

Rational Rose® Data Modeler ユーザーズ ガイド

発行日	2002 年 2 月
バージョン	2001A.04.10
資料番号	800-025183-000

support@japan.rational.com
<http://www.rational.co.jp>

Rational™
the e-development company

ご注意

著作権

Copyright ©1999-2001, Rational Software Corporation. All rights reserved.

Portions Copyright ©2000-2001, Compaq Computer Corporation. All rights reserved.

Portions Copyright ©1992-2001, Summit Software Company. All rights reserved.

許可される使用

本書には、Rational Software Corporation (または「Rational」) が所有権を有する固有情報が含まれており、個人を対象に、Rational 製品の操作、メンテナンスに関する情報を提供しています。本書の一部またはその全部を他の目的に使用することは禁止されており、形態あるいは方法を問わず、Rational の事前の許可を得ないで、複製、コピー、改変、開示、配布、送信、検索可能なシステムへの保存、人間またはコンピュータの言語への翻訳を行うことはできません。

商標

Rational、Rational Software Corporation、Rational the e-development company、ClearCase、ClearCase Attache、ClearCase MultiSite、ClearDDTS、ClearQuest、ClearQuest MultiSite、DDTS、Object Testing、Object-Oriented Recording、ObjecTime & Design、Objectory、PerformanceStudio、ProjectConsole、PureCoverage、PureDDTS、PureLink、Purify、Purify'd、Quantify、Rational、Rational Apex、Rational CRC、Rational Rose、Rational Suite、Rational Summit、Rational Visual Test、Requisite、RequisitePro、RUP、SiteCheck、SoDA、TestFactory、TestFoundation、TestMate、The Rational Watch、AnalystStudio、ClearGuide、ClearTrack、Connexis、e-Development Accelerators、ObjecTime、Rational Dashboard、Rational PerformanceArchitect、Rational Process Workbench、Rational Suite AnalystStudio、Rational Suite ContentStudio、Rational Suite Enterprise、Rational Suite ManagerStudio、Rational Unified Process、SiteLoad、TestStudio、および VADS は、Rational Software Corporation の米国およびその他の国における商標または登録商標です。その他の名前はすべて、識別の目的でのみ使用されているものであり、それぞれの会社の商標または登録商標です。

Microsoft、Microsoft のロゴ、Active Accessibility、Active Client、Active Desktop、Active Directory、ActiveMovie、Active Platform、ActiveStore、ActiveSync、ActiveX、Ask Maxwell、Authenticode、AutoSum、BackOffice、BackOffice のロゴ、bCentral、BizTalk、Bookshelf、ClearType、CodeView、DataTips、Developer Studio、Direct3D、DirectAnimation、DirectDraw、DirectInput、DirectX、DirectXJ、DoubleSpace、DriveSpace、FrontPage、Funstone、Genuine Microsoft Products のロゴ、IntelliEye、IntelliEye のロゴ、IntelliMirror、IntelliSense、J/Direct、JScript、LineShare、Liquid Motion、Mapbase、MapManager、MapPoint、MapVision、Microsoft Agent のロゴ、Microsoft eMbedded Visual Tools のロゴ、Microsoft Internet Explorer のロゴ、Microsoft Office Compatible のロゴ、Microsoft Press、Microsoft Press のロゴ、Microsoft QuickBasic、MS-DOS、MSDN、NetMeeting、NetShow、Office のロゴ、Outlook、PhotoDraw、PivotChart、PivotTable、PowerPoint、QuickAssembler、QuickShelf、RelayOne、Rushmore、SharePoint、SourceSafe、TipWizard、V-Chat、VideoFlash、Virtual Basic、Virtual Basic のロゴ、Visual C++、Visual C#、Visual FoxPro、Visual InterDev、Visual J++、Visual SourceSafe、Visual Studio、Visual Studio のロゴ、Vizact、WebBot、WebPIP、Win32、Win32s、Win64、Windows、Windows CE のロゴ、Windows のロゴ、Windows NT、Windows Start のロゴ、および XENIX は、Microsoft Corporation の米国およびその他の国における商標または登録商標です。

Sun、Sun Microsystems、Sun のロゴ、Ultra、AnswerBook 2、medialib、OpenBoot、Solaris、Java、Java 3D、ShowMe TV、SunForum、SunVTS、SunFDDI、StarOffice、および SunPCi は、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

FLEXIm および GLOBEtrotter は、GLOBEtrotter Software, Inc. の商標または登録商標です。GLOBEtrotter ソフトウェア (FLEXIm ライブラリおよびユーティリティ) の本来の用途はソフトウェア ライセンス管理であり、他の製品またはアプリケーションにこれらのソフトウェアを組み込むことは、ライセンスに含まれません。

BasicScript は、Summit Software Company の登録商標です。

本製品の一部には、<http://www.mozilla.org/MPL/MPL-1.1.txt> で入手可能な Mozilla 1.1 ライセンスに基づき、expat XML parser 1.0 が組み込まれています。expat XML parser のソースコード版は、<http://www.jclark.com/xml/expat.html> で入手できます。

特許

U.S. 特許番号 5,193,180、5,335,334、5,535,329、5,835,701、5,574,898、5,649,200、および 5,675,802 の対象部分。

U.S. 特許申請中。

国際特許申請中。

Purify は、Sun Microsystems, Inc. の U.S. 特許番号 5,404,499 の下にライセンス供与されています。

米国政府の権利

米国政府による使用、複製、または開示は、該当する Rational Software Corporation ライセンス契約書および DFARS 227.7202-1(a) および 227.7202-3(a) (1995 年)、DFARS 252.227-7013(c)(1)(ii) (1988 年 10 月)、FAR 12.212(a) (1995 年)、FAR 52.227-19、または FAR 52.227-14 で定められている規定の制約を受けます。

免責事項

本書および関連ソフトウェアは、ライセンス契約に基づいて使用することができます。ライセンス契約で明示的に定められていない限り、Rational Software Corporation は、本メディア、ソフトウェア製品、およびその関連文書について、明示的にも暗黙的にも、商品性に関する保証、特定目的への適合性に関する保証、取り扱い、使用、または取引行為に伴う保証について一切の責任を負いません。

目次

	まえがき	xiii
	対象読者	xiii
	そのほかの情報源	xiii
	Rational 技術サポートへのお問い合わせ	xiv
第 1 章	はじめに：チームの統合	1
	チームの役割	1
	ビジネス アナリストの役割	1
	アプリケーション設計者の役割	1
	データベース設計者の役割	2
	役割の依存関係	2
	Data Modeler のソリューション	2
第 2 章	UML とデータ モデリング	3
	内容	3
	UML の概要	3
	データ モデリングに UML を使う理由	3
	UML に追加されたデータ モデリング プロファイル	3
	UML データ モデリング プロファイルの利点	4
	Rose の UML とデータ モデリングの利点	4
	データ モデル図	4
	データ モデル エレメントの再利用	5
	Data Modeler の変換機能とエンジニアリング機能	6
第 3 章	論理データ モデリング	7
	内容	7
	はじめに	7
	Rose を使用した論理データ モデリング	7
	クラス図	7
	標準的な表記法	8
	マッピング機能	8
	オブジェクト モデルからデータ モデルへのマッピング	9
	コンポーネントのマッピング	9
	操作のマッピング	9
	パッケージからスキーマへのマッピング	9
	クラスからテーブルへのマッピング	10
	属性から列へのマッピング	10
	コンポジット集約から依存リレーションシップへのマッピング	13
	集約と関連から非依存リレーションシップへのマッピング	14
	関連クラスから交差テーブルへのマッピング	15
	限定子付き関連から交差テーブルへのマッピング	16
	継承のマッピング	17

	オブジェクト モデルからデータ モデルへの変換	19
	なぜ変換するのか	19
	変換処理	19
第 4 章	物理データ モデリング	23
	内容	23
	はじめに	23
	データ モデル	23
	新しいデータ モデルの構築	23
	データベースの作成	24
	スキーマの作成	24
	データ モデル図の作成	24
	ドメインの作成	24
	テーブルの作成	25
	列の作成	25
	制約の作成	26
	リレーションシップの作成	28
	参照整合性の定義	32
	カスタム トリガの作成	32
	ストアド プロシージャの作成	34
	データ モデルを作成するためのリバース エンジニアリング	35
	リバース エンジニアリング ウィザード	35
	DB2 データベースと DDL のリバース エンジニアリング	35
	Oracle のデータベースまたは DDL のリバース エンジニアリング	36
	SQL Server のデータベースまたは DDL のリバース エンジニアリング	36
	Sybase データベースと DDL のリバース エンジニアリング	36
	リバース エンジニアリング後	36
	データ モデル構築後	37
第 5 章	物理データ モデルのマッピング	39
	内容	39
	はじめに	39
	データ モデルからオブジェクト モデルへのマッピング	39
	スキーマからパッケージへのマッピング	39
	ドメインから属性型へのマッピング	39
	テーブルからクラスへのマッピング	40
	列から属性へのマッピング	40
	列挙型チェック制約からクラスへのマッピング	41
	依存リレーションシップからコンポジット集約へのマッピング	42
	非依存リレーションシップから関連へのマッピング	42
	交差テーブルのマッピング	43
	スーパータイプ/サブタイプ構造から継承構造へのマッピング	46
	データ モデルからオブジェクト モデルへの変換	47
	なぜ変換するのか	47
	変換処理	47

第 6 章	物理データ モデルの実装	51
	内容	51
	はじめに	51
	データ モデルのフォワード エンジニアリング	51
	フォワード エンジニアリング ウィザード	51
	ANSI SQL 92 の DDL へのフォワード エンジニアリング	52
	DB2 のデータベースまたは DDL へのフォワード エンジニアリング	52
	Oracle のデータベースまたは DDL へのフォワード エンジニアリング	53
	SQL Server のデータベースまたは DDL へのフォワード エンジニアリング	53
	Sybase のデータベースまたは DDL へのフォワード エンジニアリング	53
	データ モデルの比較と同期	54
	同期ウィザード	54
付録 A		57
	UML データ モデリング プロファイル	57
付録 B		59
	オブジェクト モデルからデータ モデルへのデータ型のマッピング	59
付録 C		63
	データ モデルからオブジェクト モデルへのデータ型のマッピング	63
付録 D		71
	データベースへの接続	71
	DB2	71
	Oracle	72
	SQL Server	72
	Sybase	72
	索引	75

図目次

図 1	Rose のデータ モデル図	5
図 2	Rose におけるクラス図	8
図 3	クラスからテーブルへのマッピング	10
図 4	属性から列へのマッピング	10
図 5	ID ベースの列	11
図 6	ドメイン列	12
図 7	コンポジット集約から依存リレーションシップへのマッピング	13
図 8	関連から非依存リレーションシップへのマッピング	14
図 9	多対多の関連から交差テーブルへのマッピング	15
図 10	関連クラスから交差テーブルへのマッピング	16
図 11	限定子付き関連から交差テーブルへのマッピング	17
図 12	継承から複数のテーブルへのマッピング	18
図 13	[オブジェクト モデルのデータ モデルへの変換] ダイアログ ボックス	20
図 14	ドメイン	25
図 15	テーブル	25
図 16	依存リレーションシップ	28
図 17	非依存リレーションシップ	29
図 18	ロール	30
図 19	交差テーブル	31
図 20	自己参照リレーションシップ	32
図 21	ドメイン列から属性へのマッピング	40
図 22	テーブルからクラスへのマッピング	40
図 23	列から属性へのマッピング	41
図 24	列挙型チェック制約から <<ENUM>> のクラスへのマッピング	41
図 25	依存リレーションシップからコンポジット集約へのマッピング	42
図 26	非依存リレーションシップから関連へのマッピング	43
図 27	交差テーブルから多対多の関連へのマッピング	44
図 28	交差テーブルから限定子付き関連へのマッピング	45
図 29	交差テーブルから関連クラスへのマッピング	46
図 30	[データ モデルのオブジェクト モデルへの変換] ダイアログ ボックス	48

表目次

表 1	外部キー制約の多重度	30
表 2	UML データ モデリング プロファイルのステレオタイプ	57
表 3	[分析] 言語のオブジェクト モデルからデータ モデルへのデータ型の マッピング	60
表 4	Java のオブジェクト モデルからデータ モデルへのデータ型のマッピング	61
表 5	Visual Basic のオブジェクト モデルからデータ モデルへのデータ型の マッピング	62
表 6	ANSI SQL 92 のデータ モデルからオブジェクト モデルへのデータ型の マッピング	64
表 7	DB2 のデータ モデルからオブジェクト モデルへのデータ型のマッピング	65
表 8	Oracle のデータ モデルからオブジェクト モデルへのデータ型の マッピング	66
表 9	SQL Server のデータ モデルからオブジェクト モデルへのデータ型の マッピング	67
表 10	Sybase のデータ モデルからオブジェクト モデルへのデータ型の マッピング	69

まえがき

Rational Rose[®] (以下、Rose) は、複雑なソフトウェア システムの開発をサポートする統合プログラミング環境です。本書では、データのモデル化を行う環境において Rose の特定の機能を使うために必要な概念について説明します。

対象読者

本書は、次のような方々を対象としています。

- データベースの開発者および管理者
- ソフトウェア システムのアーキテクト
- ソフトウェア エンジニアおよびプログラマ
- 設計、アーキテクチャ、構成管理、テストなどに関して決定を下す方

本書の読者は、データベースのモデル化の概念と、ソフトウェア開発プロジェクトのライフ サイクルに精通していることを前提としています。

その他の情報源

- Rational Suite にはオンライン ヘルプが用意されています。
Suite の各ツールで、[ヘルプ] メニューからオプションを選択してください。
- すべてのマニュアルは HTML または PDF のオンライン形式で利用できます。オンライン マニュアルは、Rational Solutions for Windows のオンライン ドキュメント CD-ROM に収録されています。
- トレーニングに関する詳しい情報については、ラショナル ユニバーシティの Web サイト (<http://www.rational.co.jp/ru>) を参照してください。

Rational 技術サポートへのお問い合わせ

本製品のインストール、使用方法、またはメンテナンスに関してご不明な点は、以下の Rational 技術サポートにお問い合わせください。

地域	連絡先
アジア太平洋 (日本を含む)	電話: +61-2-9419-0111 FAX: +61-2-9419-0123 support@apac.rational.com (英語のみ対応) support@japan.rational.com (日本語対応可)

メモ: Rational 技術サポートにご連絡いただく場合は、以下の情報をお知らせください。

- お名前、電話番号、会社名
- コンピュータのメーカーとモデル
- オペレーティング システムとバージョン番号
- 製品のリリース番号とシリアル番号
- ケース ID 番号 (前回のお問い合わせの続きの場合)

実際にチームを作り上げているのは、共通の目標を達成するために共に働く人たちのグループです。場合によってはチームに 2 人しかいないことや何百人もいることもあります。人数には関係なく、チームはお互いに協力して作業を行う必要があります。開発チームは、さまざまなメンバーが開発ライフ サイクルの異なる部分を担当する、特定の目的を対象としたチームです。ビジネスの問題を解決する場合、開発チームには主に 3 種類の役割があります。ビジネスアナリスト、アプリケーション設計者、データベース設計者の 3 種類です。

チームの役割

本書で説明する範囲では、ビジネスアナリストの役割は、ビジネスプロセスアナリスト、システムアナリスト、およびビジネスプロセスの要件を収集して検討する技術者によって遂行されます。アプリケーション設計者の役割は、アプリケーション開発者、ソフトウェアエンジニア、およびアプリケーションを設計してレビューを行う技術者によって遂行されます。データベース設計者の役割は、データ開発者、データベース管理者 (DBA)、データアナリスト、およびリレーショナルデータベースの設計とレビューを行う技術者によって遂行されます。

ビジネスアナリストの役割

開発プロセスにおけるビジネスアナリストの役割は、エンドユーザーやクライアントにインタビューを行ってビジネスの全体的な問題を理解し、現在のビジネスの状況を分析した上で、ビジネスに問題を及ぼしている現在のプロセスの欠点を特定し、さらに、ビジネスのあるべき姿をモデル化することです。ビジネスアナリストは、エンドユーザーの要件を収集し、ビジネスモデルでプロセスの改善を行います。ビジネスモデルでは、プロセスをイベント、人、または物のコンポーネントに分割し、ユースケースモデル、シーケンスモデル、およびアクティビティモデルを使ってプロセスを整理します。

アプリケーション設計者の役割

アプリケーション設計者の役割は、ビジネスアナリストのビジネスモデルに基づいてコードを生成し、アプリケーションを構築することです。アプリケーション設計者は、オブジェクト指向の概念モデルを使って、ユースケースからクラスを作成し、クラスモデルを使って、クラスをオブジェクトモデルの構造にまとめ上げます。次に、クラスモデル構造をコンポーネントに割り当ててコードを生成し、アプリケーションを構築します。アプリケーション設計者は、データベースのデータにアクセスできるクラスの作成も行います。

データベース設計者の役割

データベース設計者の役割は、ビジネスアナリストのビジネスモデルに基づいて、アプリケーションを構築するのに必要なすべてのデータを格納できるストレージコンテナを提供することと、データ構造の一貫性を維持する方法を提供することです。データベース設計者は、論理データモデルと物理データモデルを使用し、ユースケースやクラスからスキーマやテーブルやその他のデータベース要素を作成します。そして、スキーマ、テーブル、およびデータベース要素を、データモデル構造に組み込みます。次に、そのデータモデル構造を使って、データベース管理システム (DBMS) を選択し、データ定義言語 (DDL) のスクリプトを生成してデータベースを構築します。データベース設計者は、アプリケーション設計者がアプリケーションクラスを正しいテーブルにマッピングして、アプリケーション用のデータにアクセスできるようにする必要があります。

役割の依存関係

役割や知識の違いにより、各チームメンバーはビジネスの問題を異なる方法で解決します。ビジネスアナリストは、プロセスを変更することで問題を解決し、アプリケーション設計者はコードを生成することで問題を解決し、データベース設計者はデータベースとそこに格納されたデータを制御することで問題を解決します。ビジネスの全体的な問題を解決するこれらの方法のいずれについても、計画の段階では、ほかの2つの方法を考慮する必要があります。

ビジネス全体の問題を解決するための各役割の間には、依存関係があります。ビジネスプロセスをモデル化しただけでは、ビジネスの問題を物理的に解決することはできません。コードの生成は、データを保持するデータベースなしでは役に立ちません。データは、それにアクセスするアプリケーションがなければ意味がありません。各役割は、お互いに関係があります。どの役割も単独ではビジネスの問題を完全に解決することはできません。アプリケーション設計者とデータベース設計者には、モデル化された要件が備わっているビジネスモデルが必要です。ビジネスアナリストは、データベースの設計がビジネスに必要なすべてのルールを満たしていることを確認する必要があります。アプリケーション設計者が追加のフィールドを必要としている場合には、データベース設計者はその変更がデータベースの構造に与える影響を明らかにする必要があります。

このような依存関係は特に繰り返し開発では重要です。繰り返し開発では、常に変更があり、チーム間のコミュニケーションが重要になります。しかし、これらの役割のそれぞれで別々の表記法が使われていたのでは、コミュニケーションが困難になります。チーム間で統一された言語を使うことで、コミュニケーションの行き違いを減らし、品質を上げながら、開発にかかる時間を短縮することができます。

Data Modeler のソリューション

Rose のこれまでのリリースでは、ビジネスアナリストとアプリケーション設計者の役割は統合されていました。Data Modeler が追加されたことで、Rose では開発チームの3つの役割がすべて統合されることになり、チーム内で UML という共通言語で記述された個々のモデルを使って、自由にコミュニケーションを取ることができるようになりました。

内容

本章は、次のような構成になっています。

- UML の概要 (3 ページ)
- データ モデリングに UML を使う理由 (3 ページ)
- Rose の UML とデータ モデリングの利点 (4 ページ)

UML の概要

チームには、1 つのツール、1 つの手法、1 つの表記法が必要です。統一モデリング言語 (以下、UML) はその当初から、開発チームのメンバーを統合し、高速で高品質なアプリケーションを開発できる環境を提供してきました。しかし、UML では、ビジネス アナリストとアプリケーション設計者しかお互いにコミュニケーションを取ることができず、データベース設計者は別の表記法を使っていたため除外されていました。Rose では、Data Modeler の追加と共に、エンティティ/リレーションシップ (以下、E/R) の表記法を取り入れる UML プロファイルの追加が行われ、データベース設計者もビジネス アナリストやアプリケーション設計者とコミュニケーションを取ることができるようになっていきます。UML は、本当の意味での統合言語になったのです。

データ モデリングに UML を使う理由

UML では、Peter Chen 氏の E/R 表記法によく似た標準的な表記法を使用します。Chen 氏の E/R と同様に、UML では、相互に関連する複数のエンティティを使って構造を構築する方法が使われます。データ モデリング プロファイルを追加することで、システム全体のモデル化に関する UML の機能が向上しています。論理モデルだけでなく物理データ モデルもモデル化できるようになり、アプリケーションとデータベースのマッピングが可能になりました。

UML に追加されたデータ モデリング プロファイル

UML プロファイルは、UML のメタモデルを変更しないで特定の対象に UML を追加するための方法で、OMG (Object Management Group) によって承認されています。UML に追加された UML プロファイルでは、対象の概念や用語に基づいて、カスタマイズされたステレオタイプとタグ付きの値が使われます。

UML のデータ モデリング プロファイルを使うと、既存の UML 構造に追加されたデータ モデリング用のステレオタイプに基づいてデータベースをモデル化できます。コンポーネント、パッケージ、クラスなどの UML 構造に追加されたステレオタイプを使うことで、データベース、スキーマ、テーブルなどをモデル化できます。UML の関連に追加されたステレオタイプ

を使うことで、リレーションシップをモデル化できます。エンティティ間の関係が「強い」リレーションシップは、コンポジット集約に追加されたステレオタイプでモデル化されます。また、UML の操作に追加されたステレオタイプによって、主キー制約、外部キー制約、一意キー制約のほか、チェック制約、トリガ、ストアドプロシージャなどの追加のデータベース概念をモデル化できます。データベースの概念と UML のデータ モデリング プロファイルのステレオタイプの一覧については、「付録 A」を参照してください。

UML データ モデリング プロファイルの利点

UML データ モデリング プロファイルには、主にビジネスのモデル化とアプリケーションとの互換性に関連した 4 つの大きな利点があります。1 つ目は、UML がシステムの全体的なアーキテクチャに焦点を当てているため、上位レベルのビジネス プロセス、アプリケーション、実装をモデル化できるという点です。

