Bazy danych

Dr inż. Sławomir Samolej
D108 A, tel: 865 1486,
email: ssamolej@prz-rzeszow.pl
WWW: ssamolej.prz-rzeszow.pl

Podziękowanie:
Chcę podziękować dr inż. Krzysztofowi Świdrowi i dr inż. Grzegorzowi Decowi za udostępnienie materiałów źródłowych i slajdów dotyczących omawianych w wykładzie zagadnień.

Literatura

- **Krzysztof Świder**
  Projektowanie baz danych

  http://ns.prz-rzeszow.pl/~kswider/pbd/pbd_0405.pdf
- **Marcin Zawadzki**

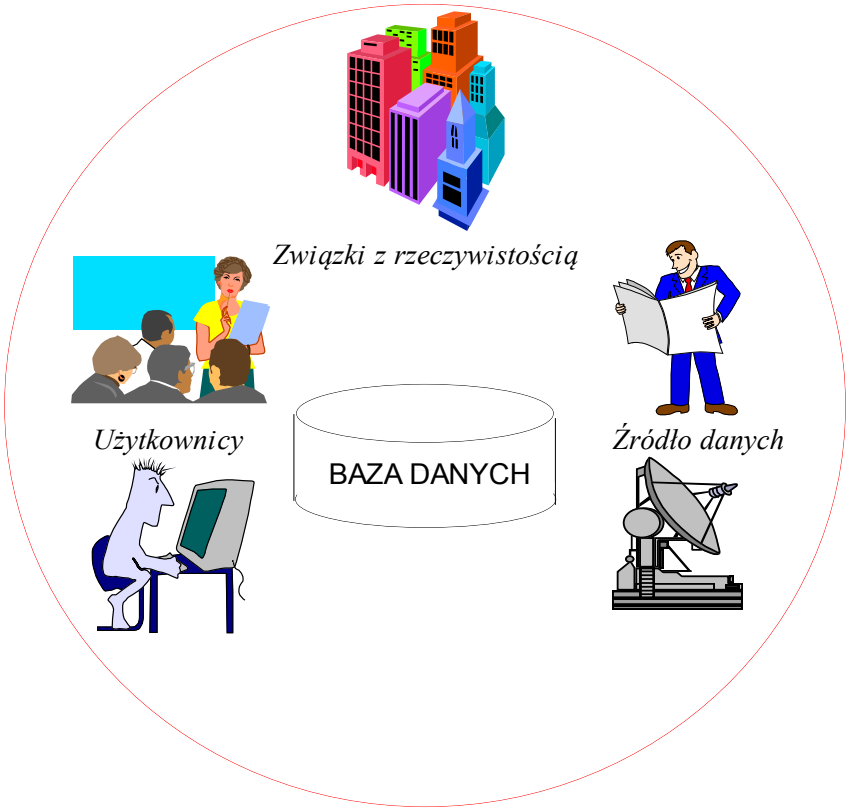
  SQL Server 2005, Mikom/PWN 2007.
- **L. Banachowski, A. Chądzyńska, K. Matejewski**, Relacyjne bazy danych. Wykłady i ćwiczenia, Wydawnictwo PJWSTK 2009

Wprowadzenie

Bazy danych i ich użytkownicy 1

Baza danych:

- zbiór powiązanych ze sobą danych
- abstrakcyjne, informatyczne odzwierciedlenie fragmentu rzeczywistości
- logicznie spójny zbiór danych posiadających określone znaczenie
- wszelka informacja, która może być przydatna firmie

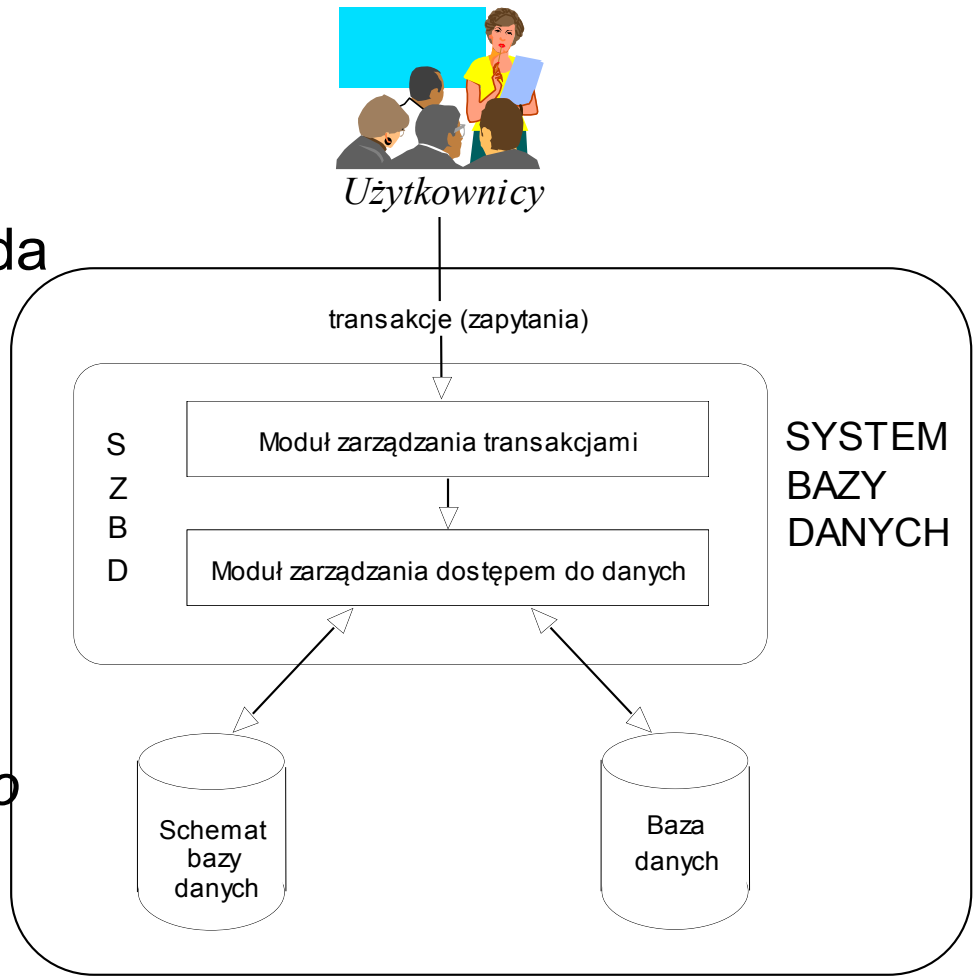


Bazy danych i ich użytkownicy 2

- Baza danych jest projektowana, budowana i wypełniana danymi dla zrealizowania określonych celów.
- Korzystają z niej różne grupy użytkowników:
  - Osoby wykorzystujące zawartą w niej informację.
  - Projektanci, którzy projektują strukturę bazy danych oraz przygotowują programy użytkowe, które ułatwiają korzystanie z bazy przyszłym użytkownikom.

System zarządzania bazą danych (SZBD)

- Jest to zbiór programów umożliwiających **tworzenie i eksploatację bazy danych**.
- Nieco szerszym pojęciem jest *system bazy danych*, który składa się z **bazy danych** i **SZBD**.
- Użytkownicy kontaktują się z systemem bazy danych za pośrednictwem *transakcji*, kierowanych za pomocą specjalnych poleceń, albo wcześniej przygotowanych aplikacji
- *Moduł zarządzania dostępem do danych* umożliwia właściwą interpretację, wprowadzanie, usuwanie i modyfikację danych



Przykład relacyjnej bazy danych

- *Schemat bazy danych* – opis struktury przechowywanych danych oraz wzajemnych powiązań między nimi.
- *Relacja* - tablica dwuwymiarowa do przechowywania danych.

Relacja *pracownik*

NUMER	NAZWISKO	ETAT	SZEF	PRACUJE_OD	PŁACA_POD	PŁACA_DOD	ID_ZESP
1000	Lech	dyrektor		01-JAN-71	3160	570	10
1080	Koliberek	sekretarka	1000	20-FEB-83	1150		10
1010	Podgajny	profesor	1000	01-MAY-75	2180	420	20
1040	Rus	adiunkt	1010	15-SEP-79	1750		20
1070	Muszyński	adiunkt	1010	01-MAY-85	1600		20
1060	Misiecki	asystent	1010	01-mar-85	1400		20
1090	Palusz	asystent	1040	15-SEP-89	1200		20
1020	Delcki	profesor	1000	01-SEP-77	2050	270	30
1030	Maleja	adiunkt	1020	01-JUL-68	1750		30
1100	Warski	asystent	1030	15-JUL-87	1350		30
1110	Rajski	stażysta	1030	01-JUL-90	900		30
1050	Lubicz	adiunkt	1000	01-SEP-83	1780		40
1120	Orka	asystent	1050	01-APR-88	1350		40

Relacja *zespol*

ID_ZESP	NAZWA	ADRES
10	administracja	Piotrowo 3a
20	bazy danych	Wieżowa 75
30	sieci komputerowe	Garbary 3
40	systemy operacyjne	Piotrowo 3a
50	translatory	Mansfelda 4

Relacja *etat*

ETAT	PŁACA_MIN	PŁACA_MAX
stażysta	800	1000
sekretarka	900	1200
asystent	1000	1600
adiunkt	1600	2000
profesor	2000	2500
dyrektor	2500	3200

Wybrane cechy baz danych

- ***Niezależność aplikacji i danych.***

  Dane mogą być wprowadzane do bazy danych, bez konieczności zmian w korzystających z nich aplikacjach. Z drugiej strony aplikacje mogą być modyfikowane niezależnie od stanu bazy danych.
- ***Abstrakcyjna reprezentacja danych.***

  Przygotowanie aplikacji baz danych odbywa się przy pomocy *języków deklaratywnych*. W odróżnieniu od języków tradycyjnych, które określa się jako *imperatywne*, piszący aplikację nie musi zajmować się kolejnością danych w bazie, ani ich wyszukiwaniem. Precyzuje on jedynie warunki selekcji informacji, tzn. pracuje w kategoriach *co zrobić*, a nie *jak zrobić*.
- ***Różnorodność sposobów widzenia danych*.**

  Te same dane mogą być różnie widziane przez różnych użytkowników. Uzyskuje się to przez stosowanie specjalnych filtrów nazywanych *perspektywami*.
- ***Fizyczna i logiczna niezależność danych*.**

  Niezależność fizyczna oznacza, że rozszerzenie systemu komputerowego, gdzie pracuje SZBD o nowy sprzęt nie narusza danych.

  Logiczna niezależność danych oznacza po pierwsze możliwość wprowadzenia nowych danych bez dezaktualizacji starych, a po drugie to, że dane, które nie są wzajemnie powiązane tzw. *związkami integralnościowymi*, mogą być usuwane z bazy niezależnie od siebie.

Korzyści wynikające ze stosowania baz danych

1. *Zmniejszenie nadmiarowości danych* – dane wykorzystywane przez różne aplikacje nie są dublowane.
2. *Współdzielenie danych* – na tych samych danych mogą pracować różne aplikacje bez groźby ich wzajemnego zniszczenia.
3. *Autoryzacja dostępu* – blokada poufnych danych dla niepowołanych użytkowników.
4. *Różnorodność interfejsów do danych* – te same dane mogą być prezentowane na różne sposoby.
5. *Reprezentacja złożonych związków* – z zastosowaniem prostych mechanizmów można modelować złożone związki znaczeniowe (semantyczne) między różnymi danymi.
6. *Ograniczenia integralnościowe* – możliwe jest zabezpieczenie przed wpisywaniem niewłaściwych wartości danych oraz niewłaściwych powiązań danych.
7. *Ochrona przed awariami* – SZBD powinien być zabezpieczony w taki sposób, że w razie awarii istnieje możliwość odtworzenia poprawnego stanu bazy danych sprzed awarii.

Kiedy stosowanie bazy danych jest niecelowe lecz możliwe

- Praca jest wykonywana przez jednego człowieka lub grupę ludzi pracujących w sposób ściśle skoordynowany (sekwencyjnie),
- Zasoby finansowe są ograniczone
- Specyfika pracy nakładałaby ograniczenia na czas realizacji transakcji (systemy czasu rzeczywistego). Wymagane są inne typy baz – obecnie na etapie eksperymentów.

Modele Danych

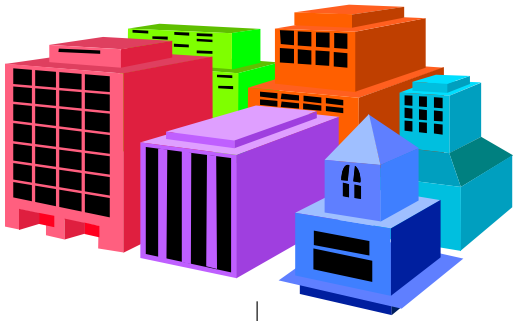
Modele danych

- Fundamentalną cechą systemów baz danych jest zapewnienie wyższego poziomu abstrakcji widzenia danych przez użytkowników, dzięki przesłonięciu szczegółów fizycznej organizacji tych danych. Uzyskuje się to dzięki oferowanym przez bazę danych modelom danych.
- Przez model rozumie się zbiór pojęć potrzebnych do opisu struktury bazy danych. Struktura ta obejmuje:
  - Typy danych, związki między danymi i nałożone ograniczenia.
  - Zbiór operacji do definicji, wyszukiwania i aktualizacji bazy danych.

Kategorie modeli danych

- *modele konceptualne (koncepcyjne)*
  - są najbardziej zbliżone poziomem abstrakcji do wymagań projektantów, opracowane w początkowej fazie projektowania systemu np. ERD,
- *modele implementacyjne*
  - służą do transformacji wcześniej przygotowanego modelu konceptualnego bazy danych. Spośród znanych podejść, do których zaliczamy modele *hierarchiczne*, *sieciowe* i *relacyjne*, praktyczne zastosowanie mają obecnie jedynie modele relacyjne.
- fizyczne modele danych
  - określają sposób organizacji danych w zewnętrznej pamięci komputera. Przy najwyższym stopniu szczegółowości rozważa się poszczególne bity przechowywane w pamięci. Na niższych stopniach szczegółowości stosuje się pojęcia: *rekord* (zapis) i *plik*.

Metodyka projektowania bazy danych



miniświat

Analiza miniświata - konstrukcja modelu konceptualnego miniświata

diagramy konceptualne

Transformacja modelu konceptualnego do modelu relacyjnego

relacje

Proces normalizacji

relacje znormalizowane

Wybór struktur fizycznych i określenie ścieżek dostępu

fizyczne struktury danych

Strojenie systemu

Języki projektowania baz danych

- *Język definiowania danych (DDL)* - definiowanie struktury danych przechowywanych w bazie.
- *Język manipulowania danymi (DML)* - zapis usuwanie i aktualizowanie danych.
- *Język sterowania danymi (DCL)* - sterowanie transakcjami np. ich wycofywanie lub zatwierdzania.
- *Język zapytań (QL),* który umożliwia pobieranie z bazy informacji zgodnych z wyspecyfikowanymi warunkami.

Modele związków encji

- Podstawową techniką przedstawiania konceptualnych modeli danych są *diagramy związków encji* (ERD). Modelują one zarówno dane jak i sposób widzenia ich struktury.
- *Encja* to cokolwiek o czym chcemy przechowywać informację.
- *Związki* opisują zależności między encjami.
- Specyficzne informacje o encjach są nazywane *atrybutami*, np. nazwisko, adres, limit kredytowy, itp..

Przykładowe dane do wygenerowania diagramu - faktura

Invoice

date: 03.03.2006
invoice number INV4234191
purchase order no. PN89893723
bill of Lading L694053442

Sold To:
Tax Id.: 813-009-123-1223
Contact name: Gregory Dec
Company Name/Address:
Cash & Long Power Supplies
915 Doncaster Drive
Suite 3143
West Deptford NJ 08066
US
tel. 609-555-6964
e-mail: clps@clps.com

No. Units	Units of measure	Description of Goods	Country of Origin	Unit Value	Total Value
50	Pr.	Aluminium Windshield Wiper Assemblies	US	$54.50	$2725.00
125	Ea.	Rubber Windshield Wiper Replacement Blades	US	$3.99	$498.75
2	Ea.	Automotive Technical Books	US	$18.95	$37.90

Additional comments:	Invoice line total	$3261.65
	Discount/Rebate	$163.00
	Invoice sub-total	$3098.65
	Freight Charges	$324.00
Declaratoin Statement	Insurance	$11.20
	Other packing	$10.00
	Invoice Total Amount	$3443.85
	Currency code	USD

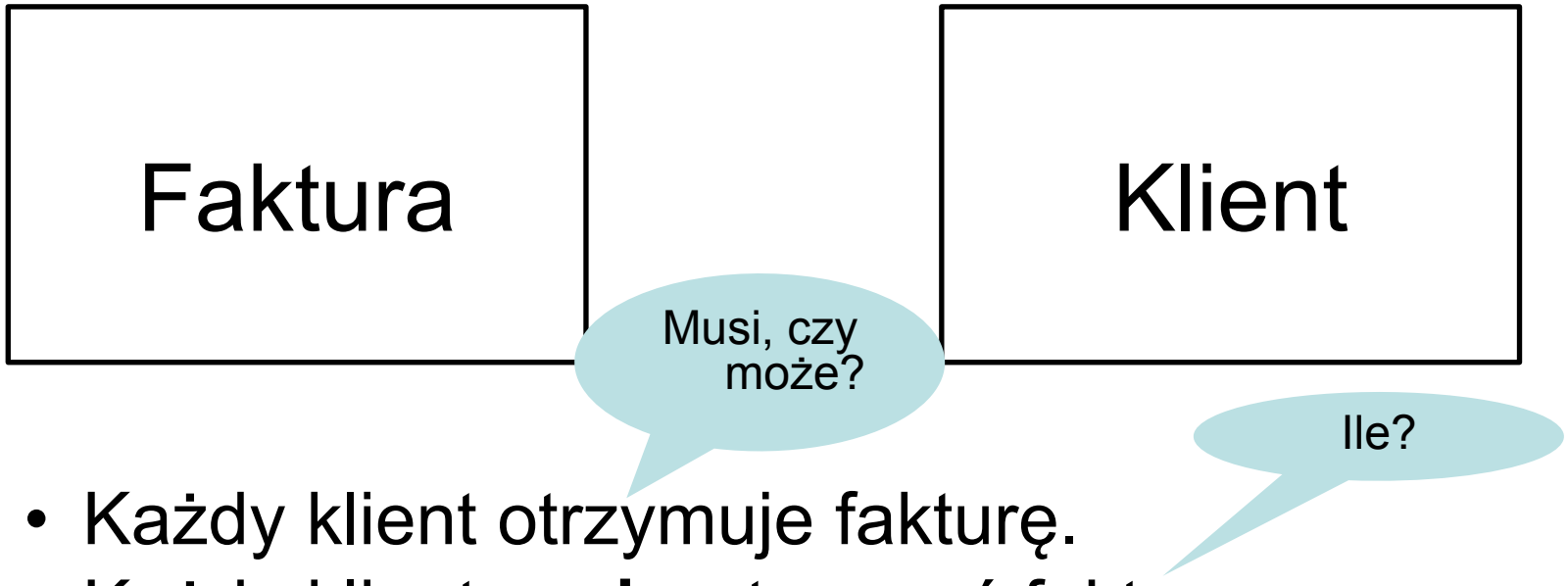
Shipper Signature Date

Steve Blume 03.03.2006

Nazywanie encji

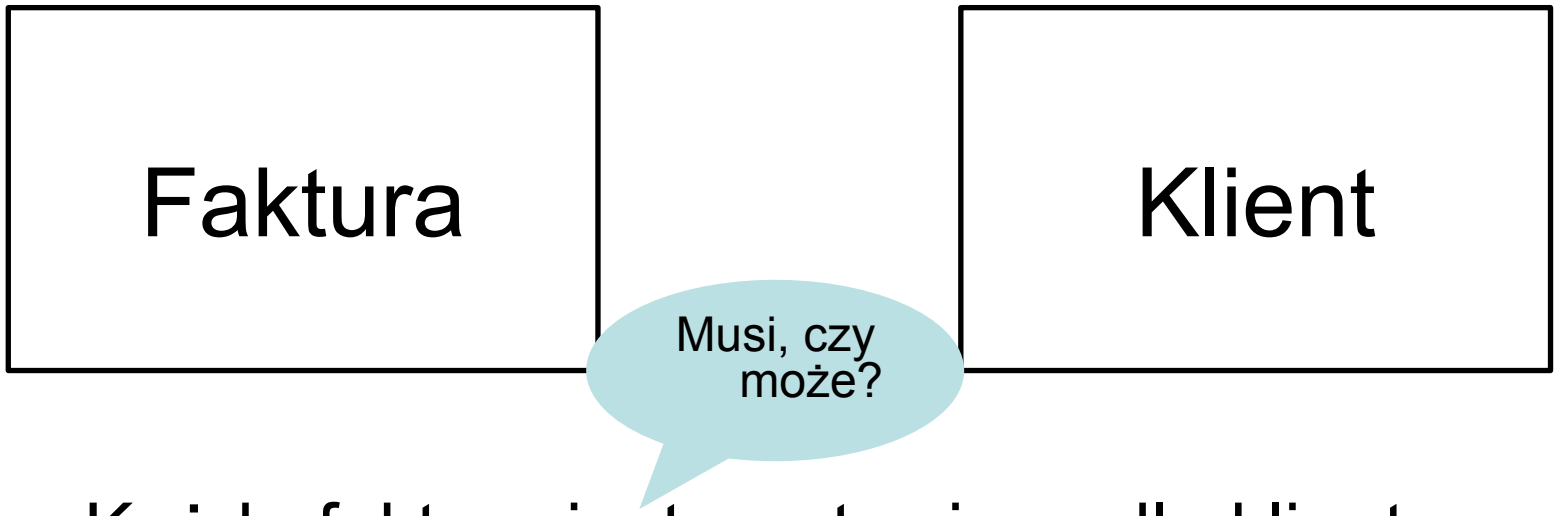
- Encja powinna być rzeczownikiem w liczbie pojedynczej
  np.: faktura, odbiorca, nadawca

