Bazy danych

Artur Gramacki

Uniwersytet Zielonogórski
Instytut Sterowania i Systemów Informatycznych
A.Gramacki@issi.uz.zgora.pl

Plan wykładów

- Cel kursu
- Czym są i do czego służą bazy danych
- System zarządzania bazą danych (SZBD)
- Projektowanie systemów informatycznych
- Modelowanie pojęciowe: model związków encji
- Pojęcie relacji oraz podstawowe operacje na relacjach
- Związki między relacjami (1:1, 1:N, N:M), klucze główne i klucze obce, inne tzw. ograniczenia (ang. *constraints*) bazodanowe
- Normalizacja relacji
- Transakcje bazodanowe
- Optymalizacja działania bazy danych

- Bazy klasy NoSQL

Materiały do wykładów

- Niektóre będą użyte bardzo pobieżnie
- http://www.uz.zgora.pl/~agramack/

Podstawy relacyjnych baz danych
Materiały pomocnicze na temat języka SQL - na przykładzie MySQL

LAMP (Linux + Apache + MySQL + PHP)
CodeIgniter + MySQL demo
AJAX
Framework Bootstrap

Bazy danych klasy NoSQL

Plan laboratoriów

- Plan może ulegać zmianom, różne elementy mogą być realizowane w różnym „nasileniu”
- http://www.uz.zgora.pl/~agramack/

Bazy danych (MySQL)

Laboratorium 1: Przygotowanie środowiska pracy dla bazy MySQL z wykorzystaniem XAMPP Portable Lite oraz MySQL-Front
Laboratorium 2: Podstawy pracy z bazą MySQL
Laboratorium 3: Podstawy języka SQL (polecenia CREATE, ALTER, DROP, INSERT, UPDATE, DELETE, SELECT)
Laboratorium 4: Zapoznanie się z wybranym programem wspomagającym projektowanie relacyjnych baz danych
Laboratorium 5: Implementacja przykładowej struktury bazy relacyjnej
Laboratorium 6: Podstawy języka SQL, część 1 (SELECT) Laboratorium 6, pliki
Laboratorium 7: Podstawy języka SQL, część 2 (GROUP BY)
Laboratorium 8: Podstawy języka SQL, część 3 (Złączenia)
Laboratorium 9: Podstawy języka SQL, część 4 (DML: INSERT, UPDATE, DELETE)
Laboratorium 10: Podstawy języka SQL, część 5 (DDL: CREATE, DROP, ALTER)
Laboratorium 11: Wybrane funkcje wbudowane
Laboratorium 12: Dostęp do danych zgromadzonych w bazie MySQL z poziomu przeglądarki internetowej z użyciem PHP Laboratorium 12, pliki
Laboratorium 13: Transakcje w bazach danych
Laboratorium 14: Tworzenie i wykorzystanie indeksów Laboratorium 14, pliki
Laboratorium 15: Import i eksport danych. Tworzenie kopii bezpieczeństwa oraz odzyskiwanie danych
Laboratorium 16: System przywilejów oraz zarządzanie użytkownikami Laboratorium 16, pliki

Bazy danych NoSQL

Laboratorium MongoDB
Laboratorium Neo4J

Cel kursu

Cel kursu

- Podać niezbędne wiadomości teoretyczne na temat relacyjnych baz danych
- Nauczyć projektować poprawne struktury relacyjne
- Nauczyć podstaw pracy oraz podstaw administrowania wybranym systemem bazodanowym (np. Oracle, MySQL)
- Nauczyć efektywnej pracy z językiem SQL
- Nauczyć tworzenia aplikacji bazodanowych, w szczególności interenetowych (języki PHP, JAVA, wzorce projektowe, MVC - Model-View-Controller, inne)

- Podać niezbędne wiadomości na temat baz klasy NoSQL
- Zaprezentować najpopularniejsze ich implementacje
- Porównać nazy NoSQL z klasycznymi bazami relacyjnymi

Czym są i do czego służą bazy danych

Czym są i do czego służą bazy danych

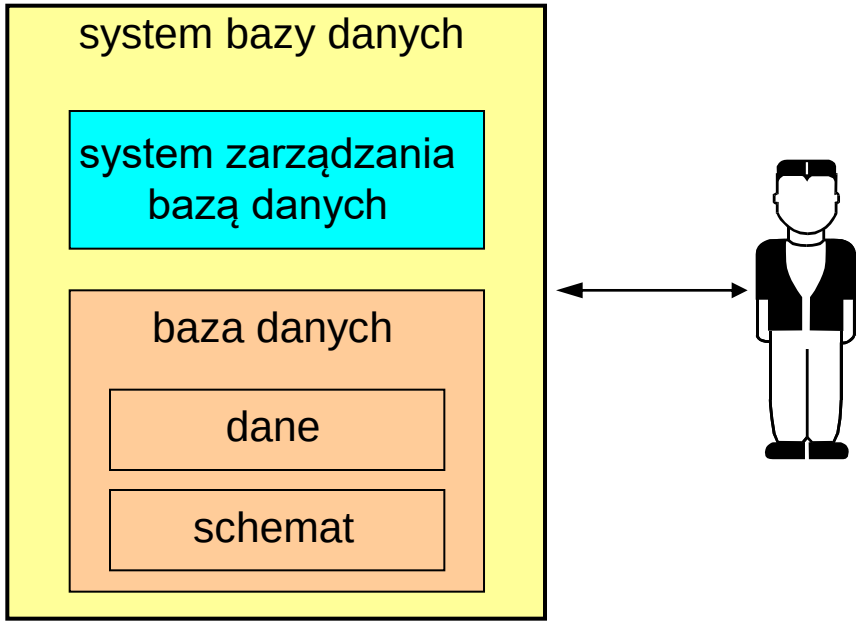
- Potrzeba gromadzenia i PRZETWARZANIA coraz większych ilości danych
- Informacje gromadzone w bazach danych są bardzo cenne
- Informacje gromadzone w bazach danych pomagają zarządzać przedsiębiorstwem (firmą, biznesem)
- Informacje gromadzone w bazach danych pomagają promować firmę
- Informacje zgromadzone i nie wykorzystywane są bezwartościowe
- Wolny i zawodny dostęp do danych jest często gorszy niż brak jakiejkolwiek bazy danych

- Oracle, DB2, SQL Server, Sybase, MySQL, PostgreSQL, Access, Delphi oraz sporo innych mniej popularnych, czy wręcz niszowych
- MongoDB, Redis, Cassandra, Neo4J … i wiele, wiele innych

System zarządzania bazami danych (SZBD)

Definicje, podstawowe informacje

- System zarządzania bazą danych SZBD (ang. *DBMS -Database Management System*) - oprogramowanie umożliwiające tworzenie oraz eksploatację bazy danych oraz jej użytkowników (np. ORACLE, MySQL)
- Baza danych - spójny zbiór danych posiadających określone znaczenie (inaczej jest to informatyczne odwzorowanie fragmentu świata rzeczywistego
- Baza danych = dane + schemat bazy danych (najczęściej relacyjny)
- System bazy danych = baza danych + system zarządzania bazą danych
- Podstawowe funkcje SZBD to:
  - łatwe odpytywanie bazy danych
  - optymalizacja zapytań
  - zapewnienie integralności danych
  - zapewnienie wielodostępu do danych
  - odporność na awarie
  - ochrona i poufność danych



Własności systemu bazy danych

- Niezależność aplikacji i danych

  Dane mogą być wprowadzane do bazy bez konieczności modyfikacji korzystających z nich programów czy systemów użytkowych, a z drugiej strony aplikacje mogą być modyfikowane niezależnie od stanu baz danych

- Abstrakcyjna reprezentacja danych

  Programy i systemy użytkowe (aplikacje) są tworzone przy użyciu tzw. deklaratywnych języków programowania (w odróżnieniu od języków imperatywnych). Twórca aplikacji nie musi np. interesować się kolejnością danych w bazie, ani sposobem ich reprezentacji i wyszukiwania. Precyzuje jedynie warunki selekcji informacji. Inaczej mówiąc: decyduje „co zrobić”, a nie „jak zrobić”

- Różnorodność sposobów widzenia danych

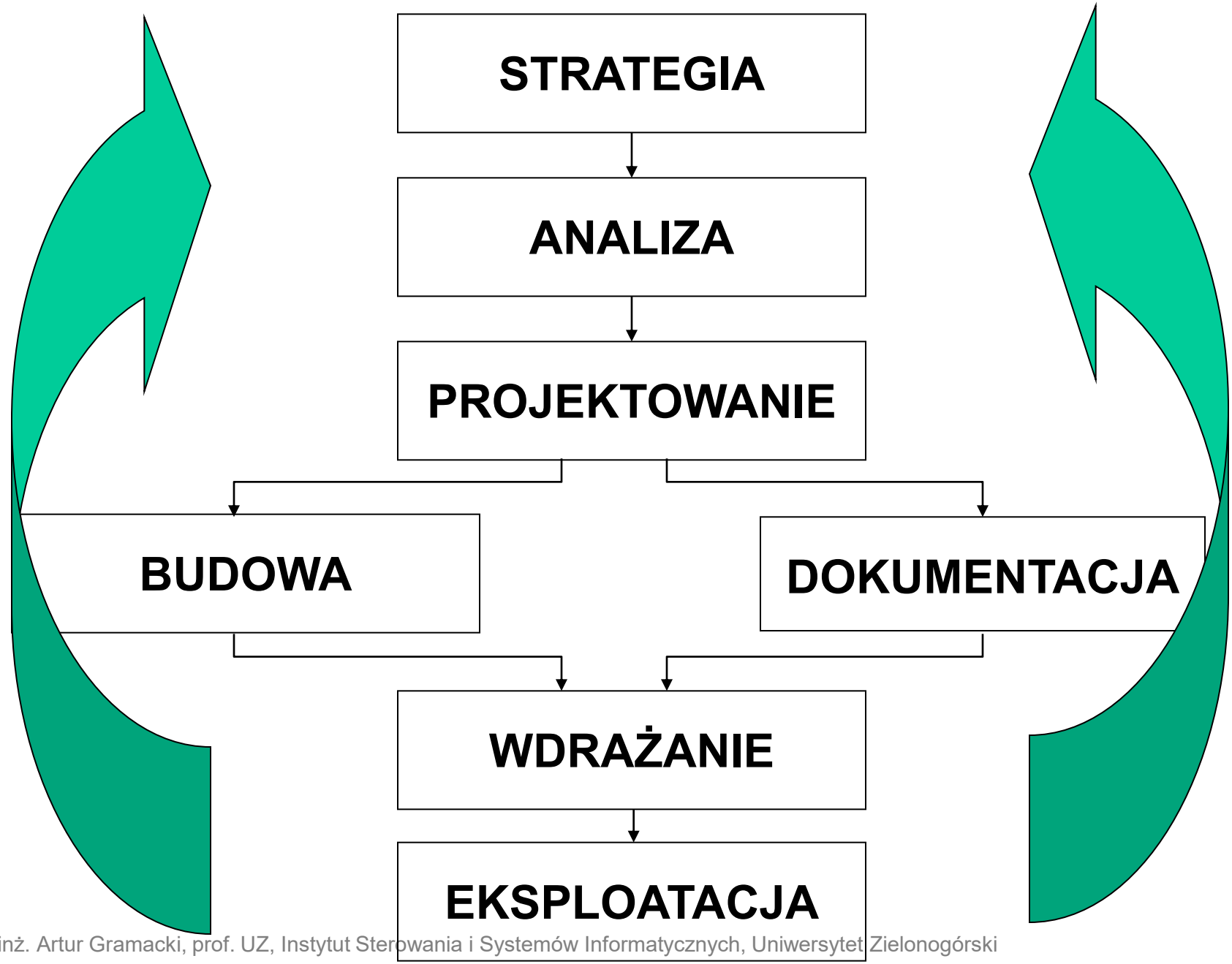
  Te same dane zawarte w bazie mogą być „widziane” w różny sposób przez różnych użytkowników. Efekt ten uzyskuje się przez stosowanie różnych „filtrów” (perspektyw) nakładanych na te same dane

- Fizyczna i logiczna niezależność danych

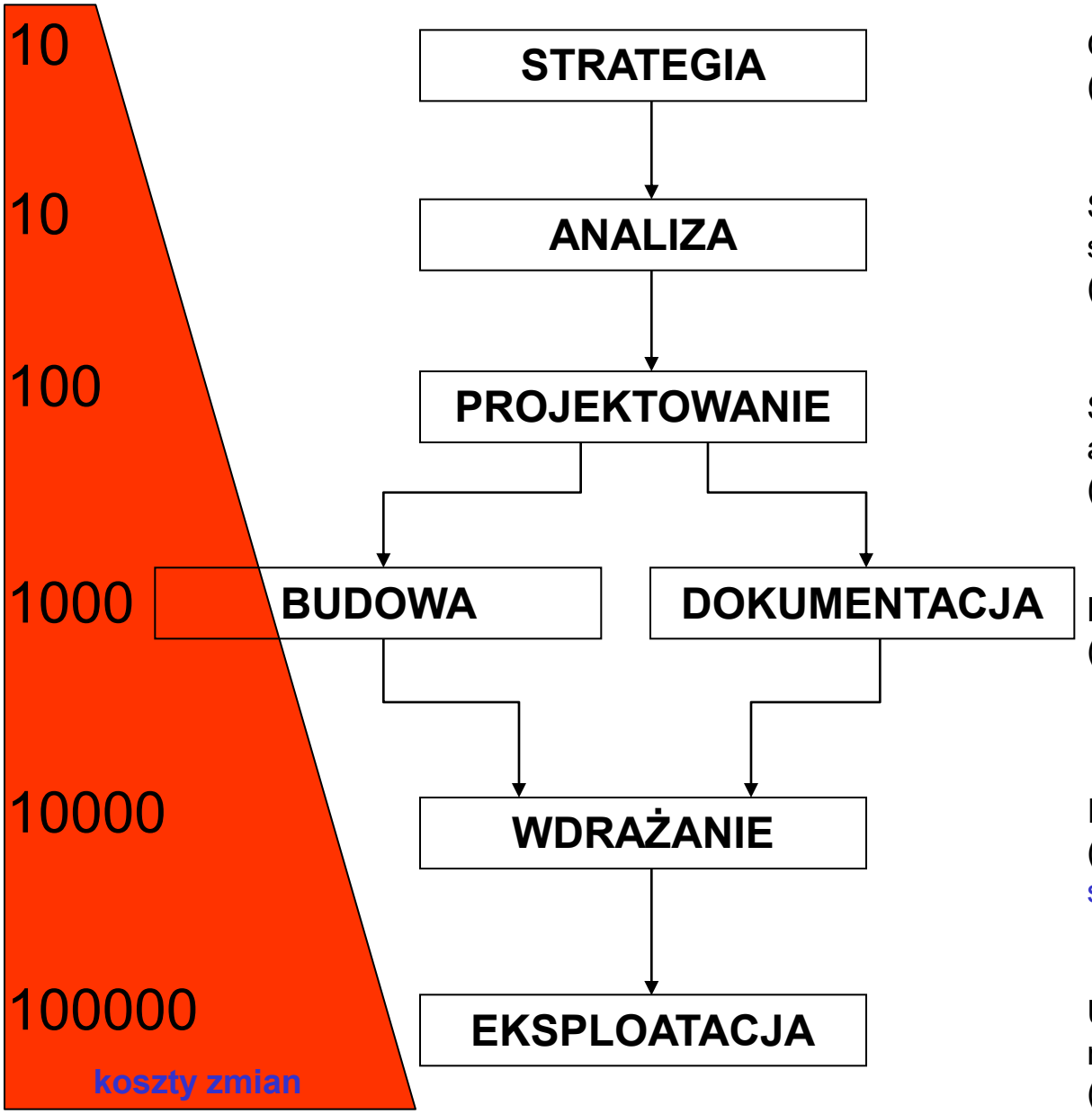
  Fizyczna niezależność danych polega na tym, że rozszerzenie systemu komputerowego, na którym pracuje SZBD o nowy sprzęt nie narusza danych w bazie. Logiczna niezależność danych polega na tym, że - po pierwsze wprowadzanie nowych danych do bazy nie deaktualizuje starych, po drugie - dane, które nie są wzajemnie powiązane tzw. więzami integralności mogą być usuwane z bazy niezależnie od siebie