2 つ目は、UML では論理設計と物理設計が分かれているため、データベースをアプリケーションにマッピングして管理方法を簡単に変更できるという点です。物理設計と論理設計を分離すると、物理設計をカスタマイズして、特定のデータベース管理システム (DBMS) と適切なレベルの正規化に対応することができます。一方で、論理設計は、データベースやアプリケーションの全体像を表すのに適した上位レベルの設計に留めることができます。また、設計を分離することにより、変更を受け入れる前に、設計に対する変更の影響を観察できます。論理設計には簡単に適用できる変更でも、物理モデルに同じ変更を適用しようとする、特定の DBMS 構造では制約がある場合があります。そのため、物理設計に対する変更は望ましくないうことになります。

3 つ目は、UML では振る舞いのモデル化ができるという点です。そのため、ビジネスルールを含め、操作と制約をモデル化できます。

最後に、UML はオブジェクト指向の表記法と互換性があります。一部のアプリケーションに関するオブジェクト指向の表記法とデータベースに関する E/R の表記法には相違があり、データベース設計者とほかのチーム メンバーとのコミュニケーションを妨げる原因となっています。データベース設計者が孤立してしまい、設計に関する重要な判断を検討するときに除外されて、ほかのチーム メンバーによる設計に関する判断に参画できない場合があります。UML では、チーム メンバー全員が同じ表記法で設計に関する判断を行うことによって、このようなコミュニケーションの障害が排除されています。この UML データ モデリング プロファイルを利用することで、Rose では UML ベースのデータ モデルを作成できます。

Rose の UML とデータ モデリングの利点

Rose の UML では、データのモデル化を行う際に大きな利点がいくつかあります。つまり、データ モデル図が Rose の一連の UML 図に導入されていること、データ モデル エレメントを再利用できること、Data Modeler がエンジニアリング機能を備えていることです。

データ モデル図

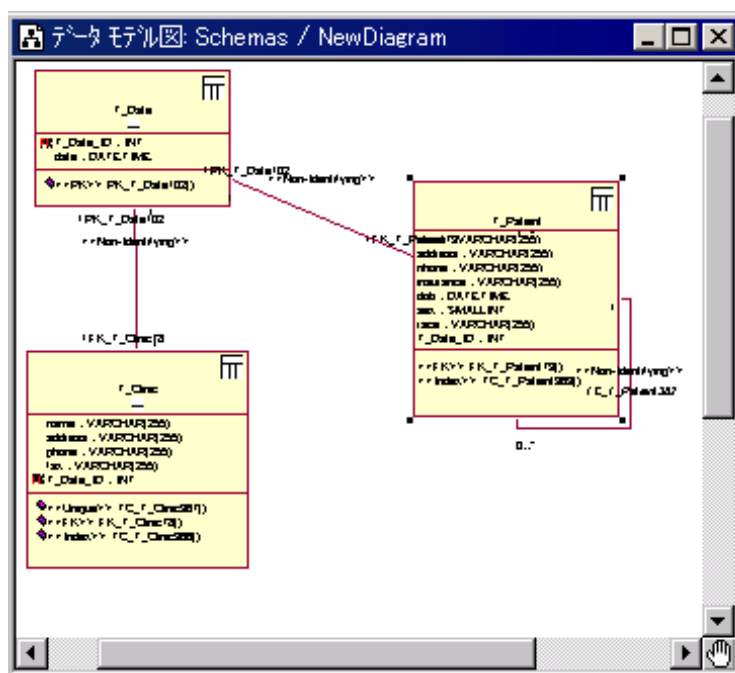
それぞれの開発モデル (ビジネス モデル、アプリケーション モデル、データ モデル) に対し、Rose では次のような図が用いられます。

- ビジネス モデル — ユースケース図、アクティビティ図、シーケンス図
- アプリケーション モデルまたは論理データ モデル — クラス図
- 物理データ モデル — データ モデル図

データモデル図の利点は、データベース設計者が、列、テーブル、リレーションシップなどの今までの慣れ親しんだ用語や構造を使ってモデル化できるという点です。また、データモデル図は、Rose の一連の図を補完し、システム全体をモデル化できます。そのため、データベース設計者とアプリケーション開発者の間にある溝を埋めることができます。

物理データモデルはデータモデル図によって視覚的に表現されるので、物理データモデルで作業する場合は、データモデル図を新たに作成するか、既存のデータモデル図をアクティブにする必要があります。Rose ではこの追加の図が利用できるので、オブジェクトモデルエレメントとデータモデルエレメントの間の混乱を低減できるほか、DBMS のサポート、キーの移行、スキーマの移行といった Data Modeler 独自の機能をサポートできます。

図 1 Rose のデータモデル図



データモデルエレメントの再利用

Rose を使うもう 1 つの利点は、モデルエレメントを再利用し、ほかのモデルに適用することができるという点です。再利用は「フレームワーク」を使うことで実現できます。フレームワークは、ほかのモデルのテンプレートとして機能するファイルです。フレームワークウィザードを使って、.mdl ファイル全体をベースにフレームワークを作成できます。フレームワークはテンプレートとして動作するので、ユーザーのビジネスに利用する標準的なフレームワークを作成できます。フレームワークは特に、モデルにドメインが含まれている場合に便利です。これは、ドメインにより、列レベルでユーザーのビジネスに関する基準を適用できるからです。標準的なエンティティとドメインを含むフレームワークの組み合わせは、ユーザーのビジネスに関する特定の基準を適用するための基礎として機能します。たとえば、診療所のビジネスプロセスをモデル化する場合、患者、医者、診療所のような特定のエンティティが必要ですが、各患者の名前が 20 文字を超えないように指定することや、各医者が正確に 9 桁の

社会保障番号を保持するように指定することも必要です。このような基準をサポートするエンティティとドメインを含むフレームワークを作成することで、基準を繰り返し再利用することができます。

Data Modeler の変換機能とエンジニアリング機能

UML データ モデリング プロファイルを利用することで、Data Modeler は、「データ モデル」として知られるデータ モデル図を、「オブジェクト モデル」として知られるクラス図にマッピングすることができます。また、逆のマッピングも可能です。このマッピングにより、Data Modeler の変換機能とエンジニアリング機能は、ラウンドトリップ エンジニアリングを行うことができます。

オブジェクト モデルとデータ モデルの間の変換

論理クラスと物理テーブルの関係は、論理データ モデルと物理データ モデルの間のマッピングの基礎になります。このマッピングは、オブジェクト モデルからデータ モデルへの変換機能またはデータ モデルからオブジェクト モデルへの変換機能を使うと自動的に行われます。これらの機能によって、クラスとテーブルが 1 対 1 にマッピングされます。ただし、非正規化の問題と DBMS の制限に関しては例外があります。

データ モデルのフォワード エンジニアリングとリバース エンジニアリング

論理クラスと物理テーブルの関係は、フォワード エンジニアリングやリバース エンジニアリングの機能を使った場合に、DBMS とオブジェクト 指向言語 (Java や C++ など) とのマッピングとしても動作します。DBMS スキーマをデータ モデルにリバース エンジニアリングし、次にデータ モデルをオブジェクト モデルに変換すると、オブジェクト モデルは、[分析] 言語に変換されます。オブジェクト モデルの言語を [分析] から Java や C++ に再割り当てすることで、Rose は、DBMS データベースを、アプリケーションの構築に利用できるアプリケーション モデルにマッピングします。アプリケーションをデータベースにマッピングする場合にも同じ処理を利用できます。C++ を使ってアプリケーションを構築したオブジェクト モデルは、再度 [分析] 言語に割り当てることができます。次に、オブジェクト モデルをデータ モデルに変換し、そのデータ モデルをフォワード エンジニアリングして、DBMS にスキーマを作成することができます。

データ モデルの比較と同期

従来の DBMS データベースとアプリケーションを使って作業している場合にも、ラウンドトリップ エンジニアリングの機能を使うことができます。ただし、DBMS へのフォワード エンジニアリングを行う代わりに、Data Modeler の比較同期機能を使って、既存の DBMS データベースを更新し、アプリケーションを構築した変換済みのオブジェクト モデルと DBMS データベースの同期を取ります。

Data Modeler の機能をすべて動作させ、論理クラスを物理テーブルにマッピングすることでデータベースをアプリケーションにマッピングして、ラウンドトリップ エンジニアリングの成果をあげることができます。

内容

本章は、次のような構成になっています。

- はじめに (7 ページ)
- Rose を使用した論理データ モデリング (7 ページ)
- オブジェクト モデルからデータ モデルへのマッピング (9 ページ)
- オブジェクト モデルからデータ モデルへの変換 (19 ページ)

はじめに

論理データ モデリングは、データベースのモデル化を行う際の重要な手順です。収集されたビジネス要件はデータ エンティティに関係しているので、論理データ モデルによって、ビジネス要件の概要を表現することができます。Rose を使って論理データ モデリングを行い、論理データ モデルをカスタマイズして物理データ モデルに変換できます。

Rose を使用した論理データ モデリング

前の章では、Rose と UML をデータ モデリングに使う一般的な利点を説明しました。本章では、Rose と UML を論理データ モデリングに使う場合の利点を説明します。Rose のこのような利点としては、クラス図を使って論理データ モデルをグラフィカルに表現できること、Rose の標準的な表記法、マッピング機能などがあります。

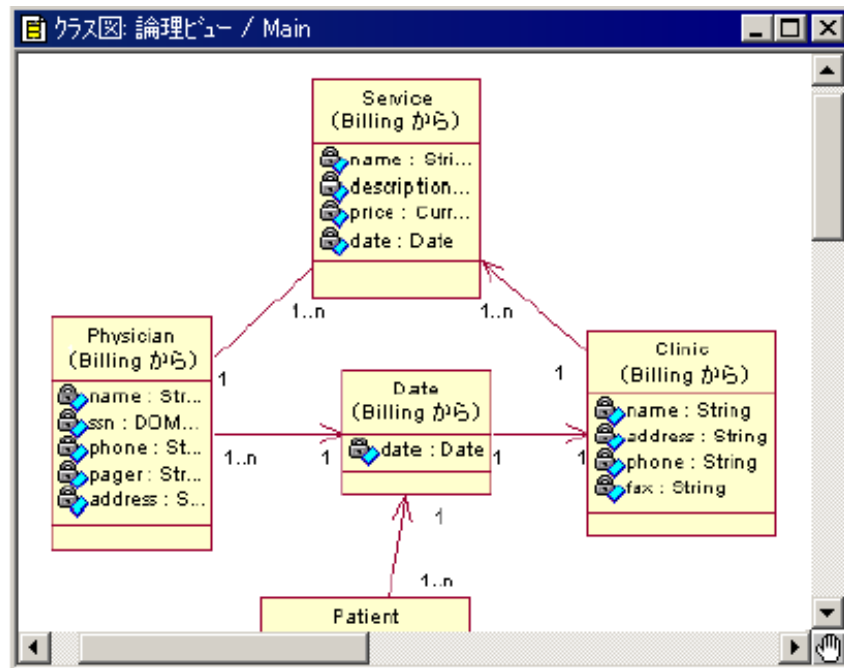
クラス図

クラス図では、論理データ モデルに似た構造が使われているため、論理データ モデルをグラフィカルに表現できます。論理データ モデルを作成するには、まず、上位レベルのエンティティを識別します。クラス図でも上位レベルのエンティティが示されます。UML の用語では、これらのエンティティは「クラス」と呼ばれます。Rose では、このようなクラスをさらに区別するために、<<entity>> というステレオタイプをクラスに割り当てることができます。

次のステップは、システムが属性を識別できるよう、エンティティに属性を割り当てることです。この手順は、クラス図でモデル化を行う場合に属性をクラスに割り当てるのと同じ手順です。

論理データ モデルとクラス図の両方に共通する最後のステップは、「関連」を使ってエンティティやクラスをほかのエンティティやクラスに関連付けることです。

図 2 Rose におけるクラス図



標準的な表記法

そのほかの利点として、Rose では UML の一般的な表記法が使われます。「第 2 章 UML とデータ モデリング」で説明したように、UML データ モデリング プロファイルでは、データ モデリングの用語に関連する UML のエレメントにステレオタイプが追加されています。これと同じ UML のエレメントがクラス図でも使われており、これらのステレオタイプを見ることで、論理データ モデルから物理データ モデルへのマッピングを理解できます。

マッピング機能

Rose のデータ モデリングの用語では、クラス図はオブジェクト モデルとも呼ばれます。オブジェクト モデルには 2 つの目的があります。クラス図やオブジェクト モデルは、論理データ モデルとして利用できますが、オブジェクト モデルはアプリケーションの概観を理解するためのモデルとしても利用できます。アプリケーションをデータベースにマッピングする場合は、オブジェクト モデルが必要です。アプリケーションからデータベースへのマッピングの基礎になるのは、物理データ モデルに対するオブジェクト モデルのマッピングです。

オブジェクト モデルからデータ モデルへのマッピング

Rose には、上位レベルのエンティティ、属性、リレーションシップを識別するだけではない高度な機能があります。Rose を使うと、いっそう厳密に物理データベースにマッピングできる堅牢な論理データ モデルを作成できます。データベースに合わせたオブジェクト モデルのカスタマイズは変更の管理に役立ち、要件の変更が既存のモデルに及ぼす影響を抑えることができます。オブジェクト モデルがデータ モデルにマッピングされている場合、オブジェクト モデルに対して変更や拡張を行うと、同じ変更や拡張をデータ モデルに適用することができます。データ モデルのエレメントにマッピングするようにオブジェクト モデルのエレメント (コンポーネントと操作を除く) をモデル化することで、データベースに厳密にマッピングするオブジェクト モデルをモデル化できます。

コンポーネントのマッピング

「コンポーネント」は、実際のアプリケーションの言語を表します。Rose の用語では、コンポーネントは、明確に定義されたインターフェイスを備えたソフトウェア モジュール (ソース コード、バイナリ コード、実行モジュール、DDL) を意味しています。UML データ モデリング プロファイルに従って、コンポーネントはデータベースにマッピングされますが、このマッピングは参照の目的のみに利用されます。オブジェクト モデルからデータ モデルへの変換では、コンポーネントは無視されます。

Rose では、いくつかのコンポーネント言語の中から希望の言語を選択して論理パッケージに割り当てることができますが、Data Modeler アドインは、3 つのコンポーネント言語、Java、Visual Basic、[分析] とのみ互換性があります。データ モデルのテーブルにマッピングしたいクラスは、これらの 3 つのコンポーネント言語の 1 つを選択して、データ モデルに変換する必要があります。Data Modeler と互換性のないコンポーネント言語を使う場合は、そのコンポーネント言語を使って別のオブジェクト モデルを作成し、変換の目的に使用する [分析] オブジェクト モデルにマッピングすることをお勧めします。

操作のマッピング

UML データ モデリング プロファイルに従って、操作はさまざまな制約にマッピングされます。ただし、コンポーネントと同様、操作は変換処理では無視されます。操作はクラスの振る舞いです。インデックス、可能性のあるトリガ、そのほかの制約を識別するための基礎として利用できるので、操作はデータベースの設計に役立ちます。

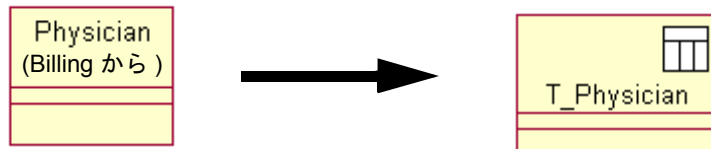
パッケージからスキーマへのマッピング

クラス図では、「論理パッケージ」はオプションのエレメントですが、Data Modeler を使う場合には、論理パッケージが必要です。論理パッケージはオブジェクト モデルの主要なコンテナなので、Data Modeler はこのレベルから変換処理を開始します。クラスをデータ モデルに変換するには、クラスを論理パッケージにグループ化する必要があります。Data Modeler では一度に 1 つの論理パッケージを変換でき、変換された論理パッケージごとに 1 つのスキーマが生成されます。

クラスからテーブルへのマッピング

クラスは、非永続と永続という2つの存在状態を持つ上位レベルのエンティティです。物理テーブルにマッピングされるのは永続クラスです。これは、永続クラスが、アプリケーションが処理を終えた後も存在する永続データストレージとして動作できるためです。永続クラスはすべて、1対1の対応でテーブルにマッピングできます。ただし、モデル内で継承構造を使用している場合は例外で、1対多のマッピングになる可能性があります。

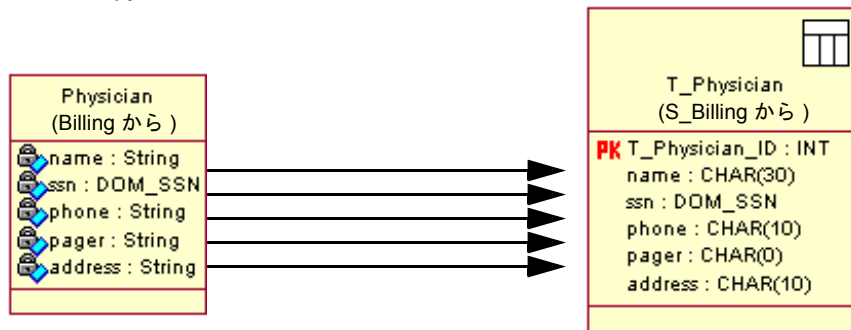
図3 クラスからテーブルへのマッピング



属性から列へのマッピング

永続属性は、列に対して1対1にマッピングされます。モデルでは、1つの列にマッピングされる属性が複数存在する場合がありますが、変換では、Data Modeler はすべての属性を1対1で変換します。Data Modeler は、派生値のような非永続的な属性を無視します。

図4 属性から列へのマッピング



Rose では、特定の属性を特別な列 (主キー列、ID ベース列、ドメイン列など) にマッピングすることで、属性のマッピングをカスタマイズできます。また、属性の型を、DBMS の特定のデータ型にマッピングすることもできます。

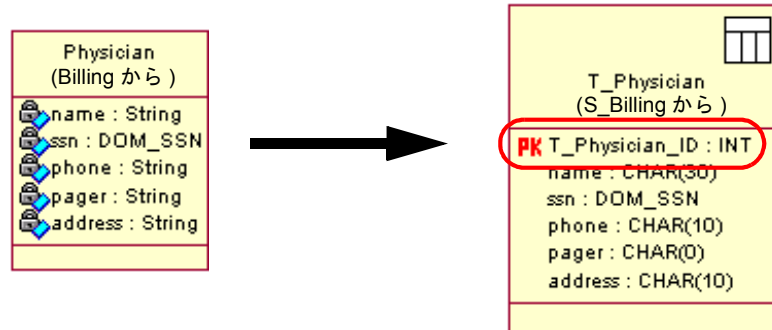
主キー

属性を主キー列としてマッピングするには、その属性を「候補キー」として割り当てます。候補キーは、オブジェクト ID の一部としてタグ付けされた属性です。Data Modeler は、オブジェクトモデルをデータモデルに変換する処理の中で、候補キーをテーブルの主キーに変換します。1つまたは複数の属性を候補キーとして割り当てることができます。複数の候補キーをクラスに割り当てた場合、属性はデータモデル内の複合主キーに変換されます。

ID ベースの列

親クラスの中で候補キーを指定していない場合は、オブジェクトモデルをデータモデルに変換する際に、Data Modeler が自動的に各親クラス用の主キーを生成します。Data Modeler は、テーブルに列を追加し(「ID ベースの列」と呼ばれます)、それを主キーとして割り当てることで、このキーを生成します。ID ベースの列には、システムによって生成された一意の値が使われます。

図 5 ID ベースの列



ID ベースのキーと候補キー

テーブル用に固有の識別子を選択する際には、候補キーより ID ベースのキーの方に明らかな利点があります。利点の 1 つは、ID ベース キーがシステムによって生成される値であるために、サイズを一定に維持できることです。

もう 1 つの利点は、ID ベース キーには 1 つの列が使われる点です。これに対して候補キーの場合、テーブルを一意に識別するには、複数の列が必要になることがあります。使われる列が 1 つであれば、データベースの設計が簡単になります。これは、リレーションシップにおいて、複数列ではなく 1 列だけが外部キーとして移行されるためです。

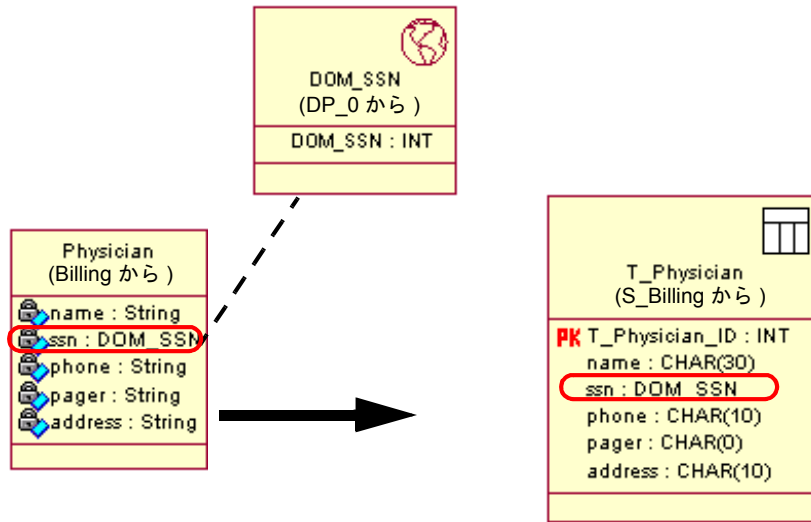
複数の列を使うと、テーブルのエントリの一意性を保証するのが難しくなります。たとえば、T_Physician テーブルで名前とアドレスを候補キーとして使うと、同名の母と娘のように、ある物理学者が名前も住所も別の物理学者と同じという場合、重複が発生します。このような状況を避けるには、候補キーとして社会保障番号(ssn)を使うことができますが、この情報が正しく入力されない可能性も高くなります。システムによって生成される ID ベースのキーであれば、このような問題は少なくなります。

ドメイン列

データモデルで「ドメイン」を既に定義している場合は、属性をドメインにマッピングできます。データモデルでドメインを作成する方法の詳細については、「第4章 物理データモデリング」を参照してください。ドメインは、特定の DBMS 言語に対応するユーザー定義のデータ型として利用できます。ドメインがデータベースと矛盾しないように、ドメインをデータモデルで使うのと同じ DBMS 言語に割り当てておくことが重要です。

ドメイン名に属性の型を設定することで、属性をドメインにマッピングします。変換処理では、ドメイン名をデータ型として使って属性が列に変換されるので、結果としてドメインに定義されたデータ型と設定が使われます。