Ustalenie relacji pomiędzy encjami Klient -> Faktura



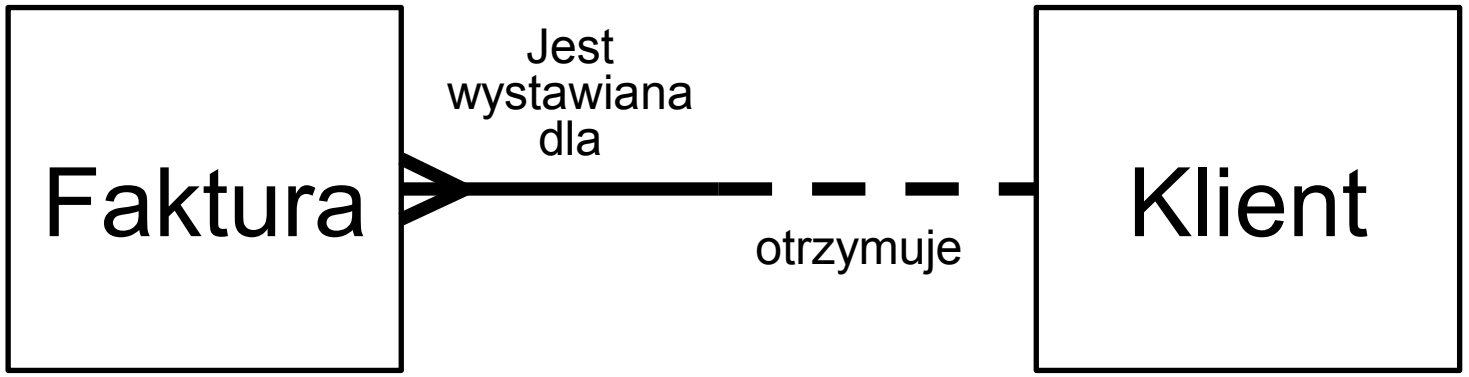
- Każdy klient otrzymuje fakturę.
- Każdy klient **może** otrzymać fakturę.
- Każdy klient może otrzymać **wiele** faktur.
- Każdy klient **może** otrzymać **0 lub wiele** faktur

Ustalenie relacji pomiędzy encjami Faktura -> Klient



- Każda faktura jest wystawiona dla klienta.
- Każda faktura **musi** (jeśli transakcja zaszła) być wystawiona dla klienta.
- Każda faktura musi być wystawiona dla **dokładnie 1** klienta.
- Każda faktura **musi** być wystawiona dla **dokładnie 1** klienta.

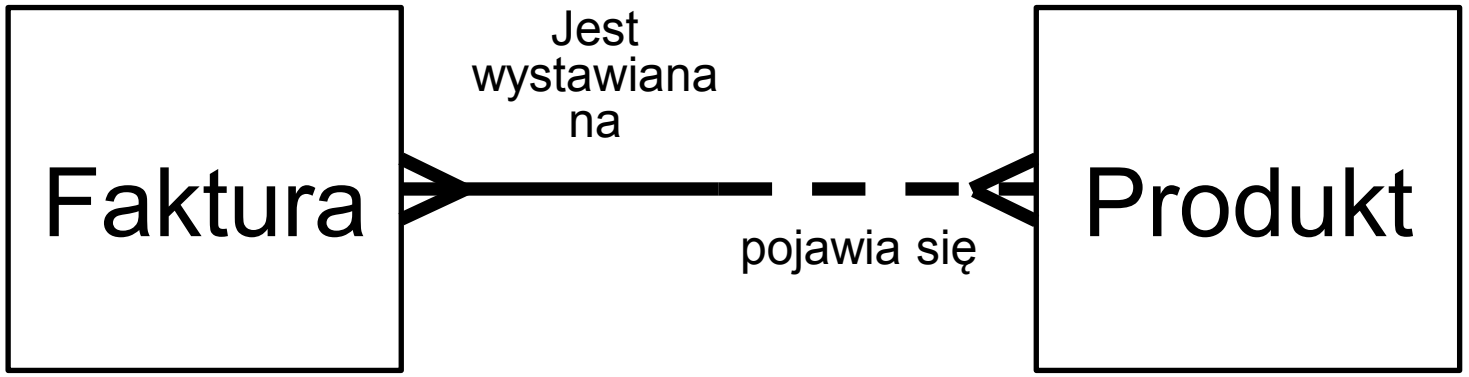
Pokazanie relacji w sposób graficzny (1)



- Każdy klient **może** otrzymać **0 lub wiele** faktur
- Każda faktura **musi** być wystawiona dla **dokładnie 1** klienta.

21

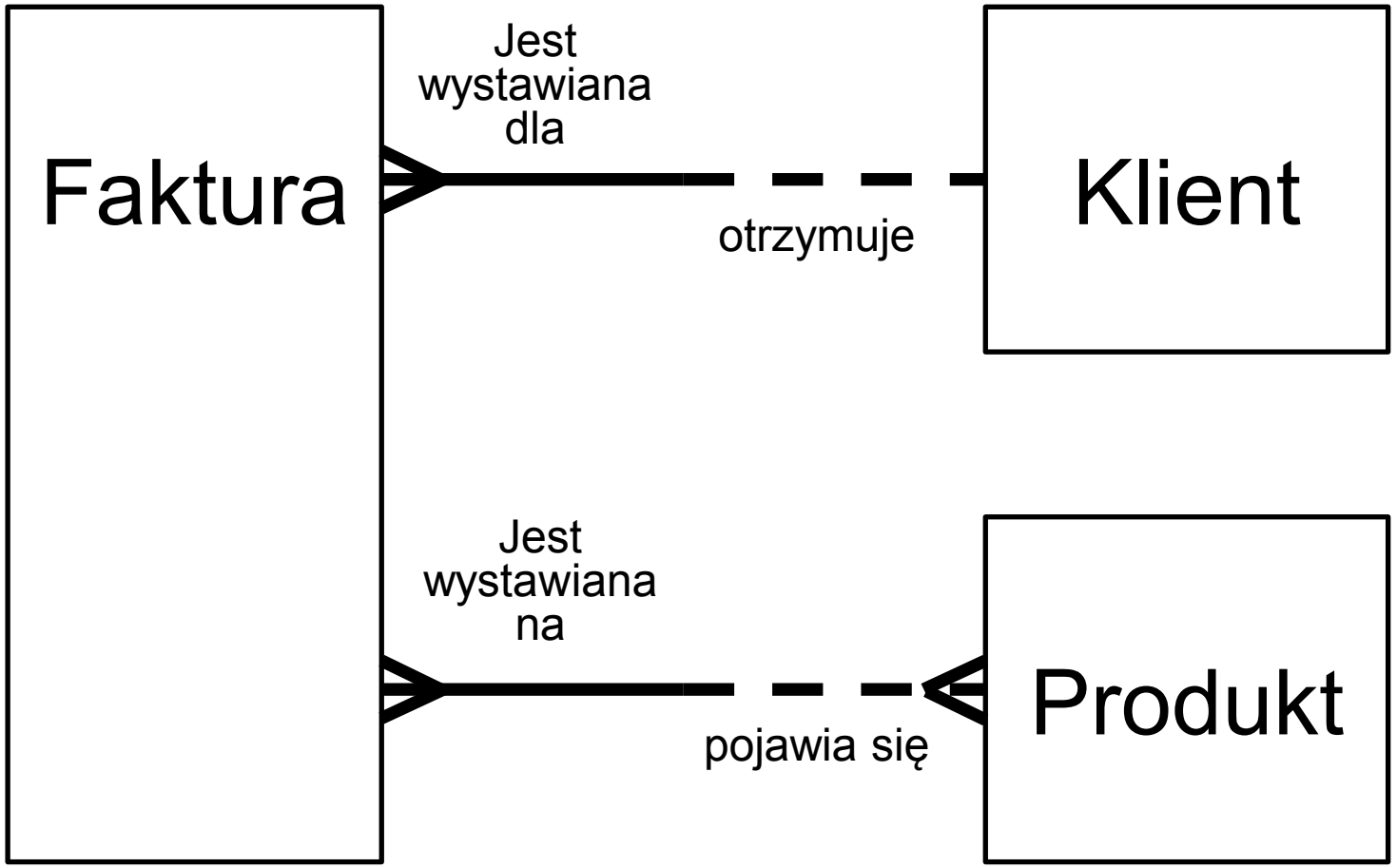
Pokazanie relacji w sposób graficzny (2)



- Każdy produkt **może** pojawić się na **0 lub wielu** fakturach.
- Każda faktura **musi** być wystawiona dla **1 lub wielu** produktów.

22

Wstępna wersja diagramu związków encji



Wprowadzanie atrybutów

- Atrybuty są najniższym poziomem opisu danych i opisują szczegóły encji
- Jeśli określa się atrybut to definiuje się jego:
  - nazwę
  - typ
  - długość
  - wartość domyślną
  - czy jest obowiązkowy, czy opcjonalny

„Wydobycie” atrybutów z dokumentów

Sold To:
Tax Id.: 813-009-123-1223
Contact name: Gregory Dec
Company Name/Address:
Cash & Long Power Supplies
915 Doncaster Drive
Suite 3143
West Deptford NJ 08066
US
tel. 609-555-6964
e-mail: clps@clps.com

Klient

tax_id
contact_mname
company_name
street
city
postal_code
country
telephone
e-mail

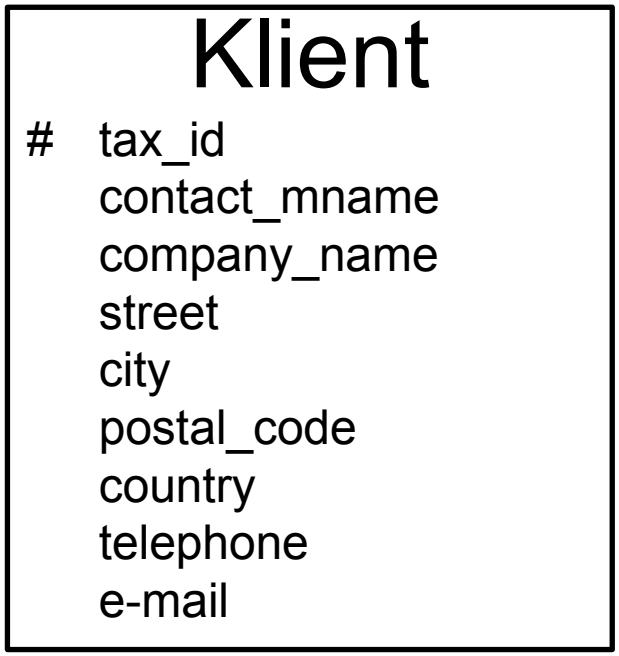
Każda encja musi zawierać co najmniej 2 atrybuty.

Unikalny identyfikator

Każda encja powinna zawierać **unikalny identyfikator**, który pozwoli jednoznacznie zidentyfikować jej wystąpienie.

Unikalny identyfikator jest zwykle wybranym atrybutem lub kombinacją atrybutów.

Dla przykładowej encji jednoznacznie określony w skali krajowej numer podatnika jest oczywistym unikalnym identyfikatorem.



Atrybuty obowiązkowe i opcjonalne

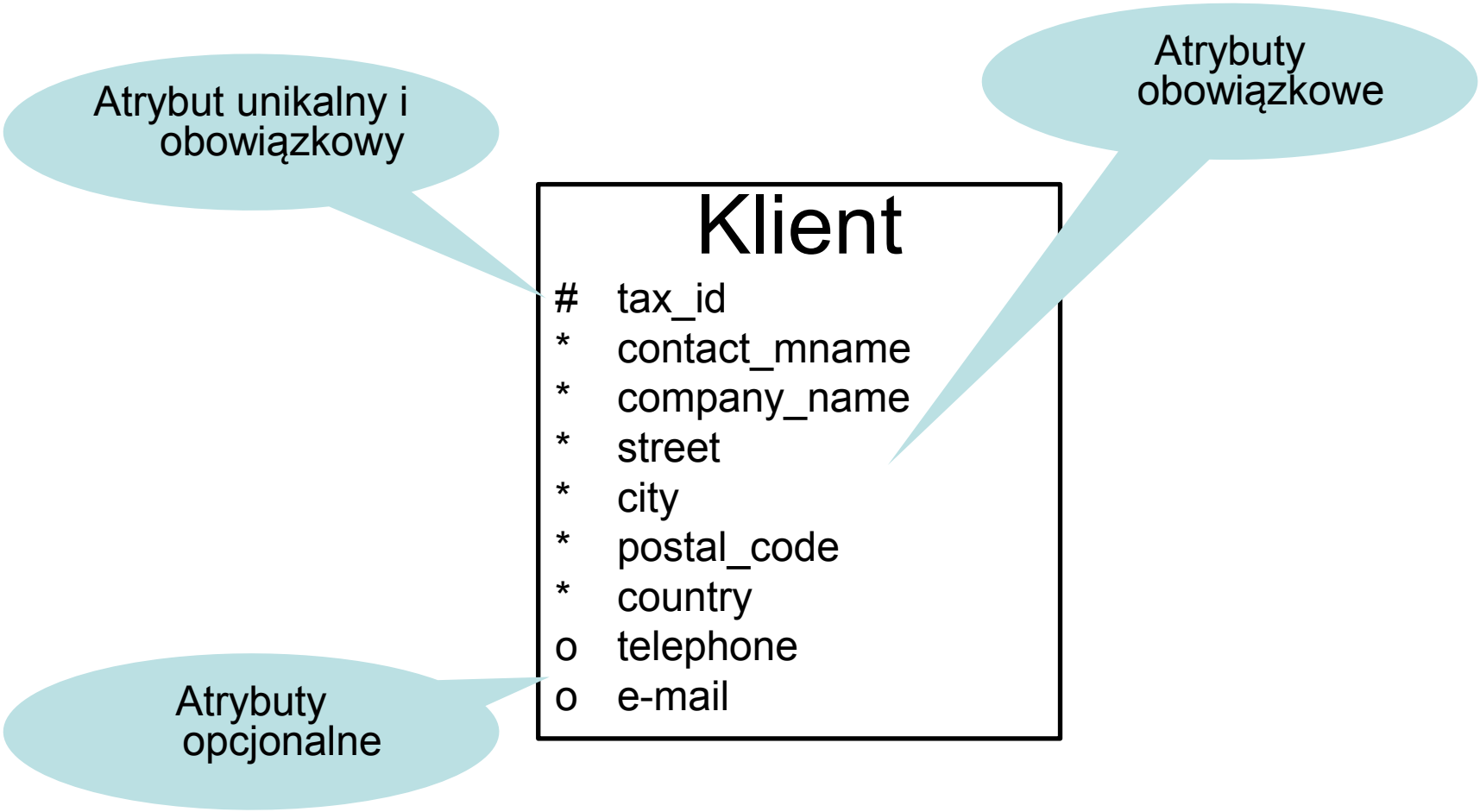
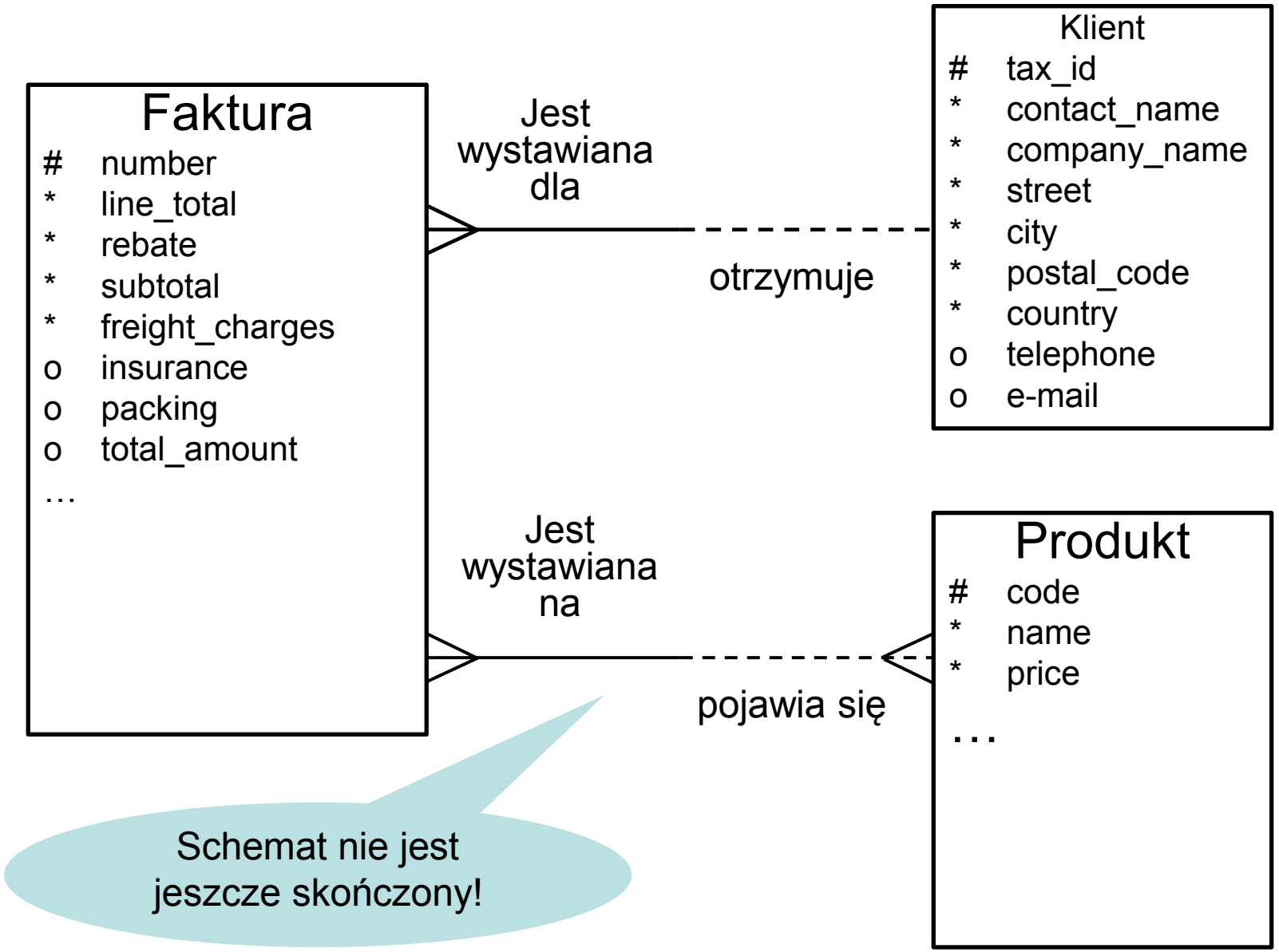
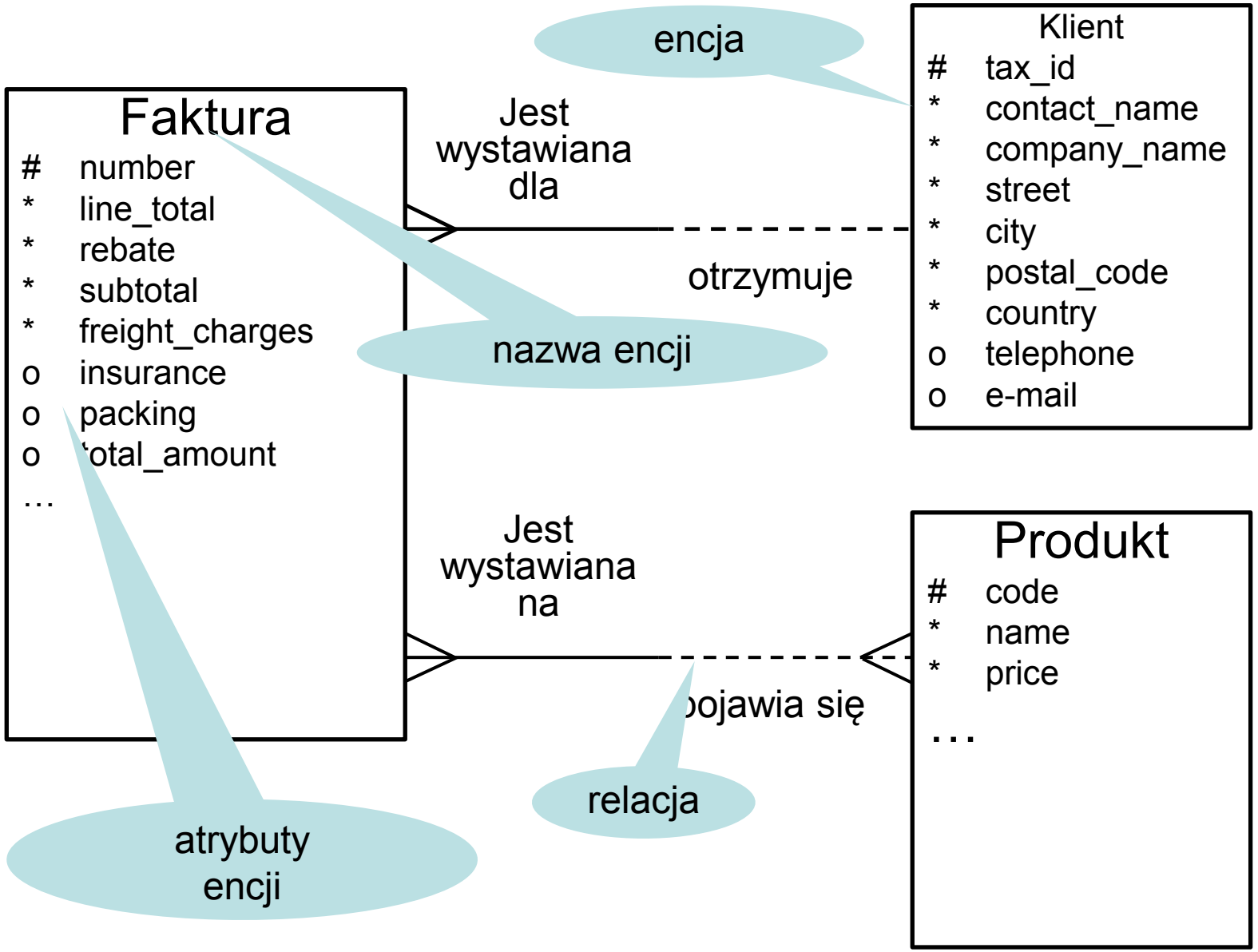