Projektowanie systemów informatycznych

Projektowanie systemów informatycznych (1/2)



Projektowanie systemów informatycznych (2/2)



Ogólny model pojęciowy systemu
(Analityk systemowy)

Szczegółowy model pojęciowy systemu
(Analityk systemowy)

Struktury logiczne i fizyczne danych i aplikacji
(Projektant)

Baza danych i funkcjonalne aplikacje
(Programista)

Instalacja systemu u odbiorcy
(Wdrożeniowiec, programista, szkoleniowiec)

Użytkowanie, pielęgnacja, modyfikacje, usuwanie usterek
(Administrator, programista)

Modelowanie pojęciowe: model związków encji

Projektowanie systemów baz danych

Miniświat – wyróżniony fragment rzeczywistości, który zamierzamy zamodelować w postaci bazy danych

Analiza miniświata - konstrukcja modelu konceptualnego — modelowanie związków encji

Transformacja modelu konceptualnego do modelu relacyjnego — relacje

Proces normalizacji modelu relacyjnego — relacje znormalizowane

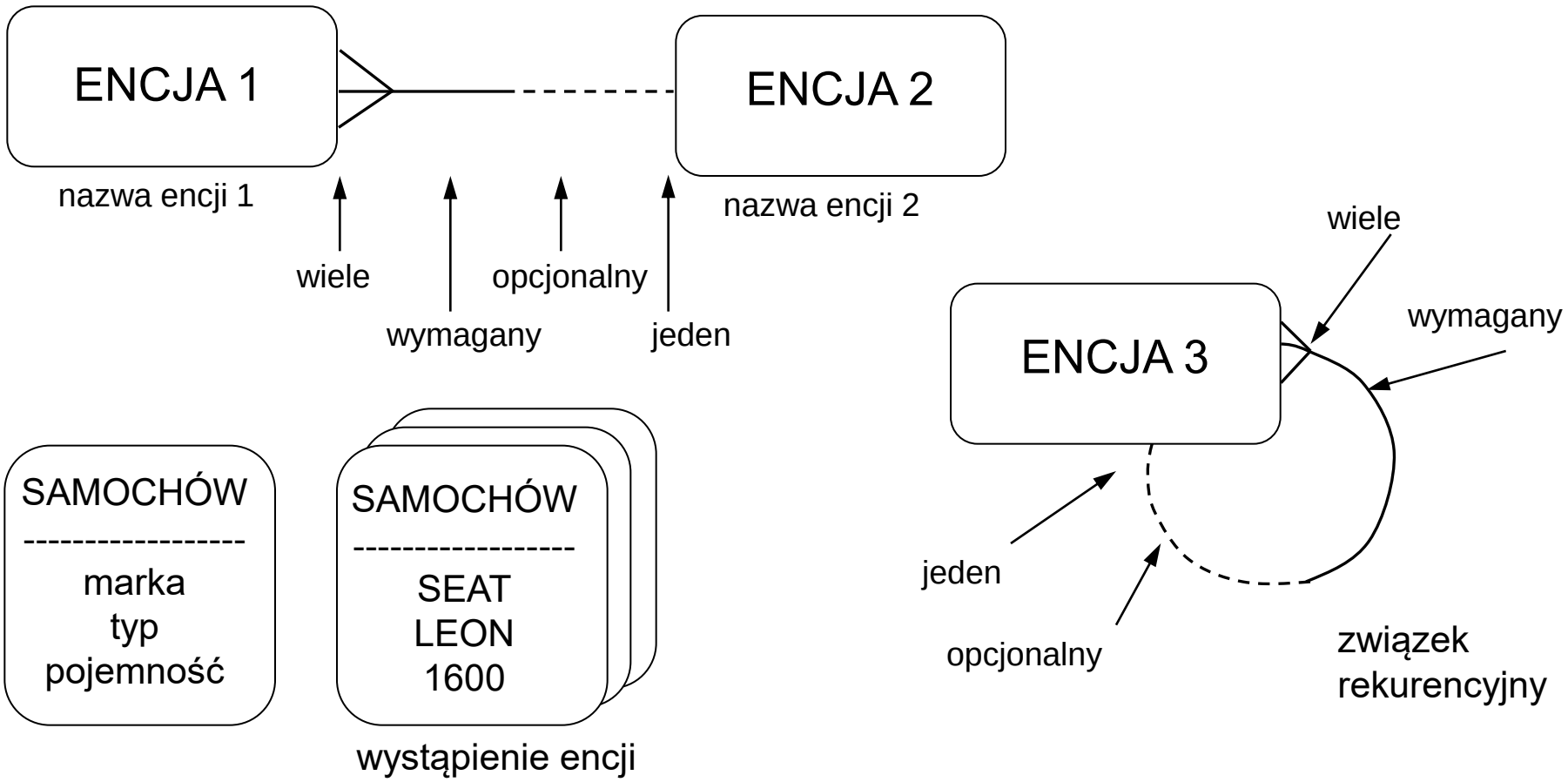
Implementacja modelu relacyjnego w wybranym Systemie Zarządzania Bazami Danych (SZBD) — np. ORACLE, MySQL, SQL Server

Tworzenie interfejsu użytkownika, strojenie itp. — np. C, C++, Java, PHP, PL/SQL

Modelowanie związków encji (1/6)

- **<u>Materiał w tym rozdziale podano w wielkim skrócie !</u>** Szczegóły patrz np. Richard Baker, *CASE Method SM, Modelowanie związków encji*, WNT, Warszawa, 1996

Encja (ang. **entity**) jest rzeczą lub obiektem mającym dla nas znaczenie, rzeczywistym bądź wyobrażonym, o którym informacje muszą być znane lub przechowywane.

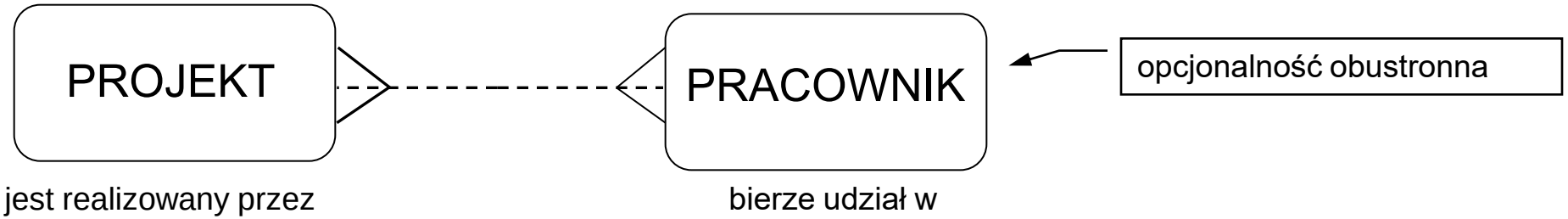


Modelowanie związków encji (2/6)

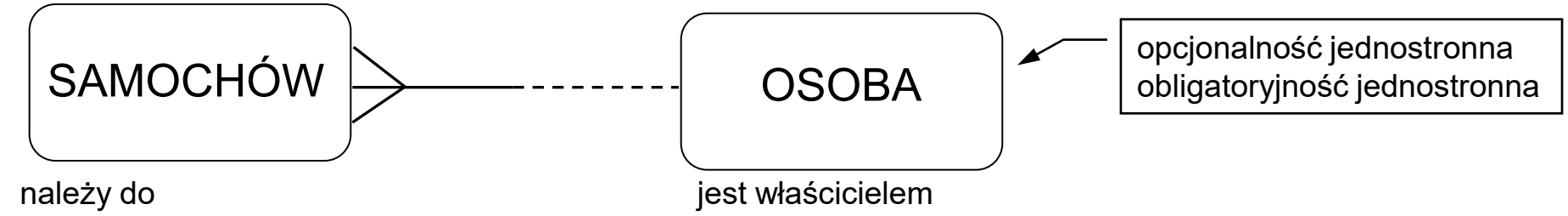
- Encją może być:
  - obiekt fizyczny (np. samochód, bilet lotniczy)
  - obiekt niematerialny (np. konto bankowe, zamówienie)
  - zdarzenie (np. urlop pracownika, sprzedaż samochodu)
  - istniejący fakt (np. znajomość języków obcych)
- Między encjami mogą zachodzić różne związki. Każdy związek ma dwa końce i ma przypisaną nazwę, stopień (liczebność) oraz opcjonalność (opcjonalny czy wymagany)
- Encje są charakteryzowane przez atrybuty
- Nazwa encji powinna być podana w liczbie pojedynczej i zapisana dużymi literami
- Nazwa encji musi dokładnie reprezentować typ lub klasę rzeczy, a nie żadne jej konkretne wystąpienie

Modelowanie związków encji (3/6)

Każdy BILET LOTNICZY **MUSI** być wystawiony dla jednego i tylko jednego PASAŻERA
Każdy PASAŻER **MOŻE** być wyszczególniony na jednym lub więcej BILETACH



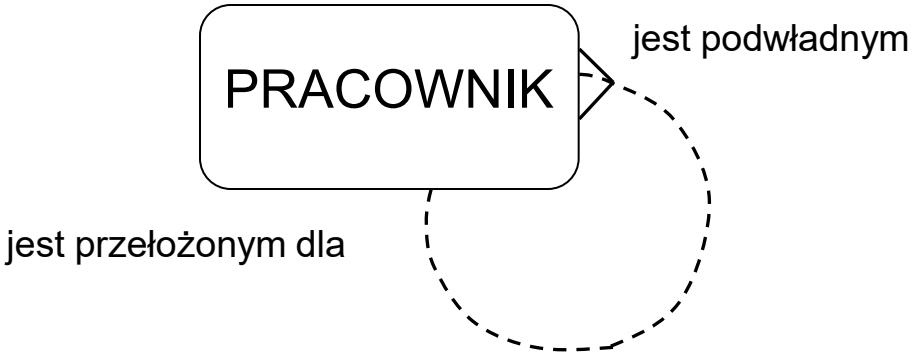
Każdy PROJEKT **MOŻE** być realizowany przez jednego lub wielu PRACOWNIKÓW
Każdy PRACOWNIK **MOŻE** brać udział w jednym lub wielu projektach



Każdy SAMOCHÓW **MUSI** być własnością jednego i tylko jednj OSOBY
Każda OSOBA **MOŻE** być właścicielem jednego lub wielu samochodów

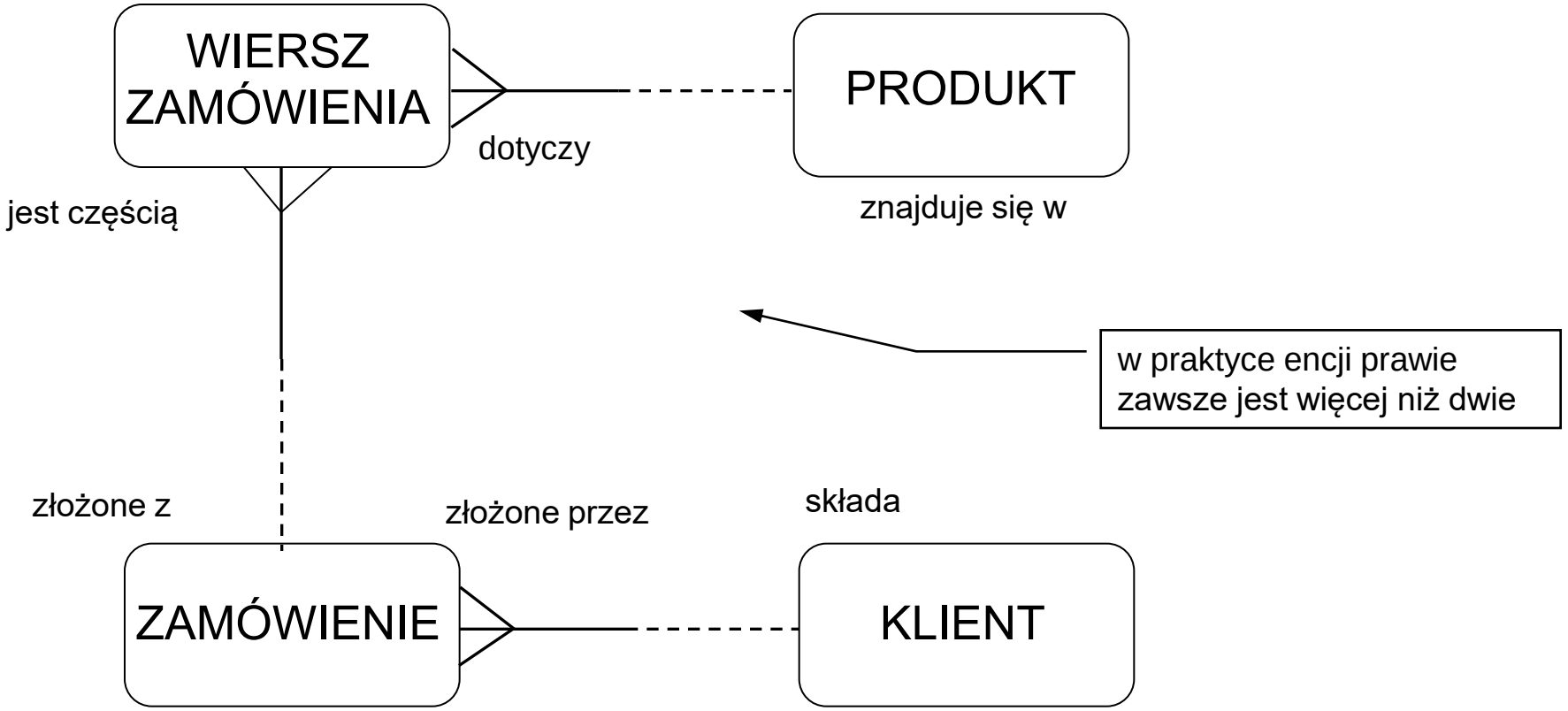
Modelowanie związków encji (4/6)

Każdy SAMOCHÓW **MUSI** posiadać jeden lub więcej PRZEGLADÓW TECHNICZNYCH
Każdy PRZEGLAD TECHNICZNY **MUSI** dotyczyć jednego i tylko jednego samochodu