図 6 ドメイン列



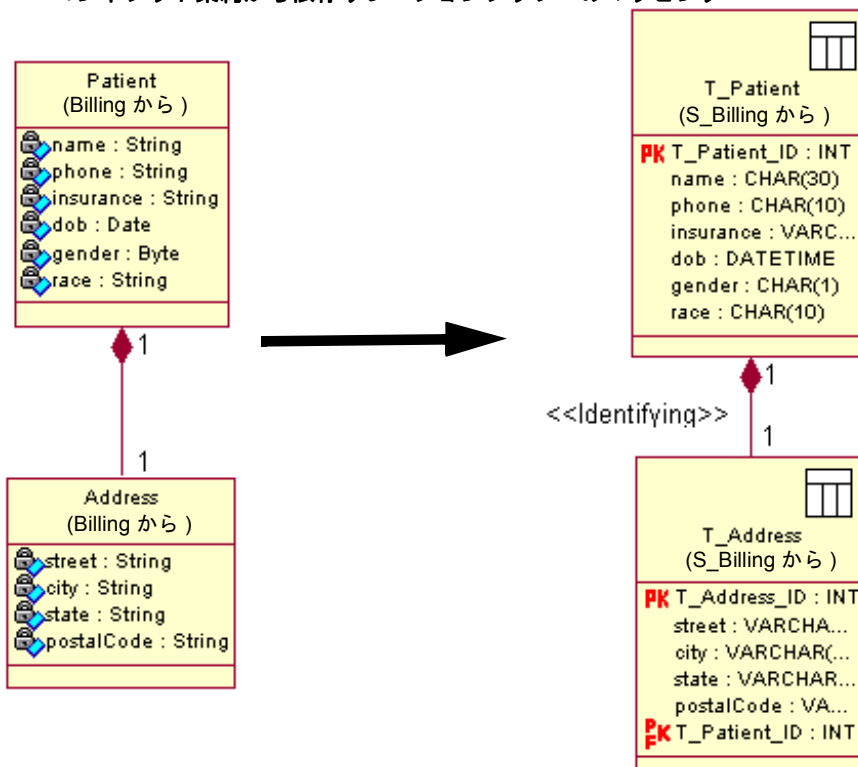
データ型

Data Modeler は、ユーザーの属性の型を、DBMS や ANSI SQL 92 標準の適切な列のデータ型に自動的にマッピングします。属性を、データモデル内で特定のデータ型に変換したい場合は、必要なデータ型にマッピングされる正しい属性型を指定する必要があります。データ型のマッピングの一覧については、「付録 B」を参照してください。また、列にデフォルト値を指定する場合は、属性の初期値として指定できます。初期値は、列のデフォルト値にマッピングされます。

コンポジット集約から依存リレーションシップへのマッピング

値による集約は「コンポジット集約」と呼ばれ、データモデルの依存リレーションシップにマッピングされます。コンポジット集約は全体と部分で構成されており、「強い」関係を示します。部分は全体なしでは存在できません。依存クラスを持つ親クラスのインスタンスが1つある場合に、コンポジット集約を使用します。依存クラスは部分として定義され、全体つまり親クラスが付随しなければなりません。親クラスが削除された場合は、そのクラスを構成する部分も削除されなければなりません。したがって、親クラスでは多重度として1を使用する必要があります。

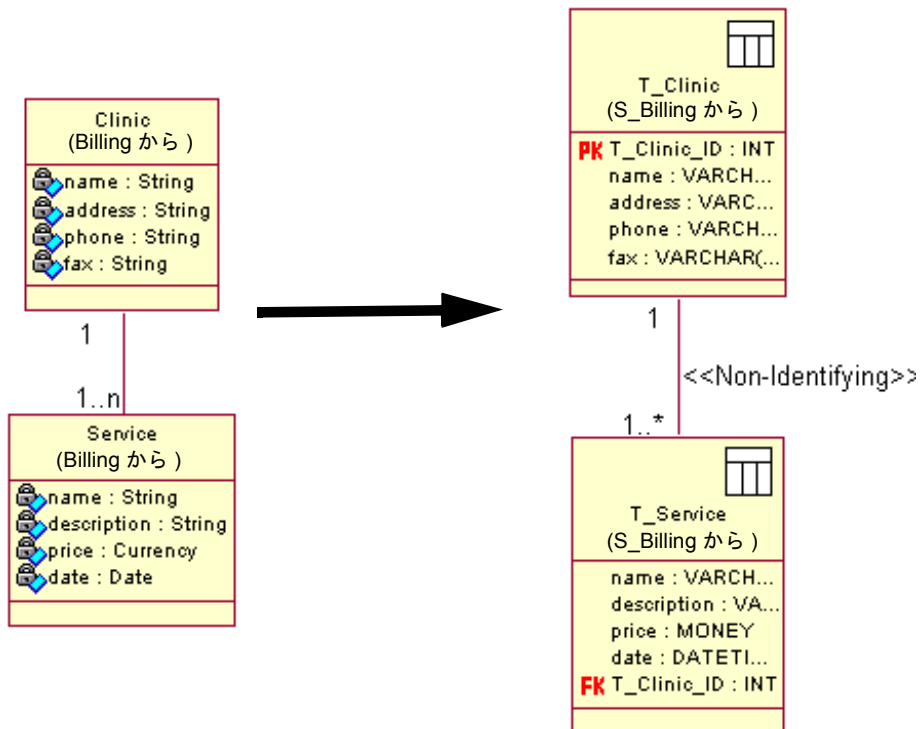
図7 コンポジット集約から依存リレーションシップへのマッピング



集約と関連から非依存リレーションシップへのマッピング

参照による集約（「集約」と呼ばれる）と関連は、多対多の関連を除いて、非依存リレーションシップにマッピングされます。どちらも、「強い」関係を使わないで2つのクラスを結合します。依存クラスを保有する親クラスに複数のインスタンスがある場合に集約を使います。相互に依存しないクラスの場合は、関連を使います。1または1..nの多重度を使うことで、集約や関連を必須にすることができます。この場合、親クラスが必要です。また、関連は、0..nの多重度を使うことでオプションにすることもできます。この場合、親クラスは必要ありません。

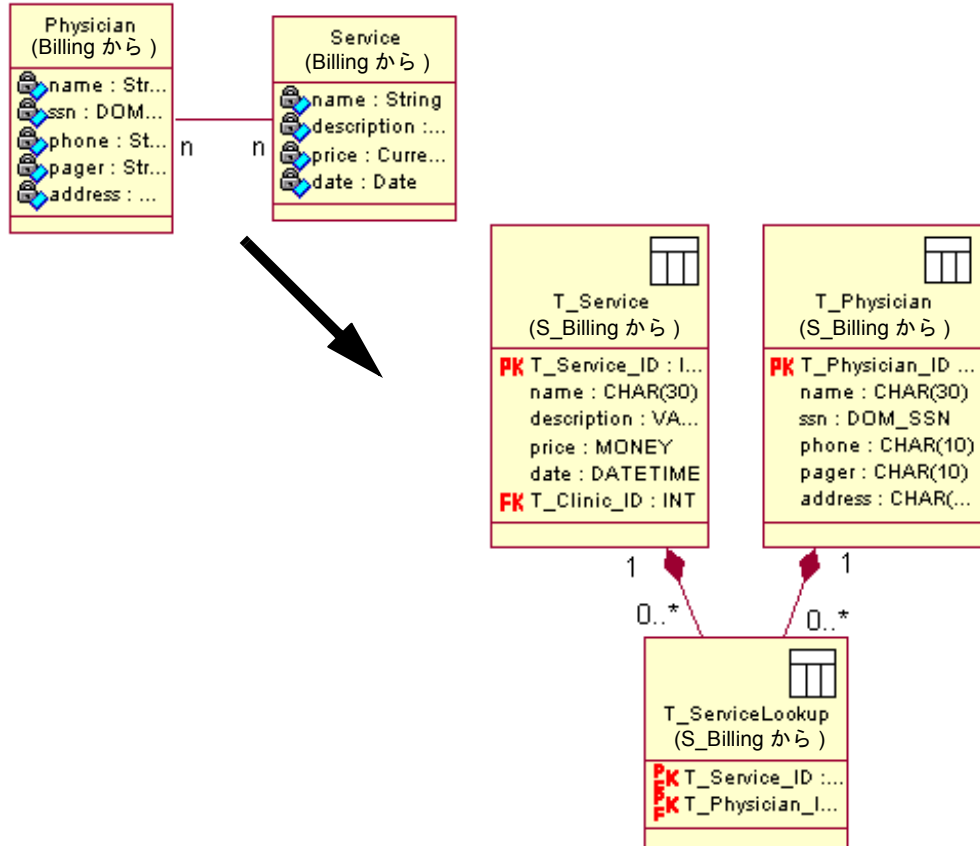
図 8 関連から非依存リレーションシップへのマッピング



多対多の関連

多対多の関連は、「交差テーブル構造」にマッピングされます。交差テーブルでは、すべての列が主キー/外部キーです。変換処理では、Data Modeler は、多対多の関係にある 2 つのクラスを読み取って、2 つのテーブルを作成します。その後、2 つの依存リレーションシップを持つこの 2 つのテーブルを、システムによって生成された交差テーブルと呼ばれるテーブルに結合します。依存リレーションシップを生成する処理の一部として、2 つのテーブルの主キーは、主キー/外部キーとして交差テーブルに移行されます。

図 9 多対多の関連から交差テーブルへのマッピング

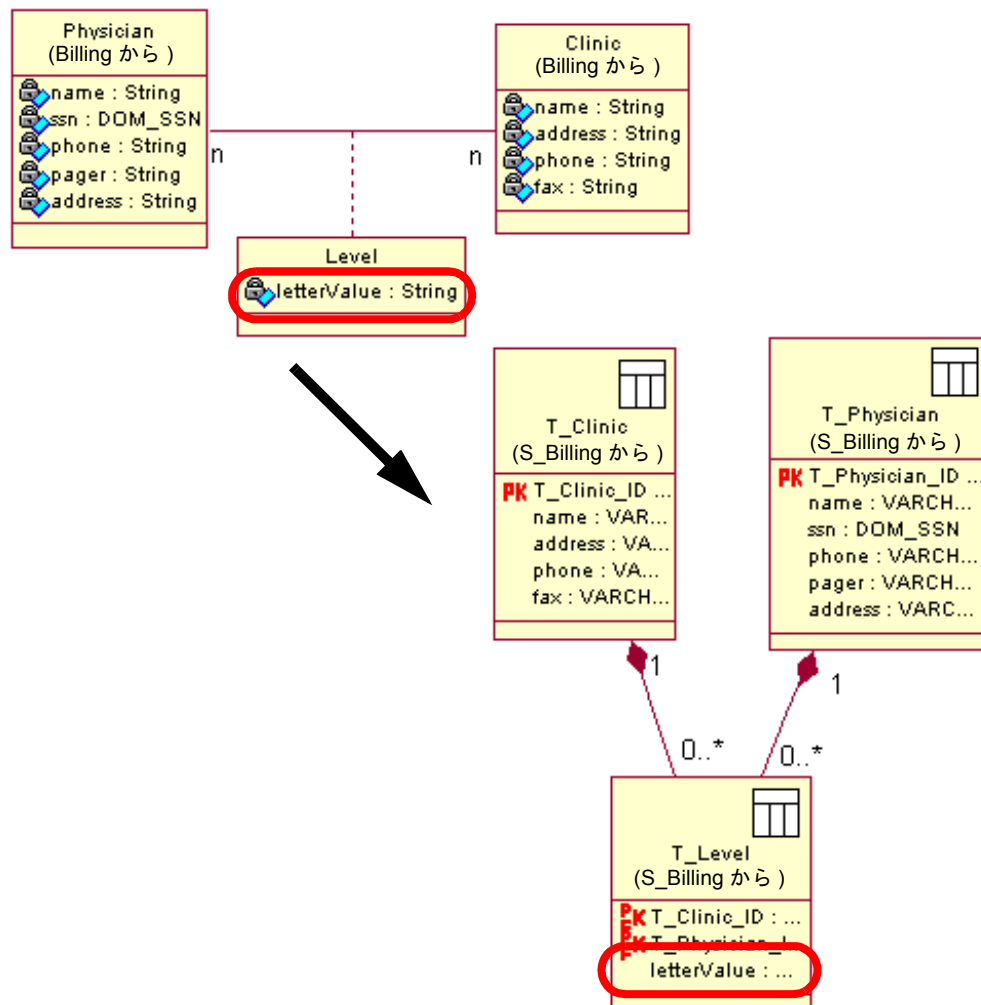


関連クラスから交差テーブルへのマッピング

多対多の関連にリンクする「関連クラス」は、交差テーブル構造にマッピングされます。この場合の交差テーブルには、主キー/外部キーではない列が 1 つ以上追加されます。関連クラスは、関連に結合される追加のクラスで、その関連の 2 つのクラスによって共有されるプロパティと操作を格納できます。クラスがこれらのプロパティと操作を共有するのは、プロパティと操作を 2 つのクラスのいずれにも格納できないからです。Data Modeler は、関連クラスと関連クラスのプロパティを変換しますが、関連クラスの操作は無視します。オブジェクトモデルからデータモデルへの変換では、Data Modeler は、2 つのクラスをテーブルに変換し、2 つ

の依存リレーションシップで交差テーブルに結合して、各テーブルの主キーを主キー / 外部キーとして交差テーブルに移行します。さらに、関連クラスの属性を、交差テーブルの列として追加します。

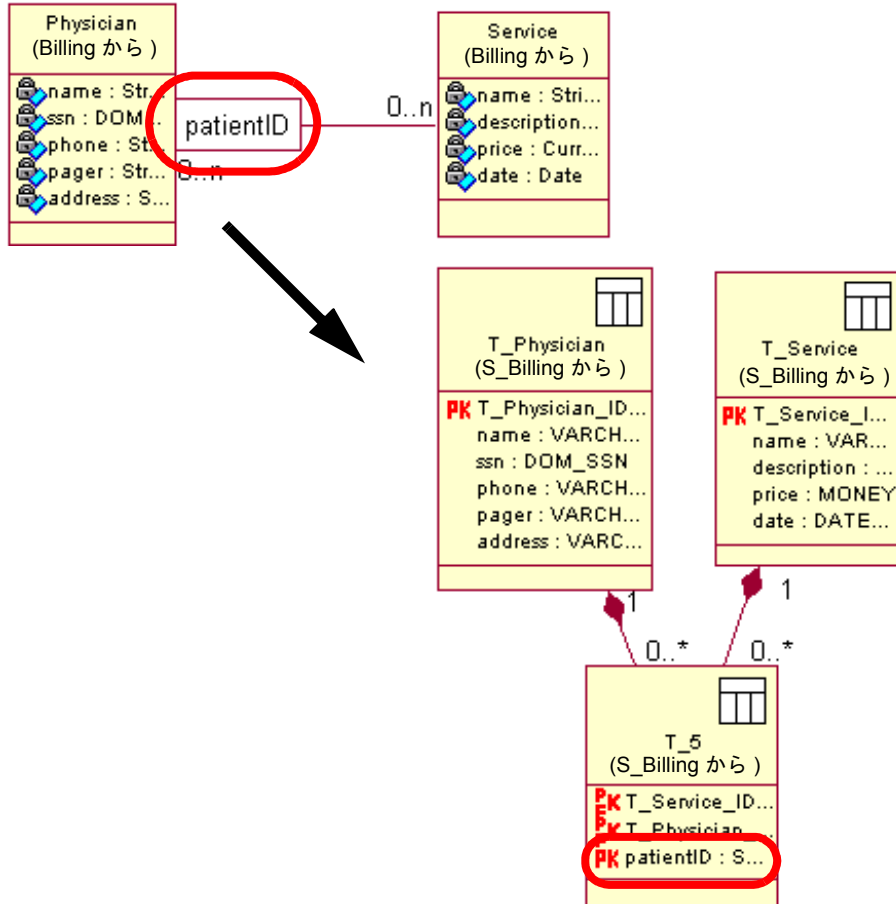
図 10 関連クラスから交差テーブルへのマッピング



限定子付き関連から交差テーブルへのマッピング

「限定子付き関連」は、追加の主キーを含む交差テーブルにマッピングされます。限定子付き関連は、限定子を使用する多対多の関連です。限定子は、関連の一方の側に適用される属性で、関連のもう一方の側で特定のオブジェクト群を返すための識別子として機能します。関連クラスの変換処理と同様に、限定子付き関連の2つのクラスは個別のクラスに変換されて、依存リレーションシップによって交差テーブルに結合されます。限定子付き関連の限定子は、交差テーブルの主キーに変換されます。交差テーブルには、2つのテーブルの主キー / 外部キーと、限定子の変換で生成された追加の主キーの列が挿入されます。

図 11 限定子付き関連から交差テーブルへのマッピング

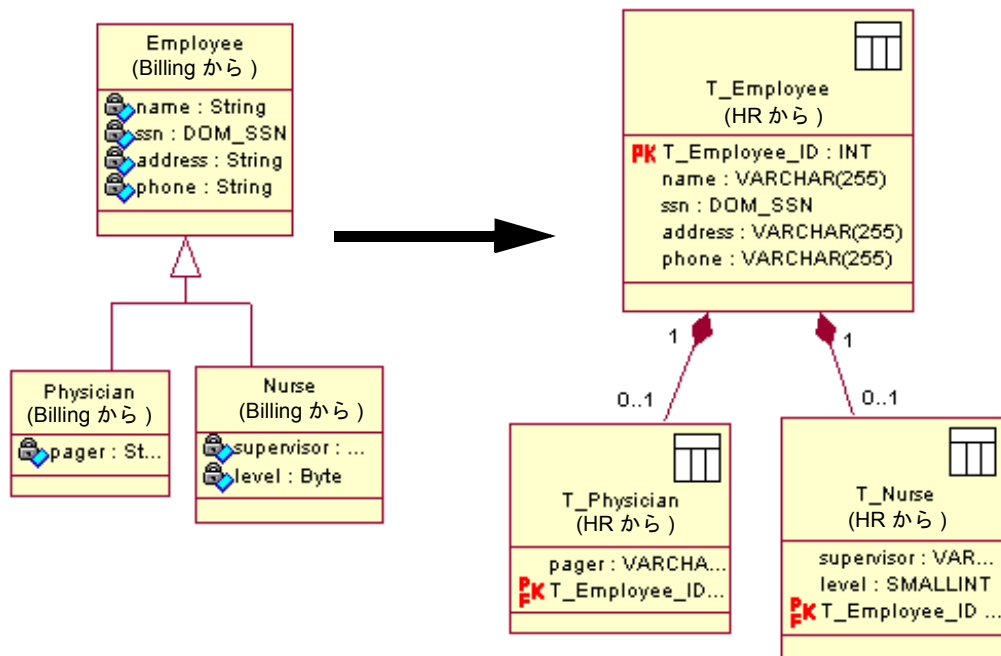


継承のマッピング

継承構造 (UML の用語では「汎化」構造) は、独立した複数のテーブルか 1 つのテーブルにマッピングされます。ただし、Data Modeler は、継承構造を独立した複数のテーブルにだけ変換します。継承構造は、一般的な親クラスと、特殊な子クラスを関連付けます。子クラスは、親クラスと同じ属性を共有すると共に、その特殊な子クラスにだけ影響するそのほかの属性を持ちます。

Data Modeler は、継承構造を複数のテーブルに変換する際、継承構造に加わっているクラスごとに 1 つのテーブルを作成します。親テーブルは、0:1 の依存リレーションシップか 1 の依存リレーションシップで各子テーブルに結合されます。そのため、各子テーブルには、親テーブルの主キーに関連する主キー / 外部キーが挿入されます。

図 12 継承から複数のテーブルへのマッピング



オブジェクトモデルをデータモデルに変換した後、継承構造を手動で1つのクラスにマッピングできます。これは、テーブル間の2つの依存リレーションシップを削除することで行うことができます。この操作によって、テーブルの主キー/外部キーが削除されます。次に、個々の列を2つの子テーブルから交差テーブルに移動します。最後に、オブジェクトモデルの子クラスを非永続に設定します。

重要：手動で継承構造を1つのテーブルにマッピングする場合には、オブジェクトモデルをデータモデルに変換する際とその逆に変換する際に、毎回この処理を繰り返す必要があります。変換するたびに手動でモデルを再マッピングしない場合には、アプリケーションとDBMSのマッピングで矛盾が発生する可能性があります。

オブジェクト モデルからデータ モデルへの変換

データ モデルに対応するようにオブジェクト モデルをカスタマイズした後、オブジェクト モデルを変換し、指定したマッピングが反映されたデータ モデルを生成できます。さまざまな理由でオブジェクト モデルからデータ モデルへの変換を行います。理由に関係なく、行われる処理は同じです。

なぜ変換するのか

オブジェクト モデルを変換することで、新しい DBMS データベースを生成したり、オブジェクト モデルとデータ モデルの同期を取ってデータ設計者と情報を共有することができます。

変換は、アプリケーションをサポートする新しいデータベースの生成に必要な手順です。オブジェクト モデルを変換すると、データ モデルが生成されます。このデータ モデルに対してフォワードエンジニアリングを行い、DBMS データベースを作成することができます。

また、オブジェクト モデルをデータ モデルに変換すると、データ モデルで生成された各エレメントはオブジェクト モデルの 1 つまたは複数のエレメントにマッピングされるので、モデルの同期が取られます。この同期により、システム内の一貫性が保たれ、アプリケーション設計者とデータベース設計者がそのオブジェクト モデルを通して情報を共有できます。

データベースの設計者は、アプリケーションのオブジェクト モデルを基にして設計します。それは、データのアクセスがこのオブジェクト モデルの構造を通して行われるからです。したがって、オブジェクト モデルからデータ モデルへの変換は、DBMS データベースに影響する可能性のあるアプリケーションの変更を伝達する手段として利用できます。アプリケーションの設計者は、変更されたオブジェクト モデルをデータ モデルに変換することで、特に変更などの情報を共有できます。また、データベースの設計者は、変更を確認することで、データベースを更新することなくその変更が DBMS にどの程度影響するかを理解できます。Data Modeler のオブジェクト モデルからデータ モデルへの変換機能を使うことで、オブジェクト モデルの変更をデータベースの設計者と共有できます。

