rows that were processed before truncating the staging tables, ready for the next refresh cycle.
4. On the **SSIS** menu, click **Variables**, and note the variables in this package. The **Get record counts and truncate Staging tables** task assigns the row counts in the staging tables to these variables.
5. In the SSIS Toolbox, drag a **Script Task** to the control flow and drop it under the **Get record counts and truncate Staging tables** task. Then right-click **Script Task**, click **Rename**, and change the name to **Log Rowcounts**.
6. Click the **Get record counts and truncate Staging tables** task, and drag the green precedence constraint to the **Log Rowcounts** task.

▶ Task 3: Enable Logging for the Script Task

1. On the **SSIS** menu, click **Logging**, then on the **Providers and Logs** tab, in the **Provider type** list, select **SSIS log provider for Windows Event Log** and click **Add**.
2. In the **Containers** tree, select the **Log Rowcounts** checkbox, and in the **Providers and Logs** tab select the **SSIS log provider for Windows Event Log** checkbox. Then on the **Details** tab, select the **ScriptTaskLogEntry** checkbox, and click **OK**.

▶ Task 4: Configure the Script Task

1. On the control flow surface, double-click the **Log Rowcounts** script task.
2. In the **Script Task Editor** dialog box, on the **Script** page, in the **ReadOnlyVariables** box, click the ellipsis (**…**) button. Then in the **Select Variables** dialog box, select the following variables and click **OK**:
   - User::CustomerCount
   - User::EmployeeCount
   - User::InternetSalesCount
   - User::PaymentCount

- User::ResellerCount
- User::ResellerSalesCount

3. Click **Edit Script**, and wait for the Visual Studio VstaProjects editor to open. If a message that the Visual C++ Language Manager Package did not load correctly is displayed, prompting you to continue to show this error, click **No**.
4. In the VstaProjects editor, in the **Main()** function, replace the comment **//TODO: Add your code here** with the following code (above the existing **Dts.TaskResult = (int)ScriptResults.Success** statement). You can copy and paste this code from **Script.txt** in the D:\Labfiles\Lab11\Starter folder.

```
String logEntry = "Data Warehouse records loaded (";
logEntry += Dts.Variables["User::CustomerCount"].Value.ToString() + " customers, ";
logEntry += Dts.Variables["User::ResellerCount"].Value.ToString() + " resellers, ";
logEntry += Dts.Variables["User::EmployeeCount"].Value.ToString() + " employees, ";
logEntry += Dts.Variables["User::PaymentCount"].Value.ToString() + " payments, ";
logEntry += Dts.Variables["User::InternetSalesCount"].Value.ToString() + " Internet
sales, and ";
logEntry += Dts.Variables["User::ResellerSalesCount"].Value.ToString() + " reseller
sales) ";
Dts.Log(logEntry, 999, null);
```

5. Save the script and close the **VstaProjects – Microsoft Visual Studio** window. Then in the Script Task Editor dialog box, click **OK**.

▶ Task 5: Test the Script

1. On the **Debug** menu, click **Start Debugging** and observe the control flow tab as the package executes, noting that each package executed opens in a new window. The entire load process can take a few minutes.
2. When execution is complete, on the **Debug** menu, click **Stop Debugging**. Then close Visual Studio, saving your work if prompted.
3. Right-click the Start button and click **Event Viewer**.
4. In Event Viewer, expand the **Windows Logs** folder and select the **Application** log. Then in the list of application events, select the first information event with a source of **SQLISPackage120** and read the information in the **General** tab.
5. Close Event Viewer.

Results: After this exercise, you should have an SSIS package that uses a script task to log row counts.

Module 12: Deploying and Configuring SSIS Packages

Lab: Deploying and Configuring SSIS Packages

Exercise 1: Creating an SSIS Catalog

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Labfiles\Lab12\Starter folder, right-click **Setup.cmd** and click **Run as administrator**.
3. Click **Yes** when prompted to confirm you want to run the command file, and wait for the script to finish.

▶ Task 2: Create the SSIS Catalog and a Folder

1. Start SQL Server Management Studio and connect to the **localhost** instance of the database engine using Windows authentication.
2. In Object Explorer, right-click **Integration Services Catalogs** and click **Create Catalog**.
3. In **Password**, type **Pa$$w0rd** and in **Retype Password** type **Pa$$w0rd**. Then click **OK**.
4. In Object Explorer, expand **Integration Services Catalogs**.
5. Right-click **SSISDB** and click **Create Folder**.
6. In the **Folder name** box, type **DW ETL**, in the **Folder description** box, type **Folder for the Adventure Works ETL SSIS Project**, and then click **OK**.