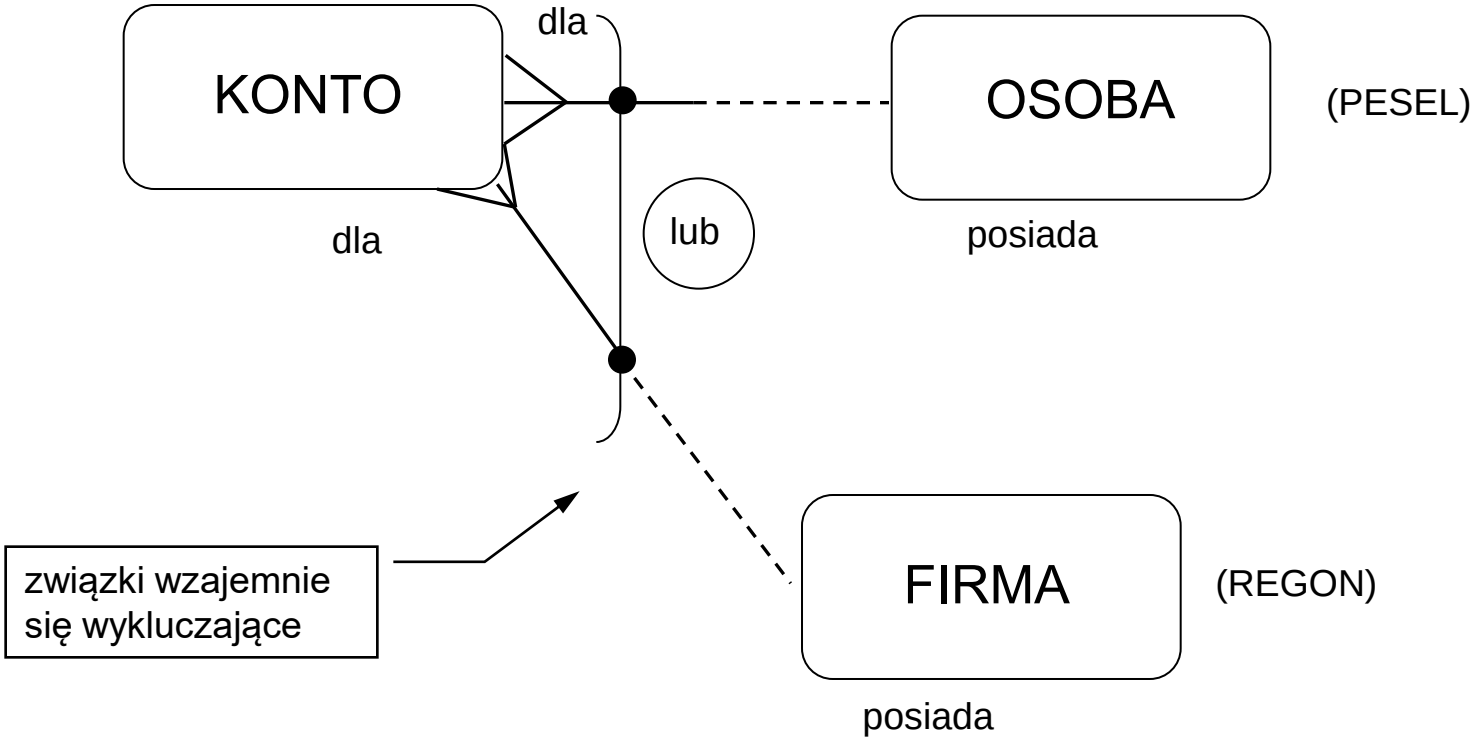
Każdy PRACOWNIK **MOŻE** być podwładnym jednego i tylko jednego PRACOWNIKA
Każdy PRACOWNIK **MOŻE** być przełożonym dla jednego lub wielu PRACOWNIKÓW

Modelowanie związków encji (5/6)



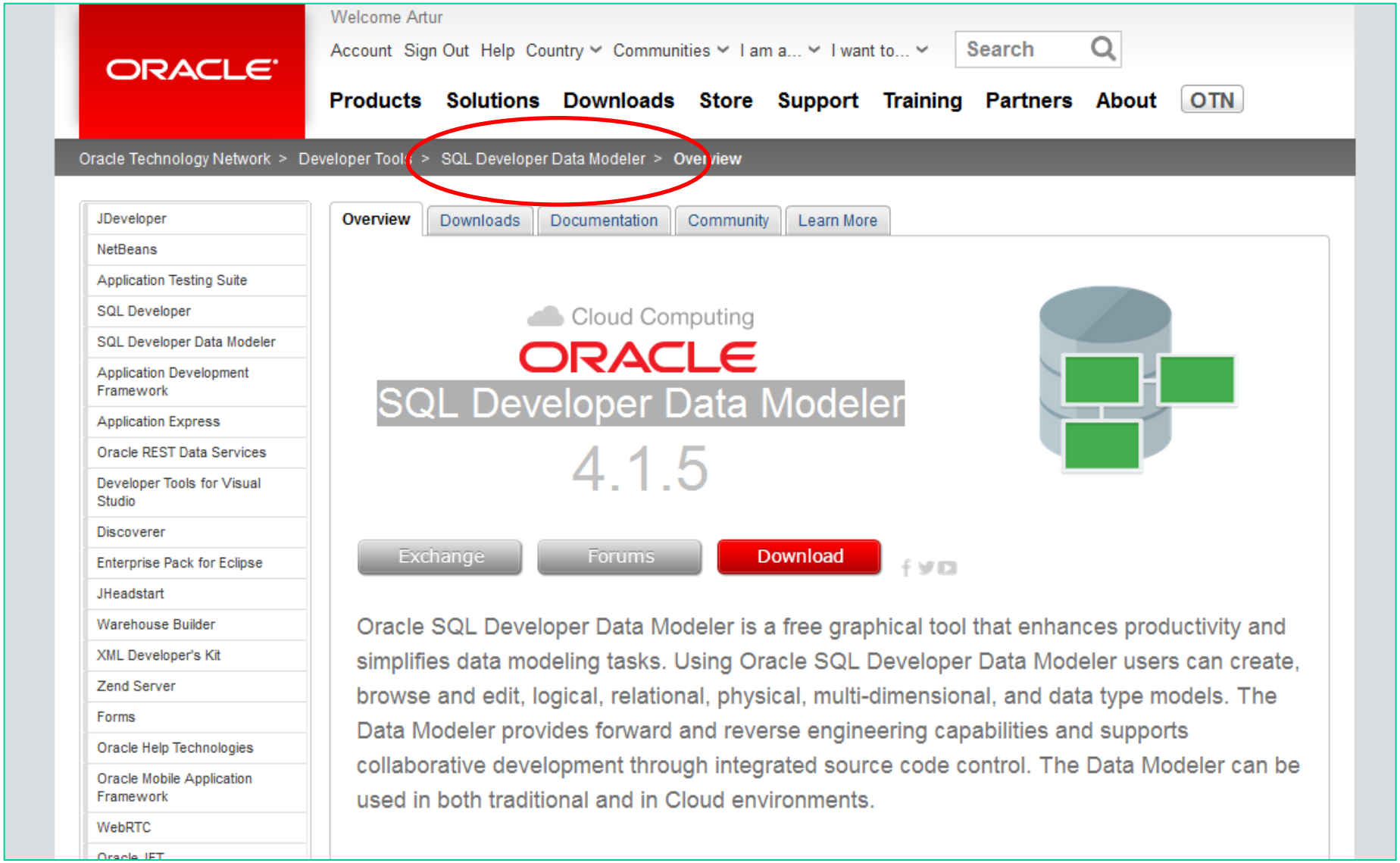
- Każdy PRODUKT **MOŻE** znajdować w jednym lub wielu WIERSZACH ZAMÓWIENIA
- Każde ZAMÓWIENIE **MOŻE** składać się z jednego lub wielu WIERSZY ZAMÓWIEŃ
- Każdy WIERSZ ZAMÓWIENIA **MUSI** dotyczyć jednego i tylko jednego PRODUKTU
- Każdy WIERSZ ZAMÓWIENIA **MUSI** być częścią jednego i tylko jednego ZAMÓWIENIA
- Każdy KLEINT **MOŻE** złożyć jedno lub wiele ZAMÓWIEŃ
- Każde ZAMÓWIENIE **MUSI** być złożone przez jednego i tylko jednego klienta

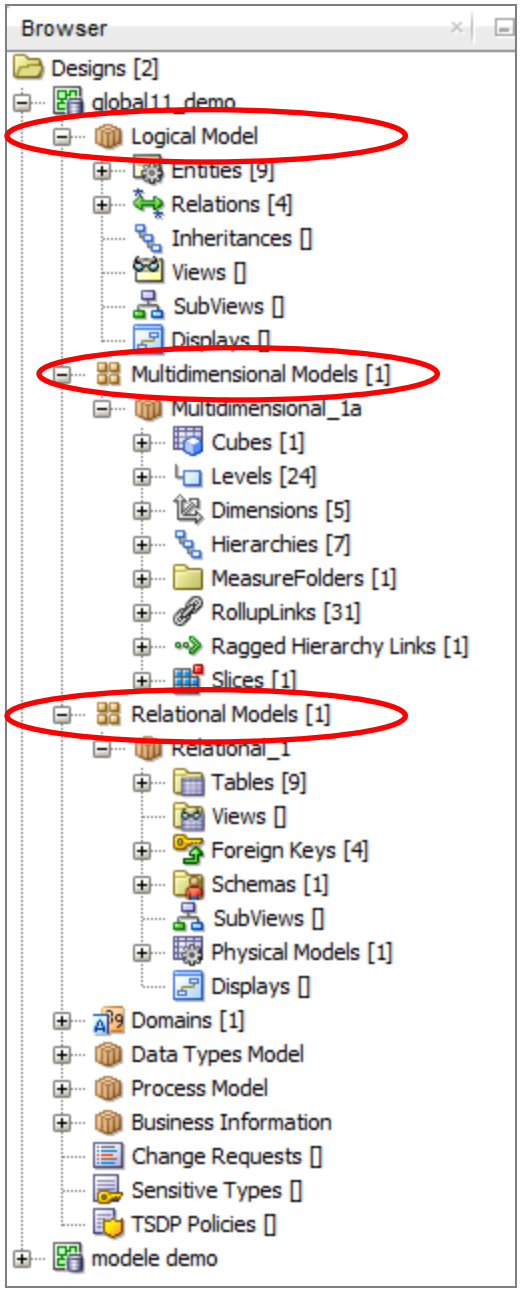
Modelowanie związków encji (6/6)