変換処理

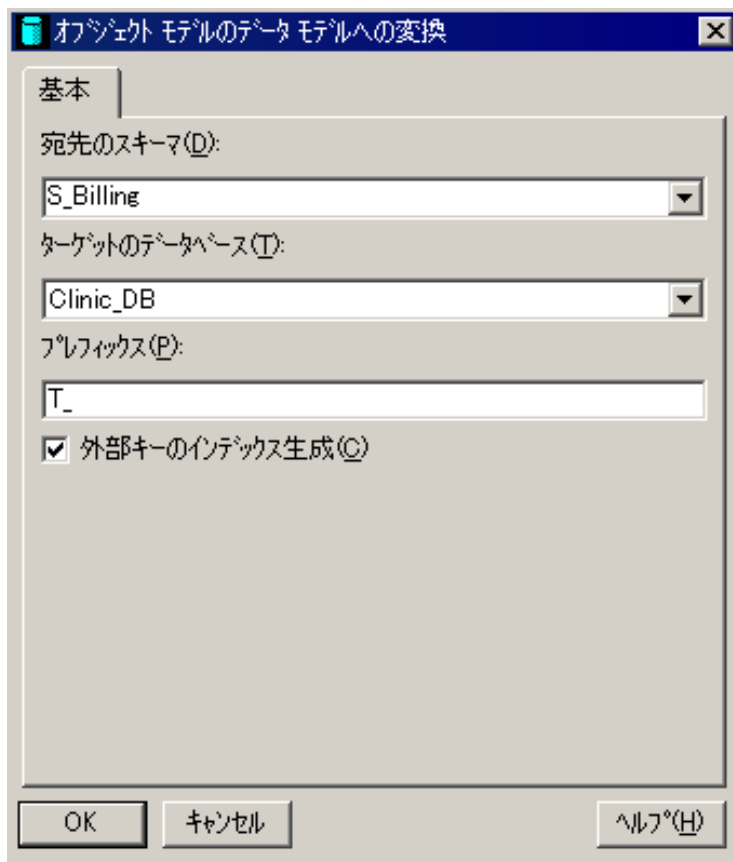
コンポーネントと操作を除き、本章のマッピングの項で説明したオブジェクト モデルのすべてのエレメントは、マッピングされるデータ モデルのエレメントに変換されます。

コンポーネントはデータベースにマッピングされますが、コンポーネント自体はデータ モデルには変換されません。実際には、オブジェクト モデルをデータ モデルに変換する前に、コンポーネントの代わりに、新しいデータベースを作成する必要があります。新しいデータベースを作成する際には、データベース名を指定し、サポートされている DBMS や ANSI SQL 92 標準にデータベースのターゲットを割り当てることができます。新しいデータベースの作成方法の詳細については、「第 4 章 物理データ モデリング」を参照してください。

[オブジェクト モデルのデータ モデルへの変換] ダイアログ ボックス

オブジェクト モデルの変換処理は、[オブジェクト モデルのデータ モデルへの変換] ダイアログ ボックスから始まります。このダイアログ ボックスでは、データ モデル用のスキーマ名、使用するデータベース名、テーブルのプレフィックス、外部キーにインデックスを作成するかどうか、などの選択や指定ができます。

図 13 [オブジェクト モデルのデータ モデルへの変換] ダイアログ ボックス



[宛先のスキーマ]

既存のスキーマ名を選択した場合、その既存のスキーマは、オブジェクト モデル用に生成される新しいスキーマで上書きされます。新しいスキーマ名を指定すると、Data Modeler は、ユーザーがデータベースに割り当てた可能性のある既存のスキーマに上書きすることなく、その名前を使って新しいスキーマを生成します。

[ターゲットのデータベース]

[ターゲットのデータベース](データベース名)を指定するときは、一覧から名前を選択する必要があります。ターゲットを空のままにしておくと、変換が完了した後で、仕様のダイアログボックスが利用できなくなります。

[プレフィックス]

新規テーブルにプレフィックスを割り当てることで、テーブルとオブジェクトモデル内のクラスを区別できるため、プレフィックスを使用することをお勧めします。テーブル名の前にプレフィックスが表示されるので、テーブルからクラスまたはクラスからテーブルへのリレーションシップの作成を防ぐことができます。このようなリレーションシップを作成すると、オブジェクトモデルやデータモデルは無効になります。

[外部キーのインデックス生成]

さらに使いやすくするために、新規データモデル内の各外部キーにインデックスを構築するように指定できます。ダイアログボックスで[OK]をクリックすると、実際の変換処理が始まります。

パッケージ、クラス、関連の変換

実際の変換処理では、Data Modelerは、[オブジェクトモデルのデータモデルへの変換]ダイアログボックスで指定したスキーマ名を使ってスキーマを生成します。既存のスキーマ名を指定した場合は、Data Modelerは既存のスキーマを更新します。その後、オブジェクトモデルのパッケージを読み出して、永続クラスと永続クラス間の関係を探します。これらのクラスは、データモデルスキーマのテーブルに変換されます。

この時点で、Data Modelerは、永続クラス内で候補キーとして割り当てられた属性を探します。候補キーとして割り当てられた属性がある場合は、各候補キーを主キーに変換し、各制約用に作成する一意のインデックスと合わせてテーブルの主キー制約に追加します。候補キーとして割り当てられた属性がない場合は、親テーブルにIDベースの列を追加し、その列をテーブルの主キーとして割り当てます。この処理によって、すべての親テーブルが主キーを保持することになるため、すべてのリレーションシップが有効になります。その後、ほかのすべての属性がテーブルに列として追加されます。

テーブルが作成されると、Data Modelerはオブジェクトモデルの関連をデータモデルのリレーションシップに変換します。各リレーションシップには、オブジェクトモデルでの指定に従って、適切な多重度が与えられます。また、各リレーションシップにより、親テーブルの主キーに基づいて、子テーブルに外部キーが移行されます。各外部キーにインデックスを構築するように指定した場合には、外部キーが親テーブルから子テーブルに移行された後、インデックスが生成されます。これには、交差テーブルに変換される構造のような、特殊な構造もすべて含まれています。

変換にかかる処理時間は、オブジェクトモデル内で変換されるクラスの数によって異なります。

変換処理の後

変換処理が完了すると、必要な変更をデータ モデルに対して行い、オブジェクト モデルとデータ モデルをより正確にマッピングすることができます。また、データ モデルのテーブル仕様または列仕様にある「マップ元」プロパティを使って、オブジェクト モデルからデータ モデルへのマッピングを確認することもできます。

【マップ元】

オブジェクト モデルやデータ モデルを変換すると、参照するオブジェクト モデル エLEMENT の名前が自動的に【マップ元】ボックスに挿入されます。これは保護されたフィールドなので、手動で操作することはできません。データ モデルで作業している場合は、クラスの名前を変更しても、特定のテーブルや列に対する【マップ元】テキスト ボックスを参照することで、テーブルや列にマッピングされているクラスや属性を知ることができます。オブジェクト モデル エLEMENT からデータ モデル エLEMENT へのマッピングは、常に 1 対 1 のマッピングとは限らないので、このプロパティが役に立ちます。テーブル間の列の移動により、データ モデルが非正規化されて、クラスとテーブルの 1 対 1 のマッピングが失われたインスタンスが発生する可能性があります。このようなインスタンスについては、【マップ元】ボックスを確認することで、加えられた変更内容に関わらず、エンティティのソースを追跡できます。

内容

本章は、次のような構成になっています。

- はじめに (23 ページ)
- データ モデル (23 ページ)
- 新しいデータ モデルの構築 (23 ページ)
- データ モデルを作成するためのリバース エンジニアリング (35 ページ)
- データ モデル構築後 (37 ページ)

はじめに

データベースのモデル化における次のステップは、物理データ モデリングです。物理データ モデルでは、論理データ モデルで得られた要件が使われ、特定のデータベース管理システム (DBMS) に適用されます。物理データ モデルには、DBMS データベースの下位レベルの詳細な仕様も取り込まれます。Rose Data Modeler では、この物理データ モデルを「データ モデル」と呼びます。

データ モデル

Data Modeler では、物理データ モデルはデータ モデルと呼ばれ、データ モデル図の中でグラフィカルに表現されます。データ モデル図は、ほかの UML 図からカスタマイズされており、データベース設計者は既に慣れている概念と用語を使って作業することができます。

Rose を使うと、さまざまな方法でデータ モデルを作成できます。前の章では、オブジェクトモデルを変換してデータ モデルを作成する方法を説明しましたが、新しいモデルを構築してデータ モデルを作成することもできます。また、データベースや DDL ファイルをリバース エンジニアリングすることで、データ モデルを作成することもできます。

新しいデータ モデルの構築

データ モデルの要素を手動で作成することで、新しいデータ モデルを構築できます。この項では、Data Modeler を使ってデータ モデルを構築する処理を説明します。

データベースの作成

「データベース」は、データ モデルの実装コンポーネントです。各データベースは、「ターゲット」に割り当てられます。ターゲットは、実際の ANSI SQL 92 標準、またはサポートされている DBMS と使用する DBMS バージョンを示しています。[データベースの仕様]でターゲットを指定できます。デフォルトのターゲットは、ANSI SQL 92 です。ターゲットを指定すると、指定した DBMS のバージョンまたは ANSI SQL 92 標準でサポートされている要素だけが、データ モデルでサポートされます。

また、データベースを作成すると、[Schemas] フォルダが自動的に Rose のブラウザに作成されるので、スキーマを保存できます。各データベースは、1 つまたは複数のスキーマを含むことができます。

スキーマの作成

「スキーマ」は、データ モデルの主要なコンテナであり、データ モデルのすべてのテーブルが含まれています。Data Modeler では、データ モデルが存在するにはスキーマが必要です。そのため、データ モデルのすべてのエレメント (ドメインを除く) をスキーマに割り当てる必要があります。

スキーマ用のテーブルを作成する前に、スキーマをデータベースに割り当てる必要があります。スキーマをデータベースに割り当てると、データベースで指定されている DBMS のバージョンまたは ANSI SQL 92 標準でサポートされている要素だけがサポートされます。スキーマをデータベースに割り当てない場合、スキーマはデフォルトとして ANSI SQL 92 標準を使用します。

データ モデル図の作成

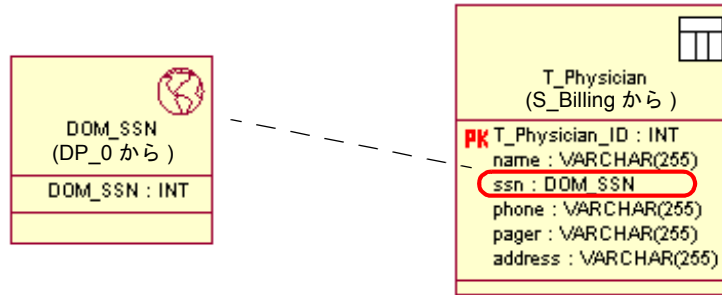
リレーションシップを作成する場合は、データ モデル図でリレーションシップを描画しなければならないため、データ モデル図を作成する必要があります。データ モデル図を作成して有効にすると、Data Modeler のカスタマイズされたツールセットと、対応するメニュー コマンドが利用できるようになります。データ モデル図の詳細については、「第 2 章 UML とデータ モデリング」を参照してください。

ドメインの作成

ドメインは、頻繁に使う列のテンプレートとして機能し、カスタマイズされたデータ型のように列や属性に適用できます。たとえば、従業員のテーブルをモデル化する場合、社会保障番号が常に必要になります。社会保障番号の列の設定を繰り返し作成する代わりに、社会保障番号の設定を含むドメインを作成し、そのドメインを列のデータ型として割り当てることができます。25 ページの図 14 を参照してください。

ドメインの作成はオプションです。使用している DBMS に対するドメインのサポートに依存します。ドメインを作成する場合、最初に、ドメイン用のコンテナとしてドメイン パッケージを作成する必要があります。ドメイン パッケージは特定の DBMS に割り当てることができます。次に、ドメインを作成し、そのドメインをドメイン パッケージに割り当てます。ドメインをドメイン パッケージに割り当てると、ドメインは、ドメイン パッケージの DBMS でサポートされているデータ型だけをサポートするようになります。また、フォワードエンジニアリングを行うために、ドメインに、サーバー生成のタグを付けることもできます。

図 14 ドメイン

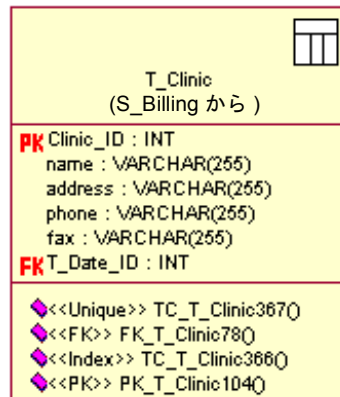


テーブルの作成

テーブルはエンティティを示します。各テーブルは列、制約、トリガ、およびインデックスを含むことができます。テーブルどうしは、リレーションシップを通して結合されます。テーブルは1つのスキーマにしか属することができませんが、スキーマは0個以上のテーブルを含むことができます。また、テーブルにテーブルスペース名を指定することもできます。

テーブルを作成する場合、テーブル名はテーブルが属するスキーマ内で固有である必要があります。また、指定したDBMSの命名要件に合っている必要があります。データモデル内のテーブルをオブジェクトモデル内のクラスと区別するために、プレフィックスを使うことが推奨されます。

図 15 テーブル



列の作成

列ではテーブルの特性が定義されます。テーブルは列によってリンクされます。列名はテーブル内で固有である必要があります。列の値は制約によって制御されます。列タイプ、データ型、長さ、スケール(必要な場合)、列がキー制約の一部かどうか、およびNullを設定できるかどうかを指定できます。

列タイプ

各列には列タイプが割り当てられます。Data Modeler は、データ列と算出列の 2 種類の列タイプをサポートしています。

データ列

データ列には、派生値を除くすべてのデータの情報が格納されます。データ列には、ターゲットの DBMS でサポートされているデータ型や、既存のドメインを割り当てることができます。データ型を指定する場合、データ型には、長さ、精度、付随するスケールの入力が必要な場合があります。デフォルト値をデータ列に割り当てることができます。また、制約をデータ列に割り当てて、それが主キーか、一意キーかと、NULL 値が許されるのかどうかを指定できます。データ列は、SQL Server の IDENTITY プロパティと DB2 の ForBitData もサポートしています。

算出列

算出列では SQL の式が使われ、値の生成と格納が行われます。列の値は生成されて、一意な値が使われないため、算出列を主キーや一意キーとして割り当ててはできません。クエリ速度を向上させる最適化の 1 つとして、算出列にインデックスを作成することができます。

制約の作成

制約には、「キー制約」と「チェック制約」の 2 種類があります。キー制約では、行を一意に識別し、リレーションシップ内で参照整合性を強制するために、列や列のグループ中のデータが制限されます。チェック制約では、ビジネスルールを強制するために、テーブルのデータへの追加やテーブルのデータの変更が制限されます。

キー制約

キー制約には、主キー制約、一意キー制約、外部キー制約の 3 種類があります。また、キー制約の種類として含まれているものにインデックスがあります。インデックスは、ほかのキー制約に直接関係しており、キーの列一覧を使用するため、キー制約の種類に含まれています。各テーブルには、1 つの主キー制約を保持できます。また、0 個以上の一意キー制約か外部キー制約を保持できます。キー制約を設定するために、データベース サーバーが一意的インデックスを作成します。

主キー制約

列の 1 つに主キーを割り当てると、主キー制約が自動的に作成されます。この場合、主キー制約は、そのキーの列と、主キーとして割り当てたほかのすべての列で構成されます。主キー制約では、テーブルの任意の 2 つの行が同じ非 NULL 値を主キー列に保持することはできず、主キーが NULL 値を保持することもできません。主キーは、すべての対応する外部キーの特性を制御し、リレーションシップの中の親テーブルを識別できます。

主キー制約は、1 つの主キーまたは複合主キーで構成できます。複合主キーは、複数の主キーと主キー / 外部キーで構成されます。主キー / 外部キーは、リレーションシップの参照整合性を強制する埋め込みキーです。そのため、NULL 値は許されません。主キー / 外部キーは、依

存リレーションシップを通して生成されるので、手動で主キー/外部キーを制御することはできません。主キー/外部キーの詳細については、28 ページの「埋め込まれた主キー/外部キー」を参照してください。

一意制約

一意キー制約も、1 つ以上の一意な列で構成されるもう 1 つの制約として知られています。一意キー制約では、テーブルの任意の 2 行が、同じ非 NULL 値を任意の制約の列に保持することはできません。この制約では NULL 値は許されていませんが、SQL Server だけは例外です。SQL Server では一意キー制約に NULL 値が許されています。

外部キー制約

外部キー制約は、1 つ以上の外部キーで構成されます。外部キーは読み取り専用ですが、外部キー名とデフォルト値は例外です。各外部キーは、2 つのテーブル間のリレーションシップを作成することで生成されて、主キーや一意キーが親テーブルから移行されます。親テーブルの主キーや一意キーに対して行われた変更はすべて、子テーブル中の外部キーに連鎖的に適用されます。外部キーに対する NULL 制約および一意キー制約は、リレーションシップの多重度によって制御されます。詳細については、29 ページの「多重度」を参照してください。

インデックス

「インデックス」は、列のキー一覧で構成されます。列のキー一覧だけを検索することで、任意の値にすばやくアクセスできます。各インデックスには固有の名前を指定する必要があります。インデックスはクラスタ化できますが、Data Modeler では、テーブルあたり 1 つのクラスタ化されたインデックスしか利用できません。インデックスをクラスタ化すると、物理的にインデックスがデータと共に格納されるのでインデックスの効率が上がります。依存リレーションシップによって関係付けられている子テーブル用にインデックスを作成する場合には、常にクラスタ化したインデックスを使用する必要があります。

インデックスの使用率/未使用率を指定することもできます。使用率/未使用率により、各インデックスのページが保持できる行の割合が決まります。使用率を低く指定すると、インデックスの柔軟性が高くなります。使用率を高く指定すると、インデックス中のレコードの変更がほとんどできなくなります。

キー制約の作成

キー制約の名前と、制約の種類が主キー、一意キー、インデックスのいずれであるかを指定することで、キー制約を作成できます。また列を選択してキー制約の中にも含めることもできます。インデックスを作成する場合には、インデックスを一意にするかどうか指定できます。自動的にインデックスが生成されるキー制約は、一意のインデックスとして設定されます。

チェック制約

チェック制約では、SQL の述語式を使って、テーブルのデータに対するアクションが制限されます。実行された SQL 式が False を返す場合、テーブルのデータは変更されません。テーブルの仕様またはドメインの仕様の [**チェック制約**] タブを使って、テーブルやドメイン用にチェック制約を作成できます。チェック制約の名前と SQL 式を指定できます。また Oracle をターゲット DBMS として使用している場合には、「延期可能」か「延期不可能」かも指定できます。

延期可能と延期不可能 (Oracle のみ)

Oracle DBMS 用のチェック制約は、延期不可能か延期可能かのいずれかです。延期不可能のチェック制約では、SQL 文が終了した時点で、アクションの有効性が確認されます。延期可能のチェック制約では、[最初のイミディエイト]を使用すると文の終了時に、[最初の指定]を使用するとコミットされる前のトランザクションの終了時に、それぞれアクションの有効性が確認されます。

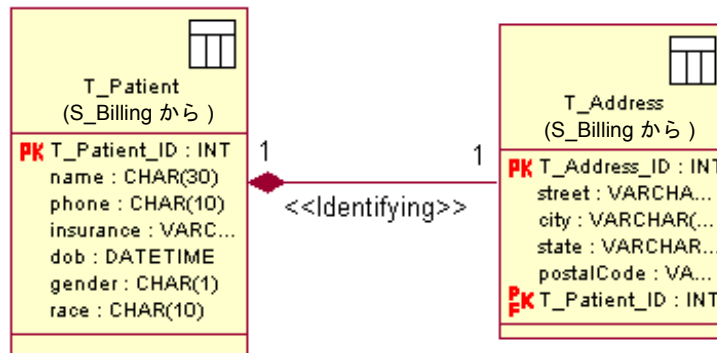
リレーションシップの作成

「依存リレーションシップ」と「非依存リレーションシップ」の2種類のリレーションシップによって、テーブルがデータ モデル内にある別のテーブルに関係付けられます。また、リレーションシップに対して多重度とロールを定義し、別のリレーションシップ構造でそれらを使うことができます。

依存リレーションシップ

依存リレーションシップでは、子インスタンスは親インスタンスなしでは存在できません。依存リレーションシップは、部分から全体へのリレーションシップで、部分は全体がないと意味を持ちません。たとえば、「診療所」のモデルでは、T_Address テーブルには住所がありますが、その住所に住んでいる患者の名前はあります。そのため、住所自体には意味がなく、患者の名前と関連付ける必要があります。

図 16 依存リレーションシップ



依存リレーションシップを使うと、親テーブルの主キーは、外部キーとして子テーブルに移行されます。外部キーは、子テーブルの既存の主キー制約に、主キー / 外部キーとして組み込まれます。子テーブルに主キー制約がない場合は、移動した外部キーが主キー / 外部キーとして割り当てられ、外部キー制約と主キー制約が作成されます。

埋め込まれた主キー / 外部キー

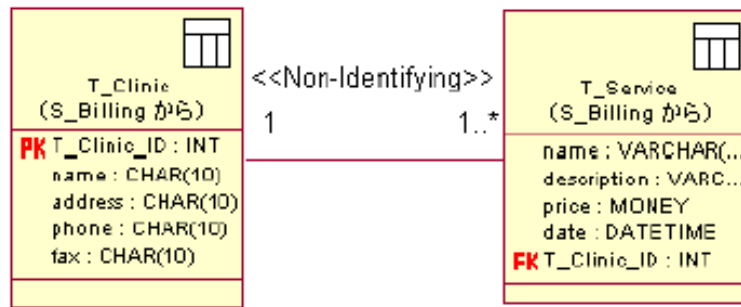
外部キーが埋め込まれると、外部キーは主キー / 外部キーとしてテーブルに現れます。外部キーが主キーに埋め込まれると、リレーションシップの参照整合性が強制されます。この埋め込みにより、親テーブルのインスタンスを削除する前に、子テーブルのインスタンスを削除する必要がありますがあるので、親のないレコードを防ぐことができます。また、主キーが親テーブルの別の列に再度割り当てられるのを防ぐこともできます。このような再割り当てが行われると、親

のないレコードが子テーブルに作成されてしまいます。埋め込まれた主キー / 外部キーは、リレーションシップが非依存リレーションシップではなく、依存リレーションシップであると区別されます。

非依存リレーションシップ

非依存リレーションシップは、子インスタンスと親インスタンスに強い相互依存関係がないリレーションシップです。つまり、外部キーは子テーブルの主キー制約に埋め込まれません。非依存リレーションシップには、オプションと必須の 2 種類があります。オプションの非依存リレーションシップでは、親インスタンスは必要ないので、このリレーションシップの親テーブルには多重度 0..1 が使われます。必須の非依存リレーションシップでは、親インスタンスが必要なので、多重度 1 が使われます。