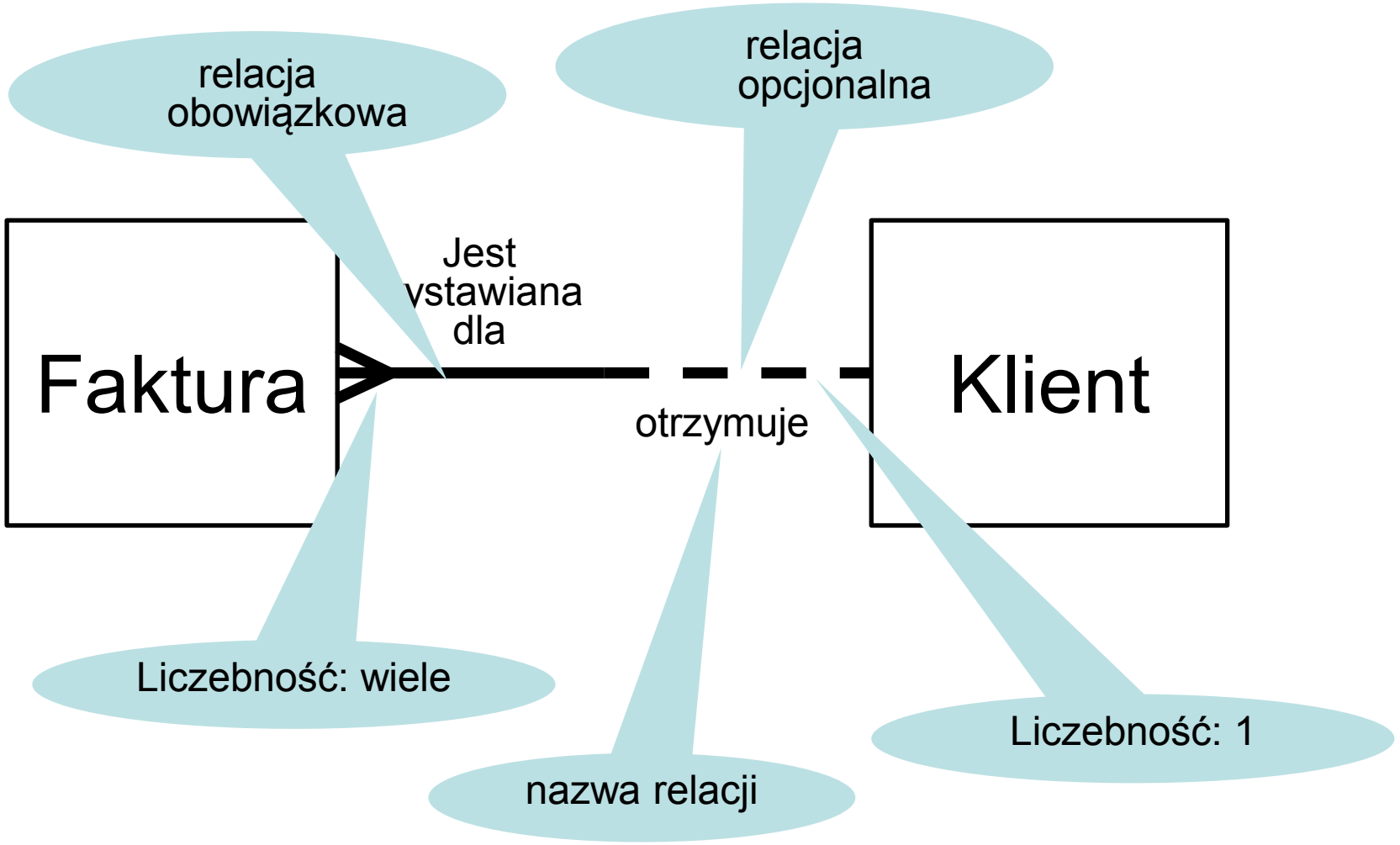
Diagram związków encji na obecnym etapie przygotowania



Terminologia diagramów związków encji (1)

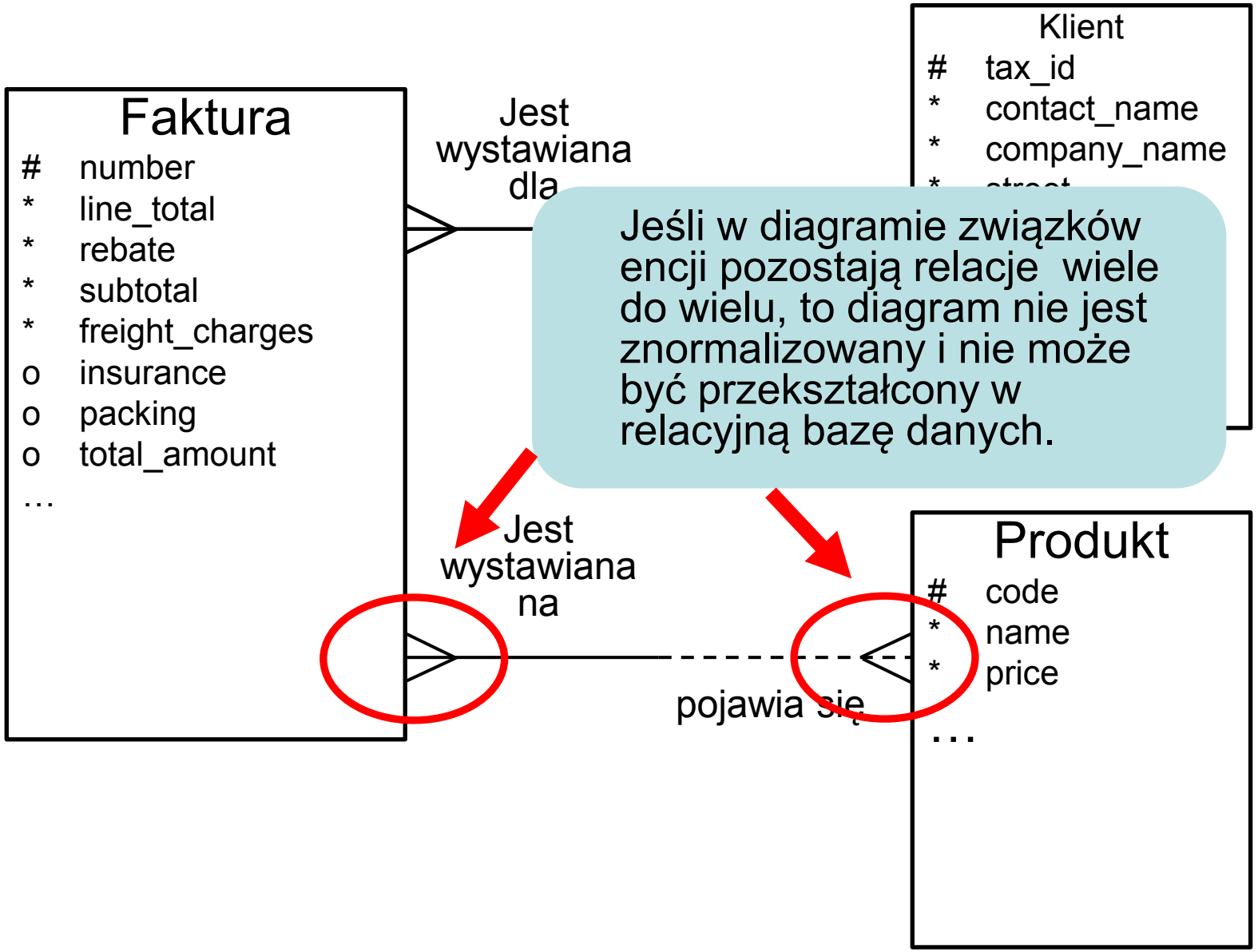


Terminologia diagramów związków encji (2)

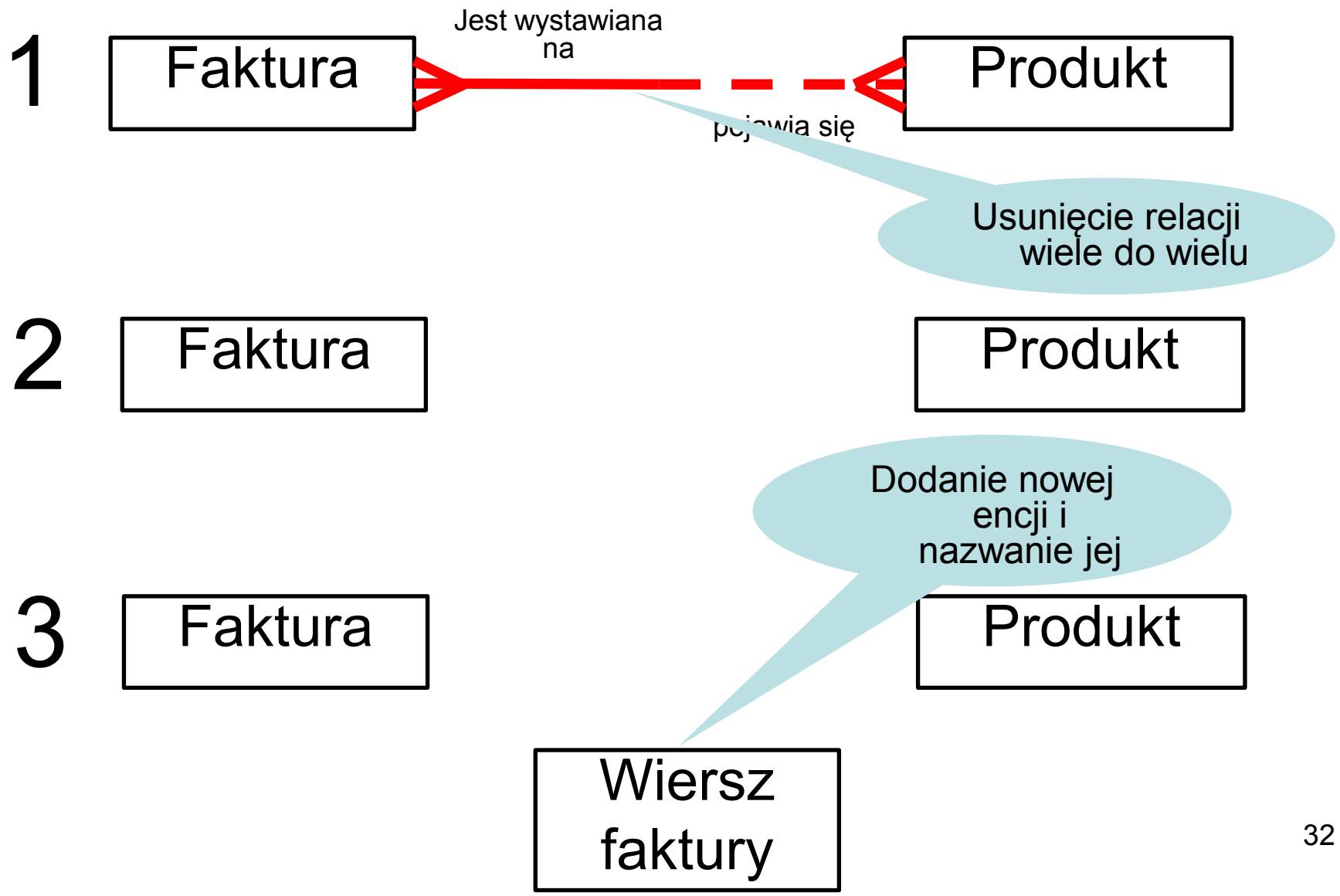


30

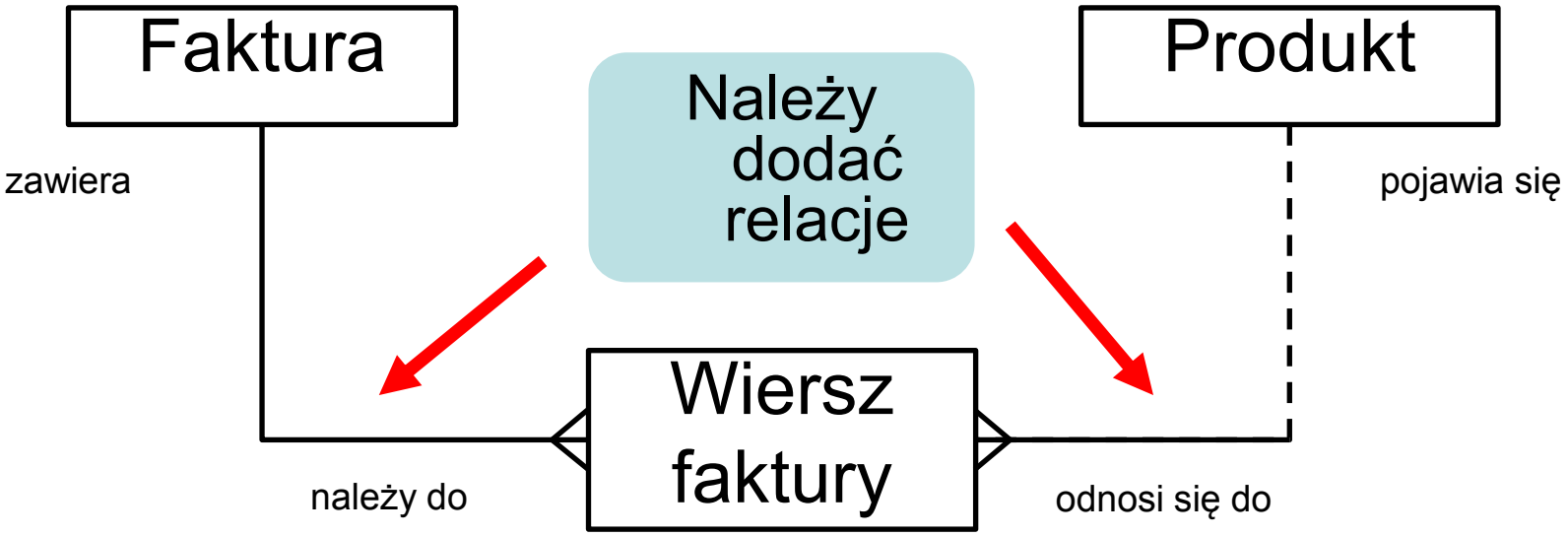
Normalizacja modelu (1)



Normalizacja modelu (2)



Normalizacja modelu (3)



Liczebność zawsze ustala się na wiele dla „nowej” encji i jeden dla „starych”.

Gdzie znaleźć nową encję?

Invoice

date: 03.03.2006
invoice number INV4234191
purchase order no. P
bill of Lading I

Sold To:
Tax Id.: 813-009-123-1223
Contact name: Gregory Dec
Company Name/Address:
Cash & Long Power Supplies
915 Doncaster Drive
Suite 3143
West Deptford NJ 08066
US
tel. 609-555-6964
e-mail: clps@clps.com

Wiersz faktury jest brakującą encją

Każdy wiersz odnosi się dokładnie do jednego produktu

No. Units	Units of measure	Description of Goods	Country of Origin	Unit Value	Total Value
50	Pr.	Aluminium Windshield Wiper Assemblies	US	$54.50	$2725.00
125	Ea.	Rubber Windshield Wiper Replacement Blades	US	$3.99	$498.75
2	Ea.	Automotive Technical Books	US	$18.95	$37.90

Additional comments:	Invoice line total	$3261.65
	Discount/Rebate	$163.00
	Invoice sub-total	$3098.65
	Freight Charges	$324.00
Declaratoin Statement	Insurance	$11.20
	Other packing	$10.00
	Invoice Total Amount	$3443.85
	Currency code	USD

Shipper Signature Date

Steve Blume 03.03.2006

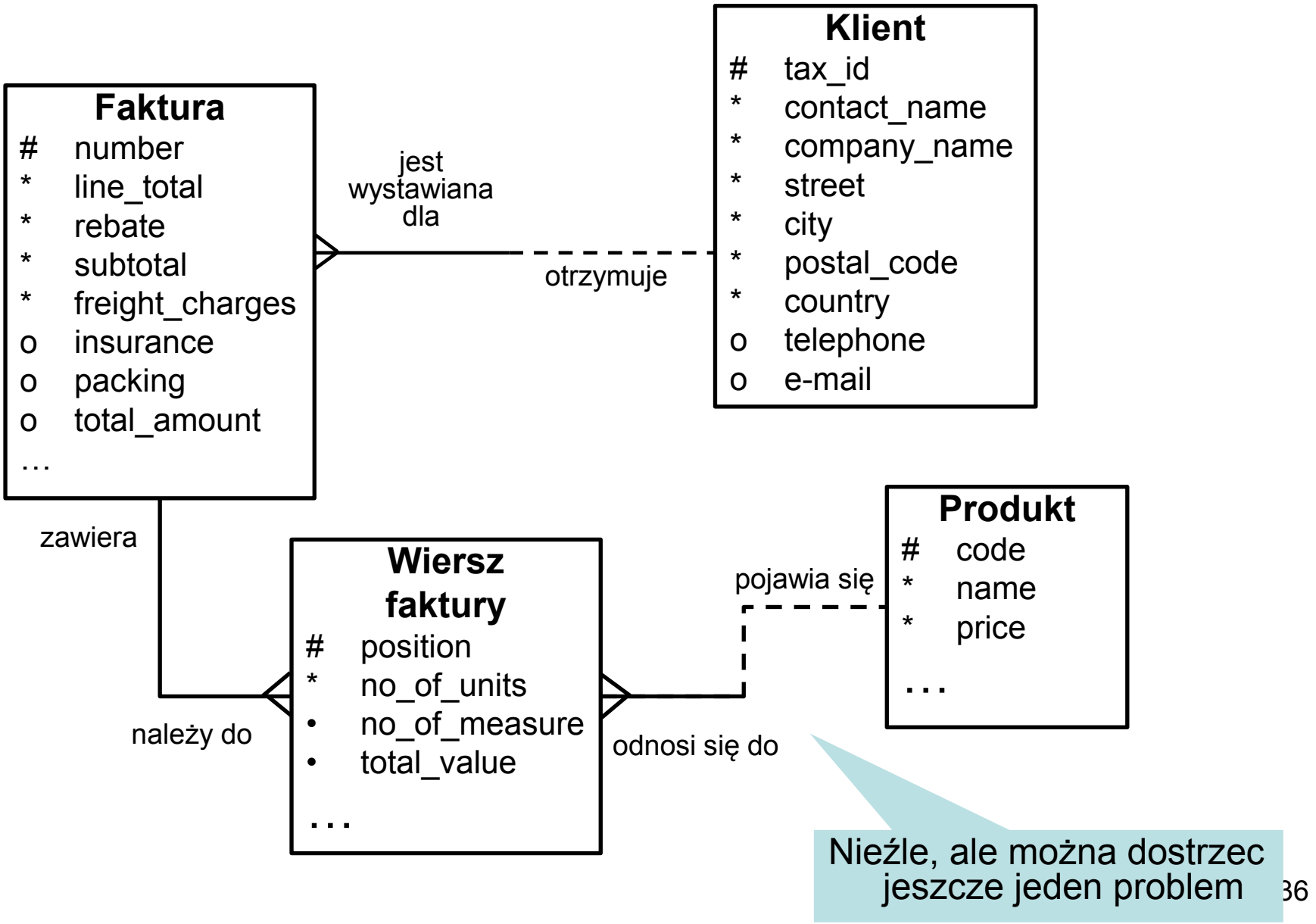
Gdzie znaleźć atrybuty nowej encji?