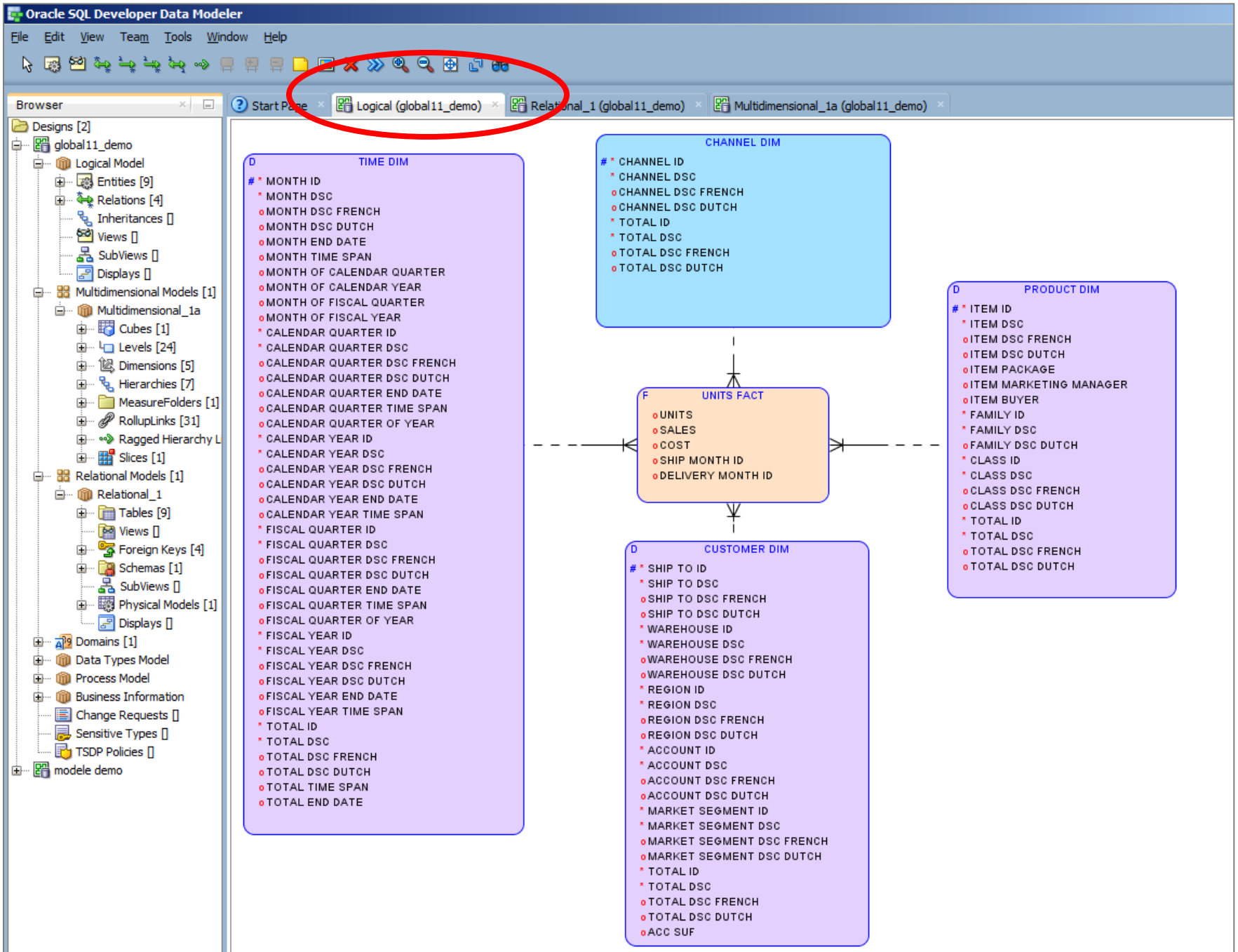


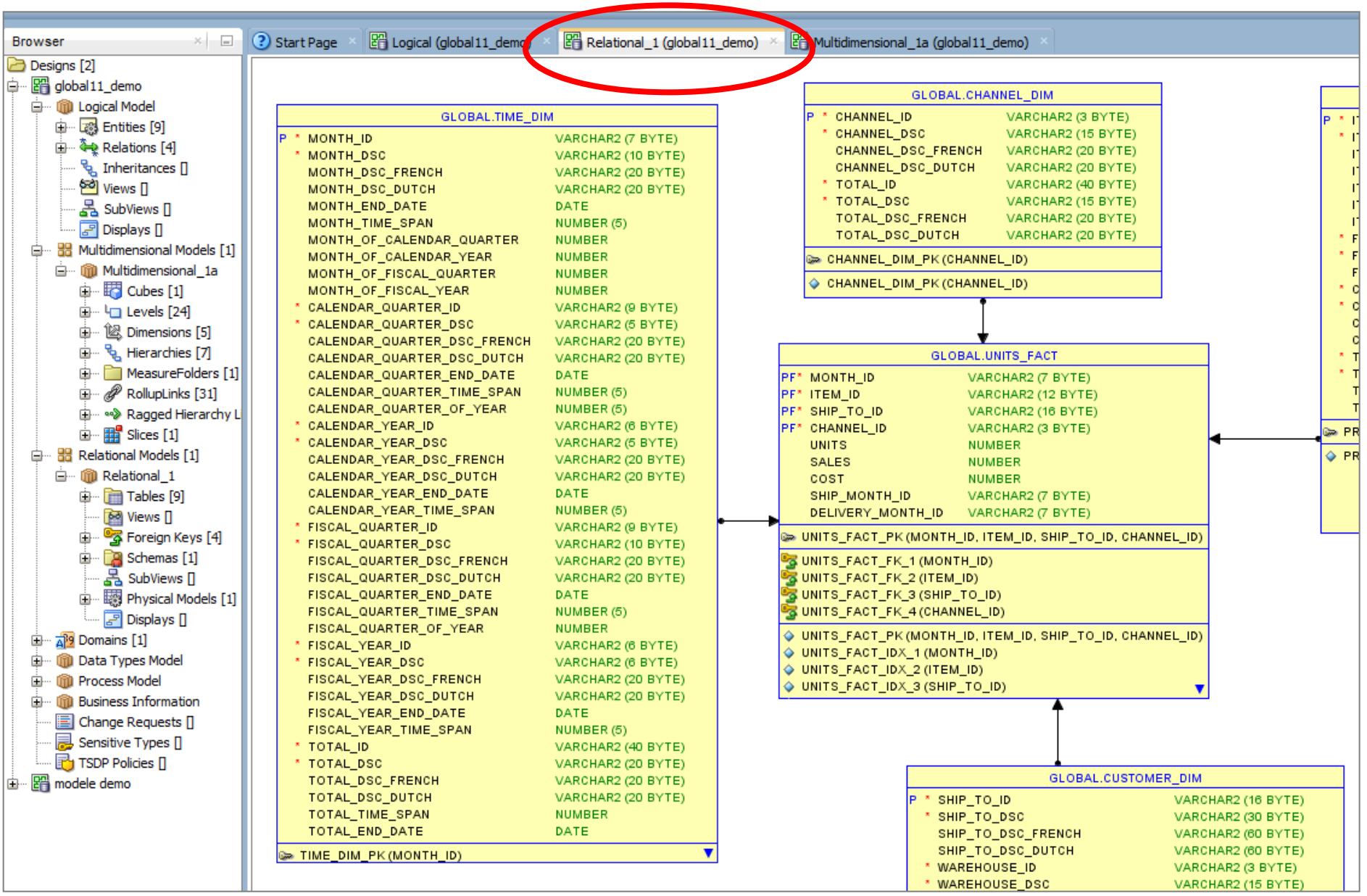
Każde KONTO **MUSI** być **ALBO** dla jednej i tylko jednej OSOBY, **ALBO** dla jednej i tylko jednej FIRMY

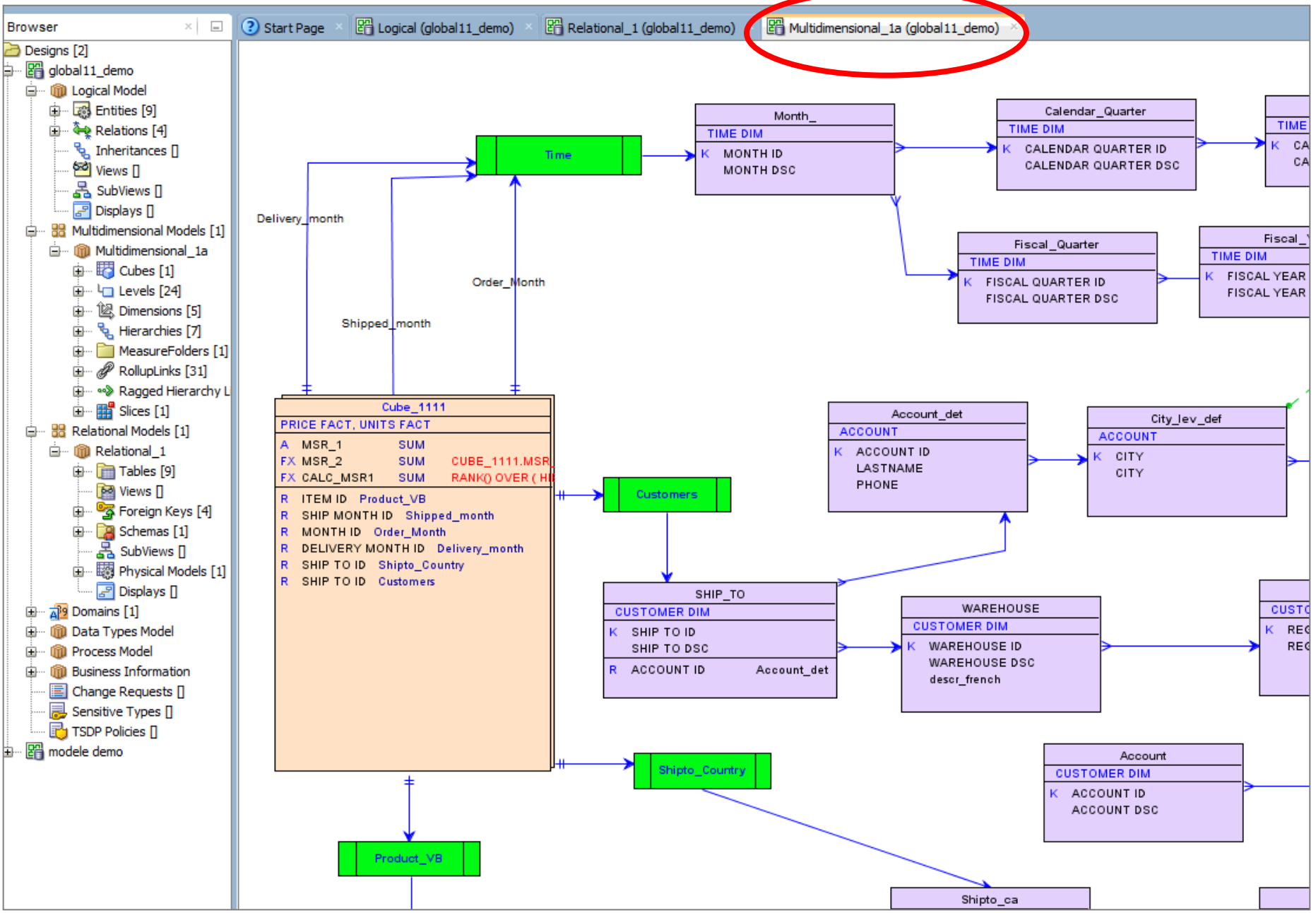
Istnieje wiele narzędzi do "rysowania" diagramów encji. Jednym z nich jest ORACLE SQL Developer Data Modeler

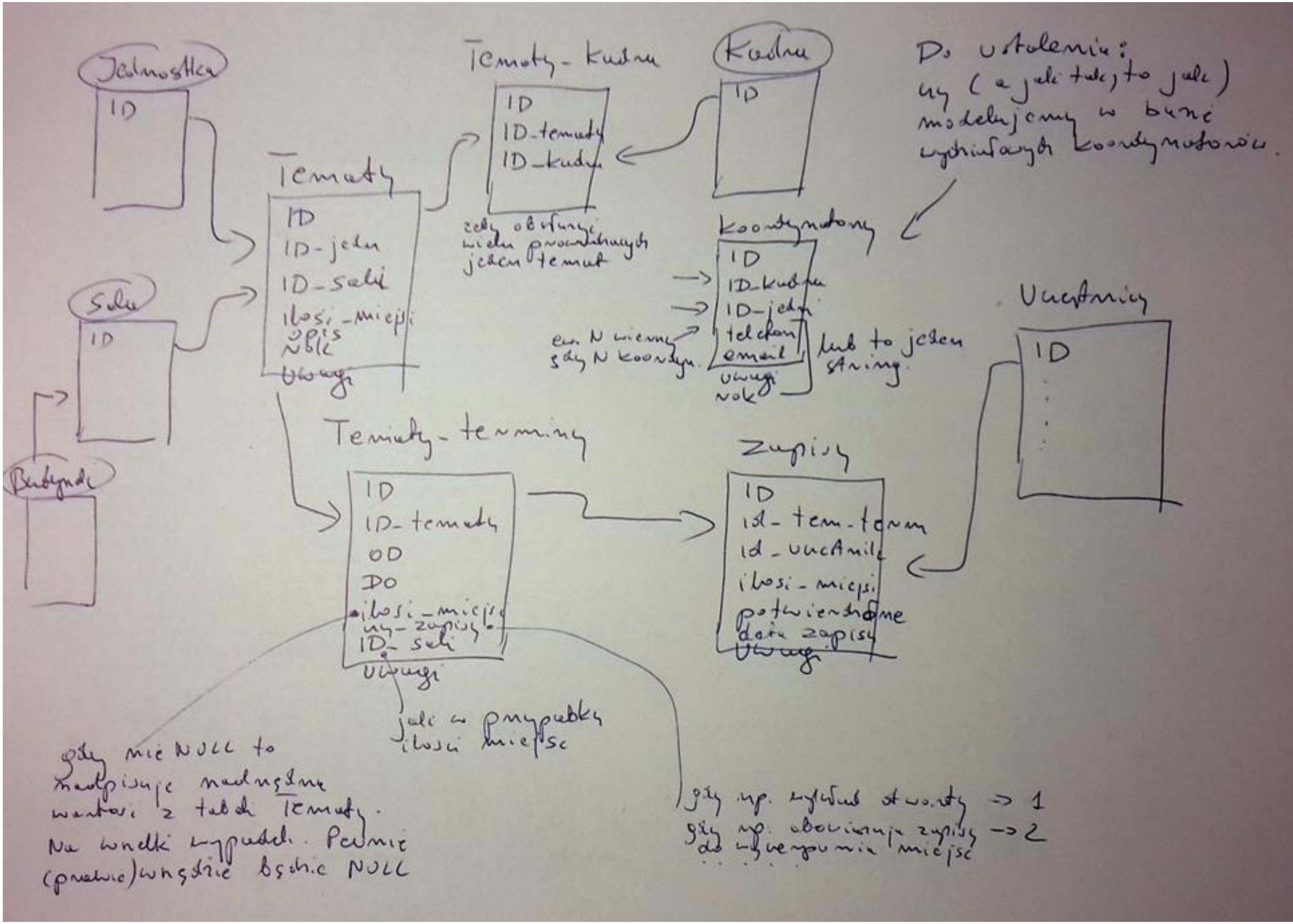

Welcome Artur
Account Sign Out Help Country Communities I am a... I want to...
Search
ORACLE
Products Solutions Downloads Store Support Training Partners About OTN
Oracle Technology Network > Developer Tools > SQL Developer Data Modeler > Overview
JDeveloper
NetBeans
Application Testing Suite
SQL Developer
SQL Developer Data Modeler
Application Development Framework
Application Express
Oracle REST Data Services
Developer Tools for Visual Studio
Discoverer
Enterprise Pack for Eclipse
JHeadstart
Warehouse Builder
XML Developer's Kit
Zend Server
Forms
Oracle Help Technologies
Oracle Mobile Application Framework
WebRTC
Overview
Downloads
Documentation
Community
Learn More
Cloud Computing
ORACLE
SQL Developer Data Modeler
4.1.5
Exchange
Forums
Download
Oracle SQL Developer Data Modeler is a free graphical tool that enhances productivity and simplifies data modeling tasks. Using Oracle SQL Developer Data Modeler users can create, browse and edit, logical, relational, physical, multi-dimensional, and data type models. The Data Modeler provides forward and reverse engineering capabilities and supports collaborative development through integrated source code control. The Data Modeler can be used in both traditional and in Cloud environments.

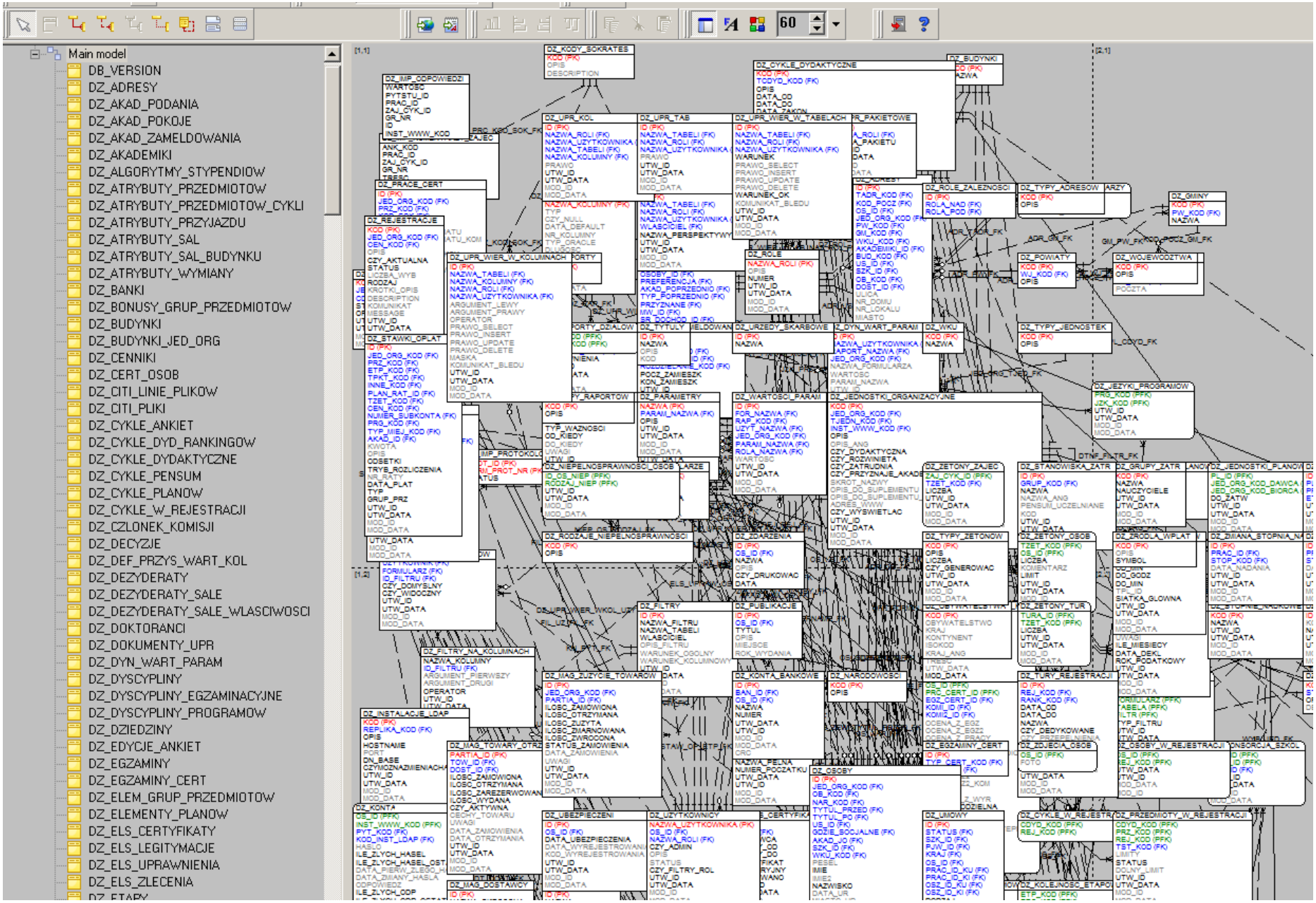

Browser
Designs [2]
global11_demo
Logical Model
Entities [9]
Relations [4]
Inheritances []
Views []
SubViews []
Displays []
Multidimensional Models [1]
Multidimensional_1a
Cubes [1]
Levels [24]
Dimensions [5]
Hierarchies [7]
MeasureFolders [1]
RollupLinks [31]
Ragged Hierarchy Links [1]
Slices [1]
Relational Models [1]
Relational_1
Tables [9]
Views []
Foreign Keys [4]
Schemas [1]
SubViews []
Physical Models [1]
Displays []
Domains [1]
Data Types Model
Process Model
Business Information
Change Requests []
Sensitive Types []
TSDP Policies []
modele demo