図 17 非依存リレーションシップ



必須の非依存リレーションシップと依存リレーションシップ

必須の非依存リレーションシップと依存リレーションシップには親インスタンスが必要なので、いつこれらを使うかの判断が困難な場合があります。考慮すべき重要な要素はビジネスユースケースについてです。子のライフサイクルの間に、親の別のインスタンスに子に関連付ける必要がある場合は、必須の非依存リレーションシップを使用します。子のライフサイクルの間、常に親の同じインスタンスに子に関連付けておく必要がある場合は、依存リレーションシップを使用します。

多重度

「多重度」は、2 つのテーブル間のリレーションシップをインスタンス化できる回数の最小数と最大数です。多重度は、参照整合性を強制する場合に使われます。テーブルの多重度が 1 の場合、そのテーブルはリレーションシップの中に存在する必要があることを示します。多重度は、子テーブルに親のないレコードができるのを防ぐために、親テーブルにおいて特に重要です。

多重度によって、外部キーが一意かどうか、および NULL 値が許されるかどうかにも決まります。親テーブルの多重度が 1 以上の場合、外部キーに NULL は許されません。次に示すのは、外部キーに NULL 値や一意の値を許すために必要な多重度を示した表です。

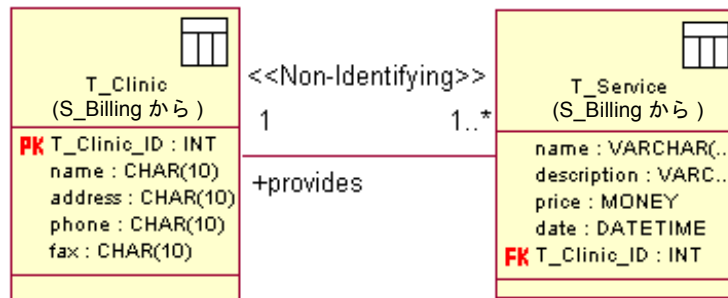
表 1 外部キー制約の多重度

必要な制約	親テーブルの 多重度	子テーブルの 多重度
外部キーは NULL 値を取り一意	0..1	0..1 または 1
外部キーは NULL 値を取り一意ではない	0..1	0..* または 1..*
外部キーは NULL 値を取らず一意ではない	1	1..* または 0..*
外部キーは NULL 値を取らないが一意	1	0..1 または 1

ロール

リレーションシップの「ロール」は、テーブルがリレーションシップの中でどのように動作するかを示し、多重度に意味を持たせます。ロールは、リレーションシップの中の 1 つのテーブルか両方のテーブルのいずれかに適用できます。多重度とロールを組み合わせることで、リレーションシップの中で何が起きているかを示す、文法を備えた文が作成されます。リレーションシップの中の各親テーブルは名詞として動作するので、ロールは動詞で子テーブルは直接目的語、あるいはその逆になります。図 18 の例では、診療所と診療所が提供するサービスとの間のリレーションシップのロールが示されています。ここでは、1 つの診療所が 1 つ以上のサービスを提供しています。

図 18 ロール



これは、ビジネスの要件をデータ モデル内のモデル化されたエレメントに変換し、ビジネス要件が満たされたことをビジネス アナリストに伝える際に便利です。

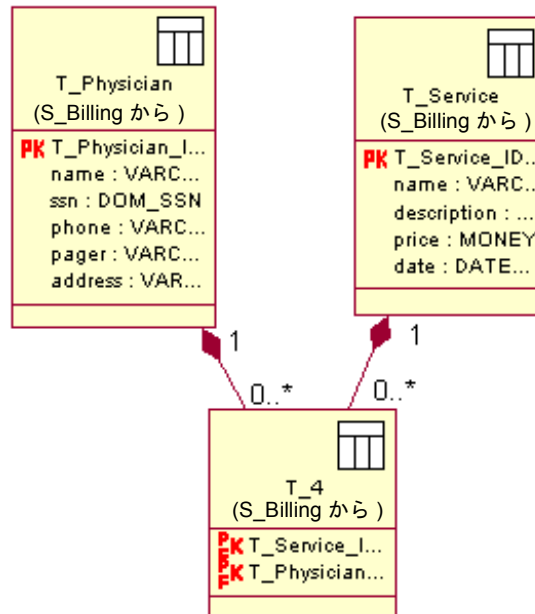
リレーションシップ構造

依存リレーションシップと非依存リレーションシップを使うと、別のリレーションシップ構造を作成できます。このような構造の 1 つが交差テーブルです。もう 1 つの構造が、自己参照構造です。

交差テーブル

「交差テーブル」は、定義上はテーブルですが、その特殊なリレーションシップ構造は、実際に何が定義されているのかを示します。交差テーブルは、1つの子テーブルが2つの親テーブルに2つの1:nの依存リレーションシップで関係付けられているリレーションシップ構造です。このような構造が作成された場合には、子テーブルには主キー/外部キーが含まれ、主キー/外部キーは親テーブルの主キーに対応します。そのため、何らかの更新を行うと、子テーブルが両方の親テーブルに影響を及ぼします。この種のリレーションシップ構造は、スーパータイプ/サブタイプの構造に利用できます。

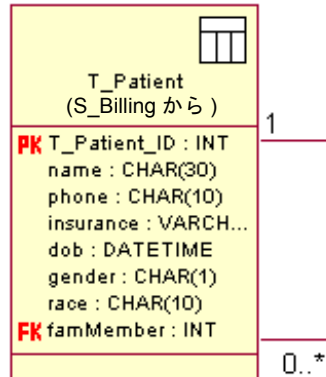
図 19 交差テーブル



自己参照

もう1つのリレーションシップ構造は、「自己参照構造」です。自己参照構造では、1つのテーブルだけが使われ、そのテーブルは非依存リレーションシップでそのテーブル自身に関係付けられます。同じテーブルの別のインスタンスに関係付ける必要のあるそのテーブルのインスタンスを保持している場合に、自己参照構造を使います。この種のリレーションシップ構造は、再帰的なリレーションシップに利用できます。

図 20 自己参照リレーションシップ



参照整合性の定義

「参照整合性」を使用すると、リレーションシップの中の親テーブルを更新または削除したときのデータの整合性が保証されます。そのためには、対応する子テーブルに特定のアクションを適用したり、親テーブルが更新および削除されないように設定します。

Data Modeler では、参照整合性のアクションをサポートするために、「宣言参照整合性 (DRI)」と「システム生成参照整合性 (RI)」トリガという 2 つの方法が提供されています。これら 2 つの方法では、次のようなアクションを実行できます。

- Cascade — 関連するすべての子进行削除 / 更新します。
- Restrict — 親の削除 / 更新を禁止します。
- Set NULL — 子のすべての外部キーを NULL に設定します。
- No Action — 何のアクションも行いません。
- Set Default — 子のすべての外部キーをデフォルト値に設定します。

宣言参照整合性

DRI では、データ モデルをフォワード エンジニアリングする際に、外部キー句の一部として参照整合性のアクションを指定します。DRI は最も効果的で簡単な方法と考えられています。DBMS に依存し、各 DBMS でのすべてのアクションをサポートするわけではありません。

システム生成参照整合性トリガ

システム生成 RI トリガでは、システム トリガの生成によって参照整合性のアクションを指定します。各 DBMS では、システム生成 RI トリガのサポートの方が充実しています。整合性を取る別の手法には、「子の制限」を指定して親のないレコードが挿入されるのを防ぐ方法があります。

カスタム トリガの作成

テーブル内のデータを更新、追加、または削除すると、「カスタム トリガ」によって一連の SQL 文が実行されます。トリガを使って複数のテーブルにビジネス ルールを適用することができます。これは、トリガによって特定のデータの修正を防ぐことができるからです。アプリケーション設計者とデータベース設計者が同じロジックを使ってビジネス プロセスを遂行できるので、ビジネス ルールを適用するための方法としてトリガを使うことは重要です。

トリガ イベント

トリガを作成する場合、トリガを発生させる 1 つのイベントやイベントの組み合わせを決める必要があります。「トリガ イベント」は、データを変更する更新、追加、削除などの固有のアクションです。

DB2 および Oracle 用の追加のトリガ設定

トリガ イベントに付随して、Data Modeler には、DB2 と Oracle の DBMS だけで利用できる追加のトリガ設定があります。設定には、トリガのタイプ、粒度、参照、When 句の使用方法などがあります。

トリガ タイプ

Data Modeler では、トリガ文を確認するタイミングを示す「トリガ タイプ」を指定できます。トリガ イベントが発生する前、またはトリガ イベントが発生した後のいずれかを選択できます。

トリガ イベントが発生する前にトリガ文を確認する場合は、トリガによって、修正の条件がデータベースにとって適切かどうかを修正前に確認できます。たとえば、このタイプのトリガを使った場合、SQL 文の要件に合っていない新しい患者のレコードを挿入しようとする、挿入のアクションは拒否されて、データベースは変更されません。

トリガ イベントが発生した後でトリガ文を確認する場合は、トリガによって、修正の条件がデータベースにとって適切かどうかを修正後に確認できます。このトリガ タイプを使う場合は、粒度のレベルと参照エイリアス名を設定できます。そのため、修正内容をコミットする前に、データが修正された結果を参照することができます。これにより、エラーが発生した場合は古いデータベースを指す呼び出しを作成し、エラーが発生していない場合は新しいデータベースを指す呼び出しを作成できるので、アプリケーション設計者にとって重要な機能です。

トリガの粒度

トリガの「粒度」によって、トリガを実行する頻度を指定できます。各行の後で実行するか、または各文の後で実行するよう指定できます。この相違は、ほかの列や行が最初に変更されない、変更条件が無効と考えられる場合に重要です。たとえば、子テーブルを更新する場合には、親テーブルを最初に更新する必要があるため、各行ではなく各文の後でだけ実行したいはずです。そのため、各行の粒度は細くなり、SQL 条件が頻繁に確認されます。粒度のレベルは、トリガをテーブルで参照するか、行で参照するかによって決まります。

トリガの参照

「トリガの参照」によって、行やテーブルが修正される前後に、その行やテーブルにエイリアス名を割り当てることができます。ターゲットの DBMS として Oracle を使用している場合には、古いテーブルや行、および修正されたテーブルや行にエイリアス名を割り当てることができます。このような参照によって、同じテーブルや行に 2 つのビューが与えられます。このビューは、ビジネス プロセスや開発の意思決定に応じて、複数のアプリケーションから使うことができます。

トリガの When 句

「When 句」は、トリガによってデータベース内で修正されるデータのフィルタリングを行うことができる検索条件を示す追加の SQL 文です。True を返すとトリガが実行されるこの SQL 検索条件文を記述することができます。つまり、この文が True を返す場合にだけ変更が行われ、False を返す場合には変更は行われません。

ストアード プロシージャの作成

Data Modeler の用語では、「ストアード プロシージャ」は、ストアード プロシージャを指す場合とストアード ファンクションを指す場合があります。ストアード プロシージャによって、モデルに同じ SQL 文が出現する回数を減らすことができるほか、アプリケーション レベルで SQL 文をコーディングする代わりに、名前を使って SQL 文をアプリケーションから呼び出すことができます。

ストアード プロシージャを作成すると、ストアード プロシージャ コンテナが自動的に作成され、ストアード プロシージャを割り当てるスキーマが示されます。

言語

ストアード プロシージャのコードを記述するのに使う言語を決める場合、内部か外部かを決める必要があります。内部のストアード プロシージャは、データベースによってサポートされるので、そのコードは SQL でのみ記述できます。外部のストアード プロシージャはアプリケーションによってサポートされるので、Java や C 言語から選ぶことができます。外部のストアード プロシージャを使う場合には、ストアード プロシージャに、パスやライブラリ名で構成される外部名を割り当てることができます。

パラメータ

「パラメータ」は、プロシージャまたはファンクションの変数です。各パラメータには、データ型、方向、およびデフォルト値を割り当てることができます。パラメータの方向は、パラメータが入力値か出力値か、あるいは両方かによって決まります。

ストアード プロシージャ

値を返さないルーチンがある場合に、ストアード プロシージャを使います。たとえば、新しい患者の ID が作成されたときに新しい患者のレコードを追加するストアード プロシージャを作成できます。

DB2 のストアード プロシージャで使われるパラメータ スタイル

C 言語で記述されたプロシージャでパラメータを渡し、値を受け取る場合の表記方法として、パラメータ スタイルの DB2Dari を選択できます。

ストアード ファンクション

ルーチンが NULL 値か非 NULL 値を返す場合に、「ストアード ファンクション」を使います。戻り値には、データ型の長さ、精度、スケールのほか、特定のデータ型を指定できます。

ファンクションのパラメータが NULL 値を返した場合に、ファンクションを呼び出すかどうかを決めることもできます。NULL 値を返した場合にファンクションを呼び出さない場合には、[NULL を返す]を使用します。NULL 値かどうかに関わらずファンクションを呼び出す場合には、[プロシージャの呼び出し]を使用します。

Not Deterministic

ファンクションが Not Deterministic ([確定しない]) と指定されている場合、そのファンクションは、値の状態に依存しており、その値の状態が結果に影響するので、結果は常に同じにはなりません。このようなファンクションは、結果の値と、その結果の値に付随するほかの値が存在する可能性がある場合に使われます。

DB2 ストアド ファンクション用のパラメータ スタイル

パラメータ スタイルは、特定のストアド ファンクション用の言語で記述されたファンクションにパラメータを送ったり、ファンクションから戻り値を受け取るための規則です。Data Modeler では、使用するストアド ファンクションの言語に関連した DB2 パラメータ スタイルを選択できます。C 言語の呼び出し形式とリンク形式に合ったファンクションには DB2GNRL を選択できます。Java クラスのメソッドとして定義されているファンクションには DB2SQL を選択できます。

データ モデルを作成するためのリバース エンジニアリング

既存の DBMS データベースや DDL ファイルをリバース エンジニアリングすることで物理データ モデルを作成できます。リバース エンジニアリングを行う前に、既存のデータベースが、使用する DBMS のすべての要件に合っていて、安定した構造になっていることを確認する必要があります。Data Modeler は間違った構造の修復は行いません。これは、テーブル名の長さと列名の長さに関しては特に重要です。

リバース エンジニアリング ウィザード

DBMS データベースや DDL ファイルをリバース エンジニアリングするときには、リバース エンジニアリング ウィザードを使います。このウィザードを使うことで、ユーザーはウィザードの指示に従って、リバース エンジニアリング処理を進めていくことができます。また、オプションとして、インデックス、トリガ、およびストアド プロシージャを新しいスキーマに挿入することもできます。データベースに接続する操作についてもウィザードが案内してくれます。データベースへの接続については、「付録 D」を参照してください。

リバース エンジニアリングの処理中にウィザードがエラーを検出した場合、Data Modeler はエラーを Rose のログに記録して処理を続けます。ウィザードからユーザーに処理の完了が通知されたら、Rose のログを参照してエラーが発生していなかったか確認する必要があります。

DB2 データベースと DDL のリバース エンジニアリング

DB 2 データベースや DDL ファイルをリバース エンジニアリングする場合、Data Modeler はシステム カタログを読み取り、データベース名をスキーマ名として使用して、新しい Data Modeler のスキーマを作成します。Data Modeler は DBMS のデータベース要素を新しいスキーマ内に再度作成します。データベースのコメントはすべてデータ モデル内に文書として再生成されます。各テーブルは、テーブルの列、制約、適切なデータ型の設定などを含め、データ

モデルのスキーマ内に再生成されます。明示的なデータ型は、データ モデルのドメインとして再生成されます。ドメイン名は、明示的なデータ型の名前です。DB2 MVS バージョン 6.x を使用している場合には、ルーチンはデータ モデル内にストアードプロシージャとして再生成され、引数はすべてパラメータになります。

Oracle のデータベースまたは DDL のリバース エンジニアリング

Oracle のデータベースや DDL ファイルをリバース エンジニアリングする場合、Data Modeler はシステム カタログを読み取り、スキーマ名として新しい Data Modeler のスキーマ データベース名を作成します。Data Modeler は DBMS のデータベース要素を新しいスキーマ内に再度作成します。データベースのコメントはすべてデータ モデル内に文書として再生成されます。各テーブルは、テーブルの列、制約、適切なデータ型の設定などを含め、データ モデルのスキーマ内に再生成されます。ユーザー定義のデータ型はすべて、データ モデルのドメインとして再生成されます。ドメイン名は、ユーザー定義のデータ型の名前です。データベースのプロシージャはストアードプロシージャとして再生成されます。

SQL Server のデータベースまたは DDL のリバース エンジニアリング

SQL Server のデータベースや DDL ファイルをリバース エンジニアリングする場合、Data Modeler はシステム カタログを読み取り、スキーマ名としてデータベース名を使用して、新しい Data Modeler のスキーマを作成します。Data Modeler は DBMS のデータベース要素を新しいスキーマ内に再度作成します。データベースのコメントはすべてデータ モデル内に文書として再生成されます。各テーブルは、テーブルの列、制約、適切なデータ型の設定などを含め、データ モデルのスキーマ内に再生成されます。ユーザー定義のデータ型はすべて、データ モデルのドメインとして再生成されます。ドメイン名は、ユーザー定義のデータ型の名前です。ルールはチェック制約としてデータ モデルに再生成されます。

Sybase データベースと DDL のリバース エンジニアリング

Sybase データベースや DDL ファイルをリバース エンジニアリングする場合、Data Modeler はシステム カタログを読み取り、スキーマ名としてデータベース名を使用して、新しい Data Modeler のスキーマを作成します。Data Modeler は DBMS のデータベース要素を新しいスキーマ内に再度作成します。データベースのコメントはすべてデータ モデル内に文書として再生成されます。各テーブルは、テーブルの列、制約、適切なデータ型の設定などを含め、データ モデルのスキーマ内に再生成されます。

リバース エンジニアリング後

リバース エンジニアリングの処理が完了したら、Rose のログでエラーを確認したり、データ モデルを参照できます。データ モデルはデータベースか DDL スキーマに包含されていますが、データ モデルを作成することで、グラフィカルにそのモデルだけを参照できます。データ モデル図を作成すると、データ モデルのエレメントを論理ビューからドラッグして図にドロップしたり、メニュー バーの [表示設定] メニューを使ってテーブルを追加できます。

データ モデル構築後

データ モデルの作成が完了したら、作成したモデルをオブジェクト モデルにマッピングして変換し、データ モデルの設計からアプリケーションを構築することができます。また、フォワードエンジニアリングによって、作成したモデルを実装し、DDL のコードや DBMS データベースを生成することができます。

内容

本章は、次のような構成になっています。

- はじめに (39 ページ)
- データ モデルからオブジェクト モデルへのマッピング (39 ページ)
- データ モデルからオブジェクト モデルへの変換 (47 ページ)

はじめに

物理データ モデルを手動で作成した場合や、既存のデータベースをリバース エンジニアリングして物理データ モデルを作成した場合には、作成した物理データ モデルをオブジェクト モデルにマッピングし、変換することができます。変換後のオブジェクト モデルは、物理データ モデルと同期が取られた堅牢な論理データ モデルとして、あるいはアプリケーション モデルの構造として動作するので、その同じロジックをビジネス プロセスのデータベース内でのモデル化に使用してアプリケーションを構築できます。

データ モデルからオブジェクト モデルへのマッピング

第 3 章では、オブジェクト モデルをデータ モデルにマッピングする Rose の機能について説明しましたが、データ モデルをオブジェクト モデルにマッピングすることもできます。このマッピングにより、オブジェクト モデルのエレメント名にデータ モデルのエレメント名を使ってデータ モデルをオブジェクト モデルに変換できます。データ モデルのエレメントのすべてをオブジェクト モデルにマッピングできますが、ストアド プロシージャ、トリガ、キーの制約、列挙型の文を使わないチェック制約は例外です。

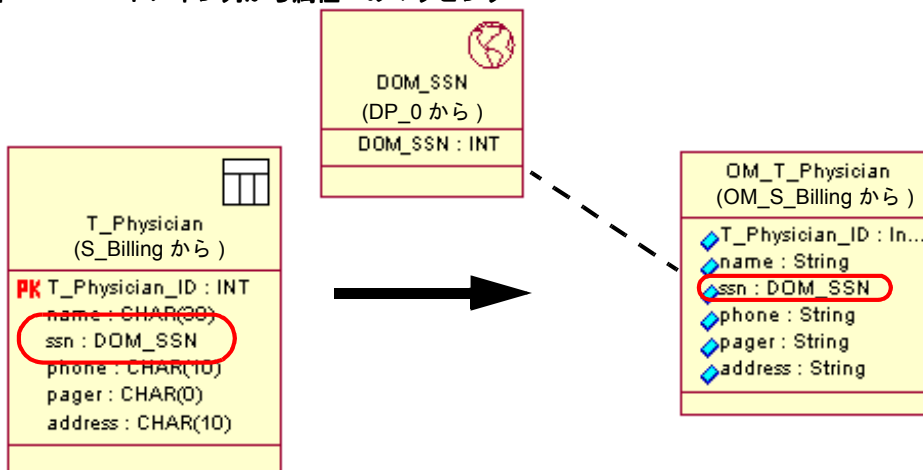
スキーマからパッケージへのマッピング

各データ モデルのスキーマは、論理パッケージにマッピングされます。Data Modeler では、順次 1 つずつスキーマを変換し、変換された各スキーマ用に 1 つのパッケージを生成します。この場合、パッケージ名にはスキーマ名が使われます。

ドメインから属性型へのマッピング

ドメイン自体はオブジェクト モデルのエレメントにはマッピングされません。しかし、変換処理では、ドメインをデータ型として使う列は、ドメイン名を参照する属性型の属性に変換されます。たとえば、ドメイン DOM_SSN を使う列 `ssn` (社会保障番号) をオブジェクト モデルに変換すると、その列は DOM_SSN を属性型として使う属性 `ssn` に変換されます。40 ページの図 21 を参照してください。

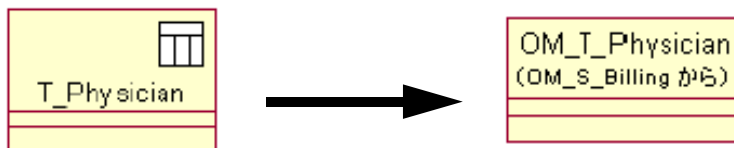
図 21 ドメイン列から属性へのマッピング



テーブルからクラスへのマッピング

テーブルはすべて、オブジェクトモデル内の永続クラスに1対1にマッピングされますが、交差テーブルは例外です。交差テーブルは特殊な関連構造にマッピングされます。直接アプリケーションにマッピング可能なモデルを構築する場合にはクラスを利用できます。