Results: After this exercise, you should have enabled CLR, creating the SSIS catalog, and a folder.

Exercise 2: Deploying an SSIS Project

▶ Task 1: Deploy an SSIS Project

1. Start Visual Studio, and open the **AdventureWorksETL.sln** solution in the D:\Labfiles\Lab12\Starter folder.
2. On the **Build** menu, click **Build Solution**.
3. When the build has succeeded, In the **Project** menu, click **Deploy**.
4. In the **Introduction** page of the Integration Services Deployment Wizard dialog box, click **Next**.
5. In the **Select Destination** page, enter **localhost** in the **Server name** box and in the **Path** box, browse to the **SSIDB\DW ETL** folder you created earlier. Then click **Next**.
6. On the **Review** page, click **Deploy**. Then, when deployment has completed, click **Close** and close Visual Studio.

Results: After this exercise, you should have deployed an SSIS project to a folder in your SSIS database.

Exercise 3: Creating Environments for an SSIS Solution

▶ Task 1: Create Environments

1. In SQL Server Management Studio, expand the **DW ETL** folder you created earlier, and expand its **Projects** folder. Note that the **AdventureWorksETL** project has been deployed.
2. Right-click the **Environments** folder, and click **Create Environment**. Then in the **Create Environment** dialog box, enter the environment name **Test**, and click **OK**.
3. Repeat the previous step to create a second environment named **Production**.

▶ Task 2: Create Variables

1. Expand the **Environments** folder to see the environments you have created, and then right-click the **Production** environment and click **Properties**.
2. In the **Environment Properties** dialog box, on the **Variables** tab, add a variable with the following settings:
   - **Name**: StgServer
   - **Type**: String
   - **Description**: Staging server
   - **Value**: MIA-SQL
   - **Sensitive**: No
3. Add a second variable with the following settings (making sure to include the trailing "\" in the value), and then click **OK**:
   - **Name**: FolderPath
   - **Type**: String
   - **Description**: Accounts folder
   - **Value**: D:\Accounts\
   - **Sensitive**: No
4. Right-click the **Test** environment and click **Properties**.
5. In the **Environment Properties** dialog box, on the **Variables** tab, add a variable with the following settings:
   - **Name**: StgServer
   - **Type**: String
   - **Description**: Staging server
   - **Value**: localhost
   - **Sensitive**: No
6. Add a second variable with the following settings (making sure to include the trailing "\" in the value), and then click **OK**:
   - **Name**: FolderPath
   - **Type**: String
   - **Description**: Accounts folder
   - **Value**: D:\TestAccts\

- **Sensitive**: No

▶ Task 3: Map Environment Variables

1. In the **Projects** folder, right-click **AdventureWorksETL** and click **Configure**.
2. In the **Configure – AdventureWorksETL** dialog box, on the **References** page, click **Add** and add the **Production** environment. Then click **Add** again and add the **Test** environment.
3. In the **Configure – AdventureWorksETL** dialog box, on the **Parameters** page, in the **Scope** list, select **AdventureWorksETL**.
4. On the **Parameters** tab, click the ellipses (**…**) button for the **AccountsFolder** parameter. In the **Set Parameter Value** dialog box, select **Use environment variable,** and click **FolderPath** in the list of variables, and click **OK**.
5. In the **Configure – AdventureWorksETL** dialog box, on the **Connection Managers** tab, select the **localhost Staging ADO NET** connection manager, and click the ellipses (**…**) button for the **ServerName** property and in the **Set Parameter Value** dialog box, select **Use environment variable** and click **StgServer** in the list of variables, and click **OK**.
6. In the **Configure – AdventureWorksETL** dialog box, on the **Connection Managers** tab, select the **localhost.Staging** connection manager and click the ellipses (**…**) button for the **ServerName** property, and in the **Set Parameter Value** dialog box, select **Use environment variable**, and click **StgServer** in the list of variables, and click **OK**.
7. In the **Configure – AdventureWorksETL** dialog box, click **OK**.

Results: After this exercise, you should have configured your project to use environments to pass values to package parameters.