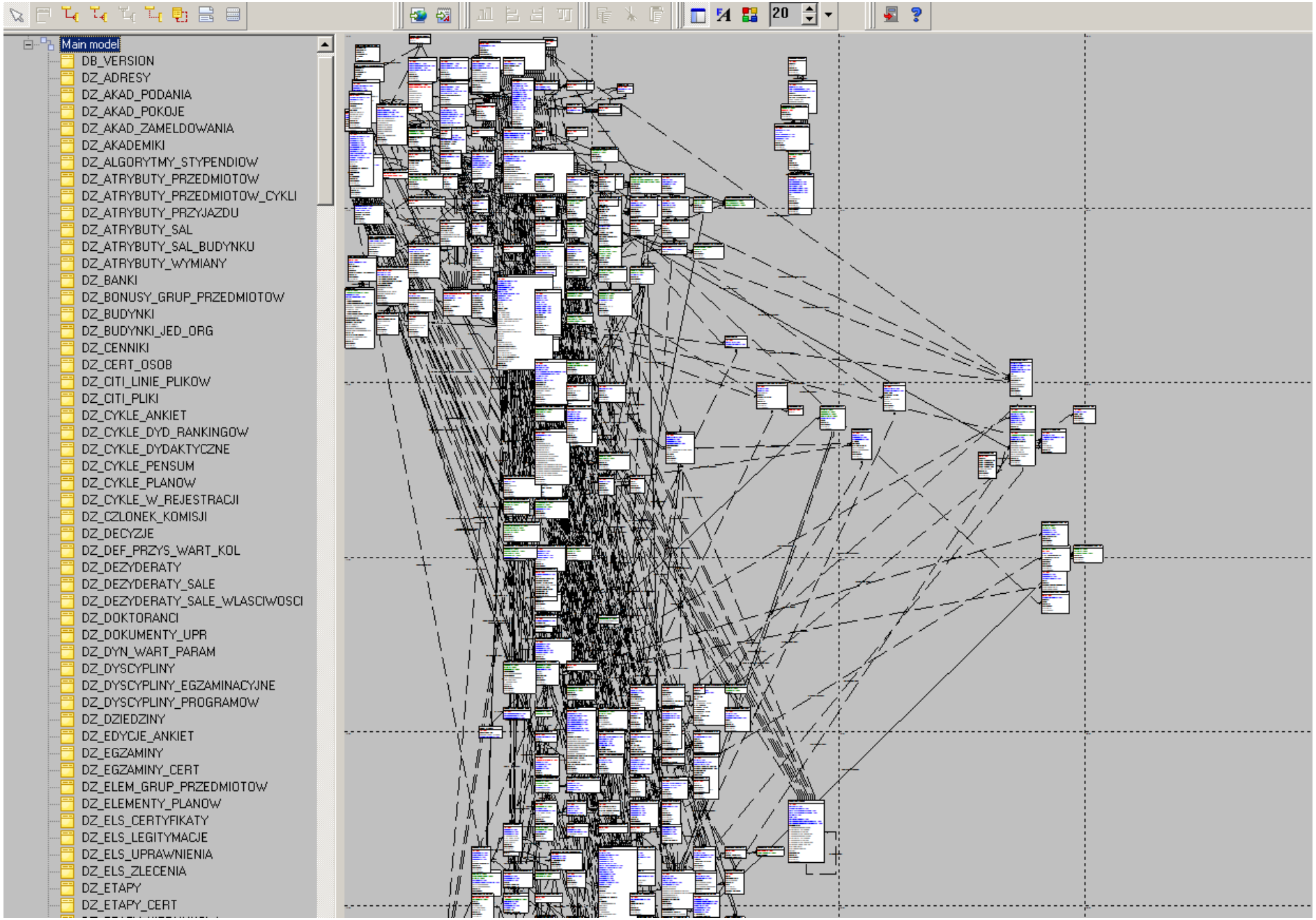

Oracle SQL Developer Data Modeler
File Edit View Team Tools Window Help
Browser
Designs [2]
global11_demo
Logical Model
Entities [9]
Relations [4]
Inheritances []
Views []
SubViews []
Displays []
Multidimensional Models [1]
Multidimensional_1a
Cubes [1]
Levels [24]
Dimensions [5]
Hierarchies [7]
MeasureFolders [1]
RollupLinks [31]
Ragged Hierarchy L
Slices [1]
Relational Models [1]
Relational_1
Tables [9]
Views []
Foreign Keys [4]
Schemas [1]
SubViews []
Physical Models [1]
Displays []
Domains [1]
Data Types Model
Process Model
Business Information
Change Requests []
Sensitive Types []
TSDP Policies []
modele demo
Start Page
Logical (global11_demo)
Relational_1 (global11_demo)
Multidimensional_1a (global11_demo)
D TIME DIM
* MONTH ID
* MONTH DSC
o MONTH DSC FRENCH
o MONTH DSC DUTCH
o MONTH END DATE
o MONTH TIME SPAN
o MONTH OF CALENDAR QUARTER
o MONTH OF CALENDAR YEAR
o MONTH OF FISCAL QUARTER
o MONTH OF FISCAL YEAR
* CALENDAR QUARTER ID
* CALENDAR QUARTER DSC
o CALENDAR QUARTER DSC FRENCH
o CALENDAR QUARTER DSC DUTCH
o CALENDAR QUARTER END DATE
o CALENDAR QUARTER TIME SPAN
o CALENDAR QUARTER OF YEAR
* CALENDAR YEAR ID
* CALENDAR YEAR DSC
o CALENDAR YEAR DSC FRENCH
o CALENDAR YEAR DSC DUTCH
o CALENDAR YEAR END DATE
o CALENDAR YEAR TIME SPAN
* FISCAL QUARTER ID
* FISCAL QUARTER DSC
o FISCAL QUARTER DSC FRENCH
o FISCAL QUARTER DSC DUTCH
o FISCAL QUARTER END DATE
o FISCAL QUARTER TIME SPAN
o FISCAL QUARTER OF YEAR
* FISCAL YEAR ID
* FISCAL YEAR DSC
o FISCAL YEAR DSC FRENCH
o FISCAL YEAR DSC DUTCH
o FISCAL YEAR END DATE
o FISCAL YEAR TIME SPAN
* TOTAL ID
* TOTAL DSC
o TOTAL DSC FRENCH
o TOTAL DSC DUTCH
o TOTAL TIME SPAN
o TOTAL END DATE
CHANNEL DIM
* CHANNEL ID
* CHANNEL DSC
o CHANNEL DSC FRENCH
o CHANNEL DSC DUTCH
* TOTAL ID
* TOTAL DSC
o TOTAL DSC FRENCH
o TOTAL DSC DUTCH
F UNITS FACT
o UNITS
o SALES
o COST
o SHIP MONTH ID
o DELIVERY MONTH ID
D CUSTOMER DIM
* SHIP TO ID
* SHIP TO DSC
o SHIP TO DSC FRENCH
o SHIP TO DSC DUTCH
* WAREHOUSE ID
* WAREHOUSE DSC
o WAREHOUSE DSC FRENCH
o WAREHOUSE DSC DUTCH
* REGION ID
* REGION DSC
o REGION DSC FRENCH
o REGION DSC DUTCH
* ACCOUNT ID
* ACCOUNT DSC
o ACCOUNT DSC FRENCH
o ACCOUNT DSC DUTCH
* MARKET SEGMENT ID
* MARKET SEGMENT DSC
o MARKET SEGMENT DSC FRENCH
o MARKET SEGMENT DSC DUTCH
* TOTAL ID
* TOTAL DSC
o TOTAL DSC FRENCH
o TOTAL DSC DUTCH
o ACC SUF
D PRODUCT DIM
* ITEM ID
* ITEM DSC
o ITEM DSC FRENCH
o ITEM DSC DUTCH
o ITEM PACKAGE
o ITEM MARKETING MANAGER
o ITEM BUYER
* FAMILY ID
* FAMILY DSC
o FAMILY DSC DUTCH
* CLASS ID
* CLASS DSC
o CLASS DSC FRENCH
o CLASS DSC DUTCH
* TOTAL ID
* TOTAL DSC
o TOTAL DSC FRENCH
o TOTAL DSC DUTCH