To jest produkt – encja, w stosunku do którego ustalmy realajcę

Pozostałe informacje stają się atrybutami encji

No. Units	Units of measure	Description of Goods	Country of Origin	Unit Value	Total Value
50	Pr.	Aluminium Windshield Wiper Assemblies	US	$54.50	$2725.00
125	Ea.	Rubber Windshield Wiper Replacement Blades	US	$3.99	$498.75
2	Ea.	Automotive Technical Books	US	$18.95	$37.90

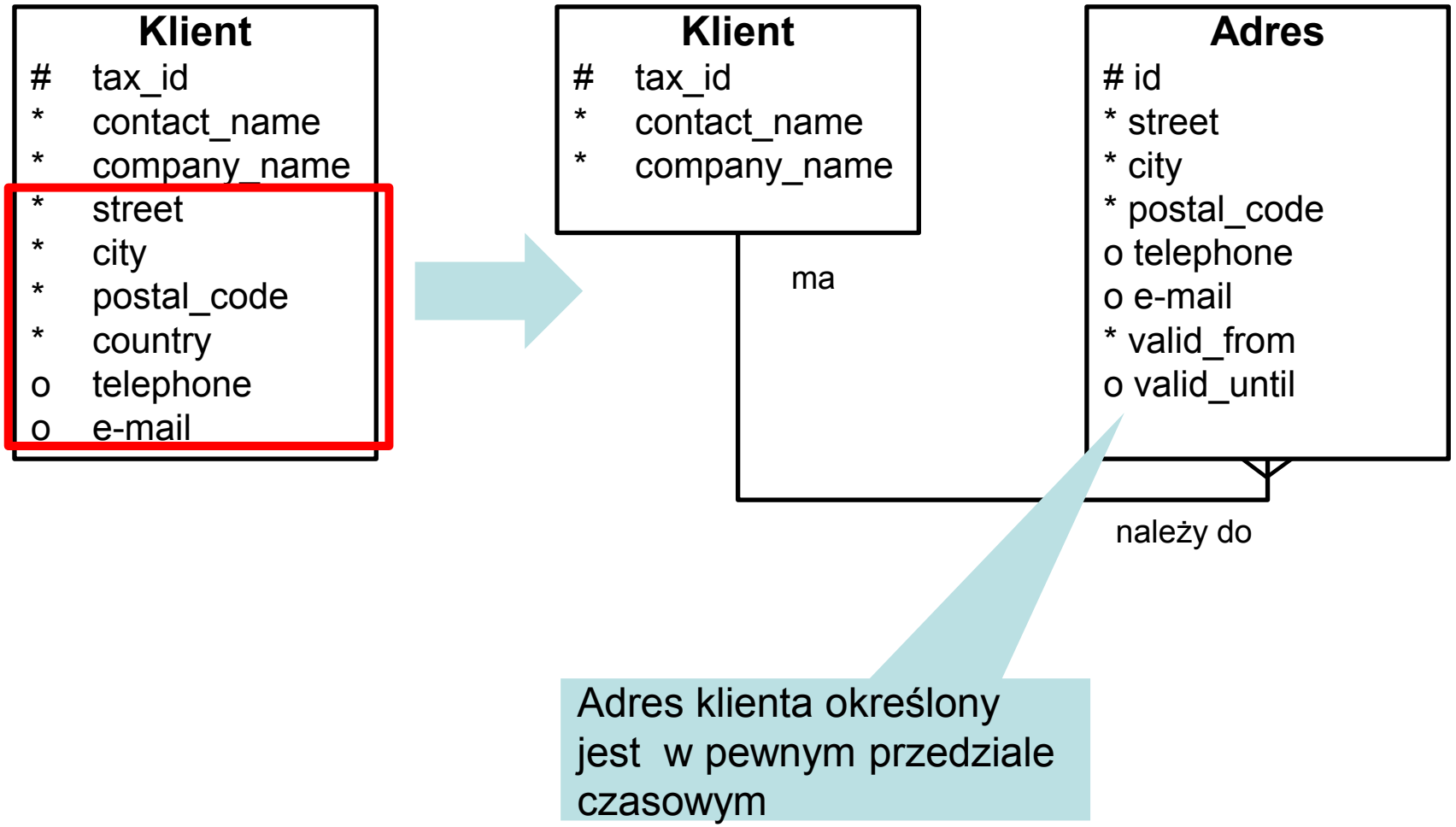
Otrzymany znormalizowany diagram związków encji



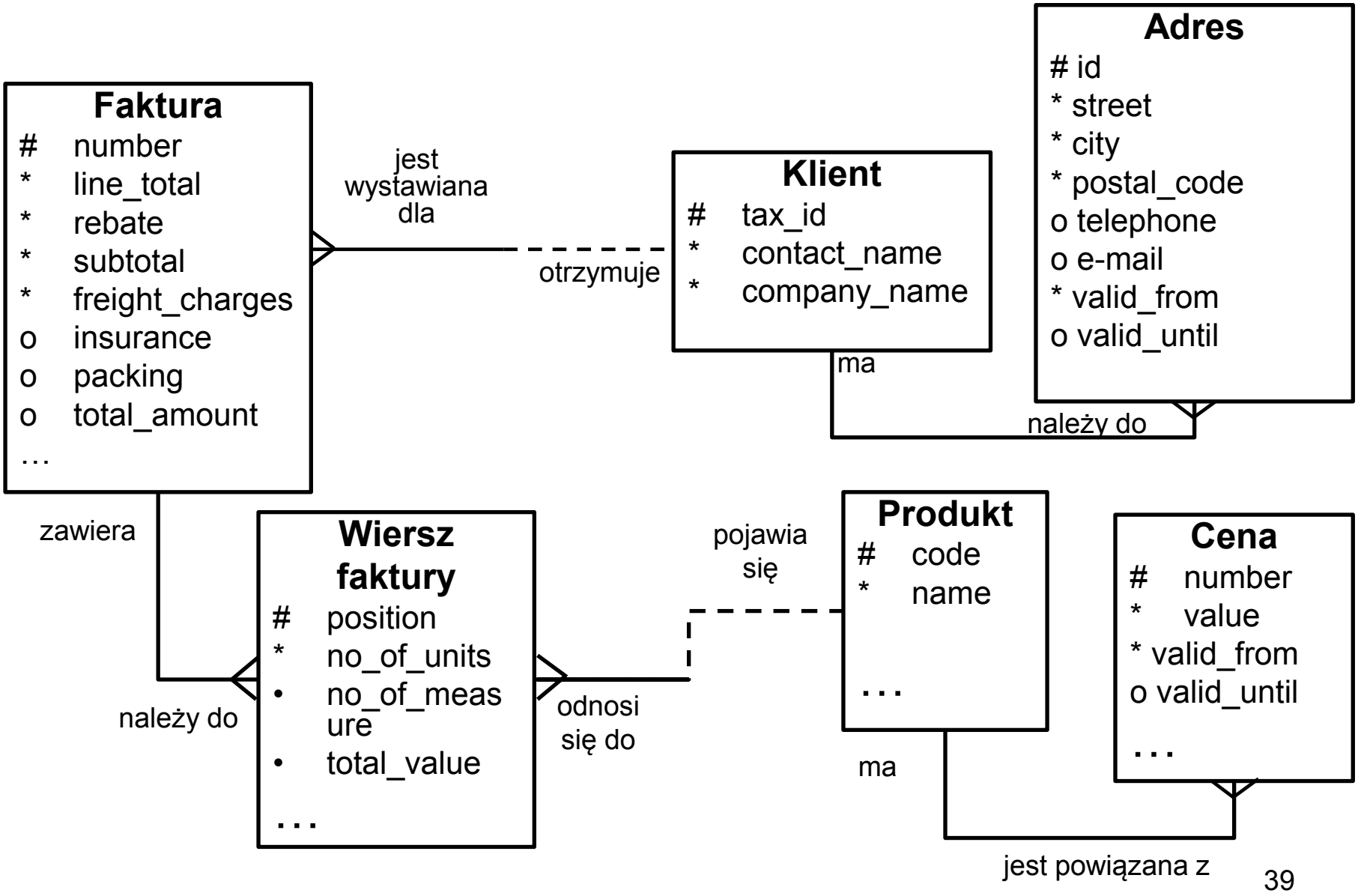
Zmiana informacji w bazie

- Przykładowy scenariusz:
  - Klient zmienił adres i poinformował na o tym
  - Dokonujemy zmiany adresu klienta w bazie
  - Po jakimś czasie otrzymujemy polecenie przeanalizowania naszych transakcji wtedy, gdy klient miał siedzibę „pod starym adresem”
  - **Nie jesteśmy w stanie tego zrobić!!!**
  - **W konsekwencji na etapie projektowania bazy musimy uwzględnić możliwość przechowywania danych historycznych o pewnych encjach!**

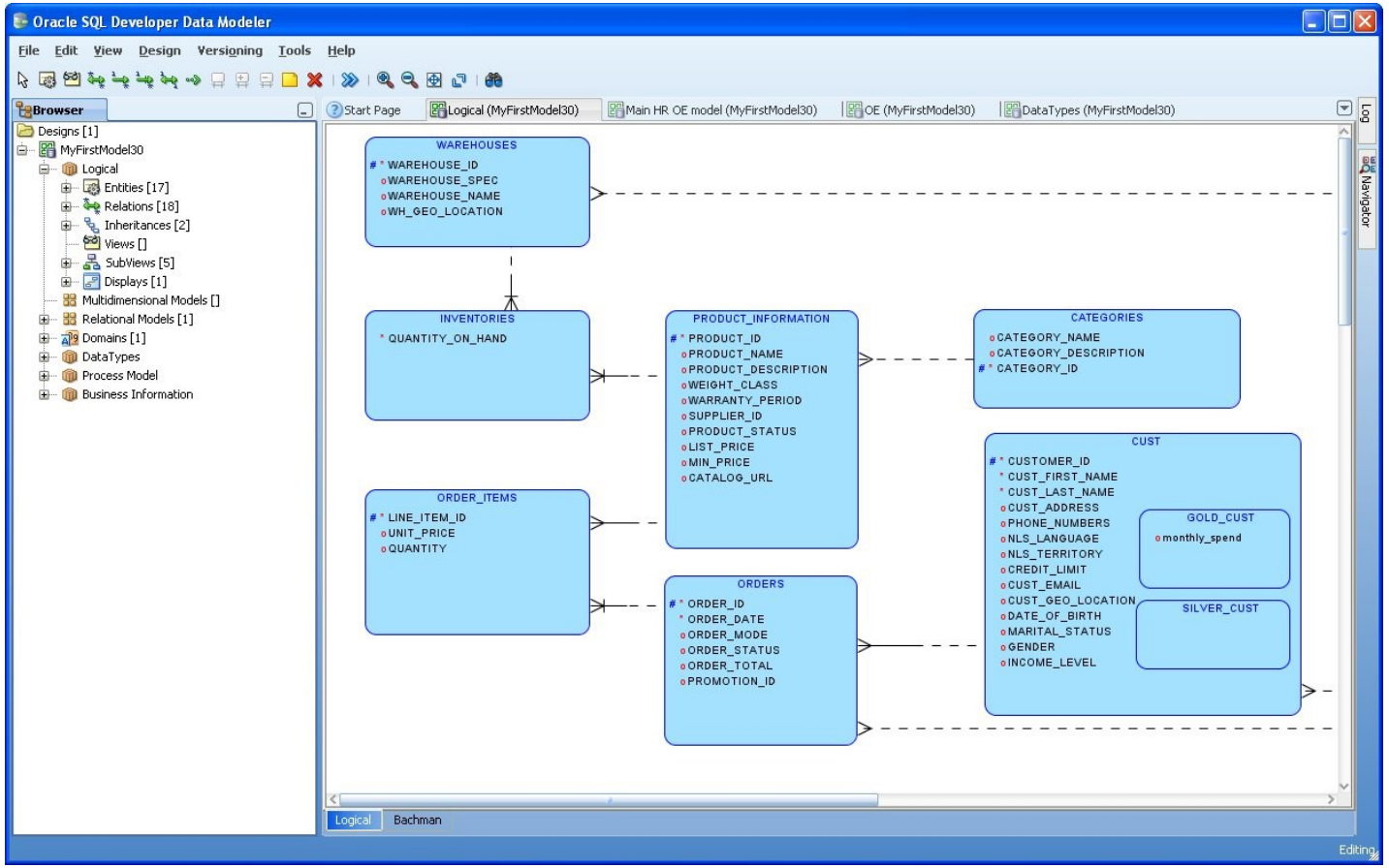
Uwzględnianie danych historycznych



Ostateczny diagram związków encji dla faktury



SQL DEVELOPER DATA MODELER



Uwaga: Istnieją darmowe narzędzia pozwalające na narysowanie diagramu związków encji i wygenerowanie na ich podstawie schematu bazy danych.

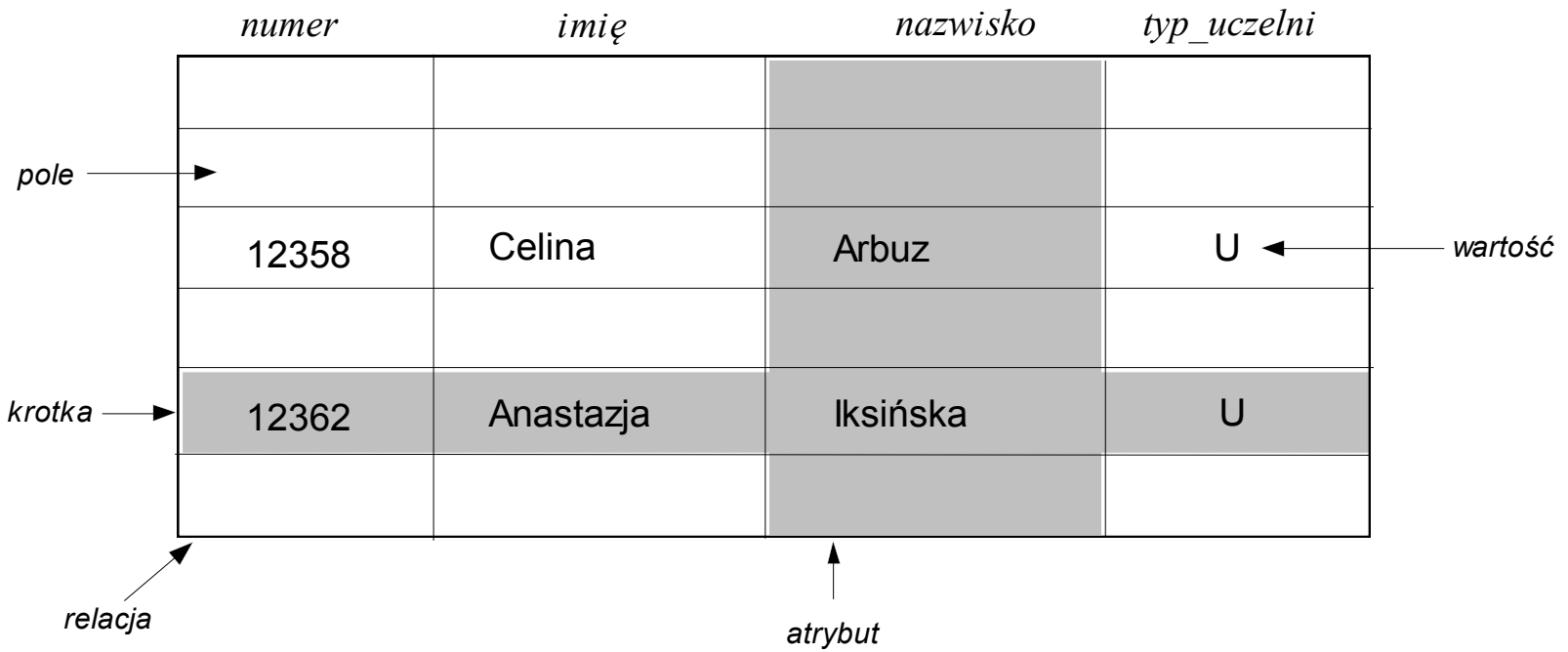
Algebra Relacji

Algebra relacji

Model danych w bazach relacyjnych obejmuje 3 składowe:

- relacyjne struktury danych,
- operatory algebry relacyjnej,
- ograniczenia integralnościowe, określające możliwe wartości danych w sposób jawny lub niejawny.

Struktura relacji

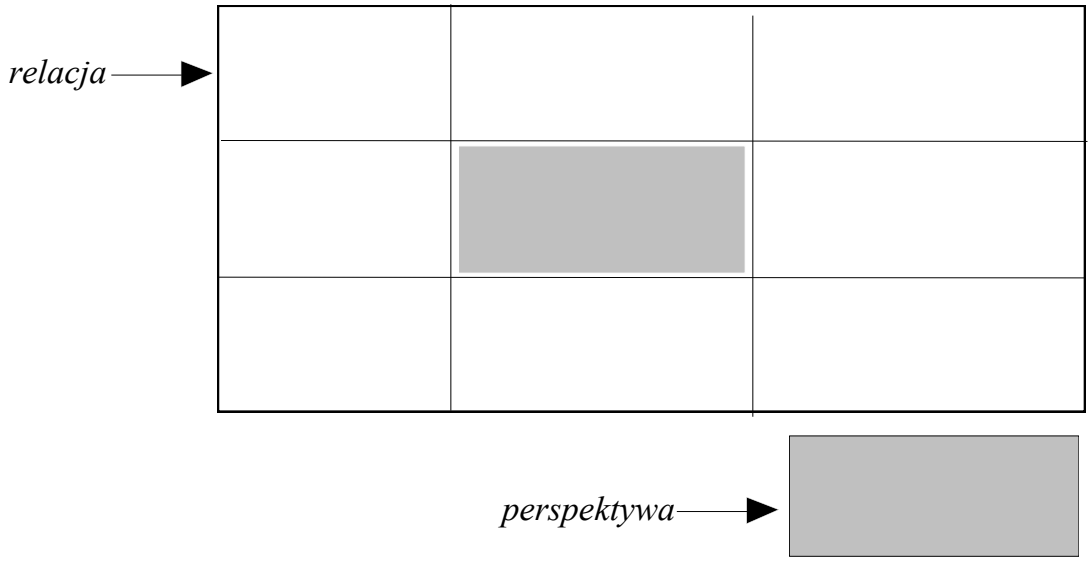


Relacja jest skończonym zbiorem wierszy (krotek) posiadających taką samą strukturę (schemat) i różne wartości. Każda krotka zawiera wartość co najmniej jednego atrybutu z określonej dziedziny, a wszystkie wartości atrybutów stanowią kolumnę określoną jako pole.

Każda relacja charakteryzuje się własnościami:

- Wszystkie krotki są różne
- Wszystkie atrybuty są różne
- Kolejność atrybutów i kolejność krotek jest nieistotna
- Wartości atrybutów są niepodzielne.

Perspektywa



Perspektywa jest rodzajem okna przez które odczytujemy lub modyfikujemy dane z relacji lub zbioru relacji.

Cechy perspektywy:

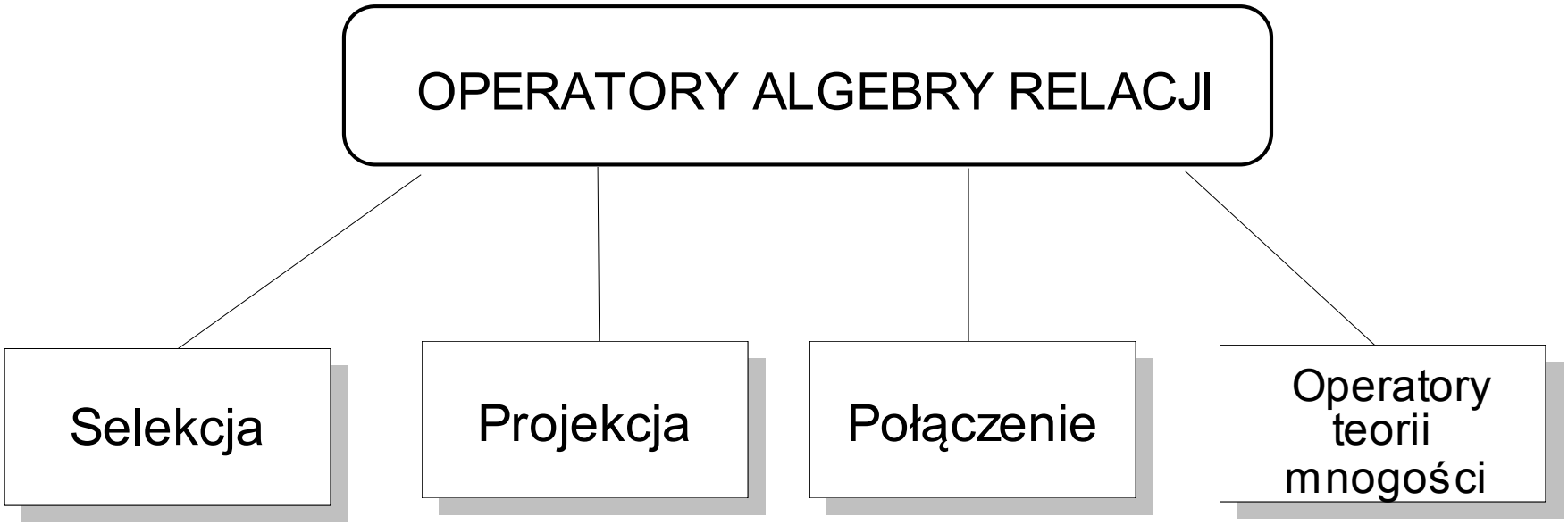
- Ogranicza zakres dostępu do określonych atrybutów i krotek
- Jest zdefiniowana z zastosowaniem co najmniej jednej relacji lub innej perspektywy
- Jest pamiętana w systemie wyłącznie w postaci swojej definicji, a zatem nie ma własnych danych; każdorazowe odwołanie się do niej wymaga pobrania danych z krotek relacji bazowych.