Exercise 4: Running an SSIS Package in SQL Server Management Studio

▶ Task 1: Run a Package

1. In Object Explorer, expand the **AdventureWorksETL** project, and expand **Packages**.
2. Right-click **Extract Payment Data.dtsx** and click **Execute**.
3. Select **Environment** and select the **.\Test** environment.
4. Click **OK**.
5. When prompted to open overview report, wait 10 seconds to allow the package to run, and then click **Yes**.
6. In the **Overview** report, in the **Execution Information** table, note the environment that was used. In the **Parameters Used** table, note the values you configured with environment variables.

Results: After this exercise, you should have performed tests to verify that a value was passed from the environment to the package.

Exercise 5: Scheduling SSIS Packages with SQL Server Agent

▶ Task 1: Create a SQL Server Agent job

1. In Object Explorer, right-click **SQL Server Agent**, point to **New**, and click **Job**.
2. In the **New Job** dialog box, in the **Name** box, type **Extract Reseller Data**.
3. In the **New Job** dialog box, on the **Steps** page, click **New**.
4. In the **New Job Step** dialog box, in the **Step name** box, type **Run Extract Reseller Data Package**.
5. In the **Type** list, select **SQL Server Integration Services Package**.
6. In the **Package source** list, select **SSIS Catalog**.
7. In the **Server** box, type **MIA-SQL**.
8. Click the ellipses (**...**) button for the **Package** box, and in the Select an SSIS Package dialog box, expand **SSISDB**, expand **DW ETL**, expand **AdventureWorksETL**, select **Extract Reseller Data.dtsx**, and then click **OK**.
9. In the **New Job Step** dialog box, on the **Configuration** tab, select **Environment**, and select the **.\Production** environment.
10. In the **New Job Step** dialog box, on the **Advanced** page, in the **On success action** list, select **Quit the job reporting success**. Then click **OK**.
11. In the **New Job** dialog box, on the **Schedules** page, click **New**.
12. In the **Name** box, type **ETL Schedule**.
13. In the **Schedule** type list, select **One time**.
14. Ensure **Enabled** is selected.
15. In the **One-time occurrence** section, select the current day and enter a time that is two minutes later than the current time.
16. Click **OK** to close the **New Job Schedule window**, and then click **OK** to close the **New Job** window.
17. Wait for two minutes.
18. In Object Explorer, expand **SQL Server Agent**, right-click **Job Activity Monitor,** and click **View Job Activity**.
19. In the **Job Activity Monitor – localhost** window, note the **Status** and **Last Run Outcome** values for the **Extract Reseller Data** job. If the job has not yet completed, wait a minute and click **Refresh** until the job has completed successfully, and then click **Close**.

▶ Task 2: View the Integration Services Dashboard

1. In SQL Server, in Object Explorer, under **Integration Services Catalogs**, right-click **SSISDB**, point to **Reports**, point to **Standard Reports**, and click **Integration Services Dashboard**.
2. In the **Packages Detailed Information (Past 24 Hours)** list, notice that the most recent two package executions succeeded, and then click the **Overview** link for the first package in the list (which should be the **Extract Reseller Data.dtsx** package).
3. In the **Overview** report, in the **Execution Information** table, note the environment that was used, and in the **Parameters Used** table, note the values you configured with environment variables.
4. Click **View Messages** to view the messages that were logged during package execution.
5. Close SQL Server Management Studio.

Results: After this exercise, you should have created an SQL Server Agent job that automatically runs your SSIS package.

Module 13: Consuming Data in a Data Warehouse

Lab: Using a Data Warehouse

Exercise 1: Exploring an Enterprise BI Solution

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 20463C-MIA-DC and 20463C-MIA-SQL virtual machines are both running, and then log on to MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the D:\Labfiles\Lab13\Starter folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. Click **Yes** when prompted to confirm that you want to run the command file, and then wait for the script to finish.