図 22 テーブルからクラスへのマッピング

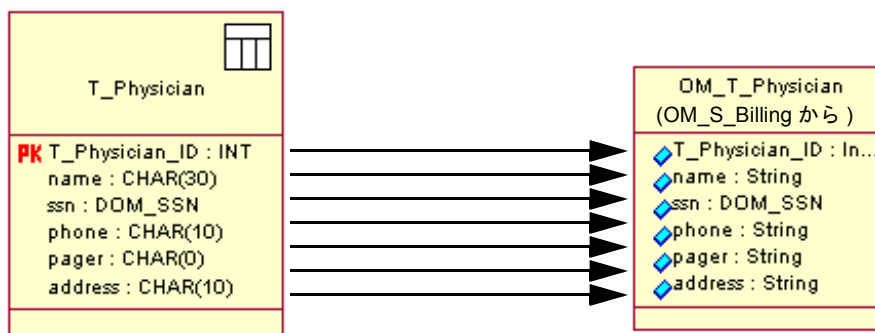


列から属性へのマッピング

列は属性に1対1にマッピングされますが、算出列と外部キー列は例外です。列のデータ型は、適切なオブジェクトモデルの属性型にマッピングされます。データ型から属性型へのマッピングの詳細については、「付録 C」を参照してください。列のデフォルト値は、属性の初期値にマッピングされます。主キー列は、オブジェクト識別子の一部としてタグ付きの属性にマッピングされます。

変換処理では、ほかの特殊な列は無視されます。算出列のような特殊な列は、派生値を使うため無視されます。外部キー列は、別のテーブルにある既存の列を参照する際のリレーションシップのポインタとして動作するため、無視されます。そのため、外部キー列は、冗長であり不要です。

図 23 列から属性へのマッピング



列挙型チェック制約からクラスへのマッピング

「in」のような列挙型の文を含むチェック制約は、<<ENUM>> というステレオタイプでクラスにマッピングされます。これによって、物理モデルに非常によく似た、より堅牢な論理モデルが作成され、ビジネスルールに関する詳細な情報が提供されます。ビジネスアナリストやアプリケーション設計者は、物理データモデルではなく、論理データモデル内の特定の列に使用可能な値だけを知ることができます。

図 24 列挙型チェック制約から <<ENUM>> のクラスへのマッピング

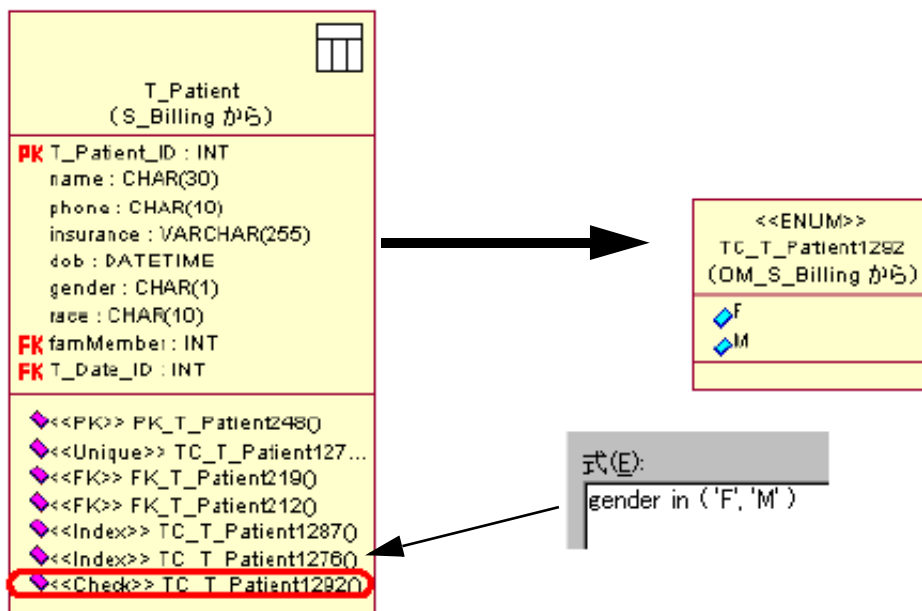
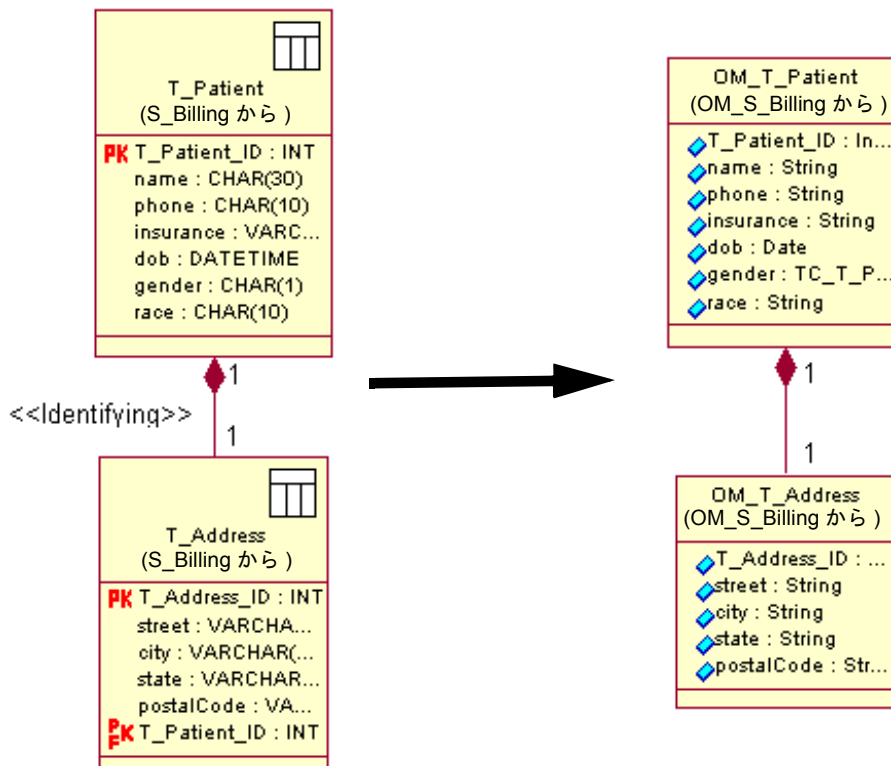


図 24 は、性別に関する列挙型チェック制約を示しています。この新規クラスでは、制約の名前が使われ、列挙型の句の各要素に対応する属性が作成されています。

依存リレーションシップからコンポジット集約へのマッピング

依存リレーションシップはコンポジット集約にマッピングされます。これは、コンポジット集約が全体と部分で構成されており、部分は、対応する全体なしでは存在できないためです。依存リレーションシップでは、この依存関係も埋め込みの外部キー（主キー/外部キーとも呼ばれます）でサポートされています。これは、親テーブルを最初に修正しないと、主キー/外部キーのある子テーブルを修正できないためです。そのため、子テーブル（部分）は、親テーブル（全体）に含まれていないレコードの一部を包含することはできません。

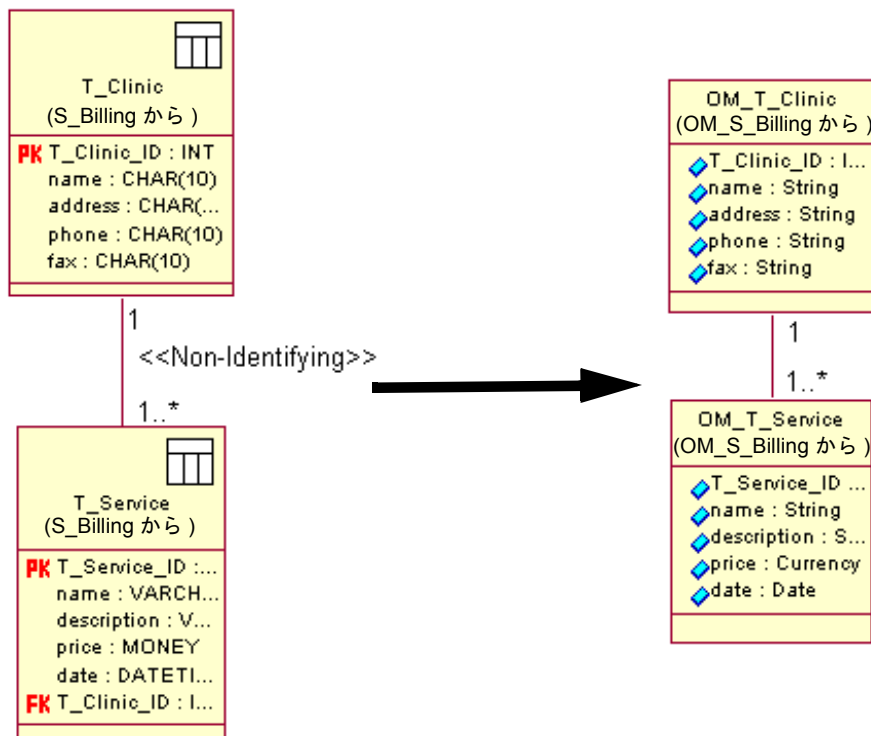
図 25 依存リレーションシップからコンポジット集約へのマッピング



非依存リレーションシップから関連へのマッピング

非依存リレーションシップは関連にマッピングされます。これは、関連がお互いに依存していない2つのクラスを統合するため、つまり、「弱い」関係を示すリレーションシップを使うためです。この弱い関係は外部キーによって識別され、外部キーは主キー制約には埋め込まれません。

図 26 非依存リレーションシップから関連へのマッピング



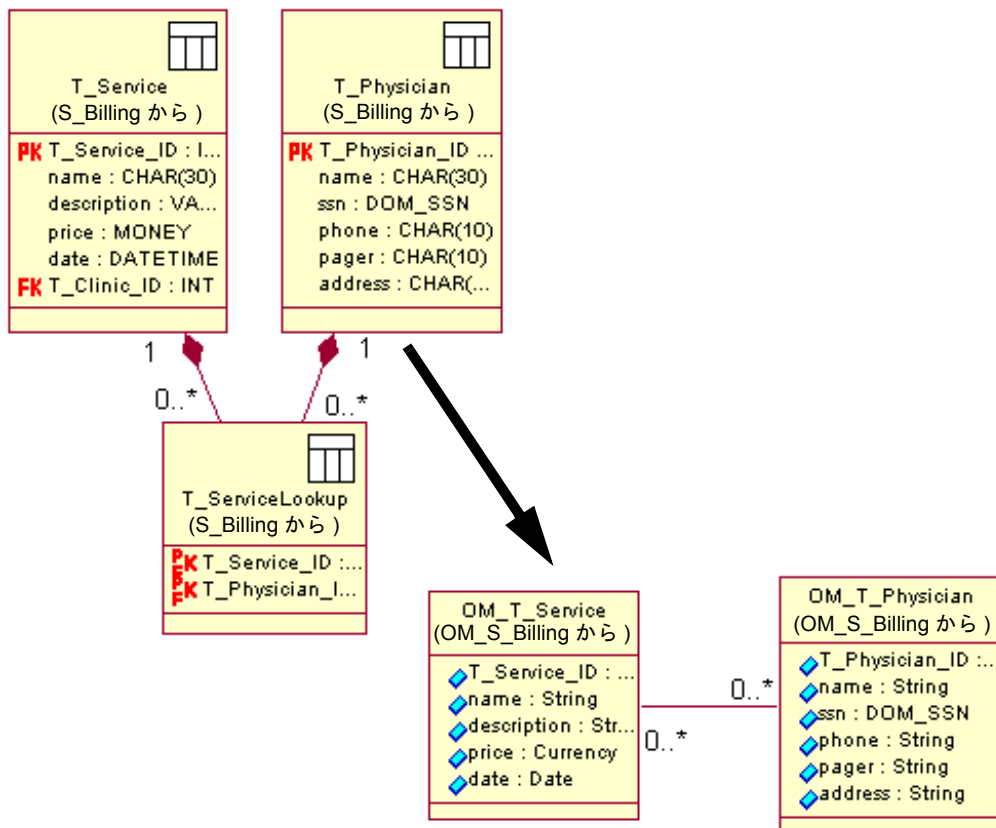
交差テーブルのマッピング

交差テーブルは、その構造自体の問題によってマッピングが困難です。交差テーブルは、最高で3つの異なるオブジェクトモデルのエレメントにマッピングできます。多対多の関連、限定子付き関連、関連クラスの3つです。共通部分は、これら3つのうちの1つか、あるいはすべてにマッピングされます。

多対多の関連

交差テーブルのすべての列が主キー / 外部キーの場合、その交差テーブルはオブジェクトモデルの多対多の関連にマッピングされます。つまり、交差テーブルに含まれているデータはすべて、リレーションシップの中のどちらかのテーブルに関連しています。そのため、このデータを含む列にマッピングされる属性は、関連のどちらかのクラス中に存在する必要がありますが、両方のクラスの中に存在することはありません。

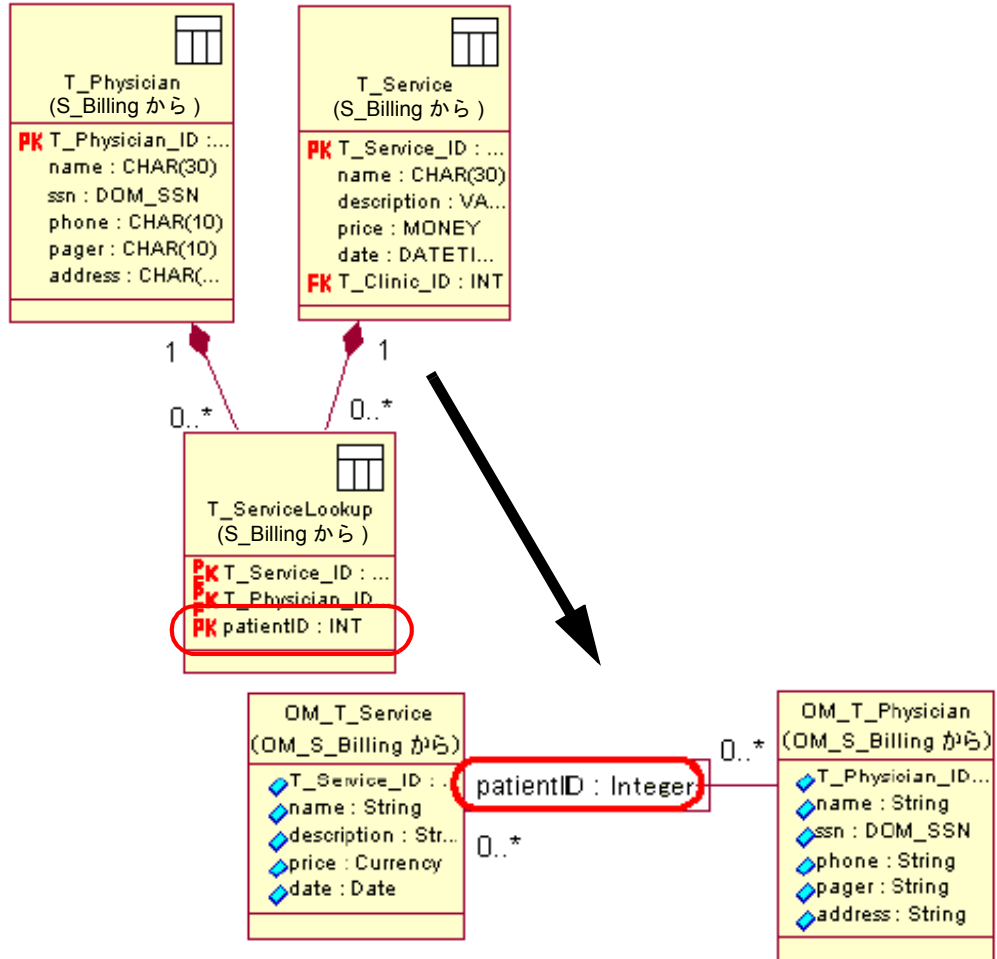
図 27 交差テーブルから多対多の関連へのマッピング



限定子付き関連

交差テーブルに追加の主キーがある場合、その交差テーブルは多対多のリレーションシップの限定子付き関連にマッピングされます。この追加の主キーは、交差テーブル専用です。依存リレーションシップを通して受け取ることはできません。つまり、このキーは主キー/外部キーではありません。交差テーブル自体は、2つのクラス間の関係にマッピングされるので、その関連は、追加の主キーの列を「所有」しています。主キーの列は、関連のフィルタとして動作します。これは、どちらのクラスもこのキーを所有することや共有することができないためです。主キーの列によって、この主キーの情報を含まないインスタンスのフィルタリングが行われます。

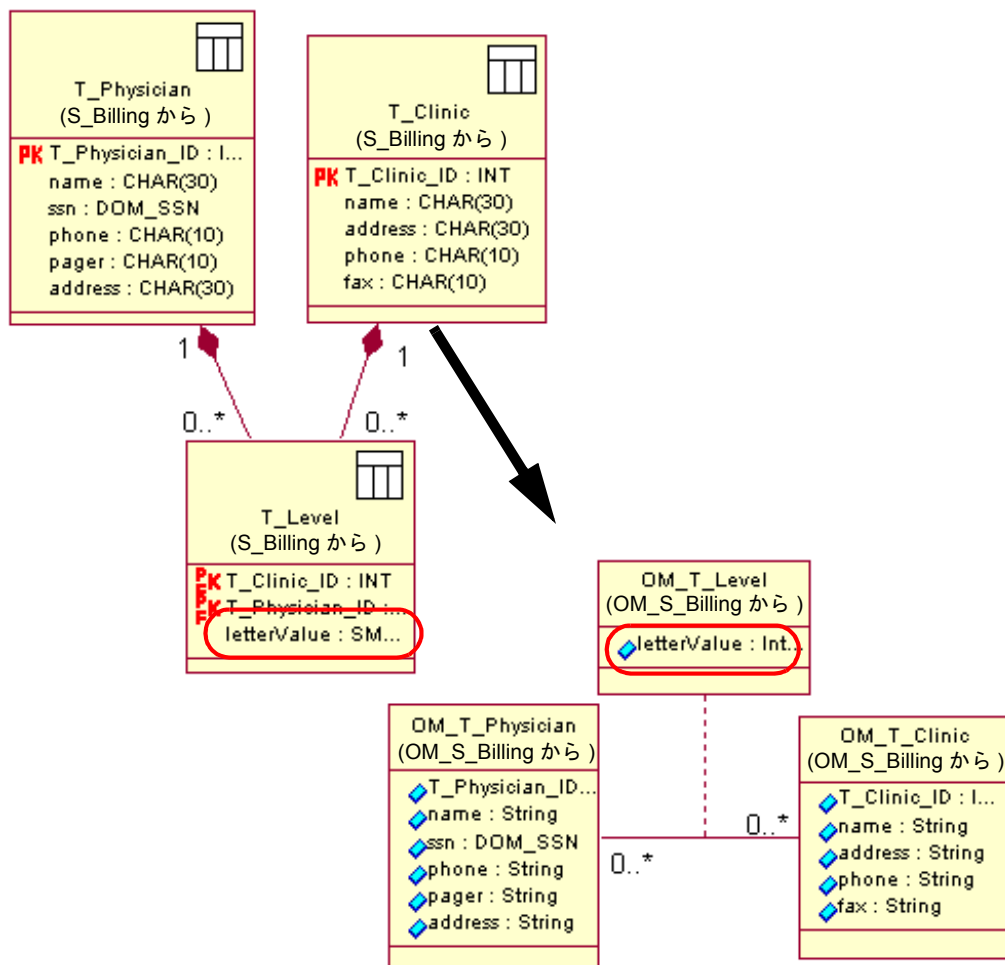
図 28 交差テーブルから限定子付き関連へのマッピング



関連クラス

交差テーブルに主キー、外部キー、および主キー / 外部キーではない追加の列がある場合、その交差テーブルは多対多の関係の関連クラスにマッピングされます。この追加の列は、リレーションシップの外にある交差テーブルを識別するので、リレーションシップによって関係付けられているどちらのテーブルもその列を所有することはできませんが、それでもなお2つのテーブルは列を共有します。関連クラスには、名前として交差テーブルの名前が使われます。

図 29 交差テーブルから関連クラスへのマッピング



スーパータイプ/サブタイプ構造から継承構造へのマッピング

スーパータイプ/サブタイプ構造は、継承構造にマッピングできます。これは、汎化とも呼ばれます。変換処理では、Data Modeler はスーパータイプ/サブタイプ構造を別々のクラスとして変換します。変換処理が終わった後、手動でコンポジット関係を削除し、継承ツリーと呼ばれる汎化構造で置換できます。

データ モデルからオブジェクト モデルへの変換

オブジェクト モデルで必要な結果が得られるように、データ モデルのカスタマイズや確認を行った後、データ モデルをオブジェクト モデルに変換できます。

なぜ変換するのか

第3章で説明したように、変換は、物理データ モデルと論理データ モデルの同期を取る場合に便利です。また、開発チーム メンバー間でコミュニケーションを取るための手段になります。

また、データベース構造からアプリケーションを構築するために変換することもできます。オブジェクト モデルに変換したら、オブジェクト モデルのクラスやパッケージ全体を適切なコンポーネント言語に割り当てることができます。その後、そのコンポーネント言語でアプリケーション コードを生成するのに必要な追加の境界クラスとインターフェイスを作成できます。

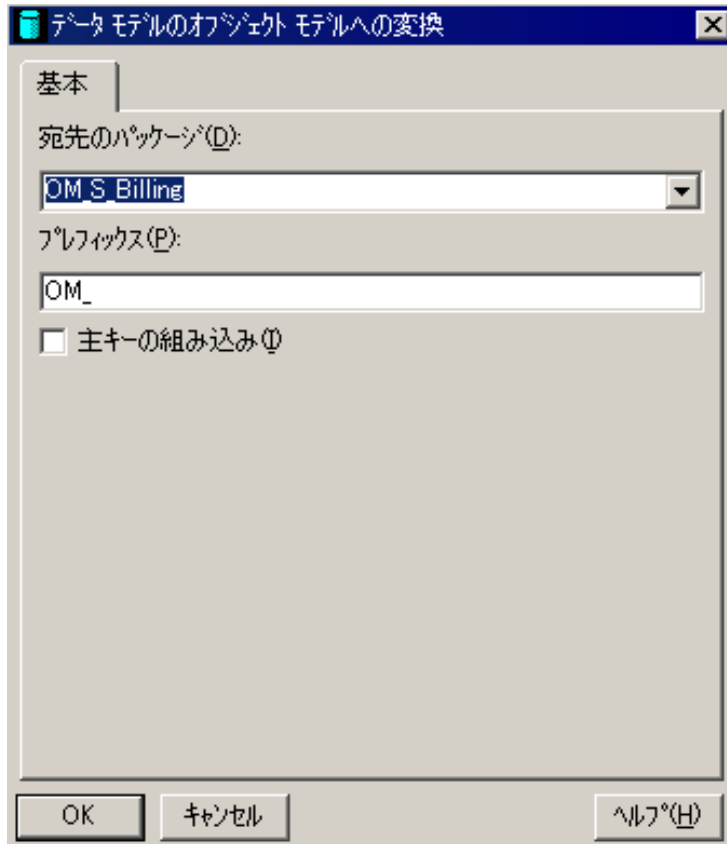
変換処理

データ モデルのエレメントはすべて、マッピングされているオブジェクト モデルに変換されます。変換処理の中で **Data Modeler** は、データ モデルをオブジェクト モデルの [分析] コンポーネント言語にのみ変換します。別のコンポーネント言語を使う場合には、変換処理が完了した後、手動でパッケージとクラスを別の言語に再度割り当てる必要があります。