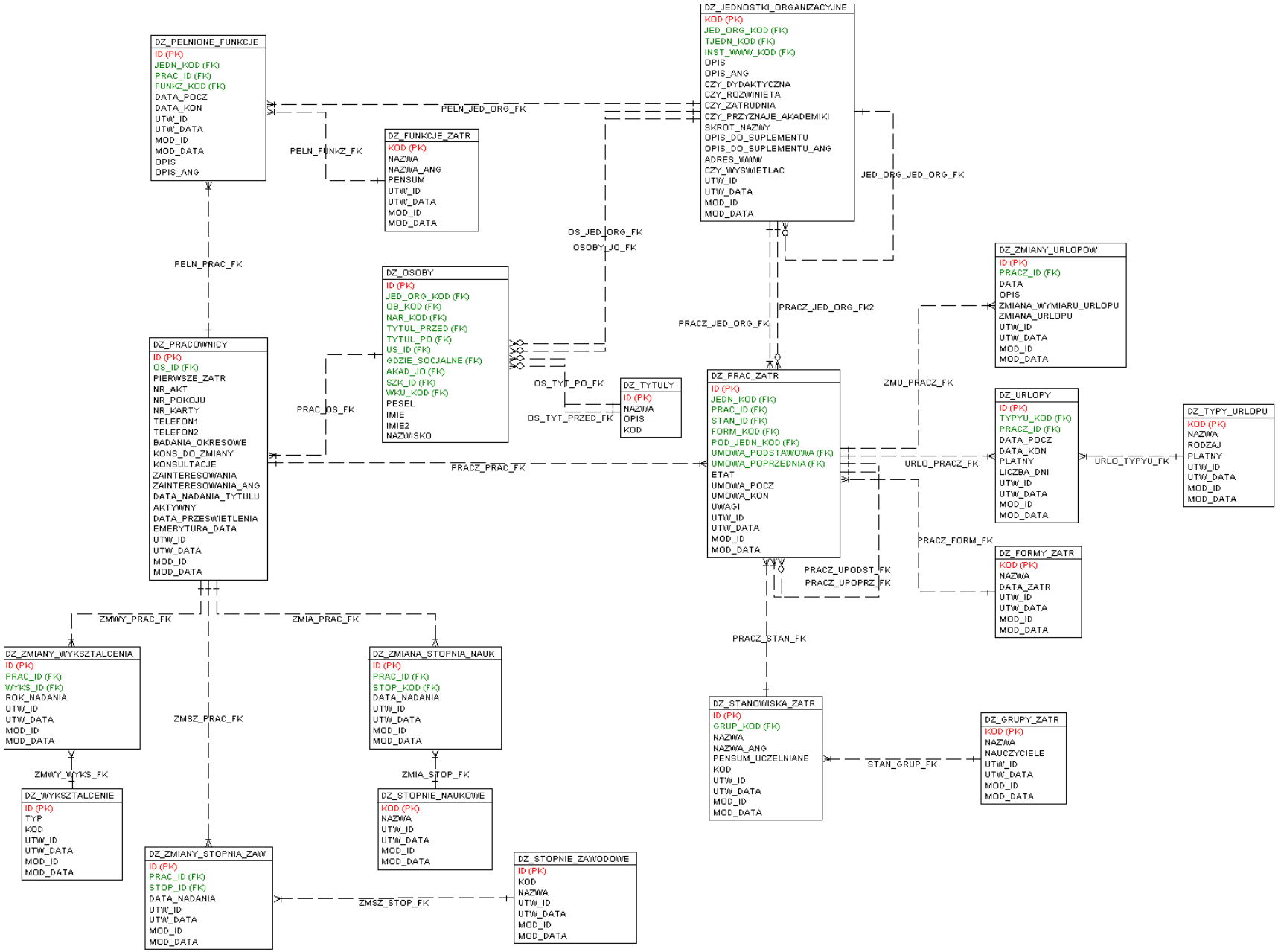

Browser
Designs [2]
global11_demo
Logical Model
Entities [9]
Relations [4]
Inheritances []
Views []
SubViews []
Displays []
Multidimensional Models [1]
Multidimensional_1a
Cubes [1]
Levels [24]
Dimensions [5]
Hierarchies [7]
MeasureFolders [1]
RollupLinks [31]
Ragged Hierarchy L
Slices [1]
Relational Models [1]
Relational_1
Tables [9]
Views []
Foreign Keys [4]
Schemas [1]
SubViews []
Physical Models [1]
Displays []
Domains [1]
Data Types Model
Process Model
Business Information
Change Requests []
Sensitive Types []
TSDP Policies []
modele demo
Start Page
Logical (global11_demo)
Relational_1 (global11_demo)
Multidimensional_1a (global11_demo)
GLOBAL.TIME_DIM
P * MONTH_ID VARCHAR2 (7 BYTE)
* MONTH_DSC VARCHAR2 (10 BYTE)
MONTH_DSC_FRENCH VARCHAR2 (20 BYTE)
MONTH_DSC_DUTCH VARCHAR2 (20 BYTE)
MONTH_END_DATE DATE
MONTH_TIME_SPAN NUMBER (5)
MONTH_OF_CALENDAR_QUARTER NUMBER
MONTH_OF_CALENDAR_YEAR NUMBER
MONTH_OF_FISCAL_QUARTER NUMBER
MONTH_OF_FISCAL_YEAR NUMBER
* CALENDAR_QUARTER_ID VARCHAR2 (9 BYTE)
* CALENDAR_QUARTER_DSC VARCHAR2 (5 BYTE)
CALENDAR_QUARTER_DSC_FRENCH VARCHAR2 (20 BYTE)
CALENDAR_QUARTER_DSC_DUTCH VARCHAR2 (20 BYTE)
CALENDAR_QUARTER_END_DATE DATE
CALENDAR_QUARTER_TIME_SPAN NUMBER (5)
CALENDAR_QUARTER_OF_YEAR NUMBER (5)
* CALENDAR_YEAR_ID VARCHAR2 (6 BYTE)
* CALENDAR_YEAR_DSC VARCHAR2 (5 BYTE)
CALENDAR_YEAR_DSC_FRENCH VARCHAR2 (20 BYTE)
CALENDAR_YEAR_DSC_DUTCH VARCHAR2 (20 BYTE)
CALENDAR_YEAR_END_DATE DATE
CALENDAR_YEAR_TIME_SPAN NUMBER (5)
* FISCAL_QUARTER_ID VARCHAR2 (9 BYTE)
* FISCAL_QUARTER_DSC VARCHAR2 (10 BYTE)
FISCAL_QUARTER_DSC_FRENCH VARCHAR2 (20 BYTE)
FISCAL_QUARTER_DSC_DUTCH VARCHAR2 (20 BYTE)
FISCAL_QUARTER_END_DATE DATE
FISCAL_QUARTER_TIME_SPAN NUMBER (5)
FISCAL_QUARTER_OF_YEAR NUMBER
* FISCAL_YEAR_ID VARCHAR2 (6 BYTE)
* FISCAL_YEAR_DSC VARCHAR2 (6 BYTE)
FISCAL_YEAR_DSC_FRENCH VARCHAR2 (20 BYTE)
FISCAL_YEAR_DSC_DUTCH VARCHAR2 (20 BYTE)
FISCAL_YEAR_END_DATE DATE
FISCAL_YEAR_TIME_SPAN NUMBER (5)
* TOTAL_ID VARCHAR2 (40 BYTE)
* TOTAL_DSC VARCHAR2 (20 BYTE)
TOTAL_DSC_FRENCH VARCHAR2 (20 BYTE)
TOTAL_DSC_DUTCH VARCHAR2 (20 BYTE)
TOTAL_TIME_SPAN NUMBER
TOTAL_END_DATE DATE
TIME_DIM_PK (MONTH_ID)
GLOBAL.CHANNEL_DIM
P * CHANNEL_ID VARCHAR2 (3 BYTE)
* CHANNEL_DSC VARCHAR2 (15 BYTE)
CHANNEL_DSC_FRENCH VARCHAR2 (20 BYTE)
CHANNEL_DSC_DUTCH VARCHAR2 (20 BYTE)
* TOTAL_ID VARCHAR2 (40 BYTE)
* TOTAL_DSC VARCHAR2 (15 BYTE)
TOTAL_DSC_FRENCH VARCHAR2 (20 BYTE)
TOTAL_DSC_DUTCH VARCHAR2 (20 BYTE)
CHANNEL_DIM_PK (CHANNEL_ID)
CHANNEL_DIM_PK (CHANNEL_ID)
GLOBAL.UNITS_FACT
PF* MONTH_ID VARCHAR2 (7 BYTE)
PF* ITEM_ID VARCHAR2 (12 BYTE)
PF* SHIP_TO_ID VARCHAR2 (16 BYTE)
PF* CHANNEL_ID VARCHAR2 (3 BYTE)
UNITS NUMBER
SALES NUMBER
COST NUMBER
SHIP_MONTH_ID VARCHAR2 (7 BYTE)
DELIVERY_MONTH_ID VARCHAR2 (7 BYTE)
UNITS_FACT_PK (MONTH_ID, ITEM_ID, SHIP_TO_ID, CHANNEL_ID)
UNITS_FACT_FK_1 (MONTH_ID)
UNITS_FACT_FK_2 (ITEM_ID)
UNITS_FACT_FK_3 (SHIP_TO_ID)
UNITS_FACT_FK_4 (CHANNEL_ID)
UNITS_FACT_PK (MONTH_ID, ITEM_ID, SHIP_TO_ID, CHANNEL_ID)
UNITS_FACT_IDX_1 (MONTH_ID)
UNITS_FACT_IDX_2 (ITEM_ID)
UNITS_FACT_IDX_3 (SHIP_TO_ID)
GLOBAL.CUSTOMER_DIM
P * SHIP_TO_ID VARCHAR2 (16 BYTE)
* SHIP_TO_DSC VARCHAR2 (30 BYTE)
SHIP_TO_DSC_FRENCH VARCHAR2 (60 BYTE)
SHIP_TO_DSC_DUTCH VARCHAR2 (60 BYTE)
* WAREHOUSE_ID VARCHAR2 (3 BYTE)
* WAREHOUSE_DSC VARCHAR2 (15 BYTE)


Browser
Designs [2]
global11_demo
Logical Model
Entities [9]
Relations [4]
Inheritances []
Views []
SubViews []
Displays []
Multidimensional Models [1]
Multidimensional_1a
Cubes [1]
Levels [24]
Dimensions [5]
Hierarchies [7]
MeasureFolders [1]
RollupLinks [31]
Ragged Hierarchy L
Slices [1]
Relational Models [1]
Relational_1
Tables [9]
Views []
Foreign Keys [4]
Schemas [1]
SubViews []
Physical Models [1]
Displays []
Domains [1]
Data Types Model
Process Model
Business Information
Change Requests []
Sensitive Types []
TSDP Policies []
modele demo
Start Page
Logical (global11_demo)
Relational_1 (global11_demo)
Multidimensional_1a (global11_demo)
Time
Month_
TIME DIM
K MONTH ID
MONTH DSC
Calendar_Quarter
TIME DIM
K CALENDAR QUARTER ID
CALENDAR QUARTER DSC
Fiscal_Quarter
TIME DIM
K FISCAL QUARTER ID
FISCAL QUARTER DSC
Delivery_month
Order_Month
Shipped_month
Cube_1111
PRICE FACT, UNITS FACT
A MSR_1 SUM
FX MSR_2 SUM CUBE_1111.MSR
FX CALC_MSR1 SUM RANK() OVER (HI
R ITEM ID Product_VB
R SHIP MONTH ID Shipped_month
R MONTH ID Order_Month
R DELIVERY MONTH ID Delivery_month
R SHIP TO ID Shipto_Country
R SHIP TO ID Customers
Customers
Account_det
ACCOUNT
K ACCOUNT ID
LASTNAME
PHONE
City_lev_def
ACCOUNT
K CITY
CITY
SHIP_TO
CUSTOMER DIM
K SHIP TO ID
SHIP TO DSC
R ACCOUNT ID Account_det
WAREHOUSE
CUSTOMER DIM
K WAREHOUSE ID
WAREHOUSE DSC
descr_french
Shipto_Country
Account
CUSTOMER DIM
K ACCOUNT ID
ACCOUNT DSC
Product_VB
Shipto_ca

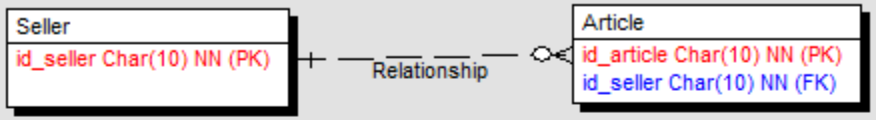

Jednostka
ID
Tematy-kadra
ID
ID-tematy
ID-kadry
Kadra
ID
Do ustalenia:
czy (a jeśli tak, to jak)
modelujemy w bazie
wydziałowych koordynatorów.
Tematy
ID
ID-jedn
ID-sali
ilość-miejsc
opis
NULL
uwagi
żeby obsłużyć
wielu prowadzących
jeden temat
Koordynatorzy
ID
ID-kadra
ID-jedn
telefon
email
uwagi
rok
ew. N wierszy
gdy N koordyn.
lub to jeden
string
Sale
ID
Budynki
Uczestnicy
ID
Tematy-terminy
ID
ID-tematy
OD
DO
ilość_miejsc
czy-zapisy
ID-sali
uwagi
jak w przypadku
ilości miejsc
Zapisy
ID
id-tem-term
id-uczestnika
ilość-miejsc
potwierdzone
data zapisu
uwagi
gdy nie NULL to
nadpisuje nadrzędne
wartości z tabeli Tematy.
Pewnie
(prawie) wszędzie będzie NULL
gdy np. wybrał otwarty -> 1
gdy np. obowiązują zapisy -> 2
do wyczerpania miejsc


Main model
DB_VERSION
DZ_ADRESY
DZ_AKAD_PODANIA
DZ_AKAD_POKOJE
DZ_AKAD_ZAMELDOWANIA
DZ_AKADEMIKI
DZ_ALGORYTMY_STYPENDIOW
DZ_ATRYBUTY_PRZEDMIOTOW
DZ_ATRYBUTY_PRZEDMIOTOW_CYKLI
DZ_ATRYBUTY_PRZYJAZDU
DZ_ATRYBUTY_SAL
DZ_ATRYBUTY_SAL_BUDYNKU
DZ_ATRYBUTY_WYMIANY
DZ_BANKI
DZ_BONUSY_GRUP_PRZEDMIOTOW
DZ_BUDYNKI
DZ_BUDYNKI_JED_ORG
DZ_CENNIKI
DZ_CERT_OSOB
DZ_CITI_LINIE_PLIKOW
DZ_CITI_PLIKI
DZ_CYKLE_ANKIET
DZ_CYKLE_DYD_RANKINGOW
DZ_CYKLE_DYDAKTYCZNE
DZ_CYKLE_PENSUM
DZ_CYKLE_PLANOW
DZ_CYKLE_W_REJESTRACJI
DZ_CZLONEK_KOMISJI
DZ_DECYZJE
DZ_DEF_PRZYS_WART_KOL
DZ_DEZYDERATY
DZ_DEZYDERATY_SALE
DZ_DEZYDERATY_SALE_WLASCIWOSCI
DZ_DOKTORANCI
DZ_DOKUMENTY_UPR
DZ_DYN_WART_PARAM
DZ_DYSCYPLINY
DZ_DYSCYPLINY_EGZAMINACYJNE
DZ_DYSCYPLINY_PROGRAMOW
DZ_DZIEDZINY
DZ_EDYCJE_ANKIET
DZ_EGZAMINY
DZ_EGZAMINY_CERT
DZ_ELEM_GRUP_PRZEDMIOTOW
DZ_ELEMENTY_PLANOW
DZ_ELS_CERTYFIKATY
DZ_ELS_LEGITYMACJE
DZ_ELS_UPRAWNIENIA
DZ_ELS_ZLECENIA
DZ_ETAPY


Main model
DB_VERSION
DZ_ADRESY
DZ_AKAD_PODANIA
DZ_AKAD_POKOJE
DZ_AKAD_ZAMELDOWANIA
DZ_AKADEMIKI
DZ_ALGORYTMY_STYPENDIOW
DZ_ATRYBUTY_PRZEDMIOTOW
DZ_ATRYBUTY_PRZEDMIOTOW_CYKLI
DZ_ATRYBUTY_PRZYJAZDU
DZ_ATRYBUTY_SAL
DZ_ATRYBUTY_SAL_BUDYNKU
DZ_ATRYBUTY_WYMIANY
DZ_BANKI
DZ_BONUSY_GRUP_PRZEDMIOTOW
DZ_BUDYNKI
DZ_BUDYNKI_JED_ORG
DZ_CENNIKI
DZ_CERT_OSOB
DZ_CITI_LINIE_PLIKOW
DZ_CITI_PLIKI
DZ_CYKLE_ANKIET
DZ_CYKLE_DYD_RANKINGOW
DZ_CYKLE_DYDAKTYCZNE
DZ_CYKLE_PENSUM
DZ_CYKLE_PLANOW
DZ_CYKLE_W_REJESTRACJI
DZ_CZLONEK_KOMISJI
DZ_DECYZJE
DZ_DEF_PRZYS_WART_KOL
DZ_DEZYDERATY
DZ_DEZYDERATY_SALE
DZ_DEZYDERATY_SALE_WLASCIWOSCI
DZ_DOKTORANCI
DZ_DOKUMENTY_UPR
DZ_DYN_WART_PARAM
DZ_DYSCYPLINY
DZ_DYSCYPLINY_EGZAMINACYJNE
DZ_DYSCYPLINY_PROGRAMOW
DZ_DZIEDZINY
DZ_EDYCJE_ANKIET
DZ_EGZAMINY
DZ_EGZAMINY_CERT
DZ_ELEM_GRUP_PRZEDMIOTOW
DZ_ELEMENTY_PLANOW
DZ_ELS_CERTYFIKATY
DZ_ELS_LEGITYMACJE
DZ_ELS_UPRAWNIENIA
DZ_ELS_ZLECENIA
DZ_ETAPY
DZ_ETAPY_CERT


DZ_JEDNOSTKI_ORGANIZACYJNE
KOD (PK)
JED_ORG_KOD (FK)
TJEDN_KOD (FK)
INST_WWW_KOD (FK)
OPIS
OPIS_ANG
CZY_DYDAKTYCZNA
CZY_ROZWINIETA
CZY_ZATRUDNIA
CZY_PRZYZNAJE_AKADEMIKI
SKROT_NAZWY
OPIS_DO_SUPLEMENTU
OPIS_DO_SUPLEMENTU_ANG
ADRES_WWW
CZY_WYSWIETLAC
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA
DZ_PELNIONE_FUNKCJE
ID (PK)
JEDN_KOD (FK)
PRAC_ID (FK)
FUNKZ_KOD (FK)
DATA_POCZ
DATA_KON
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA
OPIS
OPIS_ANG
DZ_FUNKCJE_ZATR
KOD (PK)
NAZWA
NAZWA_ANG
PENSUM
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA
PELN_JED_ORG_FK
PELN_FUNKZ_FK
PELN_PRAC_FK
JED_ORG_JED_ORG_FK
OS_JED_ORG_FK
OSOBY_JO_FK
DZ_OSOBY
ID (PK)
JED_ORG_KOD (FK)
OB_KOD (FK)
NAR_KOD (FK)
TYTUL_PRZED (FK)
TYTUL_PO (FK)
US_ID (FK)
GDZIE_SOCJALNE (FK)
AKAD_JO (FK)
SZK_ID (FK)
WKU_KOD (FK)
PESEL
IMIE
IMIE2
NAZWISKO
OS_TYT_PO_FK
OS_TYT_PRZED_FK
DZ_TYTULY
ID (PK)
NAZWA
OPIS
KOD
PRACZ_JED_ORG_FK2
PRACZ_JED_ORG_FK
DZ_ZMIANY_URLOPOW
ID (PK)
PRACZ_ID (FK)
DATA
OPIS
ZMIANA_WYMIARU_URLOPU
ZMIANA_URLOPU
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA
DZ_PRACOWNICY
ID (PK)
OS_ID (FK)
PIERWSZE_ZATR
NR_AKT
NR_POKOJU
NR_KARTY
TELEFON1
TELEFON2
BADANIA_OKRESOWE
KONS_DO_ZMIANY
KONSULTACJE
ZAINTERESOWANIA
ZAINTERESOWANIA_ANG
DATA_NADANIA_TYTULU
AKTYWNY
DATA_PRZESWIETLENIA
EMERYTURA_DATA
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA
PRAC_OS_FK
PRACZ_PRAC_FK
DZ_PRAC_ZATR
ID (PK)
JEDN_KOD (FK)
PRAC_ID (FK)
STAN_ID (FK)
FORM_KOD (FK)
POD_JEDN_KOD (FK)
UMOWA_PODSTAWOWA (FK)
UMOWA_POPRZEDNIA (FK)
ETAT
UMOWA_POCZ
UMOWA_KON
UWAGI
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA
ZMU_PRACZ_FK
URLO_PRACZ_FK
DZ_URLOPY
ID (PK)
TYPYU_KOD (FK)
PRACZ_ID (FK)
DATA_POCZ
DATA_KON
PLATNY
LICZBA_DNI
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA
URLO_TYPYU_FK
DZ_TYPY_URLOPU
KOD (PK)
NAZWA
RODZAJ
PLATNY
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA
PRACZ_FORM_FK
DZ_FORMY_ZATR
KOD (PK)
NAZWA
DATA_ZATR
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA
PRACZ_UPODST_FK
PRACZ_UPOPRZ_FK
ZMWY_PRAC_FK
ZMIA_PRAC_FK
PRACZ_STAN_FK
DZ_ZMIANY_WYKSZTALCENIA
ID (PK)
PRAC_ID (FK)
WYKS_ID (FK)
ROK_NADANIA
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA
DZ_ZMIANA_STOPNIA_NAUK
ID (PK)
PRAC_ID (FK)
STOP_KOD (FK)
DATA_NADANIA
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA
DZ_STANOWISKA_ZATR
ID (PK)
GRUP_KOD (FK)
NAZWA
NAZWA_ANG
PENSUM_UCZELNIANE
KOD
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA
STAN_GRUP_FK
DZ_GRUPY_ZATR
KOD (PK)
NAZWA
NAUCZYCIELE
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA
ZMWY_WYKS_FK
ZMSZ_PRAC_FK
ZMIA_STOP_FK
DZ_WYKSZTALCENIE
ID (PK)
TYP
KOD
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA
DZ_STOPNIE_NAUKOWE
KOD (PK)
NAZWA
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA
DZ_ZMIANY_STOPNIA_ZAW
ID (PK)
PRAC_ID (FK)
STOP_ID (FK)
DATA_NADANIA
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA
ZMSZ_STOP_FK
DZ_STOPNIE_ZAWODOWE
ID (PK)
KOD
NAZWA
UTW_ID
UTW_DATA
MOD_ID
MOD_DATA