Perspektywy stosuje się w celach:

- Ograniczenia dostępu do relacji
- Uproszczenia zapytań
- Zapewnienia niezależności danych.

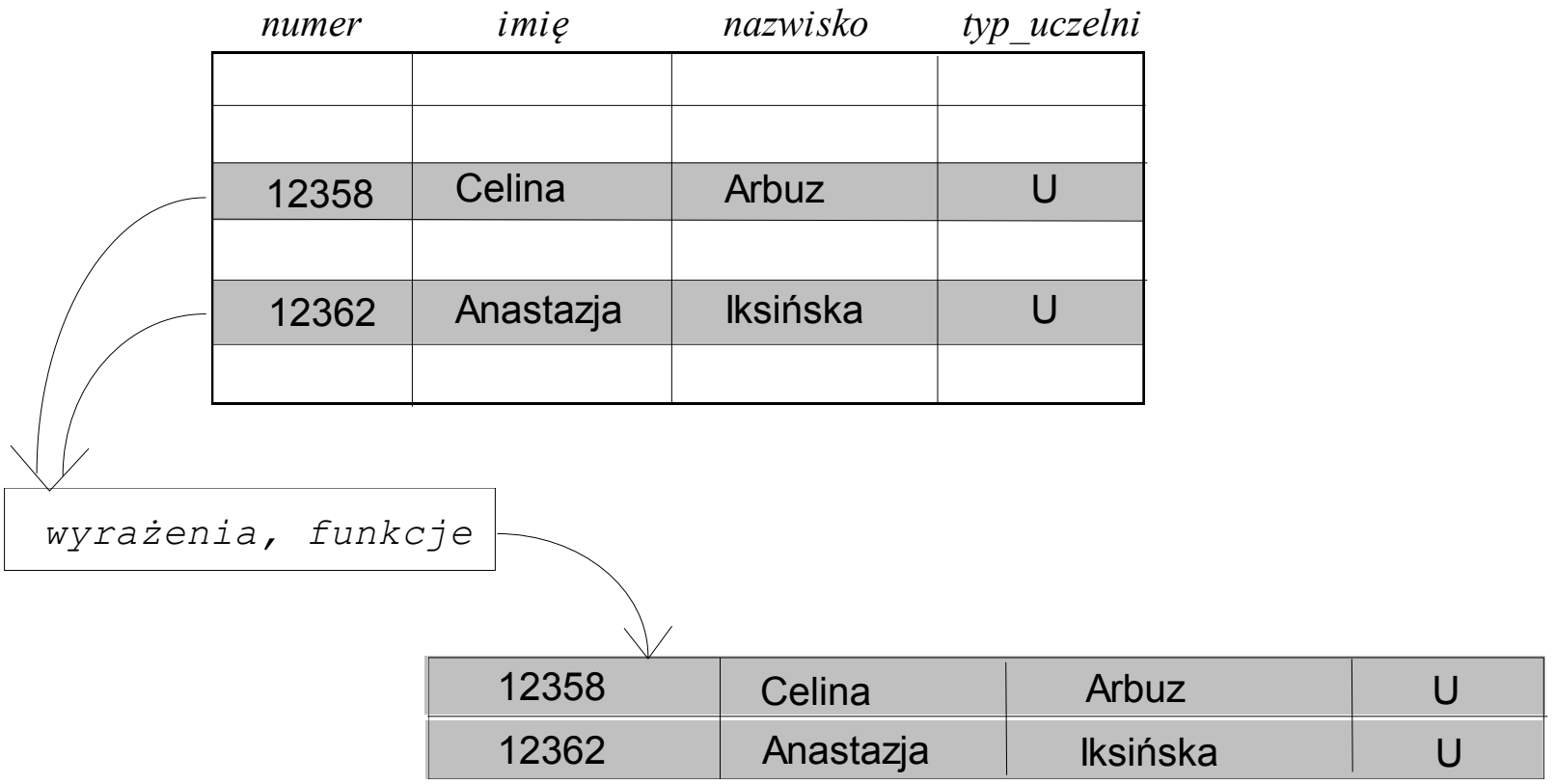
Operatory algebry relacji (1)

Operatory algebry relacji działają na jednej lub więcej relacjach, a wynikiem jest relacja.



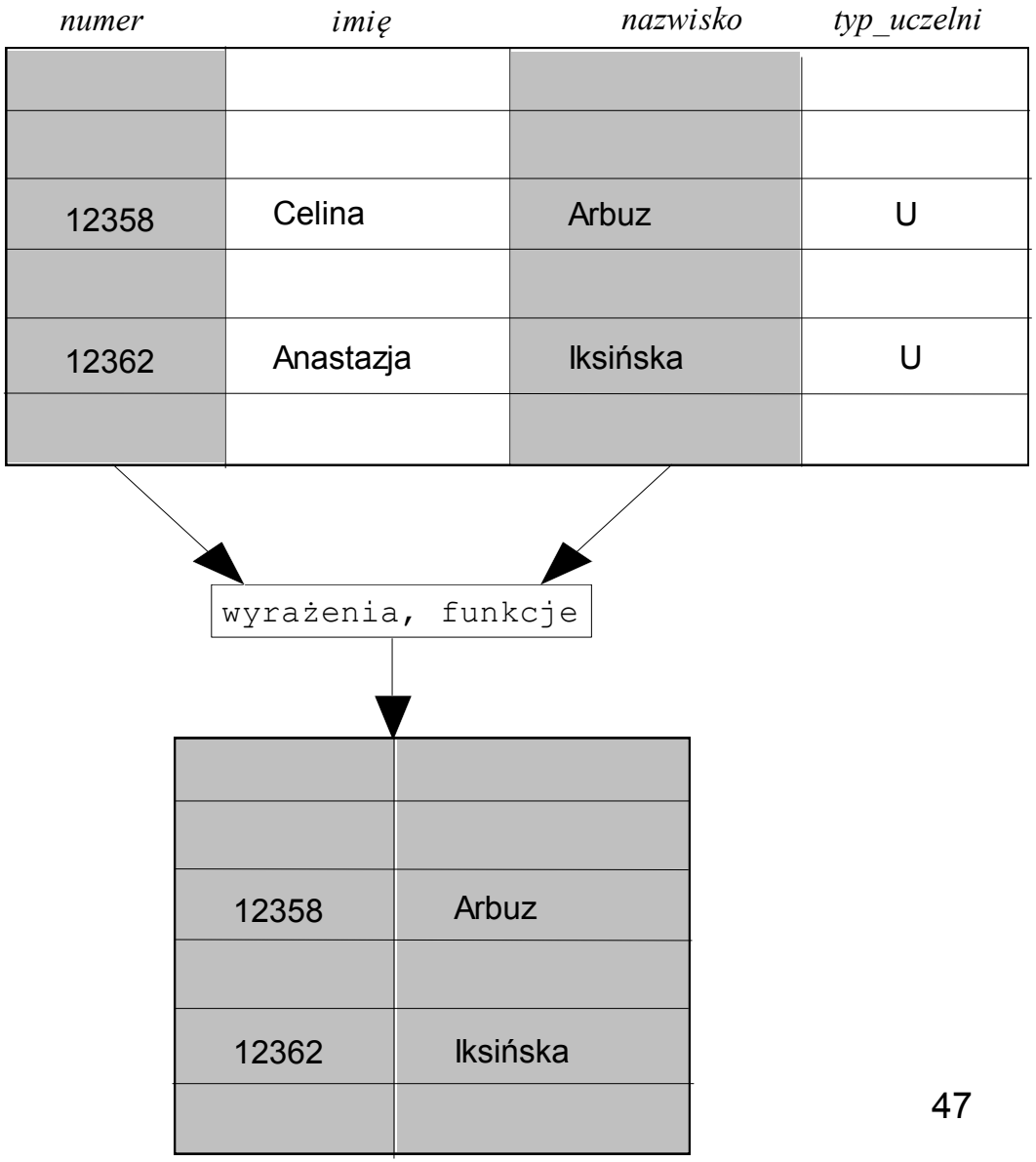
Operatory algebry relacji (2)

- **Selekcja**
  - umożliwia wybór tych krotek relacji, które spełniają określony warunek.



Operatory algebry relacji (3)

- **Projekcja**
  - umożliwia okrojenie relacji do wybranych atrybutów,
  - może być łączona z operacją selekcji – wtedy pobieranie atrybutów określonych projekcją odbywa się wyłącznie z krotek wskazanych przez selekcję.



Operatory algebry relacji (4)

- **Połączenie**
  - umożliwia konkatenację dwóch lub więcej relacji z zastosowaniem określonego warunku połączenia

numer	*imię*	*nazwisko*	*typ_uczelni*
			P
			P
12358	Celina	Arbuz	U
			P
12362	Anastazja	Iksińska	U
			A

typ_uczelni	*nazwa*
P	Politechnika
U	Uniwersytet
A	Akademia

• Celina	Arbuz	Uniwersytet
• Anastazja	Iksińska	Uniwersytet

Operatory algebry relacji (5)

- Operatory teoriomnogościowe
  - *Unia* - sumowanie krotek dwóch lub więcej relacji; warunkiem jest zgodność typów i liczby atrybutów relacji źródłowych.
  - *Przekrój* - iloczyn zbiorów krotek dwóch lub więcej relacji tzn. zbiór tych krotek, które występują jednocześnie w tych relacjach; podobnie jak w przypadku unii, warunkiem jest zgodność liczby i typów atrybutów.
  - *Różnica* - wyodrębnia krotki występujące wyłącznie w pierwszej spośród relacji wyjściowych.

Własności relacyjnej bazy danych

- Relacyjna baza danych jest widziana przez użytkownika jako zbiór relacji.
- Dostępny jest zbiór operatorów umożliwiających łączenie lub wydzielanie części relacji.
- Występuje całkowita niezależność danych.
- Nie istnieją jawne wskaźniki; powiązania danych są realizowane za pomocą samych tych danych (wspólnych wartości atrybutów).
- Stosowany język jest językiem nieproceduralnym.
- Użytkownik nie specyfikuje ścieżek dostępu do danych i nie musi znać fizycznej reprezentacji danych.

Normalizacja źle zaprojektowanej bazy danych

Normalizacja bazy danych

- Na etapie budowy modelu konceptualnego bazy danych otrzymujemy wiele elementów danych, z których ma powstać przyszły schemat bazy danych.
- Dane musimy znormalizować, tj. przypisać atrybuty odpowiednim encjom i ewentualnie wprowadzić dodatkowe encje.
- Diagramy związków encji prowadzą w naturalny sposób do znormalizowanych modeli, choć zależy to od zdolności analityka.
- Stosując bardziej ścisłe podejście do normalizacji stosuje się tzw. **postacie normalne** relacji.
- Dotychczas zdefiniowano pięć postaci normalnych, choć tylko trzy pierwsze są powszechnie używane do projektowania baz danych.
- Postacie o wyższych numerach mają tę własność, że automatycznie spełniają warunki dla postaci o numerach niższych. Otrzymujemy je zatem przez nakładanie dodatkowych warunków na postacie o numerach niższych.

Przykład 1 (1)

Zakładamy następujący wejściowy zbiór atrybutów dotyczących zamówienia.

NAZWA		ZNACZENIE
nr_zam	-	numer zamówienia
data_zam	-	data zamówienia
id_kl	-	identyfikator klienta
nazwa_kl	-	nazwa klienta
adres_kl	-	adres klienta
kod_w	-	kod wyrobu
nazwa_w	-	nazwa wyrobu
il_zam	-	ilość zamówiona
cena_w	-	cena jednostkowa wyrobu
wyr_og	-	cena całkowita wyrobu
zam_og	-	wartość całkowita zamówienia

Przykładem schematu relacji może być zatem ciąg zawierający wszystkie atrybuty:
R={nr_zam, data_zam, id_kl, nazwa_kl, adres_kl, zam_og, kod_w, nazwa_w, cena_w, il_zam, wyr_og }

Zadaniem jest zaprojektowanie znormalizowanego schematu bazy danych przechowującej wartości atrybutów wskazanych w przykładzie.

Definicje (1)

- **Definicja 1**

  Schematem relacji R nazywany ciąg atrybutów relacji (nazw) postaci R={A1,...,Ak} należących do pewnego zbioru atrybutów U={A1,...,An} dla k<n. Z każdym atrybutem Ai jest związany zbiór wartości Di nazywany dziedziną atrybutu Ai.
- **Definicja 2**

  Relacją r o schemacie R nazywamy dowolny podzbiór iloczynu kartezjańskiego dziedzin atrybutów, tj. $r \subset D1 \times D2 \times ... \times Dk$.

  Najczęściej relacja r jest przedstawiana jako tablica, której kolumny odpowiadają atrybutom tworzącym schemat, a każdy wiersz stanowi tzw. *krotkę* relacji.
- **Definicja 3**

  Schematem relacyjnej bazy danych nazywamy zbiór schematów relacji Z={R1,...,Rp} utworzony nad zbiorem atrybutów U.

Przykład 2 (1)

Załóżmy dalej, że atrybuty zgrupowano tak, aby tworzyły dwa następujące schematy relacji.

- Schemat relacji *Zamówienie* (R1).

  R1={nr_zam, data_zam, id_kl, nazwa_kl, adres_kl, zam_og}

- Schemat relacji *Pozycja-zamówienia* (R2).

  R2={nr_zam, kod_w, nazwa_w, cena_w, il_zam, wyr_og}

Przykładowe relacje otrzymane na podstawie schematów R1, R2

nr_zam	data_zam	id_kl	nazwa_kl	adres_kl	zam_og
1	02-mar-93	5	Zakłady A	Warszawa	37,00 zł
2	04-mar-93	8	Zakłady C	Kraków	100,00 zł
3	05-mar-93	5	Zakłady A	Warszawa	35,00 zł
4	10-mar-93	5	Zakłady A	Warszawa	214,00 zł
5	13-mar-93	3	Zakłady E	Katowice	50,00 zł
6	21-mar-93	8	Zakłady C	Kraków	200,00 zł
7	28-mar-93	3	Zakłady E	Katowice	70,00 zł

Relacja *Zamówienie*.

nr_zam	kod_w	nazwa_w	cena_w	il_zam	wyr_og
0001	001	śruba	2,00 zł	10	20,00 zł
0001	003	nakrętka	1,00 zł	10	10,00 zł
0001	005	wiertło	7,00 zł	1	7,00 zł
0002	002	gwoździe	1,00 zł	100	100,00 zł
0003	005	wiertło	7,00 zł	5	35,00 zł
0004	001	śruba	2,00 zł	100	200,00 zł
0004	005	wiertło	7,00 zł	2	14,00 zł
0005	002	gwoździe	1,00 zł	50	50,00 zł
0006	002	gwoździe	1,00 zł	200	200,00 zł
0007	002	gwoździe	1,00 zł	70	70,00 zł

Relacja *Pozycja-zamówienia*.

Zasady projektowania bazy danych

- Projektując relacyjną bazę danych musimy dokonać wyboru pomiędzy różnymi możliwymi zbiorami schematów relacji. Z pewnych względów jedne schematy są bardziej „odpowiednie“ niż inne.
- Podstawą projektowania (schematu) relacyjnej bazy danych jest analiza powiązań pomiędzy atrybutami. Chodzi tu o utworzenie schematów o takiej postaci, aby żaden z nich nie był zbędny oraz o wydzielenie związków (zależności) między atrybutami w oddzielne relacje. Osiągnięcie tego celu prowadzi do uniezależnienia operacji wprowadzania, usuwania i aktualizacji danych w jednych relacjach od danych w innych relacjach.

Definicje (2)

- **Definicja 4**

  Kluczem relacji nazywamy taki zbiór atrybutów relacji, dla których kombinacje ich wartości jednoznacznie identyfikują każdą krotkę tej relacji. Wymaga się przy tym aby żaden podzbiór klucza nie był kluczem.
- **Definicja 5**

  W przypadku ogólnym w relacji można wyodrębnić wiele kluczy, które *nazywamy kluczami potencjalnymi* (kandydującymi). Klucz wybrany z potencjalnych określony jest jako *główny*.

  Definicja 6

  Klucz nazywamy *prostym* jeśli jest jednoelementowy, a *złożonym*, jeśli składa się z więcej niż jednego atrybutu.
- **Definicja 7**

  Atrybut nazywamy *głównym* jeśli wchodzi w skład klucza.

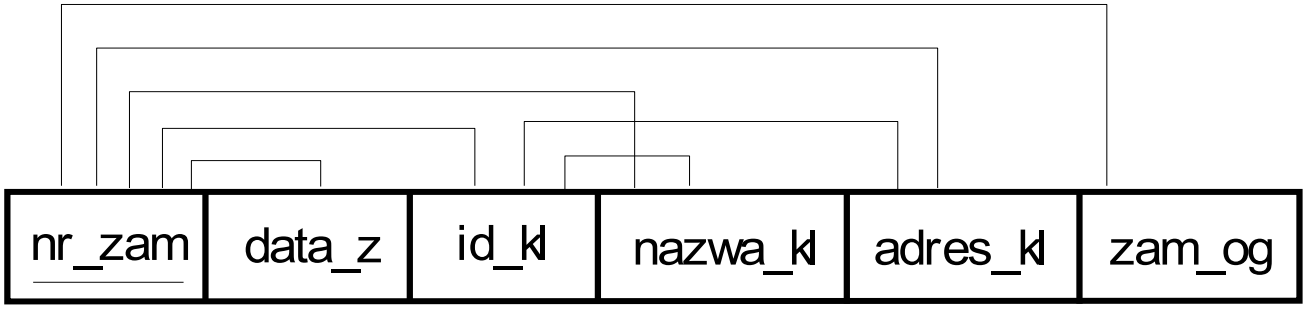
  Ponadto w praktyce w relacyjnych bazach danych ważnym pojęciem jest **klucz obcy.** Termin taki odnosi się do **takiego atrybutu relacji, który występuje jako klucz główny w innej relacji.**
- **Definicja 8**

  Atrybut B pewnej relacji jest *fukcjonalnie zależny* od atrybutu A (co zapisujemy: $A \rightarrow B$), jeśli każdej wartości „a” z A odpowiada nie więcej niż jedna wartość „b” z B,

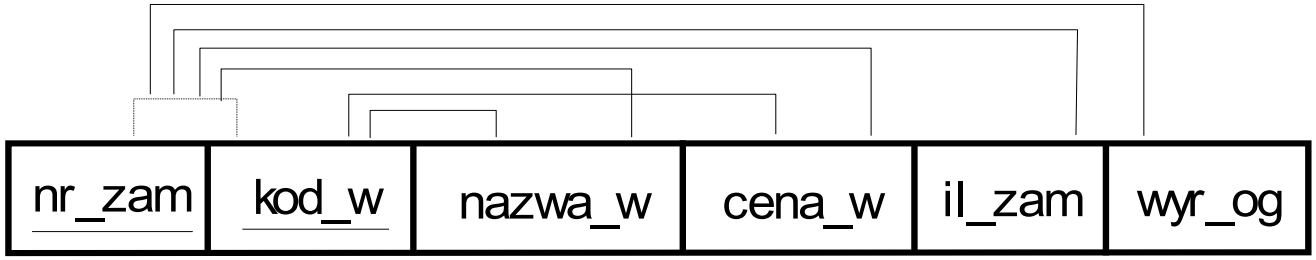
Przykład 2 (2)

Zależności funkcjonalne w schematach relacji R1 i R2:

R1: *Zamówienie*



R2:
Pozycja-zamówienia



Pierwsza postać normalna

- Definicja 9
  Relacja jest w pierwszej postaci normalnej, jeśli każdy atrybut tej relacji nie wchodzący w skład klucza jest funkcyjnie zależny od klucza.

Uwaga:

Relacje z przykładu 2 są w pierwszej postaci normalnej co jest cechą każdej relacji, wynikającą z definicji.

Niekorzystne zjawiska w pierwszej postaci normalnej

- dublowanie danych - adres klienta jest pamiętany wielokrotnie,
- możliwość wystąpienia niespójności danych (np. wskutek uaktualnienia adresu w niektórych krotkach i pozostawienia starego w pozostałych),
- utrata adresu klienta wraz z usunięciem jego zamówienia,
- brak możliwości zapamiętania adresu klienta, który nie złożył żadnego zamówienia.

Druga postać normalna

- **Definicja 10**
  Atrybut B jest *w pełni funkcjonalnie zależny* od A jeśli jest od niego zależny funkcjonalnie ale nie jest zależny funkcjonalnie od żadnego podzbioru tego atrybutu.

  W relacji według schematu R2 z przykładu 2 podkreślamy atrybuty stanowiące klucz złożony: nr_zam i kod_w.