[データ モデルのオブジェクト モデルへの変換] ダイアログ ボックス

データ モデルを変換する処理は、[データ モデルのオブジェクト モデルへの変換] ダイアログ ボックスから始めます。このダイアログ ボックスでは、オブジェクト モデルに対応するスキーマ名を選択または指定できます。そのほかにも、各オブジェクト モデルのクラスに対するプレフィックスを指定したり、主キーの列を変換の対象にするかどうかを指定できます。(オブジェクト モデルのクラスとデータ モデルのテーブルを区別するためにプレフィックスを使うことをお勧めします。)

図 30 [データ モデルのオブジェクト モデルへの変換] ダイアログ ボックス



データ モデル エLEMENT の変換

実際の変換処理では、Data Modeler はデータ モデルのスキーマを読み込み、パッケージを生成するか、変換しているスキーマと同じ名前の既存のパッケージが存在する場合にはそれを更新します。次に Data Modeler は、データ モデル内の有効な各テーブル (交差テーブルを除く) に永続クラスを作成します。

この時点で、外部キー列はすべてクラスから削除されており、ほかの列はすべて属性に変換されています。[データ モデルのオブジェクト モデルへの変換] ダイアログ ボックスで主キー列を変換の対象にするように指定した場合は、主キー列も属性に変換されます。制約はすべて属性から削除されますが、主キー列は例外です。主キー列は、オブジェクト識別子の一部としてタグが付けられます。また、列挙型チェック制約もすべて、ステレオタイプ <<ENUM>> の付いたクラスに変換されます。

クラスが確立されると、Data Modeler はそのリレーションシップを、リレーションシップの種類に基づいて関連あるいはコンポジット集約に変換し、データ モデルで指定された適切な多重度を割り当てます。

変換処理の後

変換処理が完了すると、必要な変更をオブジェクト モデルに加えることができます。たとえば、継承構造の変更などです。また、データ モデルのテーブルの仕様または列の仕様にある [マップ元] プロパティを使って、データ モデルからオブジェクト モデルへのマッピングを参照することもできます。[マップ元] プロパティの詳細については、「第 3 章 論理データ モデリング」を参照してください。

内容

本章は、次のような構成になっています。

- はじめに (51 ページ)
- データ モデルのフォワード エンジニアリング (51 ページ)
- データ モデルの比較と同期 (54 ページ)

はじめに

データベースのモデル化における最後のステップは、物理データ モデルの実装です。物理データ モデルは、フォワード エンジニアリングを行って新しい DDL やデータベースを作成することで実装できます。あるいは、特定のテーブルの変更だけを行い、モデルを既存の DDL やデータベースと比較して同期を取ることができます。

データ モデルのフォワード エンジニアリング

データ モデルからデータベースへのフォワード エンジニアリングを行う前に、データ モデルが、使用する DBMS のすべての要件 (特にテーブル名や列名の長さなど) に合っていることを確認する必要があります。データ モデルの構造が安定していることも確認する必要があります。Data Modeler は、間違ったデータ構造の修正は行いません。

フォワード エンジニアリング ウィザード

データ モデルをデータベースや DDL ファイルにフォワード エンジニアリングするときには、フォワード エンジニアリング ウィザードを使います。このウィザードを使うことで、ユーザーはウィザードの指示に従って、フォワード エンジニアリング処理を進めていくことができます。また、DBMS に接続する操作についてもウィザードが案内してくれます。使用する DBMS への接続については、「付録 D」を参照してください。

フォワード エンジニアリングの処理中にウィザードがエラーを検出した場合、Data Modeler はエラーを Rose のログに記録して処理を続けます。ウィザードからユーザーに処理の完了が通知されたら、Rose のログを参照してエラーが発生していなかったか確認する必要があります。

また、ウィザードを使うと、コメントのほか、テーブル、インデックス、ストアド プロシージャ、トリガに対して CREATE 文を含めることや無視することができます。また、完全な名前や引用符、drop 文などを含めることや無視することができます。これらはすべて重要ですが、完全な名前や引用符、drop 文は特に、実行形式の DDL に大きな影響を与える場合があります。

完全な名前

完全な名前は、DDL に対して大きな影響があります。それは、DDL で完全な名前を使うと、DDL を実行するためにほかのツールを使うことができなくなり柔軟性が制限されるからです。生成された DDL を実行することでデータベースへのフォワードエンジニアリングを行う場合にだけ、完全な名前を使うようにしてください。

引用符

引用符は、DDL に対して大きな影響があります。引用符を使うことで、空白などの標準的なコードセット以外の文字を使うことができます。また、エレメント名を引用符で囲むことで、エレメント名の一部に 2 バイト コードセット (DBCS) を使うことができます。これは、ビジネス上必要な場合、あるいは国際化文字で作業を行っている場合など、特定の命名規則に従う必要がある場合に便利です。

drop 文

CREATE Table 文と組み合わせて使われる drop 文は、DDL に対して大きな影響があります。フォワードエンジニアリングの際に drop 文が正しく指定されていないと、エラーが発生したり、DDL に復元不可能な上書きが行われることがあります。テーブルと drop 文があると、すべてのテーブルが DDL スクリプトの最初で削除され、次に CREATE Table 文が生成されます。テーブルが含まれており、drop 文が含まれていない場合には、CREATE Table 文は既存の DDL スクリプトの最後から始まります。この場合に、DDL にテーブルが重複しているとエラーが発生します。テーブルが含まれておらず、drop 文が含まれている場合には、すべてのテーブルが DDL スクリプトの最初で削除され、CREATE Table 文は生成されません。そのため、DDL が空になります。すべての DBMS で drop 文がサポートされていますが、SQL サーバーと Sybase は例外で、条件付き drop 文が使われます。条件付き drop 文の詳細については、53 ページの「SQL Server のデータベースまたは DDL へのフォワードエンジニアリング」または 53 ページの「Sybase のデータベースまたは DDL へのフォワードエンジニアリング」を参照してください。

ANSI SQL 92 の DDL へのフォワードエンジニアリング

データ モデルを ANSI SQL 92 の DDL ファイルにフォワードエンジニアリングする場合、Data Modeler は、データ モデルのスキーマを読み込み、モデル化された各エレメントに対して DDL 文を生成します。文書はすべてコメントとして再生成されます。しかし、ストアドプロシージャとトリガはフォワードエンジニアリングされないので、DDL には含まれません。「;」は、DDL スクリプトの文を分割するための区切り文字の働きをします。

DB2 のデータベースまたは DDL へのフォワードエンジニアリング

データ モデルを DB2 のデータベースや DDL ファイルにフォワードエンジニアリングする場合、Data Modeler はデータ モデルのスキーマを読み込み、システム カタログを生成します。文書はすべて、DBMS データベース内にコメントとして再生成されます。各テーブルは、テーブルの列、選択された制約、データ型の設定などを含め、データベースや DDL 中に再生成されます。「;」は DDL スクリプトの文を分割するための区切り文字の働きをします。

ドメインは、固有のデータ型として再生成されます。ドメインには、サーバー生成のタグが付けられ、その後フォワードエンジニアリングされる際に、CREATE DISTINCT TYPE 文が作成されてドメイン名やデータ型などの情報を提供します。ドメインにサーバー生成のタグが付けられていない場合には、フォワードエンジニアリングされた際に、CREATE Table 文のドメインの列ではデータ型だけが提供され、ドメイン名には関連付けられません。

Oracle のデータベースまたは DDL へのフォワードエンジニアリング

データ モデルを Oracle のデータベースや DDL ファイルにフォワードエンジニアリングする場合、Data Modeler はデータ モデルのスキーマを読み込み、ユーザー カタログを生成します。テーブルと列に関する文書はすべて、DBMS データベース内にコメントとして再生成されます。各テーブルは、テーブルの列、選択された制約、データ型の設定などを含め、データベースか DDL 中に再生成されます。クラスタ化されたインデックスが、インデックス構成テーブル (IOT) として再生成されます。DDL スクリプトの文を分割するための区切り文字として、「;」または「/」を指定できます。

SQL Server のデータベースまたは DDL へのフォワードエンジニアリング

データ モデルを SQL Server のデータベースや DDL ファイルにフォワードエンジニアリングする場合、Data Modeler はデータ モデルのスキーマを読み込み、システム カタログを生成します。文書はフォワードエンジニアリングされません。しかし、各テーブルは、テーブルの列、選択された制約、データ型の設定などを含め、データベースや DDL 中に再生成されます。「Go」は DDL スクリプトの文を分割するための区切り語の働きをします。

ドメインは、ユーザー定義のデータ型として再生成されます。ドメインに列挙型チェック制約やデフォルト値が含まれている場合、チェック制約の 1 つがルールとしてフォワードエンジニアリングされ、それ以外はチェック制約のまま残ります。ドメインのデフォルト値は、ユーザー定義のデータ型のデフォルトになります。ドメインにサーバー生成のタグが付けられている場合は、ドメインがフォワードエンジニアリングされる際に、追加の EXEC sp_addtype 文が作成され、ドメイン名、データ型、長さなどの情報が提供されます。また、ドメインにサーバー生成のタグが付けられていない場合は、ドメインがフォワードエンジニアリングされる際に、CREATE Table 文中のドメインの列ではデータ型だけが提供され、ドメイン名には関連付けられません。

フォワードエンジニアリング中に drop 文があると、drop 文は、条件付き drop 文として動作します。テーブルが存在する場合には削除されます。テーブルが存在しない場合には、drop 文は作成されないので DDL スクリプトでエラーは発生しません。

Sybase のデータベースまたは DDL へのフォワードエンジニアリング

データ モデルを Sybase のデータベースや DDL ファイルにフォワードエンジニアリングする場合、Data Modeler はデータ モデルのスキーマを読み込み、システム カタログを生成します。文書はフォワードエンジニアリングされません。しかし、各テーブルは、テーブルの列、選択された制約、データ型の設定などを含め、データベースや DDL 中に再生成されます。「Go」は DDL スクリプトの文を分割するための区切り語の働きをします。

ドメインは、ユーザー定義のデータ型として再生成されます。ドメインに列挙型チェック制約やデフォルト値が含まれている場合、チェック制約の 1 つがルールとしてフォワードエンジニアリングされ、それ以外はチェック制約のまま残ります。ドメインのデフォルト値は、ユーザー定義のデータ型のデフォルトになります。ドメインにサーバー生成のタグが付けられている場合は、ドメインがフォワードエンジニアリングされる際に、追加の EXEC sp_addtype 文が作成され、ドメイン名、データ型、長さなどの情報が提供されます。また、ドメインに

サーバー生成のタグが付けられていない場合は、ドメインがフォワードエンジニアリングされる際に、CREATE Table 文中のドメインの列ではデータ型だけが提供され、ドメイン名には関連付けられません。

フォワードエンジニアリング中に drop 文があると、drop 文は、条件付き drop 文として動作します。テーブルが存在する場合には削除されます。テーブルが存在しない場合には、drop 文は作成されないため DDL スクリプトでエラーは発生しません。

データ モデルの比較と同期

既存のデータベースに対応するデータ モデルを実装する場合には、比較と同期を行うことをお勧めします。これは、スキーマ内のすべてのテーブルに変更が行われるフォワードエンジニアリングとは異なり、比較と同期では、テーブル単位の変更や差異があった場合にのみデータベースの更新が行われるためです。データ モデルからデータベースへの同期ウィザードを使って、データ モデルをデータベースや DDL ファイルと比較して同期を取ります。

同期ウィザード

同期ウィザードを使うと、ユーザーはウィザードの指示に従って同期処理を進めていくことができます。また、オプションとして、コメント、インデックス、トリガ、ストアドプロシージャを含めることもできます。DDL を生成する場合は、完全な名前と引用符も含めることができます。DBMS データベースに接続する操作についてもウィザードが案内してくれます。データベースへの接続については、「付録 D」を参照してください。ウィザードによってこの処理は、比較と同期の 2 つの部分に分割されます。

比較の処理

比較の処理では、Data Modeler がデータ モデルのスキーマと DBMS データベースを読み込みます。次に、Data Modeler はこれら 2 つのスキーマを比較し、データベースの各要素とデータ モデルの各エレメントに関して、その相違を同期ウィザード中に視覚的に一覧表示します。この 2 つの一覧は、横に並べて配置された上で、テーブルごとにグループ化され、データ モデルのエレメント名と DBMS データベースの要素名の対応が取られます。DBMS データベース内には存在しないエレメントがデータ モデルに存在する場合、そのエレメントはデータ モデルの一覧の下に 1 行で表示され、もう 1 つの一覧はブランクになります。列などの、テーブルのエレメントに相違がある場合は、相違のある列と列が属するテーブルだけが一覧に表示され、テーブルのほかのエレメントは一覧には表示されません。

ウィザードでは、各行の項目ごとにアクションを設定できます。このアクションは、データ モデル内あるいはデータベース内の、エレメント (要素) が更新される場所を示しています。アクションを設定しない場合には、データベースやデータ モデルに変更は行われません。データベースのスキーマをデータ モデルのエレメントで更新したい場合には、このアクションを Export に設定します。ただし、データベースの列に対して行われた変更をすべて考慮しないと、オリジナルのテーブルをデータベースから最初に削除することになり、データを失う結果になる場合があります。データ モデルのスキーマをデータベースのスキーマ内の要素で更新する場合は、このアクションを Import に設定します。データ モデルとデータベースのスキーマの両方からエレメント (要素) を削除する場合は、このアクションを Delete に設定します。テーブル全体に適用されるアクションを設定し、テーブルの特定のエレメントは同期させたくない場合は、各エレメントに対するアクションを Ignore に設定します。

同期ウィザードを使うと、比較の結果を .txt ファイルに保存することもできます。この .txt ファイルは、そのまま使うことや、ワープロや表計算ツールで開いて使うことができます。

同期処理

各エレメントにアクションを設定した後、ウィザードでは実際の同期処理を始める前に変更内容を確認できます。また、ウィザードによって DDL のパスを設定したり、DDL が生成された後の実行時オプションを設定することができます。同期処理では、Data Modeler が同期オプションを読み込み、指定されたアクションに従って、データモデルと DBMS データベースを更新します。データベースや DDL に変更がある場合は、これらの変更だけが加えられた新しい DDL が作成されるので、この DDL を実行して既存の DBMS データベースを更新することができます。

付録 A

UML データ モデリング プロファイル

以下の表は、データベースの概念と、既存の UML 構造に割り当てられたデータ モデリング ステレオタイプの対応を示しています。これは、UML データ モデリング プロファイルと呼ばれます。

表 2 UML データ モデリング プロファイルのステレオタイプ

データベースの概念	UML の構造	データ モデリングのステレオタイプ
データベース	コンポーネント	<<Database>>
スキーマ	パッケージ	<<Schema>>
テーブル (エンティティ)	クラス	<<Table>>
ドメイン	クラス	<<Domain>>
リレーションシップ	関連	<<Non-Identifying>>
リレーションシップ (強い関係)	コンポジット集約	<<Identifying>>
インデックス	操作	<<Index>>
主キー制約	操作	<<PK>>
外部キー制約	操作	<<FK>>
一意キー制約	操作	<<Unique>>
チェック制約	操作	<<Check>>
トリガ	操作	<<Trigger>>
ストアドプロシージャ	ユーティリティ クラス	<<SP>>

付録 B

オブジェクト モデルからデータ モデルへのデータ型のマッピング

ここには、サポートされているオブジェクト モデルの言語と、サポートされている DBMS のデータ型の対応表を示します。これは、Data Modeler がオブジェクト モデルをデータ モデルに変換する際のマッピングと同じものです。

ここには、以下の表があります。

- [分析] 言語のオブジェクト モデルからデータ モデルへのデータ型のマッピング - 表 3 (60 ページ)
- Java のオブジェクト モデルからデータ モデルへのデータ型のマッピング - 表 4 (61 ページ)
- Visual Basic のオブジェクト モデルからデータ モデルへのデータ型のマッピング - 表 5 (62 ページ)

表 3 [分析] 言語のオブジェクト モデルからデータ モデルへのデータ型のマッピング

[分析]	ANSI SQL 92	DB2	Oracle	SQL Server	Sybase
STRING	VARCHAR (255)	VARCHAR (255)	VARCHAR (255)	VARCHAR (255)	VARCHAR (255)
INTEGER	INTEGER	INTEGER	NUMBER (10,0)	INT	INT
DOUBLE	DOUBLE PRECISION	DOUBLE	FLOAT	FLOAT	FLOAT
DATE	DATE	TIMESTAMP	DATE	DATE	DATE
BOOLEAN	SMALLINT	SMALLINT	NUMBER (1,0)	BIT	BIT
BYTE	SMALLINT	SMALLINT	NUMBER (3,0)	SMALLINT	SMALLINT
SINGLE	FLOAT	REAL	FLOAT	FLOAT	FLOAT
LONG	INTEGER	BIGINT	NUMBER (20,0)	INT	INT
CURRENCY	DOUBLE PRECISION	DOUBLE	FLOAT	MONEY	MONEY

表 4 Java のオブジェクト モデルからデータ モデルへのデータ型のマッピング

Java	ANSI SQL 92	DB2	Oracle	SQL Server	Sybase
LONG	INTEGER	BIGINT	NUMBER (20,0)	INT	INT
CHAR	CHAR	GRAPHIC (1)	CHAR	CHAR (1)	CHAR (1)
INT	INTEGER	INTEGER	NUMBER (10,0)	INT	INT
SHORT	SMALLINT	SMALLINT	NUMBER (5,0)	SMALLINT	SMALLINT
DOUBLE	FLOAT	DOUBLE	FLOAT	FLOAT	FLOAT
BYTE	SMALLINT	SMALLINT	NUMBER (3,0)	SMALLINT	SMALLINT
FLOAT	FLOAT	REAL	FLOAT	FLOAT (0)	FLOAT
BOOLEAN	SMALLINT	SMALLINT	NUMBER (1,0)	BIT	BIT
java.util.Date	DATE	TIMESTAMP	DATE	DATETIME	DATETIME
java.lang.String	VARCHAR (255)	VARCHAR (255)	VARCHAR2 (255)	CHAR (255)	CHAR (255)

表 5 Visual Basic のオブジェクト モデルからデータ モデルへのデータ型のマッピング

Visual Basic	ANSI SQL 92	DB2	Oracle	SQL Server	Sybase
STRING	VARCHAR (255)	VARCHAR (255)	VARCHAR (255)	VARCHAR (255)	VARCHAR (255)
INTEGER	INTEGER	INTEGER	NUMBER (10,0)	INT	INT
DOUBLE	DOUBLE PRECISION	DOUBLE	FLOAT	FLOAT	FLOAT (0)
COLLECTION	SMALLINT	SMALLINT	NUMBER (5)	SMALLINT	SMALLINT
DECIMAL	FLOAT	REAL	FLOAT	FLOAT	FLOAT
BYTE	DATE	TIMESTAMP	DATE	DATETIME	DATETIME
SINGLE	FLOAT	REAL	FLOAT	FLOAT	FLOAT
BOOLEAN	SMALLINT	SMALLINT	NUMBER (1,0)	BIT	BIT
BYTE	SMALLINT	SMALLINT	NUMBER (3,0)	SMALLINT	SMALLINT
LONG	INTEGER	BIGINT	NUMBER (20,0)	INT	INT
CURRENCY	DOUBLE PRECISION	DOUBLE	FLOAT	MONEY	MONEY

付録 C

データ モデルからオブジェクト モデルへのデータ型のマッピング

ここには、サポートされている DBMS の各データ型と、[分析]、Java、および Visual Basic の各言語との対応表を示します。Data Modeler は、[分析] 言語への変換だけを行います。ほかの言語のマッピングは、参考としてのみ示してあります。

ここには、以下の表があります。

- ANSI SQL 92 のデータ モデルからオブジェクト モデルへのデータ型のマッピング - 表 6 (64 ページ)
- DB2 のデータ モデルからオブジェクト モデルへのデータ型のマッピング - 表 7 (65 ページ)
- Oracle のデータ モデルからオブジェクト モデルへのデータ型のマッピング - 表 8 (66 ページ)
- SQL Server のデータ モデルからオブジェクト モデルへのデータ型のマッピング - 表 9 (67 ページ)
- Sybase のデータ モデルからオブジェクト モデルへのデータ型のマッピング - 表 10 (69 ページ)

表 6 ANSI SQL 92 のデータ モデルからオブジェクト モデルへのデータ型のマッピング

ANSI SQL 92	[分析]	VB	Java
BIT	BOOLEAN	BOOLEAN	BOOLEAN
BIT VARYING	BYTE	BYTE	BYTE
CHAR	STRING	STRING	STRING
DATE	DATE	DATE	DATE
DECIMAL	BYTE INTEGER LONG SINGLE DOUBLE	BYTE INTEGER LONG SINGLE DOUBLE	BYTE SHORT INTEGER LONG FLOAT DOUBLE
DOUBLE PRECISION	DOUBLE	DOUBLE	DOUBLE
FLOAT	SINGLE DOUBLE	SINGLE DOUBLE	SINGLE DOUBLE
INTEGER	INTEGER	INTEGER	INTEGER
INTERVAL	STRING	STRING	STRING
REAL	SINGLE	SINGLE	FLOAT
SMALLINT	INTEGER	INTEGER	INTEGER
TIME	DATE	DATE	DATE
TIME WITH TIME ZONE	DATE	DATE	DATE
TIMESTAMP	DATE	DATE	DATE
TIMESTAMP WITH TIME ZONE	DATE	DATE	DATE
VARCHAR	STRING	STRING	STRING