Toad Data Modeler (wer. 2.25)



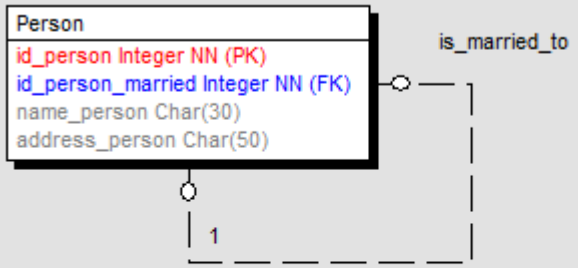
The seller may have N articles and any article has 1 seller (1:N relationship)
The seller needn't have any article, each article must have at least one seller.

OR

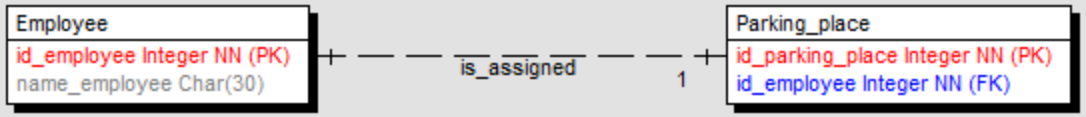
The seller may have 0 or N articles but at the same time,
any article has 1 and only 1 seller:

Cardinality (min,max)-notation:

SELLER ARTICLE
0,N 1,1



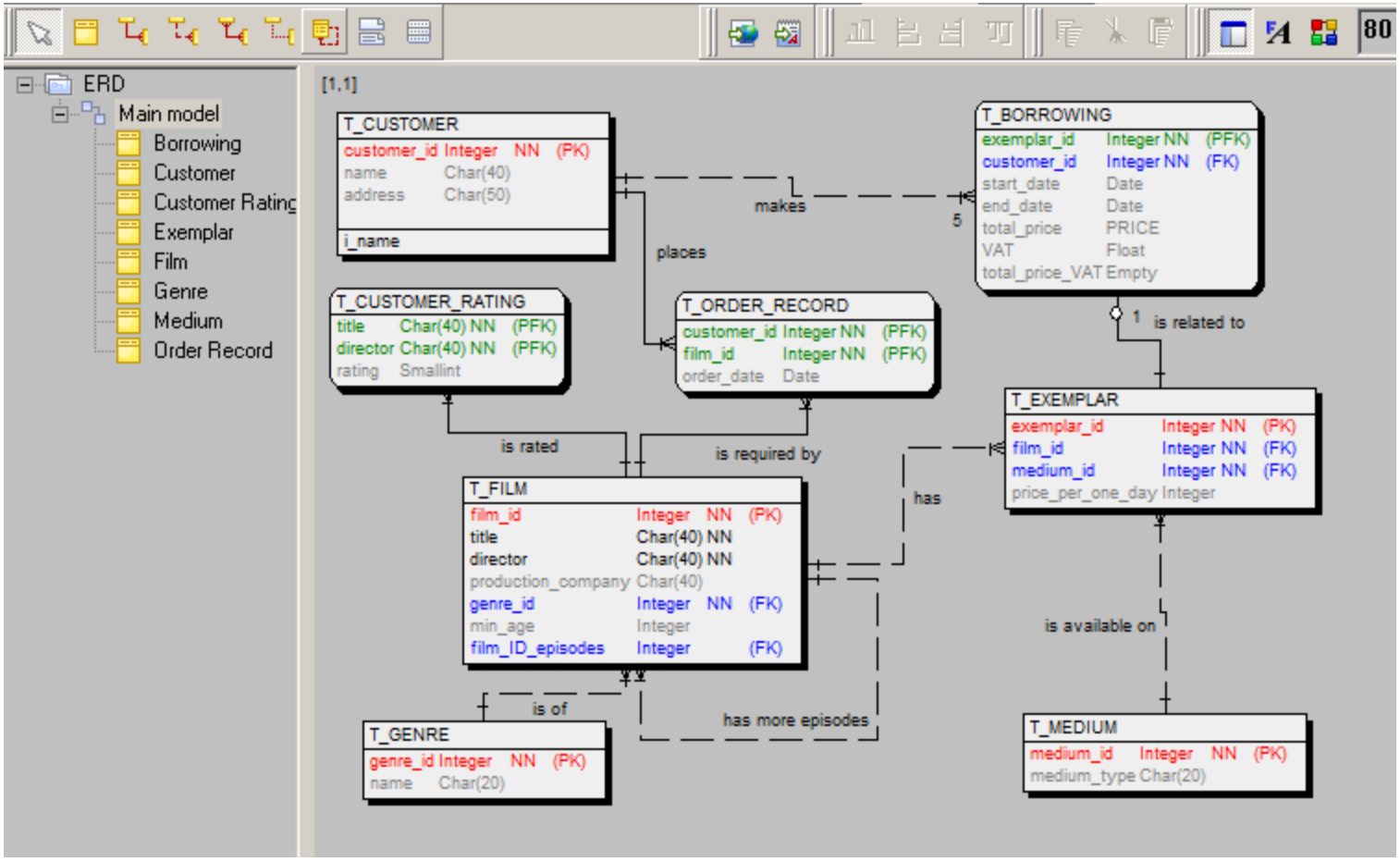
Pay attention to the use of the Rolename feature !
(Edit the Foreign key attribute and see the Rolename input field)



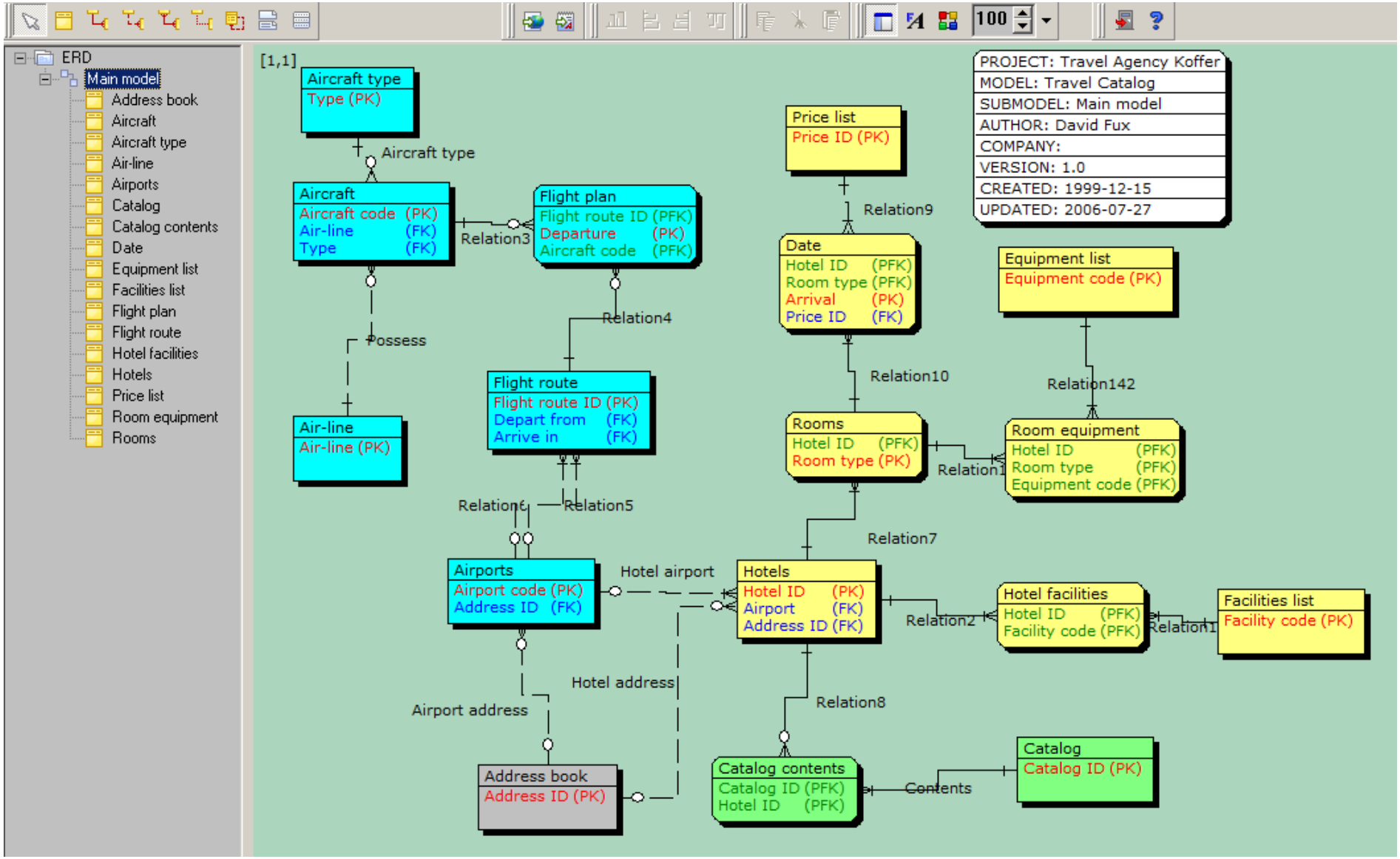
Example of cardinality 1:1

Each employee has assigned exactly one parking place
and each parking place is assigned to exactly one employee.

Toad Data Modeler (wer. 2.25)

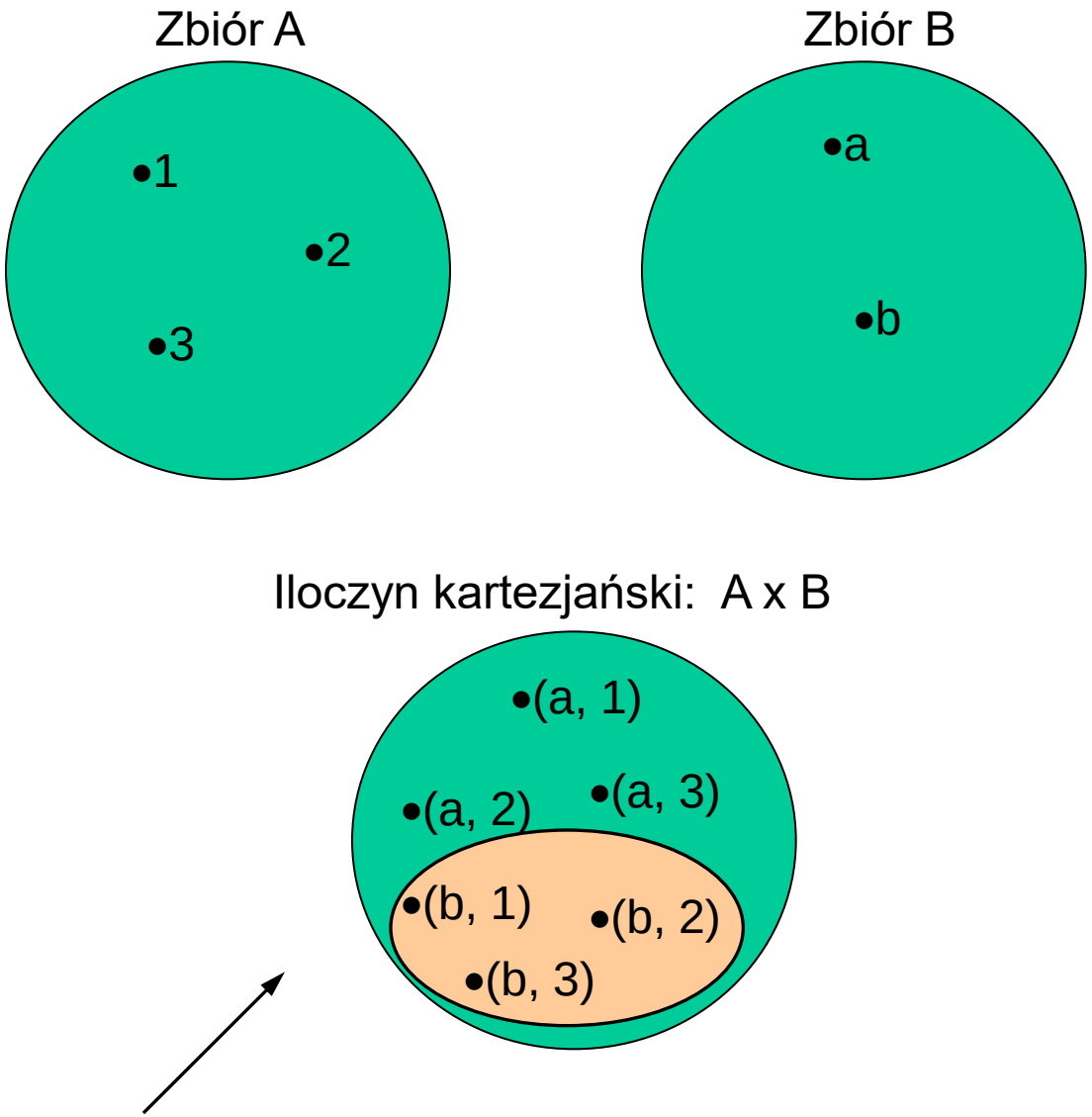


Toad Data Modeler (wer. 2.25)