▶ Task 2: View an Enterprise Data Model

1. Start Visual Studio and open the **AW Enterprise BI.sln** solution in the D:\Labfiles\Lab13\Starter folder. If the **Tabular model designer** dialog box is displayed, in the **Workspace server** list, specify **localhost\SQL2** and then click **OK**.
2. In Solution Explorer, in the **AWDataModel** project, double-click **Model.bim** to open it in the designer. If you are prompted to run a script on the Analysis Services server, click **Yes**.
3. Click each tab in the designer to view the tables in the data model. These are based on tables in the data warehouse.
4. On the **Model** menu, point to **Model View** and click **Diagram View**. This shows the relationships between the tables in the data model.
5. On the **Model** menu, point to **Model View** and click **Data View**. Then click the **Reseller Sales** tab (you may need to click the drop-down list at the end of the tabs).
6. Note that in the measure grid area (under the data), some aggregations have been defined as measures for the table.
7. Under the **Profit** column, right-click the **Reseller Margin** measure and click **Edit KPI Setting**. Then in the **Key Performance Indicator (KPI)** dialog box, note that this KPI compares the **Reseller Margin** measure to a target of 0.5, indicating that Adventure Works Cycles seeks to achieve a 50 percent margin on reseller sales. Then click **Cancel**.
8. On the **Project** menu, click **AWDataModel Properties**, and on the **Deployment** page note that the data model is configured to be deployed as a cube named **AdventureWorks** in a database named **AWOLAP**. Then click **Cancel**.
9. On the **Build** menu, click **Deploy AWDataModel**. If you are prompted for impersonation credentials, specify the user name **ADVENTUREWORKS\ServiceAcct**, the password **Pa$$w0rd**, and then click **OK**. Then, when deployment is complete, click **Close** and minimize Visual Studio.
10. Start Excel and create a new blank workbook.
11. On the **Data** tab, in the **Get External Data** area (which may be condensed into a drop-down list, depending on the screen resolution), in the **From Other Sources** list, click **From Analysis Services**.
12. In the Data Connection Wizard, on the **Connect to Database Server** page, enter the Server name **MIA-SQL\SQL2**, select **Use Windows Authentication**, and click **Next**.
13. On the **Select Database and Table** page, select the **AWOLAP** database and the **AdventureWorks** cube, and click **Next**.

14. On the **Save Data Connection File and Finish** page, click **Finish**. Then, in the **Import Data** dialog box, select **PivotTable Report**, ensure that **Existing worksheet** is selected, and click **OK**.
15. In the **PivotTable Fields** pane, under **Reseller Sales**, select **Reseller Revenue**, **Reseller Profit**, and **Reseller Margin**. The total for each measure is displayed in the PivotTable.
16. In the **PivotTable Fields** pane, under **Internet Sales**, select **Internet Revenue**, **Internet Profit**, and **Internet Margin**. The total for each measure is added to the PivotTable.
17. In the **PivotTable Fields** pane, expand **KPIs**, expand **Internet Margin**, and select **Status**. Then expand **Reseller Margin** and click **Status**. A status indicator for each KPI is shown in the PivotTable.
18. In the **PivotTable Fields** pane, under **Order Date**, select **Calendar Date**. The measures are shown for each year.
19. In the **PivotTable Fields** pane, under **Geography**, select **Location**. The measures under each year are shown for each sales territory.
20. In the PivotTable, expand the years and locations to drill down into the detailed results for specific time periods and locations.
21. When you have finished exploring the data, close Excel without saving the workbook.

▶ Task 3: View Enterprise Reports

1. Maximize Visual Studio, in which the **AW Enterprise BI.sln** solution should be open.
2. In Solution Explorer, in the **AWReports** project, in the **Shared Data Sources** folder, note that the project contains two shared data sources, **AWDataWarehouse.rds** and **AWOLAP.rds**. These provide connection details for the **AWDataWarehouse** SQL Server database and the **AWOLAP** Analysis Services database.
3. In Solution Explorer, in the **Reports** folder, double-click **Dashboard.rdl** to open it in the designer. Note that this report contains visualizations for various business metrics. The report uses a hidden parameter named **Year** to filter the data so that only the data for the latest year is shown.
4. In Solution Explorer, in the **Reports** folder, double-click **Monthly Sales Report.rdl** to open it in the designer. Note that this report provides details of sales performance for a specific month, which is determined by two parameters named **Year** and **Month**.
5. On the **Build** menu, click **Deploy AWReports**. Wait for deployment to complete, and then close Visual Studio.
6. Start Internet Explorer® and browse to **http://mia-sql/sites/adventureworks**. Note that the first time you browse to this site, it may take a few minutes to open.
7. In the **Adventure Works Portal** page, in the quick launch area on the left side of the page, click **Reports**.
8. In the **Reports** page, click **Dashboard**. Then wait for the report to load.
9. View the dashboard report, and expand the product categories to view KPIs for each subcategory. Note that you can hold the mouse over the KPI indicators to see the actual margin as a tooltip.
10. At the top of the page, click **Reports** to return to the **Reports** page.
11. In the **Reports** page, click **Monthly Sales Report**. Then wait for the report to load.
12. View the report, noting that you can change the **Year** and **Month** parameters to view the sales results for a different month.
13. In the **Actions** drop-down list, point to **Export** and click **Word**. Then, when prompted to open or save the report, click **Open**.