表 7 DB2 のデータ モデルからオブジェクト モデルへのデータ型のマッピング

DB2	[分析]	VB	Java
SMALLINT	BYTE OBJECT VARIANT	BYTE OBJECT COLLECTION DECIMAL	BYTE SHORT
INTEGER	INTEGER BOOLEAN	INTEGER BOOLEAN	BOOLEAN INT
BIGINT	LONG	LONG	LONG
REAL	SINGLE	SINGLE	FLOAT
DOUBLE	DOUBLE CURRENCY	DOUBLE	DOUBLE
DECIMAL	BYTE INTEGER LONG SINGLE DOUBLE	BYTE INTEGER LONG SINGLE DOUBLE	BYTE INTEGER LONG SINGLE DOUBLE
CHARACTER	STRING	STRING	STRING
VARCHAR	STRING	STRING	STRING
LONG VARCHAR	STRING	STRING	STRING
GRAPHIC	STRING	STRING	CHAR
VARGRAPHIC	STRING	STRING VARIANT	STRING
LONG VARGRAPHIC	STRING	STRING	STRING
DATE	DATE	DATE	DATE
TIME	DATE	DATE	DATE
TIMESTAMP	DATE	DATE	DATE
BLOB	STRING	STRING	STRING
CLOB	STRING	STRING	STRING
DBCLOB	STRING	STRING	STRING

表 8 Oracle のデータ モデルからオブジェクト モデルへのデータ型のマッピング

Oracle	[分析]	VB	Java
CHAR	STRING	STRING	java.lang.String
VARCHAR2	STRING	STRING	java.lang.String
NCHAR	STRING	STRING	java.lang.String
NVARCHAR2	STRING	STRING	java.lang.String
NUMBER (1,0)	BYTE	BYTE	BYTE
NUMBER (3,0)	BYTE	BYTE	BYTE
NUMBER (5,0)	INTEGER	INTEGER	SHORT
NUMBER (10,0)	LONG	LONG	INT
NUMBER (20,0)	LONG	LONG	LONG
NUMBER (m, n) (n は非ゼロ)	SINGLE DOUBLE	SINGLE DOUBLE	FLOAT DOUBLE
FLOAT (n)	SINGLE DOUBLE	SINGLE DOUBLE	FLOAT DOUBLE
LONG	STRING	STRING	java.lang.String
LONG RAW	STRING	STRING	java.lang.String
RAW	STRING	STRING	java.lang.String
DATE	DATE	DATE	java.util.Date
BLOB	STRING	STRING	java.lang.String
CLOB	STRING	STRING	java.lang.String
NCLOB	STRING	STRING	java.lang.String
BFILE	STRING	STRING	java.lang.String
ROWID	LONG	LONG	LONG

表 9 SQL Server のデータ モデルからオブジェクト モデルへのデータ型のマッピング

SQL Server	[分析]	VB	Java
BIT	BOOLEAN	BOOLEAN	BOOLEAN
INT	INTEGER LONG	INTEGER LONG	INT LONG
SMALLINT	BYTE OBJECT VARIANT	BYTE OBJECT COLLECTION	BYTE SHORT
TINYINT	BYTE	BYTE	BYTE
DECIMAL	BYTE INTEGER LONG SINGLE DOUBLE	BYTE INTEGER LONG SINGLE DOUBLE	BYTE INTEGER LONG SINGLE DOUBLE
NUMERIC	BYTE INTEGER LONG SINGLE DOUBLE	BYTE INTEGER LONG SINGLE DOUBLE	BOOLEAN
REAL	SINGLE	SINGLE	FLOAT
MONEY	CURRENCY	CURRENCY	DOUBLE
SMALLMONEY	CURRENCY	CURRENCY	DOUBLE
DATETIME	DATE	DATE	DATE
SMALLDATETIME	DATE	DATE	DATE
TIMESTAMP	LONG	LONG	LONG
UNIQUEIDENTIFIER	LONG	LONG	LONG
CHAR	STRING	STRING	STRING
VARCHAR	STRING	STRING	STRING
TEXT	STRING	STRING	STRING
BINARY	STRING	STRING	STRING
VARBINARY	STRING	STRING	STRING
IMAGE	STRING	STRING	STRING

SQL 7.0 の場合におけるデータ モデルからオブジェクト モデルへの属性型のマッピング

SQL Server	[分析]	VB	Java
NTEXT (バージョン 7.0 のみ)	STRING	STRING	STRING
NVARCHAR (バージョン 7.0 のみ)	STRING	STRING VARIANT	STRING
NCHAR (バージョン 7.0 のみ)	STRING	STRING	CHAR

表 10 Sybase のデータ モデルからオブジェクト モデルへのデータ型のマッピング

Sybase	[分析]	VB	Java
BIT	BOOLEAN	BOOLEAN	BOOLEAN
INT	INTEGER LONG	INTEGER LONG	INT LONG
SMALLINT	BYTE OBJECT VARIANT	BYTE OBJECT COLLECTION	BYTE SHORT
TINYINT	BYTE	BYTE	BYTE
DECIMAL	BYTE INTEGER LONG SINGLE DOUBLE	BYTE INTEGER LONG SINGLE DOUBLE	BYTE INTEGER LONG SINGLE DOUBLE
NUMERIC	BYTE INTEGER LONG SINGLE DOUBLE	BYTE INTEGER LONG SINGLE DOUBLE	BOOLEAN
REAL	SINGLE	SINGLE	FLOAT
MONEY	CURRENCY	CURRENCY	DOUBLE
SMALLMONEY	CURRENCY	CURRENCY	DOUBLE
DATETIME	DATE	DATE	DATE
SMALLDATETIME	DATE	DATE	DATE
TIMESTAMP	LONG	LONG	LONG
UNIQUEIDENTIFIER	LONG	LONG	LONG
CHAR	STRING	STRING	STRING
VARCHAR	STRING	STRING	STRING
TEXT	STRING	STRING	STRING
BINARY	STRING	STRING	STRING
VARBINARY	STRING	STRING	STRING
IMAGE	STRING	STRING	STRING

付録 D

データベースへの接続

ここでは、Data Modeler のエンジニアリング ウィザードを使う場合の、データベース接続に関する情報について説明します。データ モデルとデータベース間でのリバース エンジニアリング処理、フォワード エンジニアリング処理、および同期処理を行うためのウィザードを使うときには、サポートされているデータベースに接続することができます。各エンジニアリング処理について、正しいユーザー権限を保有していることが重要です。データベースのリバース エンジニアリングを行う場合には、システム カタログやマスター カタログに対する読み出しの権限だけが必要です。フォワード エンジニアリングや、データ モデルの同期を行う場合には、実際のスキーマや影響するデータベースに対する読み出し / 書き込みの権限が必要です。

DB2

DB2 データベースに接続する前に、DB2 クライアントと ODBC ドライバがインストールされていることを確認する必要があります。IBM DB2 の ODBC ドライバ (または、ほかのサードパーティ製 DB2 ODBC ドライバ) を使うことができます。

メモ: 使用する DB2 に合った ODBC ドライバを使っていることを確認してください。

ODBC ドライバを DB2 用に設定する必要があります。ODBC ドライバが正しく設定されている場合は、データ ソース名が Data Modeler のウィザードの[データ]ドロップダウンリストに表示されます。ODBC ドライバを設定して初めて使う場合には、ODBC Data Source Administrator からデータベースの接続をテストしてください。その後のデータベースの接続については、Data Modeler のウィザードで接続をテストしてください。

DB2 データベースへの接続

Data Modeler のウィザードでは、ODBC データ ソース名、ユーザー名、およびパスワードが必要です。ユーザー名と、それに対応するパスワードで、読み出しまたはデータベースを使う権限が与えられます。そのユーザー名で権限が与えられたデータベースだけ選択できます。DB2 UDB データベースのリバース エンジニアリングを行う場合、`syscat.*` ファイルに対する読み出しの権限が必要です。DB2 MVS データベースのリバース エンジニアリングを行う場合には、`sysIBM.*` ファイルに対する読み取り権限が必要です。

DB2 7.0 を使っている場合は、ユーザー名とパスワードの認証に NT の認証機能を利用できます。そのためには、ユーザー名とパスワードを空白のままにしておきます。NT の認証機能を使うと、Data Modeler はユーザー ID をオペレーティング システムに問い合わせます。この 1 回のサインオンによる方式は、データベース管理に便利です。

Oracle

Oracle データベースに接続する前に、Oracle のクライアントと Oracle Config サービスがインストールされていることを確認する必要があります。Oracle Net8 または Net Easy Config を設定する必要があります。

Oracle データベースへの接続

Data Modeler のウィザードでは、サービス名、ユーザー名、およびパスワードが必要です。サービス名によって、Oracle Server 名か Oracle Easy Config Service 名が決まります。ユーザー名とそれに対応するパスワードによって、データベースを読み取る権限または使用する権限が決まります。ユーザー名で認証されたデータベースだけ選択できます。

メモ： ウィザードの処理を続ける前に、必ずデータベースの接続をテストしてください。

SQL Server

SQL Server データベースに接続する前に、SQL Server のクライアントが存在することを確認する必要があります。OLE DB ドライバが Rose と共にインストールされている必要があります。OLE DB ドライバが SQL Server 用に設定されている必要があります。

SQL Server データベースへの接続

Data Modeler のウィザードでは、サーバー名、データベース名、ユーザー名、およびパスワードが必要です。サーバー名は、SQL Server が動作しているサーバーの名前です。サーバー名がローカル マシンの場合は、サーバー名を空白のままにしておくか、ローカル マシン名に設定することができます。

ユーザー名とそれに関連するパスワードによって、データベースを読み取る権限または使用する権限が決まります。ユーザー名で認証されたデータベースだけを選択できます。システム管理者の権限を持っている場合は、マスター データベースにアクセスできるので、[データベース] のドロップダウン リストにはその名前が表示されます。システム管理者の権限がない場合には、データベース名を手動で入力する必要があります。

SQL Server のサーバーが NT の認証機能をサポートしている場合には、ユーザー名とパスワードを空白のままにしておく、NT の認証機能を利用できます。NT の認証機能を使うと、Data Modeler はユーザー ID をオペレーティング システムに問い合わせます。この 1 回のサインオンによる方式は、データベース管理に便利です。

メモ： ウィザードの処理を続ける前に、必ずデータベースの接続をテストしてください。

Sybase

Sybase データベースに接続する前に、Sybase クライアントと ODBC ドライバがインストールされていることを確認する必要があります。ODBC ドライバを Sybase 用に設定する必要があります。ODBC ドライバが正しく設定されている場合は、データ ソース名が Data Modeler のウィザードの [データ] ドロップダウン リストに表示されます。ODBC ドライバを設定して初めて使う場合は、ODBC Data Source Administrator からデータベースの接続をテストしてください。その後のデータベースの接続については、Data Modeler のウィザードで接続をテストしてください。

Sybase データベースへの接続

Data Modeler のウィザードでは、データ ソース、データベース名、ユーザー名、およびパスワードが必要です。データ ソースは、Sybase のデータ ソースの名前です。ユーザー名とそれに関連するパスワードによって、データベースを読み取る権限または使用する権限が決まります。ユーザー名で認証されたデータベースだけを選択できます。システム管理者の権限を持っている場合は、マスター データベースにアクセスできるので、[データベース] のドロップダウン リストにはその名前が表示されます。システム管理者の権限がない場合は、データベース名を手動で入力する必要があります。

メモ： ウィザードの処理を続ける前に、必ずデータベースの接続をテストしてください。

索引

A

ANSI SQL 92

DDL へのフォワードエンジニアリング 52
データ モデル 24

C

C++ 6

DB2 ストアド プロシージャ 34
ストアド ファンクションのパラメータ スタイル 35
ストアド プロシージャ 34

CREATE Table 文 52

D

DB2

ストアド ファンクションのパラメータ スタイル 35
ストアド プロシージャ 34
データベースまたは DDL のリバース エンジニアリング 35
データベースまたは DDL へのフォワードエンジニアリング 52

DB2 のトリガ 33

DBMS 4

DB2 33, 34, 35, 52
Oracle 33, 36, 53
SQL Server 36, 53
Sybase 36, 53
新しい DBMS データベースの生成 19
言語へのマッピング 6
参照整合性 32
従来のデータベースでの作業 6
ターゲット データベース 24

データ型 12, 26
ドメイン 24
ドメイン列 12
物理データ モデル 23
“Oracle” も参照 28

DDL

ANSI SQL 92 52
DB2 52
drop 文 52
Oracle 53
SQL Server 53
Sybase 53
引用符 52
完全な名前 52
drop 文 52
SQL Server 53
Sybase 54

I

ID ベースの列 11
変換における作成 21

J

Java 6

コンポーネントに対する指定 9
ストアド ファンクションのパラメータ スタイル 35
ストアド プロシージャ 34

N

Not Deterministic (ストアド ファンクション) 35

O

Oracle

- チェック制約 28
- データベースまたは DDL のリバース エンジニアリング 36
- データベースまたは DDL へのフォワードエンジニアリング 53

Oracle のトリガ 33

R

Rose 8

Rose の図 4

S

SQL Server

- ストアド プロシージャ 34
- データベースまたは DDL のリバース エンジニアリング 36
- データベースまたは DDL へのフォワードエンジニアリング 53

Sybase

- データベースまたは DDL のリバース エンジニアリング 36
- データベースまたは DDL へのフォワードエンジニアリング 53

U

UML 8

プロファイル 3

UML (統一モデリング言語) 3

V

Visual Basic

- コンポーネントに対する指定 9

W

When 句トリガ 34

あ

アプリケーション

- オブジェクト モデルでの表示 8
- 従来のアプリケーションでの作業 6
- 変更の伝達 19
- アプリケーション設計者の役割 1

い

- 依存クラス 13
- 依存リレーションシップ 28
 - コンポジット集約へのマッピング 42
 - 集約へのマッピング 13
- 一意キー制約 27
- イベント
 - トリガ 33
- インデックス制約 27
- 引用符 52

う

埋め込まれた主キー / 外部キー 28

え

- 永続クラス 10
- 延期可能なチェック制約 28
- 延期不可能なチェック制約 28
- エンティティ / リレーションシップの表記法 3

お

オブジェクト モデル

- アプリケーションの表現 8
- コンポーネント 9
- データ モデルへの変換 19
- データ モデルへのマッピング 6, 9
- フォワード エンジニアリング 6
- リバース エンジニアリング 6
- オプションの非依存リレーションシップ 29

親クラス 13, 17

関連 14

集約 14

親テーブル 28

参照整合性 32

非依存リレーションシップ 29

リレーションシップ 29

ロール 30

親のないレコード

参照整合性での防止 32

か

外部キー

インデックスの指定 21

埋め込み 28

子テーブル 28

参照整合性 32

多重度 29

データ モデル エLEMENT の変換 48

外部キー制約 27

外部キー列

マッピング 40

カスタム トリガ

定義 32

データ モデルでの作成 32

完全な名前 52

関連

限定子 16

限定子付き 44

多対多 15, 43

データ モデルのリレーションシップへの変換 21

非依存リレーションシップへのマッピング 14

関連クラス

共通部分テーブルのマッピング 45

共通部分テーブルへのマッピング 15

き

キー

埋め込み 28

キー制約

インデックス 26, 27

外部キー 27

主キー 26

説明 26

データ モデルでの作成 27

行

行とインデックス 27

共通部分テーブル 15, 31

関連クラスへのマッピング 15

関連構造へのマッピング 40

限定子付き関連へのマッピング 16

マッピング 43

く

クラス 7

依存 13

永続 10

親 13, 17

関連 15, 45

子 17

属性の割り当て 7

テーブルへの変換 21

テーブルへのマッピング 10

非永続 10

クラス図

コンポーネント 9

説明 7

データ モデル図へのマッピング 6

け

継承

マッピング 17

限定子 16

限定子付き関連

共通部分テーブルのマッピング 43, 44

共通部分テーブルへのマッピング 16

こ

構造

- 自己参照 31
- マッピング 46

候補キー

- 主キーへの変換 21

候補キー列 11

子クラス 17

子テーブル 28

- 共通部分テーブル 31

- ロール 30

コンポーネント

- 説明 9

- 変換 19

コンポジット集約

- “集約” も参照 13

さ

再帰的なリレーションシップ

- 自己参照構造 31

算出列 26

参照整合性 31

- アクション 32

- 親テーブル 29, 32

- システム生成参照整合性 (RI) トリガ 32

- 宣言参照整合性 (DRI) 32

し

システム生成参照整合性 (RI) トリガ 32

システム トリガ

- 参照整合性 32

集約 13

- 非依存リレーションシップへのマッピング 14

主キー 10

- ID ベースの列 11

- 埋め込み 28

- 親テーブル 28

- 限定子付き関連 44

- 算出列 26

- 主キー / 外部キー 28, 43

- 主キー制約 26

す

図

- データ モデル 4, 23, 24

- データ モデルとクラス間のマッピング 6

スーパータイプ / サブタイプ構造

- 継承構造へのマッピング 46

スキーマ

- 格納 24

- スキーマを作成するためのリバース エンジニアリング 35

- 定義 24

- データベースへの割り当て 24

- データ モデルでの作成 24

- デフォルトの標準 24

- パッケージへのマッピング 9, 39

- 変換のための指定 20

ステレオタイプ 3

- entity 7

- ENUM 41, 48

ストアドファンクション

- DB2 パラメータ スタイル 35

- Not Deterministic 35

- データ型 34

- パラメータ 35

ストアドプロシージャ

- 言語 34

- ストアドファンクション 34

- データ モデルでの作成 34

- パラメータ 34

せ

制約

- 一意キー 27

- キー 26

- 説明 26

- チェック 26, 27

- データ モデルでの作成 26

- 宣言参照整合性 32

そ

操作

変換 19

マッピング 9

属性

クラスへの割り当て 7

限定子としての属性 16

ドメインへのマッピング 12

列として変換 21

列へのマッピング 10

属性型

データ型へのマッピング 12

た

ターゲット データベース 24

説明 24

タグ付きの値 3

多重度 29

外部キー 29

多対多の関連

関連クラス 45

共通部分テーブルのマッピング 43

ち

チームの役割

アプリケーション設計者 1

依存関係 2

データベース設計者 2

ビジネス アナリスト 1

チェック制約 27

クラスへのマッピング 41

て

データ

チェック制約 27

データ型

ストアド ファンクション用のストアド パラ
メータ 34

ストアド プロシージャのパラメータ 34

属性型へのマッピング 12

データベース

従来のデータベースでの作業 6

スキーマの割り当て 24

ターゲット 24

データ モデルでの作成 24

デフォルト ターゲット 24

変換のための指定 21

モデル化 7

データベース設計者の役割 2

データベースのモデル化

物理データ モデリング 23

データ モデリング

論理 7

データ モデリング プロファイル 3

データ モデル 23

新しいデータ モデルの構築 23

アプリケーションへのマッピング 8

オブジェクト モデルへの変換 47

オブジェクト モデルへのマッピング 6, 9

定義 23

データ モデルを作成するためのリバース エ
ン지니어リング 35

比較と同期 54

フォワード エン지니어リング 51

リレーションシップ 24

データ モデル エレメント

変換 48

データ モデルからオブジェクト モデルへの変換
47

データ モデル図 4, 23

クラス図へのマッピング 6

説明 4

データ モデルでの作成 24

データ モデルの同期

説明 55

データ モデルの比較 54

説明 54

データ モデルのリバース エン지니어リング 35

データ列 26

テーブル

共通部分 15, 43

クラスへのマッピング 10, 40

スキーマ 25

説明 25

チェック制約 27

- データ モデルでの作成 25
- 名前のプレフィックスの指定 21
- リレーションシップ 28
- リレーションシップの多重度 29
- テーブルスペース名
 - テーブル 25
- テーブルの結合 15

と

- 同期
 - データ モデル 54
- 同期ウィザード 54
- ドメイン
 - DB2 53
 - SQL Server 53
 - Sybase 53
 - サーバーによる生成 24
 - 説明 24
 - 属性型へのマッピング 39
 - 属性へのマッピング 12
 - チェック制約 27
 - データ型 24
 - データ モデルでの作成 24
- ドメイン パッケージ
 - DBMS への割り当て 24
- ドメイン列 12
- トリガ
 - When 句 34
 - イベント 33
 - カスタム 32
 - 参照 33
 - 参照整合性 32
 - タイプ 33
 - 粒度 33
 - データ モデルでの作成 32
- トリガ イベント
 - 定義 33

は

- パッケージ 9
- 変換 21

- パラメータ
 - ストアド プロシージャ 34

ひ

- 非依存リレーションシップ
 - オプション 29
 - 関連へのマッピング 14, 42
 - 構造 31
 - 集約へのマッピング 14
 - 必須 29
- 非永続クラス 10
- ビジネス アナリストの役割 1
- ビジネス要件
 - データ エンティティ 7
- 必須の非依存リレーションシップ 29
- 表記法
 - エンティティ / リレーションシップ 3

ふ

- フォワード エンジニアリング
 - データ モデル 6
- フォワード エンジニアリング ウィザード 51
- フォワード エンジニアリング、データ モデル 51
- 物理データ モデリング 23
- 物理データ モデル 4
 - 実装 51
 - マッピング 4
 - “データ モデル” も参照 23
- 物理データ モデル、変換 4
- フレームワーク 5
- プログラミング言語
 - C 34, 35
 - C++ 6
 - Java 6, 9, 34, 35
 - SQL 34
 - Visual Basic 9
- プロファイル 3

へ

変換 6

オブジェクト モデルからデータ モデルへの
変換 19, 20

関連 21

クラス 21

パッケージ 21

ま

マッピング

オブジェクト モデルからデータ モデルへの
マッピング 9

関連クラスから共通部分テーブルへのマッピ
ング 15

継承 17

限定子付き関連から共通部分テーブルへの
マッピング 16

コンポーネントから言語へのマッピング 9

操作 9

属性からドメインへのマッピング 12

パッケージ 9

[マップ元] ボックス

変換時の使用 22

も

モデル

同期 19

変換 6

モデル化、データベース 7

モデルの変換 6

り

リバース エンジニアリング

データ モデル 6

リバース エンジニアリング ウィザード 35

リレーションシップ

依存 28

親テーブル 29

説明 28

多重度 29

データ モデルでの作成 28

非依存 28, 29

ロール 30

リレーションシップ構造 30

れ

列

ID ベース 11

キー制約 26

候補キー 11

算出列 26

主キー 10

説明 25

属性へのマッピング 8, 40

データ モデルでの作成 25

データ列 26

ドメイン 12

列挙型チェック制約

クラスへのマッピング 41

ろ

ロール

テーブル 30

論理データ モデリング 7

論理データ モデル 4

論理パッケージ 9