  R2={nr_zam, kod_w, nazwa_w, cena_w, il_zam, wyr_og}

  Zauważmy, że występuje tam niepełna zależność funkcjonalna atrybutu nazwa_w od klucza relacji (nr_zam, kod_w)

- **Definicja 11**
  Relacja jest w drugiej postaci normalnej jeśli każdy atrybut tej relacji nie wchodzący w skład klucza jest w pełni funkcjonalnie zależny od wszystkich kluczy potencjalnych

Przykład 3 (1)

- W relacji R2 występują następujące niepełne zależności funkcjonalne:
  kod_w$\rightarrow$nazwa_w

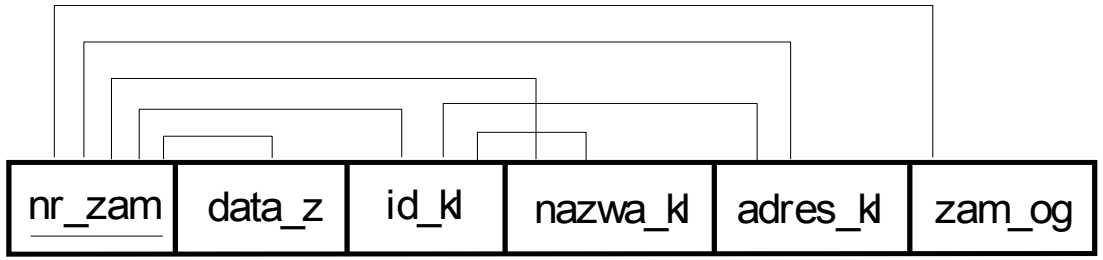
  oraz

  kod_w$\rightarrow$cena_w.

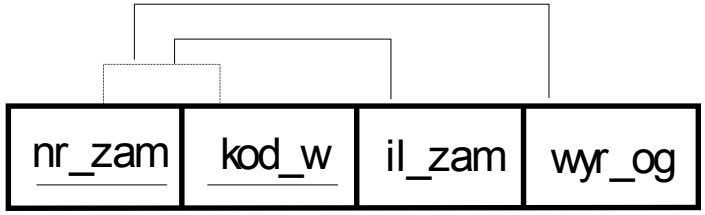
  Eliminując te zależności otrzymujemy następujący nowy zbiór schematów relacji (R21 R22 R23)

Przykład 3 (2)

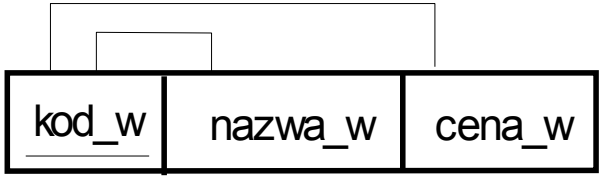
R21: *Zamówienie*



R22: *Pozycja-zamówienia*



R23: W*yrób*



Zauważmy, że relacje R21, R22, R23 nadal zawierają dublujące się dane. Wynika to z istnienia tranzytywnych zależności pomiędzy atrybutami.

Trzecia postać normalna

- **Definicja 12**
  Niech A,B,C będą rozłącznymi podzbiorami atrybutów pewnej relacji. Mówimy, że atrybut C jest tranzytywnie funkcjonalnie zależny od A jeśli A$\rightarrow$B, B$\rightarrow$C i nieprawda, że (B$\rightarrow$A lub C$\rightarrow$B). Używa się wtedy określenia, że „A wyznacza C w sposób przechodni”.
- Rozważymy schemat relacji R1 z przykładu 2 i podkreślimy klucz główny:

  R1={nr_zam, data_zam, id_kl, nazwa_kl, adres_kl, zam_og}.

  Zachodzą tutaj zależności nr_zam$\rightarrow$id_kl oraz id_kl$\rightarrow$nazwa_kl. Ponieważ zależności odwrotne nie są prawdziwe, stąd nazwa_kl jest tranzytywnie funkcjonalnie zależna od nr_zam.
- **Definicja 13**

  Relacja jest w trzeciej postaci normalnej jeśli jest ona w drugiej postaci normalnej i każdy jej atrybut nie wchodzący w skład żadnego klucza potencjalnego nie jest tranzytywnie funkcjonalnie zależny od żadnego klucza potencjalnego tej relacji.

Przykład 4 (1)

- Rozważmy relację R21 z przykładu 5.5. W relacji tej występują m. in. następujące zależności funkcjonalne:

  nr_zam$\rightarrow$id_kl id_kl$\rightarrow$nazwa_kl

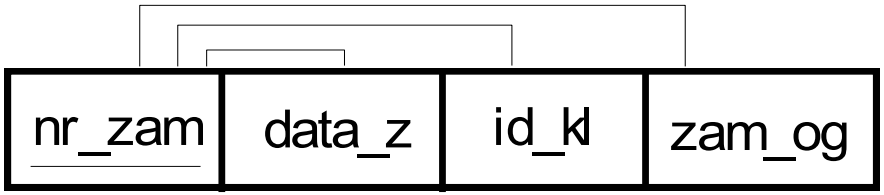
  oraz

  nr_zam$\rightarrow$id_kl id_kl$\rightarrow$adres_kl,

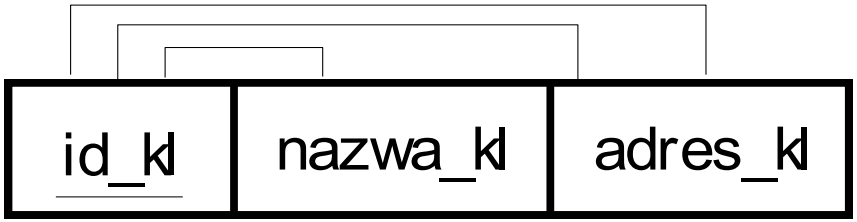
  co oznacza, że atrybuty nazwa_kl i adres_kl są tranzytywnie funkcjonalnie zależne od atrybutu nr_zam. Zależności te można usunąć modyfikując odpowiednio schemat relacji *Zamówienie*. Prowadzi to do trzeciej postaci normalnej pokazanej na KOLEJNYM SLAJDZIE.

Przykład 4 (2)

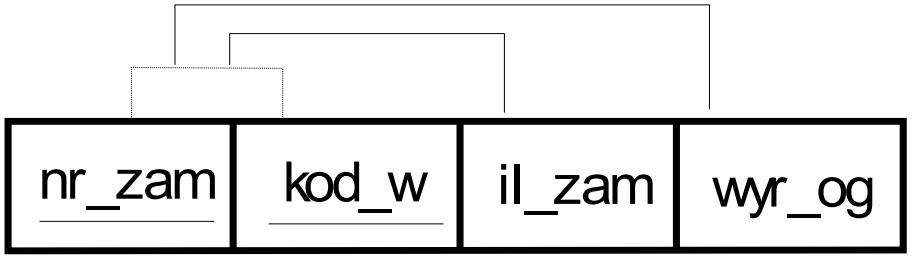
R31: *Zamówienie*



R32: *Klient*



R33: *Pozycja-zamówienia*



R33: W*yrób*

Przykład 4 – baza znormalizowana

Relacja *Klient*

id_kl	nazwa_kl	adres_kl
05	Zakłady A	Warszawa
08	Zakłady C	Kraków
03	Zakłady E	Katowice

Relacja *Wyrób*.

kod_w	nazwa_w	cena_w
001	śruba	2,00 zł
002	gwoździe	1,00 zł
003	nakrętka	1,00 zł
005	wiertło	7,00 zł

Relacja *Zamówienie*.

nr_zam	data_zam	id_kl	zam_og
0001	02-mar-93	05	37,00 zł
0002	04-mar-93	08	100,00 zł
0003	05-mar-93	05	35,00 zł
0004	10-mar-93	05	214,00 zł
0005	13-mar-93	03	50,00 zł
0006	21-mar-93	08	200,00 zł
0007	28-mar-93	03	70,00 zł

Relacja *Pozycja-zamówienia*.

nr_zama	kod_w	il_zam	wyr_og
0001	001	10	20,00 zł
0001	003	10	10,00 zł
0001	005	1	7,00 zł
0002	002	100	100,00 zł
0003	005	5	35,00 zł
0004	001	100	200,00 zł
0004	005	2	14,00 zł
0005	002	50	50,00 zł
0006	002	200	200,00 zł

Praktyczne uwagi dotyczące tworzenia relacyjnych baz danych na przykładzie MS SQL Server

Logiczna struktura relacyjnych baz danych

- Logiczna struktura bazy danych określa strukturę informacji zawartej w bazie danych – logikę bazy.
- Do logicznej struktury bazy danych zalicza się:
  - tabele,
  - widoki,
  - indeksy,
  - procedury składowane,
  - funkcje użytkownika,
  - wyzwalacze,
  - synonimy.

Tabele

- **Tabele** są podstawowymi strukturami do przechowywania danych. Definicja tabeli składa się przede wszystkim z definicji kolumn, które ją tworzą. Definicja kolumny obejmuje określenie typu zmiennych, informację o domyślnej wartości pól kolumny i o tym, czy mogą one przyjmować wartości nieokreślone (NULL), ograniczenia nałożone na dane oraz klucze podstawowe i obce. Klucze podstawowe służą do jednoznacznego rozróżnienia wierszy wchodzących w skład tabeli. Klucze obce odwołują się do wartości kluczy podstawowych w innych tabelach, tworząc w ten sposób relację pomiędzy nimi.

Indeksy

- **Indeksy** są pomocniczymi strukturami, usprawniającymi wyszukiwanie danych w tabeli. Przechowują one wskazania do wierszy w tabeli, zawierających konkretną wartość. Ponieważ indeksy mają strukturę drzewa, ich przeszukiwanie jest znacznie szybsze niż przeszukiwanie całej tablicy. Przy braku indeksów, każde przeszukiwanie wiązałoby się z koniecznością przeszukiwania całej tabeli danych.

Widoki

- **Widoki** są strukturami podobnymi do tabel, z tą jednak różnicą, że nie przechowują same danych, a jedynie odwołują się do danych zapisanych w innych tabelach. Jeżeli dla przykładu dane o klientach są zapisane w kilku tabelach (np. osoba, adres, miasto, ulica, firma) w postaci znormalizowanej, jeden widok może złączyć wszystkie dane, pokazując je jako jedną tabelę. Odczytywanie danych z widoku odbywa się w ten sam sposób, jak odczytywanie danych z tabeli.

Procedury składowane

- **Procedury składowane** są to fragmenty wykonywalnego kodu SQL, przechowywanego na serwerze. Do procedur można przekazywać parametry, a procedura może zwracać parametry na zewnątrz.

Wyzwalacze

- **Wyzwalacze** są to fragmenty kodu SQL, które są wykonywane, gdy na serwerze wystąpi określona akcja. Zwykle tymi akcjami są operacje na danych zawartych w tabelach i widokach. Dobrym przykładem użycia wyzwalacza jest zapisywanie starych wartości danych do tabeli archiwalnej.

Wyzwalacze

- **Wyzwalacze** są to fragmenty kodu SQL, które są wykonywane, gdy na serwerze wystąpi określona akcja. Zwykle tymi akcjami są operacje na danych zawartych w tabelach i widokach. Dobrym przykładem użycia wyzwalacza jest zapisywanie starych wartości danych do tabeli archiwalnej.

Funkcje użytkownika i synonimy

- **Funkcje użytkownika** są podobne do procedur, z jedną różnicą – muszą bezpośrednio zwracać wartość i mogą być bezpośrednio używane w przypisaniach.
- **Synonimy** są to inne, zastępcze nazwy obiektów znajdujących się w bazie danych.

Dane zapisywane w tabelach

- Wszystkie dane zawarte w bazie są zapisane w tabelach. Definicja tabeli zawiera definicje kolumn, które ją tworzą. Podstawową informacją o kolumnie jest typ przechowywanych przez nią danych.

Dane liczbowe

Tabela 1.1 Całkowite typy danych

Typ	**Zakres**	**Liczba bajtów**
Tinyint	0..255	1
Smallint	-32768..32767	2
Int	-2 147 483 648.. 2 147 483 647	4
bigi nt	-9 223 372 036 854 775 808 .. 9 223 372 036 854 775 807	8

Tabela 1.2 Typy pieniężne

Typ	**Zakres**	**Liczba bajtów**
smallmoney	-214 478.3648.. 214 748.3647	4
money	-922 337 203 685 477.5808 .. 922 337 203 685 477.5807	8

Tabela 1.3 Typy przybliżone

Typ	**Zakres**	**Liczba bajtów**
float	Od -1.79E+38 do -2.23E-38, 0, od 2.23E-38 do 1.79E+38	Zależna od precyzji
real	Od -3.40E+38 do -1.18E-38, 0, od 1.18E-38 do 3.40E+38	8

Data – czas / łańcuchy tekstowe

Tabela 1.4 Typy reprezentujące czas i datę

Typ	**Zakres**	**Dokładność**
datetime	1 stycznia 1753 do 31 grudnia 9999	3,33 ms
smalldatetime	1 stycznia 1900 do 6 czerwca 2079	1 min

Tabela 1.5 typy łańcuchowe

	Znakowe	Znakowe w formacie UNICODE	Binarne
Stała długość	char	nchar	Binary
Zmienna długość	varchar	nvarchar	Varbinary
Duże obiektu	text lub vartext(max)	ntext lub nvartex(max)	image lub varbinary(max)

Uwaga: Dostępne są również inne typy danych: sql_variant, timestamp, typ tabelaryczny, uniqueidentifier, kursor, XML, oraz typy definiowane przez użytkownika.

Ograniczenia związane z kolumną

- Z kolumną są związane ograniczenia (ang. constraint). Ograniczenia te mogą być następujące:
  - NULL/NOT NULL
  - CHECK
  - UNIQUE
  - PRIMARY KEY
  - FOREGIN KEY

NULL/NOT NULL

- **NOT NULL** – to ograniczenie określa, że kolumna nie może przyjmować wartości nieokreślonej (NULL). Wartość nieokreślona (NULL) oznacza, że polu nie została przypisana konkretna wartość danego typu.

CHECK

- **CHECK** – jest to wyrażenie logiczne związane z kolumną. Dane we wszystkich wierszach muszą pasować do wyrażenia tak, aby zawsze było ono prawdziwe. Jeżeli jakaś zmiana w bazie danych doprowadza do pogwałcenia ograniczenia narzuconego przez CHECK, jest wywoływany automatycznie błąd i bieżąca transakcja jest cofana.

UNIQUE

- **UNIQUE** – ograniczenie to oznacza, że wszystkie wartości w kolumnie muszą być różne.

PRIMARY KEY

- **PRIMARY KEY** – klucz główny, pola kolumn posiadających ten atrybut jednoznacznie identyfikują wiersze. Jako klucz główny może być użyta zarówno jedna kolumna jak i zbór kolumn.

FOREGIN KEY

- **FOREGIN KEY** – klucz obcy. Odwołuje się do wartości klucza głównego w innej tabeli, definiując relację pomiędzy tabelami. Kiedy relacja zostanie ustanowiona, wszystkie wartości w tej tabeli muszą mieć albo wartości nieokreślone, albo znajdujące się w kolumnie klucza głównego tabeli, do której się odwołujemy.

Tworzenie tabel

- Jeśli nie dysponujemy oprogramowaniem automatycznie generującym schemat bazy danych na podstawie diagramu ERD, to bazę tworzy się „ręcznie”
  - Pisząc odpowiedni skrypt w języku SQL lub,
  - Posługując się nakładkami graficznymi wspomagającymi projektowanie schematu bazy danych.

Praktyczna ścieżka tworzenia tabel

1. Zdefiniowanie nazw kolumn, typu danych, które będą przechowywać poszczególne kolumny oraz wskazanie, czy zezwala się, że dane mogą być „puste” (NULL) dla wszystkich tabel.
2. Wskazanie kluczy głównych tabel.
3. Zdefiniowanie w wybranych kolumnach założonych na etapie projektowania wartości domyślnych.
4. Uzupełnienie ograniczeń dla poszczególnych kolumn.
5. Zdefiniowanie powiązań pomiędzy relacjami – wskazanie kluczy obcych w danej tabeli, wskazujących klucze główne w innych tabelach.
6. Po zdefiniowaniu struktury bazy danych, można przystąpić do wprowadzania kolejnych wierszy z danymi.

Język SQL

Wprowadzenie do SQL (1)

- Język SQL powstał w firmie IBM w latach 70-tych, jako część projektu badawczego dotyczącego relacyjnych baz danych.
- Obecnie stał się światowym standardem dla języków baz danych i występuje w produktach większości firm sprzedających oprogramowanie dla baz danych.
- Nazwa SQL pochodzi od skrótu *Structural Query Language*, co oznacza „strukturalny język zapytań”.

Wprowadzenie do SQL (2)

- Polecenie SQL może być zapisane w pojedynczym wierszu lub kilku wierszach.
- W każdym poleceniu można wyróżnić tzw. klauzule (*clauses*) rozpoczynające się słowem kluczowym.
- W celu zwiększenia czytelności polecenia zaleca się pisanie klauzul w osobnych wierszach.
- Polecenie SQL jest wprowadzane po wyświetleniu „znaku zachęty” (*prompt*), postaci np.:

  SQL⟩
- Polecenie może być pisane zarówno dużymi, jak i małymi literami w formacie swobodnym, co oznacza, że trzy następujące polecenia:

  select *nazwisko* ***from*** *pracownik;*

  SELECT NAZWISKO
  FROM PRACOWNIK;

  select
  nazwisko
  from
  pracownik;

  są równoważne.

Proste użycie polecenia SELECT (1)

- Polecenia dotyczą bazy wskazanej na slajdzie 7.
- Najczęściej stosowanym poleceniem języka SQL jest polecenie ***select*** do wyszukiwania informacji w bazie danych. W najprostszej formie, umożliwiającej **projekcję danych** pojedynczej relacji, polecenie to musi zawierać:
- klauzulę ***select*** wskazującą atrybuty projekcji,
- klauzulę ***from***, wskazującą relację, której dotyczy polecenie.

Przykład

select *id_zesp, nazwa*
from *zespol;*

są wyświetlane zawartości atrybutów *id_zesp* i *nazwa* wszystkich krotek relacji *zespol*

Przykład

select * from *pracownik;*

są wyświetlane wartości wszystkich atrybutów dla wszystkich krotek relacji *pracownik*, a więc pełna zawartość informacyjna tej relacji

Proste użycie polecenia SELECT (2)

W rozbudowanej formie klauzula użyta do projekcji może zawierać:

- literały (łańcuchy znaków lub daty umieszczone w apostrofach lub liczby),
- wyrażenia arytmetyczne (nazwy atrybutów i literały numeryczne połączone znakami operacji +, -, *, /),
- funkcje (przekształcają wartości atrybutów i literałów),
- aliasy nazw atrybutów (alternatywne nazwy atrybutów występujące w zapytaniu bezpośrednio po ich właściwych nazwach),
- operator konkatenacji || (umożliwia łączenie wyświetlanych wartości różnych atrybutów w pojedyncze łańcuchy znaków;

Proste użycie polecenia SELECT (3)

Obliczenia w języku SQL realizuje się przez umieszczanie wyrażeń arytmetycznych w poleceniach tego języka. Wyrażenie składa się z nazw kolumn o wartościach liczbowych i liczb połączonych znakami: +, -, *, /.

Przykład

Przypuśćmy, że adiunktom zabiera się 20% pensji na podatek dochodowy. Obliczenie takiego podatku można przeprowadzić w oparciu o tabelę *pracownik* następująco:

```
select placa_pod, placa_pod*0.20
from pracownik
where etat = 'adiunkt';
```

Wynik:

PLACA_POD	PLACA_POD*0.20
1750	350
1600	320
1750	350
1780	356

W instrukcji *select* z przykładu użyto dodatkowo klauzuli *where*, która realizuje operację **selekcji**, tj. wybiera tylko krotki spełniające określone warunki.

Definiowanie schematu bazy danych (1)

Polecenia do definiowania schematu bazy są podzbiorem SQL nazywanym językiem definiowania danych DDL.

Tworzenie relacji

Relacje są tworzone za pomocą polecenia ***create table***:

create table *relacja*
(nazwa atrybutu typ(rozmiar) [***default*** *wartość_domyślna*]
[[***constraint*** *nazwa_ogr*] *ograniczenie_atr*],
(nazwa atrybutu typ(rozmiar) [***default*** *wartość_domyślna*]
[[***constraint*** *nazwa_ogr*] *ograniczenie_atr*],
. . .
[[***constraint*** *nazwa_ogr*] *ograniczenie_rel, ...*]);

Definiowanie schematu bazy danych (2)

Uwaga: W polecaniach zostaną zastosowane typy danych charakterystyczne dla MS SQL (por. slajdy 79-80)

Przykład

Zdefiniować tabelę etat

```
create table
etat (
 nazwa [nvarchar](20) constraint pk_nazwa primary key,
 placa_min [smallmoney] not null
        constraint pl_min check(placa_min>0),
 placa_max [smallmoney] not null
        constraint pl_max check(placa_max<=5000),
);
```

Definiowanie schematu bazy danych (3)

Rozszerzanie i modyfikowanie schematu relacji:

- Dodanie atrybutu do relacji

alter table *relacja*

add *(nazwa atrybutu typ(rozmiar)* [***default*** *wartość_domyślna*]
[[***constraint*** *nazwa_ogr*] *ograniczenie_atr*]);

Przykład

- Do relacji pracownik dodać nowy atrybut o nazwie *tytul_nauk*.

alter table *pracownik*

add *tytul_nauk* [nvarchar](20);

Definiowanie schematu bazy danych (4)

Rozszerzanie i modyfikowanie schematu relacji (cd):

- Dodanie ograniczenia integralnościowego relacji:

alter table *relacja*
add [***constraint*** *nazwa_ogr*] *ograniczenie_rel*;

Przykład

- Atrybut *id_zesp* relacji *pracownik* zdefiniować jako klucz obcy tej relacji, wskazujący na atrybut kluczowy relacji *zespol*.

alter table *pracownik*
add constraint *prac_fk* ***foreign key***
(id_zesp) ***references*** *zespol(id_zesp)****;***

Definiowanie schematu bazy danych (5)

Rozszerzanie i modyfikowanie schematu relacji (cd):

- Modyfikowanie atrybutów:

ALTER TABLE *table_name*

ALTER COLUMN *column_name nowa_definicja_atrybutu*

- Przykład
  Należy zmodyfikować rozmiar atrybutu tytul_nauk z 20 do 50 znaków, oraz zmienić jego właściwość na NOT NULL

alter table pracownik

alter column tytul_nauk nvarchar(50) NOT NULL;

Definiowanie schematu bazy danych (6)

Zmiana nazwy i usuwanie relacji:

- Zmiana nazwy relacji

EXEC sp_rename , ’pracownik', ’pracownik1';

- Usunięcie relacji

***drop table** relacja [**cascade constraints**];*

- **Uwaga:** Jeśli występuje opcjonalna klauzula ***cascade constraints***, wówczas są usuwane ograniczenia integralnościowe w innych relacjach, które w swojej definicji wykorzystują atrybuty kluczowe i unikalne usuwanej relacji.