14. In Microsoft® Word®, click **Enable Editing**. Then view the report which has been exported as a Word document. When you have finished, close Word without saving the document.
15. In Internet Explorer, click **Reports** to return to the **Reports** page. Keep Internet Explorer open for the next exercise.

Results: After this exercise, you should have deployed an Analysis Services database and some Reporting Services reports in SharePoint Server.

Exercise 2: Exploring a Self-Service BI Solution

▶ Task 1: Use Report Builder to Create a Report

1. In Internet Explorer®, in the **Reports** page of the Adventure Works Portal SharePoint Server site, click the **Files** tab on the ribbon, and in the **New Document** drop-down list, click **Report Builder Report**. If you are prompted to open the application, click **Open**.
2. When you are prompted to run the Report Builder application, click **Run** and wait for the application to be downloaded and started.
3. In the **Getting Started** pane, ensure that **New Report** is selected and click **Table or Matrix Wizard**.
4. On the **Choose a dataset** page, ensure that **Create a dataset** is selected, and click **Next**.
5. On the **Choose a connection to a data source** page, click **Browse**. Browse to the **Reports\Data Sources** folder, select the **AWDataWarehouse.rsds** data source, and click **Open**. Then click **Next**.
6. On the **Design a query** page, expand **Tables**, expand **DimProduct**, and select **ProductName**, and then expand **FactInternetSales,** and select **Order Quantity**. Then expand the **Relationships** section and verify that a relationship has been detected based on the **ProductKey** field.
7. In the **Selected fields** list, in the **Aggregate** column for the **OrderQuantity** field, select **Sum**. Then verify that the **Grouped by** aggregate is automatically specified for the **ProductName** field.
8. Click **Edit as Text** to view the Transact-SQL code that has been generated by the query designer, and modify it to match the following code:

```
SELECT TOP 10 DimProduct.ProductName, SUM(FactInternetSales.OrderQuantity) AS
SalesCount
FROM DimProduct INNER JOIN FactInternetSales
ON DimProduct.ProductKey = FactInternetSales.ProductKey
GROUP BY DimProduct.ProductName
ORDER BY SalesCount DESC;
```

9. Click the **Run** icon to view the query results, and then click **Next**.
10. On the **Arrange fields** page, drag both fields to the **Values** area, and then click **Next**.
11. On the **Choose the layout** page, click **Next**.
12. On the **Choose a style** page, select any style and click **Finish**.
13. In the report designer, add the title **Top 10 Internet Sellers**. Then widen the columns as required so that the headers are fully visible.
14. Click **Run** to preview the report, and then click **Design** to return to design view, making any changes you think necessary to improve the report formatting.

15. When you are satisfied with the report, click the **Save** icon, browse to the **Reports** folder, and save the report as **Top 10 Sellers.rdl**.
16. Close Report Builder, and refresh the **Reports** page in Internet Explorer. Then click **Top 10 Sellers** to view your report.
17. When you have viewed the report, close Internet Explorer.

▶ Task 2: Use Excel to Create a Data Model

1. Start Excel and create a new blank workbook. Save the workbook as Analysis.xlsx in the D:\Labfiles\Lab13\Starter folder.
2. On the ribbon, click **File**. Then click **Options**.
3. In the **Excel Options** dialog box, on the **Add-Ins** page, in the **Manage** list, select **COM Add-Ins** and click **Go**.
4. In the **COM Add-Ins** dialog box, if the following add-ins are not selected, select them and click **OK**.
   - **Microsoft Office PowerPivot for Excel 2013**
   - **Microsoft Power Query for Excel**
5. On the **PowerPivot** tab, click **Manage**. The PowerPivot for Excel window opens.
6. In the PowerPivot for Excel window, in the **Get External Data** area (which may be condensed into a drop-down list, depending on the screen resolution), in the **From Database** list, click **From SQL Server**.
7. In the Table Import Wizard, on the **Connect to a Microsoft SQL Server Database** page, change the friendly connection name to **AWDataWarehouse** and enter the server name **MIA-SQL**. Then, ensure that **Use Windows Authentication** is selected, in the **Database name** list, select **AWDataWarehouse**, and click **Next**.
8. On the **Choose How to Import the Data** page, ensure that **Select from a list of tables and views to choose the data to import** is selected, and click **Next**.
9. On the **Select Tables and Views** page, select the **DimCustomer**, **DimGeography** and **FactInternetSales** tables. Then change the **Friendly Name** value for the selected tables to **Customer**, **Geography** and **Internet Sales**, and click **Finish**.
10. Wait for the data from the selected table to be imported, and then click **Close**.
11. In the PowerPivot for Excel window, click **Diagram View** and note that the relationships between the tables have been detected.
12. Minimize the PowerPivot for Excel window and return to the Excel Workbook.
13. In Excel on the **Power Query** tab, in the **Get External Data** area (which may be condensed into a drop-down list, depending on the screen resolution), in the **From File** list, click **From Excel**.
14. Browse to the D:\Labfiles\Lab13\Starter folder, select **GasPrices.xlsx**, and click **OK**.
15. In the **Navigator** pane, select Sheet1, and click **Edit**.
16. In the Query Editor, click **Use First Row as Headers**.

17. In the drop-down list for the **Country/Region** column, select only the following values, and click **OK**.
    - Australia
    - Canada
    - France
    - Germany
    - United Kingdom
    - United States
18. Right-click the **US$/US gallon (95 RON)** header and click **Rename**. Then rename this column to **Price per Gallon**.
19. Click the **Price per Gallon** column header, and then in the **Data Type** list, select **Number**.
20. In the **Query Settings** pane, change the **Name** to **Gas Prices**. Then, clear **Load to worksheet**, and select **Load to data model**.
21. Click **Apply and Close**.
22. Close the **Workbook Queries** pane and maximize the PowerPivot for Excel window. Note that the **Gas Prices** table has been added to the data model.
23. On the **Design** tab, click **Create Relationship**.
24. In the **Create Relationship** dialog box, in the **Table** list, select **Gas Prices** and in the **Column** list, select **Country/Region**. Then in the **Related Lookup Table** list, select **Geography** and in the **Related Lookup Column** list, select **CountryRegionName**, and click **Create**.
25. In the PowerPivot for Excel window, on the **Home** tab, in the **PivotTable** drop-down list, click **PivotTable**. Then in the **Insert PivotTable** dialog box, select **Existing Worksheet** and click **OK**.
26. In the **PivotTable Fields** pane, expand **Gas Prices** and select **Country/Region** and **Price per Gallon**, expand **Internet Sales** and select **Sales Amount**. Note that the PivotTable displays the sales revenue and price per gallon for each sales territory.
27. Save the workbook.

▶ Task 3: Use Excel to Visualize Data

1. In Excel, on the ribbon, click **File**. Then click **Options**.
2. In the **Excel Options** dialog box, on the **Add-Ins** page, in the **Manage** list, select **COM Add-Ins** and click **Go**.
3. In the **COM Add-Ins** dialog box, if the **Power View** add-in is not selected, select it and click **OK**.
4. On the **Insert** tab, click **Power View**, and wait for the Power View sheet to be added to the workbook.
5. In the Power View report, close the **Filters** pane and add the title **Sales By Geography**.
6. In the **Power View Fields** pane, expand **Geography** and select **CountryRegionName**, and then expand **Internet Sales** and select **SalesAmount**.
7. In the **Switch Visualization** area of the ribbon, in the **Column Chart** list, select **Stacked Column**. Then resize the column chart to fill the top half of the report.
8. Click the empty area under the column chart, and in the **Power View Fields** pane, expand **Gas Prices** and select **Country/Region** and **Price per Gallon**. Then, under **Internet Sales**, select **SalesAmount**.
9. In the **Fields** area, in the **SalesAmount** drop-down list, select **Average**.

10. In the **Switch Visualization** area of the ribbon, in the **Bar Chart** list, select **Clustered Bar**. Then resize the bar chart to fill the bottom half of the report.
11. In the bar chart, click the bar that shows the average sales amount for the **United States** country/territory. Note that all charts in the report are filtered to highlight the data for the United States.
12. Click the average sales amount bar for **France** and note that the chart updates to highlight data for France.
13. Click the average sales amount bar for **France** again to remove the filter.
14. Close Excel, saving the changes to the workbook if prompted.

Results: After this exercise, you should have a report named Top Sellers and an Excel workbook called Analysis.xslx.