Klauzule polecenia SELECT (1)

Klauzula *where*

Klauzula ***where*** jest klauzulą opcjonalną polecenia ***select***, realizującą operację selekcji algebry relacji.

select *atrybuty_projekcji*

from *relacja*

where *warunki_do_spełnienia*

Klauzule polecenia SELECT (2)

- Operatory w specyfikacji *warunki_do_spełnienia*

Operator	Opis
=	Równość.
!=	Różność.
>	Większość.
>=	Większość lub równość.
<	Mniejszość.
<=	Mniejszość lub równość.
is null	Sprawdzenie, czy wartość atrybutu jest wartością pustą.
between... and...	Sprawdzenie, czy lewy operand jest zawarty w przedziale określonym po słowach between i and.
in(zbiór)	Sprawdzenie, czy lewy operand należy do zbioru literałów podanego jako prawy operand.
like	Porównanie wartości atrybutu podanego jako lewy operand z tzw. wzorcem dopasowania. Wzorzec jest konstruowany za pomocą dwóch znaków specjalnych: „%” i „_”. Pierwszy z nich reprezentuje dowolny łańcuch znaków (także pusty), a drugi reprezentuje pojedynczy znak.

Cztery ostatnie operatory mogą być dodatkowo poprzedzone słowem ***not***.

Klauzule polecenia SELECT (3)

Przykład

Wybrać wszystkich pracowników, których pensja podstawowa jest mniejsza od dwukrotnej pensji dodatkowej.

select *nazwisko, placa_pod, placa_dod*

from *pracownik*

where *placa_pod<2*placa_dod;*

W wyniku otrzymujemy informację postaci:

no rows selected

co oznacza, że w relacji *pracownik* nie znaleziono krotek spełniających podany warunek.

Klauzule polecenia SELECT (4)

Przykład

Wyszukać wszystkich pracowników, których nazwiska zaczynają się na literę „*L*”.

select *nazwisko*

from *pracownik*

where *nazwisko like ‘L%’*;

NAZWISKO

Lech

Lubicz

Klauzule polecenia SELECT (5)

Można stosować także operatory logiczne ***and***, ***or*** i ***not***.

Priorytety operatorów :

=, !=, <, >, <=, >=, ***between and***, ***in***, ***like***, ***is null***.

not,

and,

or.

Przykład

Wybrać wszystkich adiunktów i profesorów, których płaca podstawowa jest wyższa od 1750.

select *nazwisko, etat, placa_pod* ***from*** *pracownik*

where *placa_pod>1750*

and *(etat='adiunkt'* ***or*** *etat='profesor');*

NAZWISKO ETAT PLACA_POD

---------- ---------- ----------

Podgajny profesor 2180

Delcki profesor 2050

Lubicz adiunkt 1780

Klauzule polecenia SELECT (6)

Klauzula *order by*

Klauzula ***order by***, jeśli występuje, powinna być zawsze umieszczana jako ostatnia w poleceniu ***select***. Wskazuje ona atrybut, według którego zostaną posortowane wiersze otrzymane w wyniku zapytania.

Przykład

Sporządzić spis wszystkich asystentów w kolejności ustalonej datą ich zatrudnienia. Otrzymujemy następującą listę:

select *nazwisko, pracuje_od* ***from*** *pracownik*
where *etat='asystent'*
order by *pracuje_od;*

NAZWISKO	*PRACUJE_OD*
Misiecki	*01-MAR-85*
Warski	*15-JUL-87*
Orka	*01-APR-88*
Palusz	*15-SEP-89*

Klauzule polecenia SELECT (7)

Klauzula *group by*

Klauzula ***group by*** umożliwia podział krotek relacji na grupy, a następnie ewentualne zastosowanie dla tych grup tzw. *funkcji grupowych*. Grupowanie może następować rekurencyjnie, według kolejnych atrybutów wskazanych w klauzuli.

Przykład

Wyświetlić informację o liczbie poszczególnych etatów w firmie.

select etat, **count(*) as** 'liczba etatów' **from** pracownik

where etat!='dyrektor'

group by etat;

Wynik zapytania z przykładu:

etat	*liczba etatów*
adiunkt	*4*
asystent	*4*
profesor	*2*
sekretarka	*1*
stażysta	*2*

Zastosowana w tym przykładzie funkcja *count* obliczająca liczbę etatów w każdej podgrupie jest przykładem funkcji grupowej.

Klauzule polecenia SELECT (8)

Klauzula *having*

W odróżnieniu od klauzuli ***where***, która operuje na pojedynczych krotkach, za pomocą klauzuli ***having*** można dokonywać selekcji na wcześniej wydzielonych grupach. Należy przy tym zaznaczyć, że tworzenie grup i obliczanie funkcji grupowych jest realizowane przed selekcją grup.

Przykład

Podać informację o średniej płacy w zespołach liczących powyżej trzech pracowników

```
select id_zesp,avg(placa_pod)
from pracownik
group by id_zesp,
having count(*) > 3;
ID_ZESP          AVG(PLACA_POD)
-----------      ------------------------
      20          1626
      30          1512.5
```

Łączenie relacji (1)

W przypadkach, gdy wymagana informacja musi być pozyskana z więcej niż jednej relacji można połączyć te relacje i ewentualnie wykonać na nich wymagane operacje algebry relacji.

Poziome łączenie relacji

Polega na utworzeniu relacji wynikowej, której krotki są wynikiem połączenia (konkatenacji) krotek relacji źródłowych.

Zwykle krotki jednej relacji są łączone z krotkami innej relacji tylko wtedy, gdy wartości korespondujących atrybutów tych krotek spełniają warunek określony w klauzuli ***where***.

Format ogólny:

select *atrybut(y)*

from *łączone_relacje*

where *warunek_połączenia;*

Łączenie relacji (2)

W przypadku, gdy łączone relacje mają atrybuty o tych samych nazwach, nazwy te są poprzedzone nazwami relacji. Takie prefiksowanie atrybutów jest stosowane także w klauzulach ***select***, ***group by*** i ***order by***. Jeśli prefiksowanie ma być stosowane wielokrotnie jest możliwe użycie aliasów zdefiniowanych w klauzuli ***from***, jak uczyniono to w przykł.

Przykład

Wyświetlić informację o tym, w jakich zespołach pracują poszczególni pracownicy.

select *nazwisko,nazwa*
from *pracownik p, zespol z*
where *p.id_zesp=z.id_zesp;*

NAZWISKO	*NAZWA*
Lech	*administracja*
Koliberek	*administracja*
Podgajny	*bazy danych*
Rus	*bazy danych*
Muszyński	*bazy danych*
. . .	. . .

Łączenie relacji (3)

Pionowe łączenie relacji

- Pionowe łączenie polega na utworzeniu relacji wynikowej, której krotki są sumą, częścią wspólną lub różnicą krotek relacji źródłowych.
- Stosujemy jeden z operatorów mnogościowych algebry relacji, tj. operator unii, przekroju lub różnicy. Zapytanie zawiera wtedy dwie lub więcej klauzule ***select***, posiadające tę samą liczbę atrybutów zgodnych typów.
- W wyniku pojawiają się nazwy atrybutów wyłącznie z pierwszej klauzuli ***select***.
- Jeśli zostaje użyta klauzula ***order by***, to musi ona wystąpić jako ostatnia i zamiast nazw atrybutów zawierać ich numery porządkowe.

Łączenie relacji (4)

Pionowe łączenie relacji (cd)

Format ogólny:

select *atrybut1,...,atrybutn*
from *relacja1*
where *warunki*
operator
select *atrybut1,...,atrybutn*
from *relacja1*
where *warunki*
order by *1,...,n;*

gdzie *operator* przyjmuje jedną z wartości: ***union***, ***union all***, ***intersect*** *lub* ***minus*** (SQL*PLUS) lub ***except*** (T-SQL).

Łączenie relacji (5)

Przykład

Określić te etaty w zespołach 30 i 40, dla których pewnym pracownikom należącym do różnych zespołów przyznano jednakowe płace podstawowe.

select *etat,placa_pod* ***from*** *pracownik* ***where*** *id_zesp=30*

intersect

select *etat,placa_pod* ***from*** *pracownik* ***where*** *id_zesp=40;*

ETAT PLACA_POD

--------- ----------

asystent 1350

stażysta 900

Sytuacja o której mowa w zapytaniu ma miejsce w przypadku etatu *asystent* (pracownicy Warski i Orka mają jednakowe płace podstawowe wynoszące po 1350) oraz *stażysta* (pracownicy Rajski i Kolski mają płace podstawowe wynoszące po 900). Należy zwrócić uwagę, że wyniki dwóch zapytań skierowanych do relacji *pracownik* połączono operatorem przekroju ***intersect***. Wyświetloną informację wynikową można uporządkować, np. według wartości atrybutu *etat*.

Zagnieżdżanie zapytań (1)

Polecenia ***select*** mogą być w sobie zagnieżdżane, co pozwala realizować bardziej złożone operacje na relacjach. Zapytanie zagnieżdżone jest także nazywane *podzapytaniem* (*subquery*) lub zapytaniem *wewnętrznym*.

Zagnieżdżanie zapytań (2)

Tryb nieskorelowany

Podzapytanie jest wykonywane jako pierwsze, jednokrotnie, a jego wyniki są przekazywane do zapytania zewnętrznego.

Format ogólny:

select *atrybutA1,...,atrybutAn*
from *relacjaA*
where *atrybut*
operator
*(**select** atrybuB1,...,atrybutBn*
from *relacjaB*
where *warunek);*

W przypadku, gdy podzapytanie wyznacza dokładnie jedną krotkę w warunku selekcji zapytania zewnętrznego stosujemy najczęściej jeden z operatorów porównania, np. = lub >=.

Jeśli natomiast podzapytanie wyznacza więcej niż jedną krotkę stosujemy operatory: ***in***, ***any*** *lub* ***all***.

Zagnieżdżanie zapytań (3)

Przykład

Wyświetlić wszystkich pracowników zatrudnionych na tym samym etacie co pracownik Orka (asystenci), wraz z ich płacą podstawową:

```
select nazwisko,placa_pod
from pracownik
where etat =
(select etat
 from pracownik
where nazwisko='Orka');
```

NAZWISKO	PLACA_POD
Misiecki	1400
Palusz	1200
Warski	1350
Orka	1350

Zagnieżdżanie zapytań (4)

Przykład

Wyznaczyć tych pracowników, którzy zarabiają mniej niż każdy pracownik z zespołu 20.

```
select nazwisko,placa_pod
from pracownik
where placa_pod < all
(select placa_pod
 from pracownik
where id_zesp=20);
```

NAZWISKO	PLACA_POD
Koliberek	1150
Rajski	900
Kolski	900

W odróżnieniu od poprzedniego przykł. , gdzie podzapytanie wyznacza pojedynczą wartość, tutaj zapytanie wewnętrzne wyznacza grupę wartości.

Zagnieżdżanie zapytań (5)

Tryb skorelowany

- W trybie skorelowanym najpierw jest wykonywane zapytanie zewnętrzne, a dopiero potem skorelowane z nim podzapytanie.
- Podzapytanie skorelowane jest wykonywane tyle razy, ile razy jest wykonywane zapytanie zewnętrzne.
- Składniowo zapytania skorelowane od nieskorelowanych różni konieczność zastosowania aliasów relacji, na których operuje zapytanie zewnętrzne i odwoływania się do nich w podzapytaniu.

Zagnieżdżanie zapytań (6)

Przykład 6.19

Podać informacje o tych pracownikach, których płaca podstawowa jest wyższa niż przeciętna dla etatu, na którym są zatrudnieni.

```
select nazwisko,placa_pod,etat
from pracownik p
where placa_pod >
(select avg(placa_pod)
 from pracownik
where etat=p.etat)
order by p.etat;
```

Zapytanie zewnętrzne przegląda kolejne krotki pracowników przekazując je do podzapytania skorelowanego.

W podzapytaniu jest wyznaczana przeciętna płaca pracowników zatrudnionych na tym samym etacie co pracownik analizowany przez zapytanie zewnętrzne.

Krotki otrzymane w wyniku zostają dodatkowo uporządkowane według wartości atrybutu *etat*.

Wynik:

NAZWISKO	PLACA_POD	ETAT
Rus	1750	adiunkt
Maleja	1750	adiunkt
Lubicz	1780	adiunkt
Misiecki	1400	asystent
Orka	1350	asystent
Warski	1350	asystent
Podgajny	2180	profesor

Zagnieżdżanie zapytań (7)

Przeciętne płace można wyznaczyć za pomocą polecenia:

select *etat, avg(placa_pod)*
from *pracownik*
group by *etat;*

ETAT	*AVG(PLACA_POD)*
adiunkt	*1720*
asystent	*1325*
dyrektor	*3160*
profesor	*2115*
sekretarka	*1150*
stażysta	*900*

Zagnieżdżanie zapytań (8)

- W przypadku zapytań skorelowanych możemy wykorzystać nowy operator ***exists*** lub jego negację ***not exists*** celem sprawdzenia, czy podzapytanie wyznacza jakąkolwiek wartość (bądź nie wyznacza żadnej).

Przykład

Wyznaczyć pracowników, którzy są zatrudnieni na etatach, na których nie jest zatrudniony nikt inny.

```
select numer,nazwisko,etat
from pracownik p
where not exists
(select numer
 from pracownik
where etat=p.etat and numer!=p.numer)
order by numer;
```

NUMER	NAZWISKO	ETAT
1000	Lech	dyrektor
1080	Koliberek	sekretarka

Modyfikowanie zawartości relacji (1)

Polecenia służące do wypełniania relacji krotkami, modyfikowania zawartości relacji oraz usuwania krotek z relacji wchodzą w skład języka manipulowania danymi - DML.

Wstawianie krotek

W celu wstawienia krotki do relacji stosowane jest polecenie ***insert*** postaci:

insert into *nazwa_relacji* [(*atrybut*, *atrybut*, ...)]

values (*wartość*, *wartość*, ...);

Opcjonalna lista atrybutów jest zbędna w przypadku określenia wszystkich wartości atrybutów relacji *nazwa_relacji*.

Modyfikowanie krotek

Do modyfikowania krotek służy polecenie ***update*** następującej postaci:

update *relacja* [*alias*]

set *atrybut* [, *atrybut*] = {*wyrażenie* | *podzapytanie*}

[***where*** *warunki*];

Modyfikowanie zawartości relacji (2)

Usuwanie krotek

W celu usunięcia krotki (krotek) z relacji jest stosowane polecenie ***delete*** następującej postaci:

delete from *relacja*

[***where*** *warunki*];

Przykład

delete from *pracownik*

where *etat = 'ASYSTENT';*

Polecenie usuwa z relacji *pracownik* wszystkie krotki opisujące asystentów.

Dodatkowe struktury danych (1)

Perspektywy

Perspektywa jest specjalną strukturą w bazie danych, która pozwala ograniczyć zakres dostępnych danych do wybranych atrybutów i krotek.

Dane udostępniane przez perspektywy pochodzą z relacji, tzn. perspektywy nie posiadają „własnych” danych i są pamiętane w postaci definicji.

Dodatkowe struktury danych (2)

Perspektywy(cd.)

Utworzenie perspektywy jest możliwe za pomocą polecenia ***create view*** postaci:

create [***or replace***] ***view*** *nazwa_perspektywy*
[(*atrybut1*, *atrybut2*, ...)]
as select *ciało_polecenia_SELECT*
[***with check option***];

Opcjonalna klauzula ***or replace*** zastępuje istniejącą perspektywę nową definicją. Na tomiast klauzula ***with check option*** uniemożliwia wstawianie i modyfikowanie krotek w sposób niezgodny z warunkami selekcji perspektywy, tzn. stanowi mechanizm zachowania spójności danych .

Dodatkowe struktury danych (3)

Wyróżniamy perspektywy *proste* i *złożone*.

Cecha	**Perspektywa**	
	prosta	**Złożona**
Pochodzenie danych	Udostępnia dane z pojedynczej relacji	Udostępnia dane wielu relacji
Ograniczenia dla definicji	W definicji nie stosuje się operacji na zbiorach, funkcji ani grupowania krotek	W definicji można stosować operacje połączenia relacji, operacje na zbiorach, funkcje i grupowanie krotek.
Możliwość pobierania danych	Tak	Tak
Możliwość modyfikowania danych	Tak	W ograniczonym zakresie

Dodatkowe struktury danych (4)

Przykład

Utworzyć perspektywę prostą zawierającą wybrane informacje (numer i nazwisko) dla wszystkich asystentów.

```
create view asystenci
as     select numer, nazwisko
       from pracownik
       where etat='asystent';
```

Dodatkowe struktury danych (5)

Przykład

Na podstawie relacji *pracownik* i *zespol* utworzyć perspektywę złożoną podającą informacje o płacach w poszczególnych zespołach.

create view *pods_zesp*

(nazwa, placa_min, placa_max, placa_przec)

as ***select*** *nazwa, min(placa_pod), max(placa_pod), avg(placa_pod)*

from *pracownik, zespol*

where *pracownik.id_zesp=zespol.id_zesp*

group by *nazwa*;

Dodatkowe struktury danych (6)

Dostęp do danych perspektywy odbywa się podobnie jak w przypadku relacji, np. za pomocą polecenia ***select***, podając nazwę perspektywy w klauzuli ***from***. Można także użyć poleceń ***insert***, ***update*** i ***delete***, jeśli tylko perspektywa umożliwia modyfikowanie relacji, na której bazuje.

Przykład

select ***

from *pods_zesp*;

Wynik:

NAZWA	*PLACA_MIN*	*PLACA_MAX*	*PLACA_PRZEC*
administracja	*1150*	*3160*	*2155*
bazy danych	*1200*	*2180*	*1626*
sieci komputerowe	*900*	*2050*	*1512.5*
systemy operacyjne	*900*	*1780*	*1343.3333*

Dodatkowe struktury danych (7)

Przykład

create or replace view *adunkci*

as ***select*** *numer, nazwisko, etat*

from *pracownik*

where *etat='adiunkt'*

with check option;

W tym przykładzie utworzono perspektywę udostępniającą wybrane informacje o adiunktach. Klauzula ***with check option*** zapewnia weryfikację spójności (integralności) danych, co powoduje, że modyfikacja zawartości relacji bazowej tej perspektywy przez samą perspektywę jest ograniczona.

Dodatkowe struktury danych (8)

Oznacza to w szczególności, że nie jest możliwe wykonanie następującego polecenia:

update *adiunkci*

set *id_zesp=30*

where *nazwisko='Orka'*;

Powodem jest ograniczenie integralnościowe ustalające, że na perspektywie mogą być realizowane tylko takie polecenia ***insert*** i ***update***, w wyniku których otrzymujemy krotki „widziane" przez tę perspektywę.

Perspektywę można usunąć za pomocą polecenia ***drop view*** postaci:

drop view *nazwa_perspektywy*;

Dodatkowe struktury danych (9)

Liczniki

Liczniki stanowią osobny typ obiektów w bazie danych, które automatycznie zwiększają swoją wartość każdorazowo gdy są odczytywane.

Licznik definiuje się poleceniem ***create sequence*** postaci:

create sequence [*nazwa_uzytkownika*] *nazwa_licznika*
[***increment by*** *liczba*]
[***start with*** *wartość_początkowa*]
[***cycle***/***nocycle***];

Odczytu wartości licznika można dokonać za pomocą polecenia:

select *nazwa_licznika.****nextval from dual***;

Symbolem ***nextval*** oznaczono pseudoatrybut przechowujący kolejną wartość licznika. Jeżeli odczytana ma być wartość aktualna (bez jej zwiększania) używamy polecenia:

select *nazwa_licznika.****curval from dual***;

Dodatkowe struktury danych (10)

Przykład

create sequence *mój_licznik*;
Sequence created.
select *mój_licznik.****nextval from dual***;
NEXTVAL

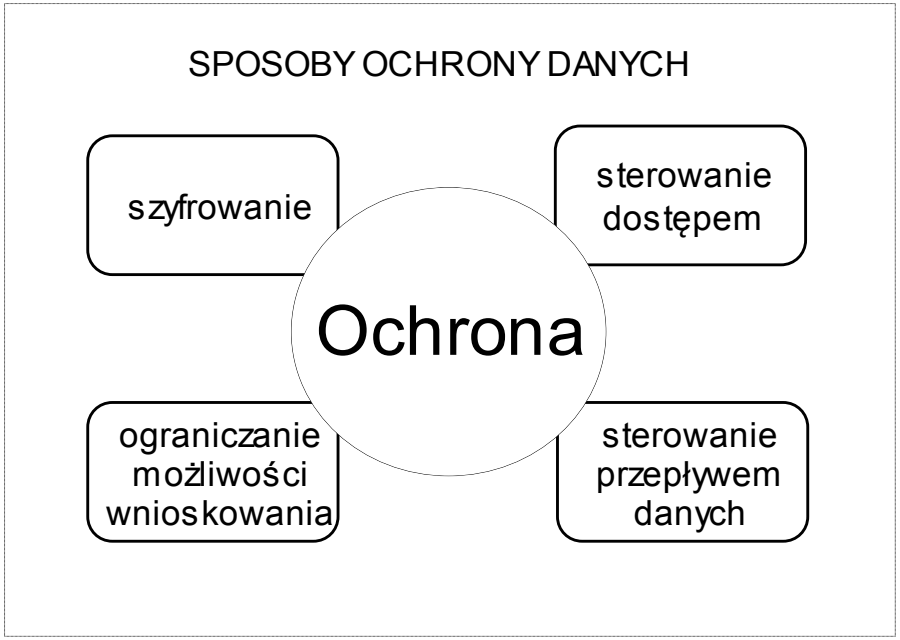
1
select *mój_licznik.****nextval from dual***;
NEXTVAL

2
select *mój_licznik.****currval from dual***;
CURRVAL

2

Po odczytaniu wartości licznika nie jest możliwy powrót do jego poprzedniej wartości.

Autoryzacja dostępu (1)



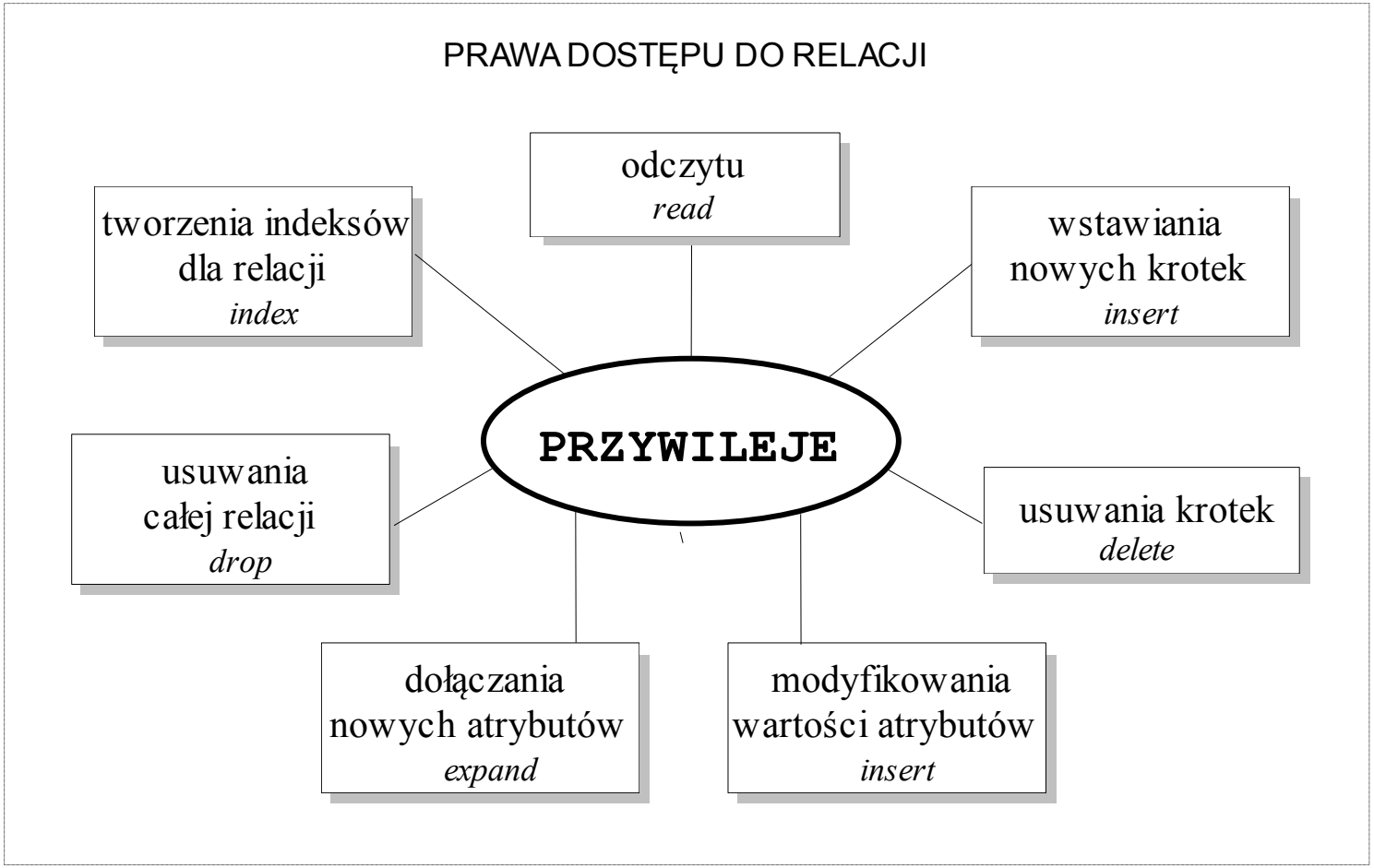
- *Sterowanie dostępem* polega na identyfikacji i ewidencji poszczególnych użytkowników oraz przyznawaniu im praw dostępu do określonych danych
- *Sterowanie przepływem danych* to kontrola przesłań danych w celu zabezpieczenia ich przed dostępem nieupoważnionych osób.
- *Ograniczanie możliwości wnioskowania* ma zapobiec sytuacji, w której użytkownik bazy na podstawie dostępnych mu danych jest w stanie wyciągać wnioski dotyczące informacji, których nie powinien poznać.
- *Szyfrowanie* umożliwia przechowywanie lub przesyłanie danych w postaci czytelnej jedynie dla użytkowników znających odpowiedni kod.

Autoryzacja dostępu (2)

- Najbardziej popularnym sposobem ochrony w systemach zarządzania bazą danych jest sterowanie dostępem.
- Prowadzi się ewidencję użytkowników oraz nadaje im identyfikatory i hasła zabezpieczające dostęp do systemu.
- Użytkowników często łączy się w grupy, których członkowie zwykle posiadają jednakowe prawa dostępu do pewnych obiektów.
- Nieograniczone prawa (przywileje) do wszystkich obiektów bazy danych posiada specjalny użytkownik określany jako administrator bazy danych.
- Jednym z ważniejszych uprawnień administratora jest nadawanie (odbieranie) praw innym użytkownikom.

Autoryzacja dostępu (3)

- Można ograniczyć prawa dostępu na poziomie danej relacji:

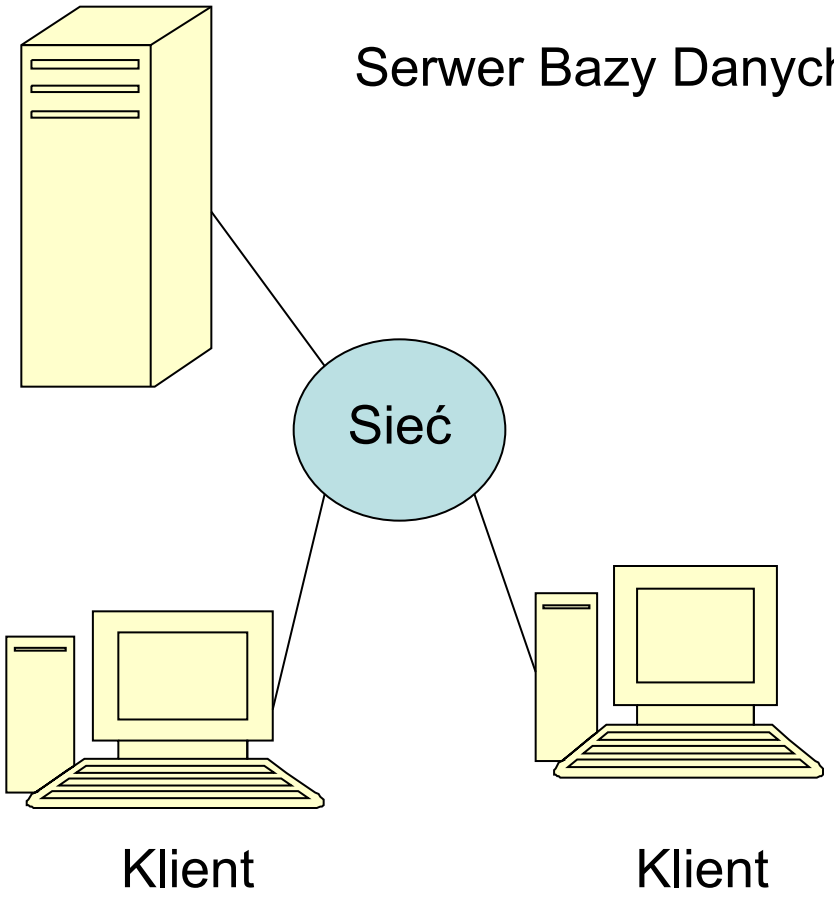


Autoryzacja dostępu (4)

- Do szczególnych praw w bazie danych należą:
  - prawo dołączania się do bazy danych, nadawane przez administratora systemu oraz
  - prawo przekazywania przyznanych przywilejów innym użytkownikom bazy danych.

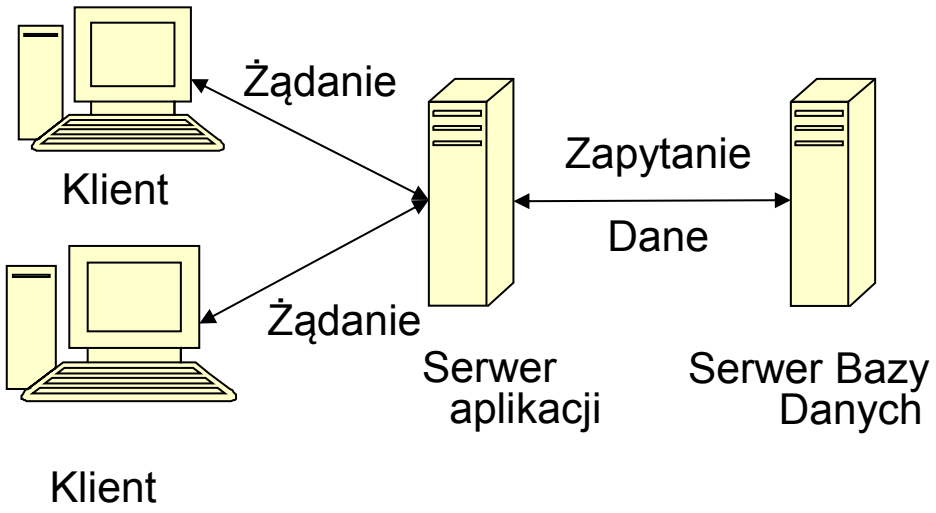
Aplikacje baz danych

Architektura dwuwarstwowa



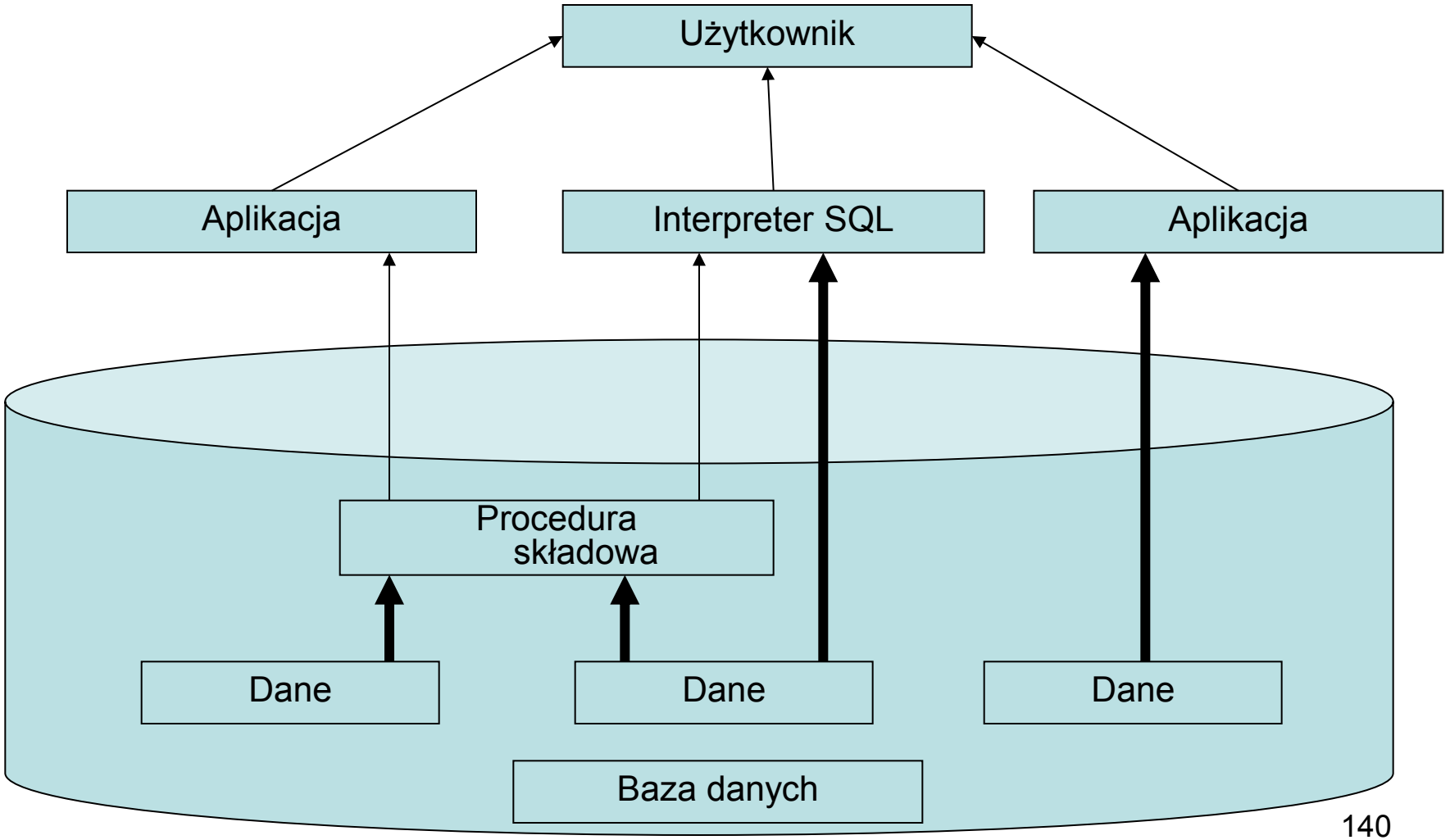
- Aplikacja jest wykonywana po stronie klienta, który realizuje komunikację z użytkownikiem oraz wykorzystuje serwer w celu uzyskania dostępu do danych w bazie
- Zasadniczą funkcję serwera wypełnia system zarządzania bazą danych zapewniający aplikacjom dostęp do danych
- Klient i serwer mogą być zainstalowani na tym samym komputerze, bardziej wydajnym rozwiązaniem jest zastosowanie sieci komputerowej z wydzielonym sieciowym serwerem bazy danych.

Architektura wielowarstwowa



- Wprowadzony jest serwer aplikacji, który udostępnia dane klientom pełniąc rolę interfejsu między klientami a serwerami bazy danych.
- Serwer aplikacji:
  - Sprawdza uprawnienia klienta
  - Łączy się z serwerem bazy danych
  - Wykonuje operacje zgłoszone w konkretnym żądaniu (przejmuje częściowo przetwarzanie zapytań, odciążając serwer aplikacji)

Dostęp aplikacji do danych



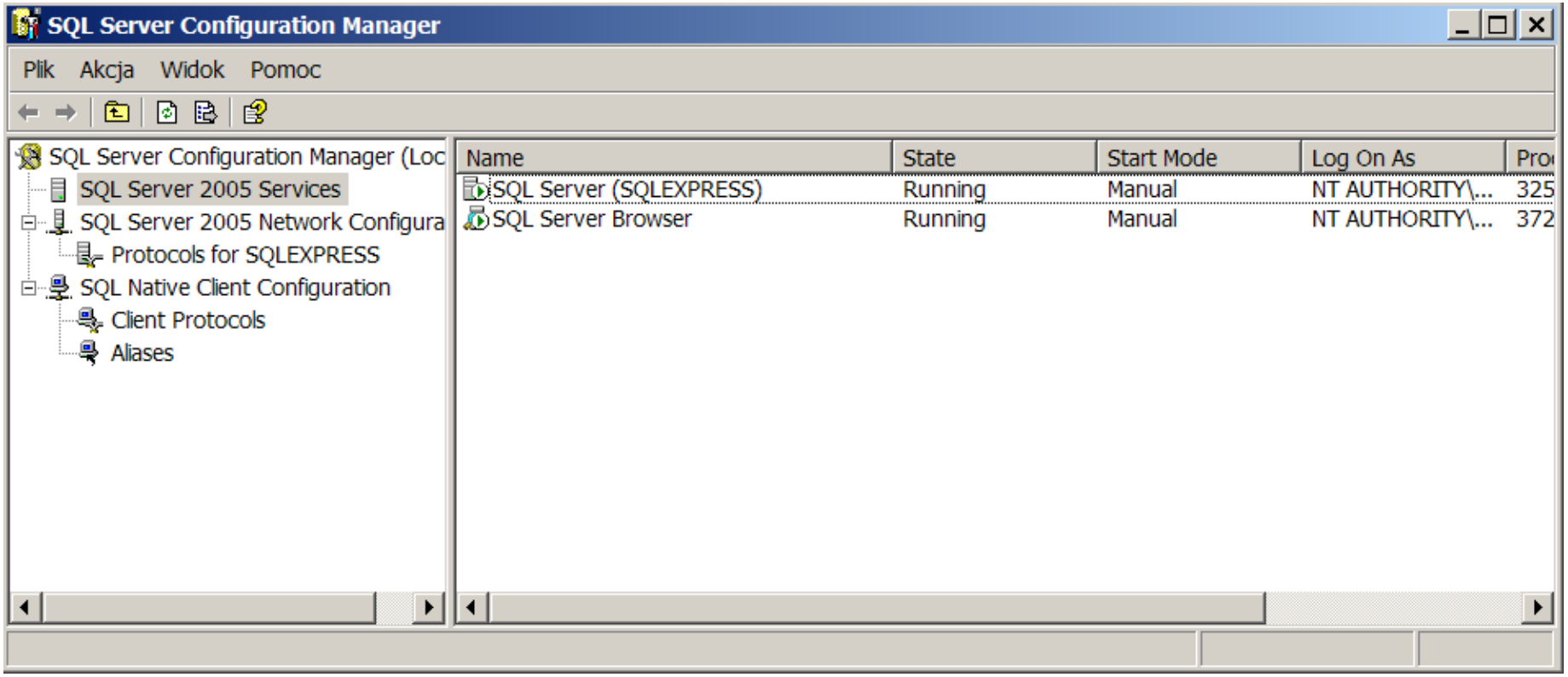
Prosta aplikacja bazodanowa

```
#using <mscorlib.dll>
using namespace System;
using namespace System::Collections;

#using "System.dll"
#using "System.Data.dll"
#using "System.Transactions.dll"
#using "System.EnterpriseServices.dll"
#using "System.Xml.dll"

using namespace System::ComponentModel;
using namespace System::Data;
using namespace System::Data::SqlClient;

void main()
{
    // Zapytanie SQL:
    String ^query = "SELECT * FROM
    Categories";
    // Łańsuch tekstowy definiujący połączenie
    // do bazy:
    String ^connectString = "Data
    Source=983E8A44B5024CD\\SQLEXPRES
    S;Initial Catalog=Northwind;Integrated
    Security=True";
    SqlConnection^ sqlconn =
    gcnew SqlConnection(connectString);
    sqlconn->Open();
    SqlCommand ^sqlCommand =
    gcnew SqlCommand(query, sqlconn);
    // Bufor na dane
    SqlDataReader ^dataReader =
    sqlCommand->ExecuteReader();
    // Odczytanie liczby kolumn:
    int numCols = dataReader->FieldCount;
    // Wypisz liczbą kolumn
    Console::Write("No. of columns:");
    Console::WriteLine(numCols);
    // Wypisz Dane:
    while(dataReader->Read())
    {       for (int c = 0; c < numCols-1; c++)
            {Console::Write(dataReader[c]);
             Console::Write("\t");
            }
            Console::WriteLine("");
    }
}
```

Najpopularniejsze serwery SQL

- Płatne
  - Microsoft SQL Serwer
  - Oracle
  - IBM DB2
- Darmowe
  - MySQL (Linux)
  - Microsoft SQL Serwer Express

Microsoft SQL Serwer Express



Microsoft SQL Server Management Studio Express