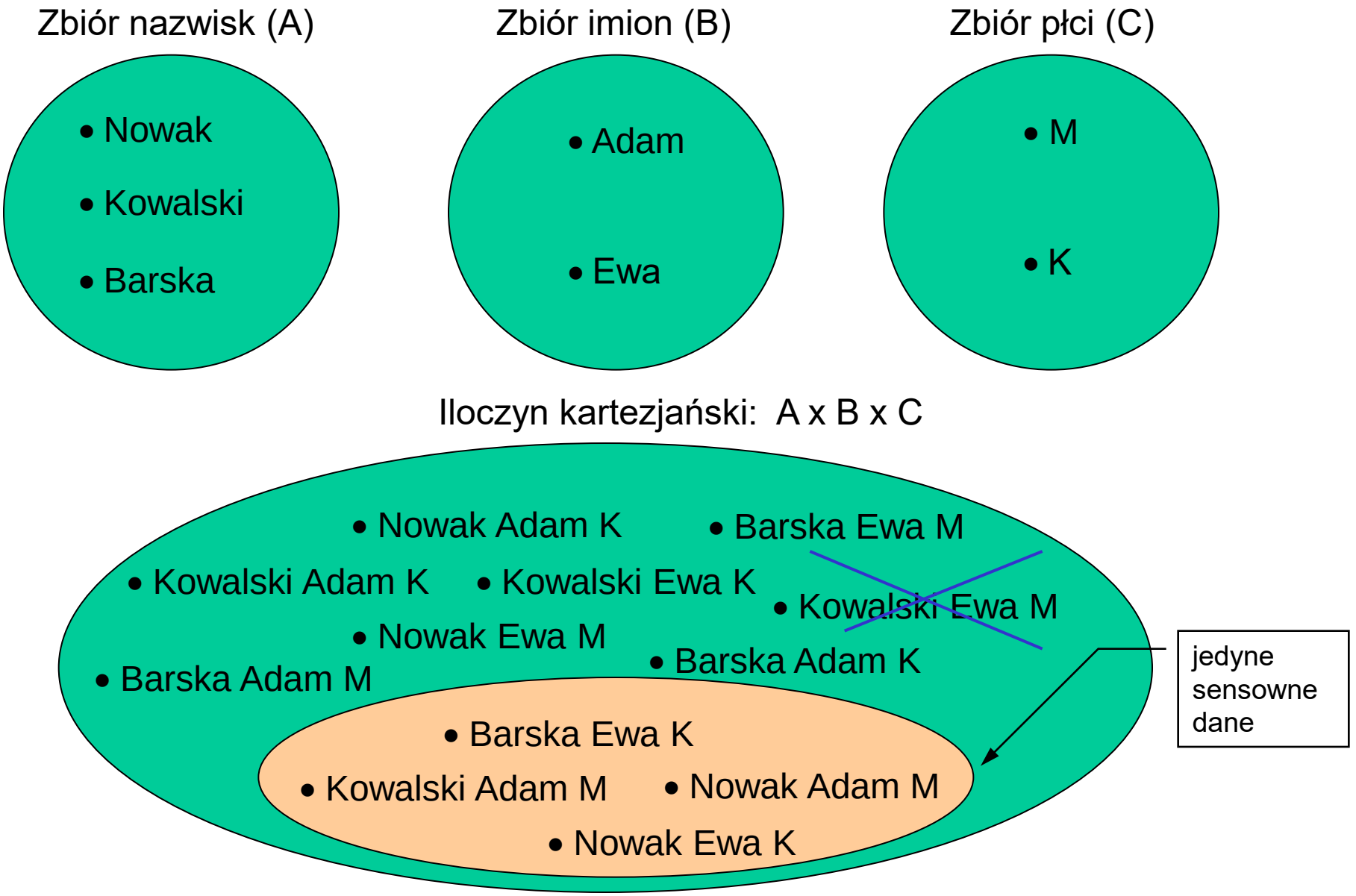
Pojęcie relacji oraz podstawowe operacje na relacjach

Iloczyn kartezjański (1/2)

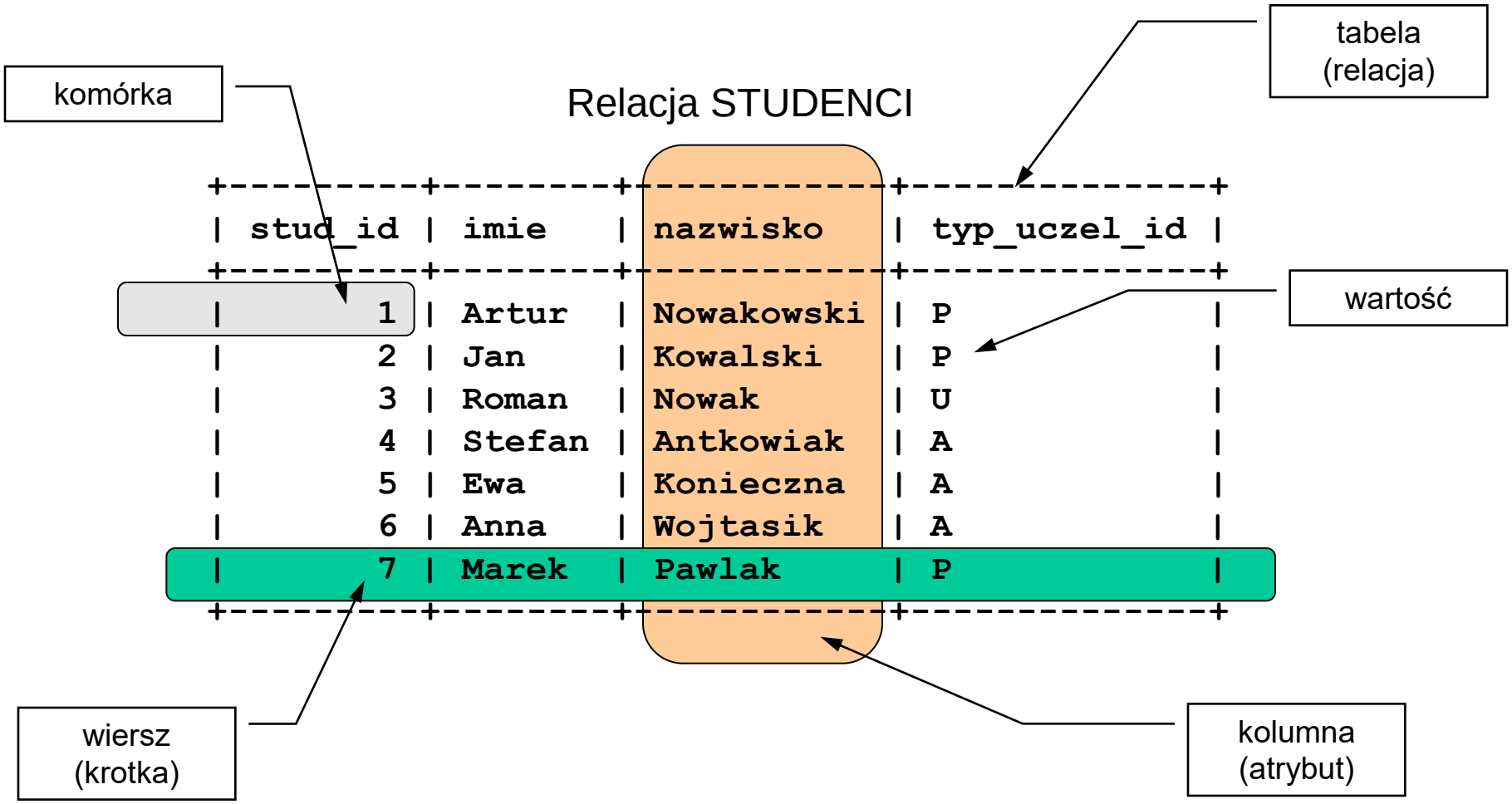


Relacja – **podzbiór** iloczynu kartezjańskiego

Iloczyn kartezjański (2/2)



Przykład relacji



stud_id	imie	nazwisko	typ_uczel_id
1	Artur	Nowakowski	P
2	Jan	Kowalski	P
3	Roman	Nowak	U
4	Stefan	Antkowiak	A
5	Ewa	Konieczna	A
6	Anna	Wojtasik	A
7	Marek	Pawlak	P

Operacje na relacjach - SELEKCJA

stud_id	imie	nazwisko	typ_uczel_id
1	Artur	Nowakowski	P
2	Jan	Kowalski	P
3	Roman	Nowak	U
4	Stefan	Antkowiak	A
5	Ewa	Konieczna	A
6	Anna	Wojtasik	A
7	Marek	Pawlak	P

```
SELECT
  stud_id,
  imie,
  nazwisko,
  typ_uczel
FROM
  studenci
WHERE
  typ_uczel_id="P";
```

lub po prostu:
`SELECT *` zamiast wymieniać kolumny do wyświetlenia

średnik jest ZAWSZE zakończeniem instrukcji SQL
(separatorem nie musi być zawsze średnik, choć w większości przypadków tak właśnie jest)

stud_id	imie	nazwisko	typ_uczel_id
1	Artur	Nowakowski	P
2	Jan	Kowalski	P
7	Marek	Pawlak	P

Operacje na relacjach - PROJEKCJA (RZUT)

stud_id	imie	nazwisko	typ_uczel_id
1	Artur	Nowakowski	P
2	Jan	Kowalski	P
3	Roman	Nowak	U
4	Stefan	Antkowiak	A
5	Ewa	Konieczna	A
6	Anna	Wojtasik	A
7	Marek	Pawlak	P

```
SELECT
  nazwisko, typ_uczel_id
FROM
  studenci;
```

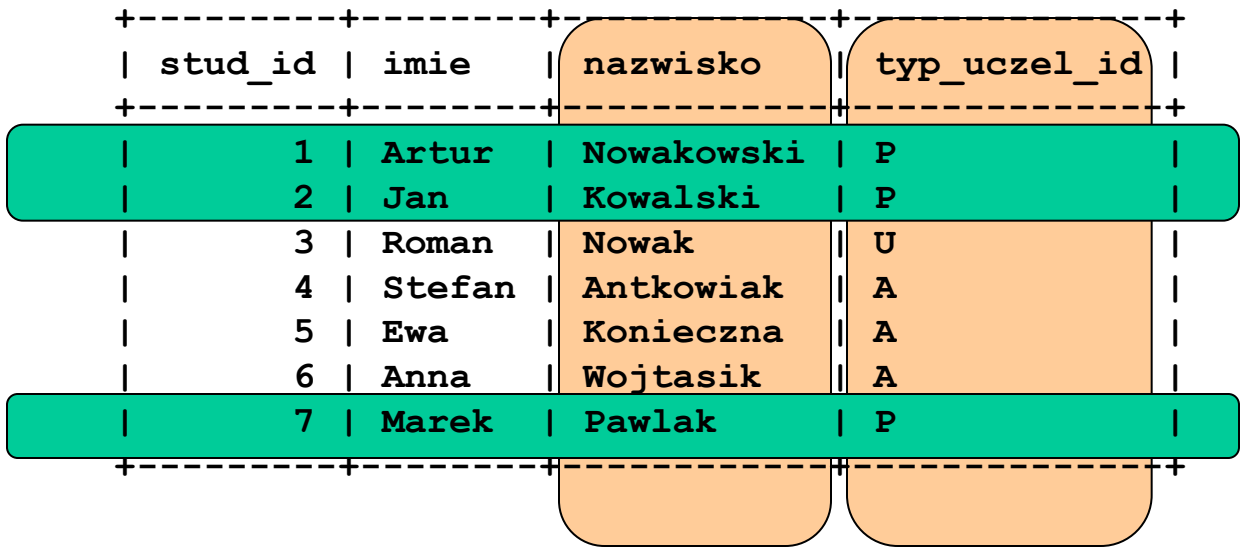
nazwisko	typ_uczel_id
Nowakowski	P
Kowalski	P
Nowak	U
Antkowiak	A
Konieczna	A
Wojtasik	A
Pawlak	P

różna kolejność kolumn

```
SELECT
 typ_uczel_id, nazwisko
FROM
  studenci;
```

typ_uczel_id	nazwisko
P	Nowakowski
P	Kowalski
U	Nowak
A	Antkowiak
A	Konieczna
A	Wojtasik
P	Pawlak

Operacje na relacjach - SELEKCJA oraz PROJEKCJA



stud_id	imie	nazwisko	typ_uczel_id
1	Artur	Nowakowski	P
2	Jan	Kowalski	P
3	Roman	Nowak	U
4	Stefan	Antkowiak	A
5	Ewa	Konieczna	A
6	Anna	Wojtasik	A
7	Marek	Pawlak	P

```
SELECT
  nazwisko, typ_uczel_id
FROM
  studenci
WHERE
  typ_uczel_id="P";
```

nazwisko	typ_uczel_id
Nowakowski	P
Kowalski	P
Pawlak	P

Operacje na relacjach - ZŁĄCZENIA (NATURALNE)

Operacja ta polega na łączeniu rekordów dwóch lub więcej relacji z zastosowaniem określonego warunku łączenia. Wynikiem połączenia jest podzbiór iloczynu kartezjańskiego.

STUDENCI

stud_id	imie	nazwisko	typ_uczel_id
1	Artur	Nowakowski	P
2	Jan	Kowalski	P
3	Roman	Nowak	U
4	Stefan	Antkowiak	A
5	Ewa	Konieczna	A
6	Anna	Wojtasik	A
7	Marek	Pawlak	P

UCZELNIE

typ_uczel_id	nazwa
A	Akademia
P	Politechnika
U	Uniwersytet

imie	nazwisko	nazwa
Artur	Nowakowski	Politechnika
Jan	Kowalski	Politechnika
Roman	Nowak	Uniwersytet
Stefan	Antkowiak	Akademia
Ewa	Konieczna	Akademia
Anna	Wojtasik	Akademia
Marek	Pawlak	Politechnika

nazwy tabel nie są tu absolutnie niezbędne

```
SELECT
  studenci.imie,
  studenci.nazwisko,
  uczelnie.nazwa
FROM
  uczelnie,
  studenci
WHERE
  studenci.typ_uczel_id =
  uczelnie.typ_uczel_id;
```

Operacje na relacjach - ZŁĄCZENIA ZEWNĘTRZNE (1/3)

STUDENCI

stud_id	imie	nazwisko	typ_uczel_id
1	Artur	Nowakowski	P
2	Jan	Kowalski	P
3	Roman	Nowak	NULL
4	Stefan	Antkowiak	NULL
5	Ewa	Konieczna	NULL
6	Anna	Wojtasik	A
7	Marek	Pawlak	P

UCZELNIE

typ_uczel_id	nazwa
A	Akademia
P	Politechnika
U	Uniwersytet

wartość NULL:
„wartość nieznana w tym momencie”

imie	nazwisko	nazwa
Artur	Nowakowski	Politechnika
Jan	Kowalski	Politechnika
Anna	Wojtasik	Akademia
Marek	Pawlak	Politechnika

```
SELECT
  S.imie, S.nazwisko, U.nazwa
FROM
  uczelnie AS U,
  studenci AS S
WHERE
  S.typ_uczel_id = U.typ_uczel_id;
```

brak informacji o 3 pracownikach oraz o tym, że istnieje też Uniwersytet

tu użyliśmy tzw. aliasów. Nie jest to obowiązkowe ale bardzo wygodne

Operacje na relacjach - ZŁĄCZENIA ZEWNĘTRZNE (2/3)

```
SELECT
  S.imie, S.nazwisko, U.nazwa
FROM
  studenci S LEFT OUTER JOIN uczelnie U
ON
  S.typ_uczel_id = U.typ_uczel_id;
```

```
SELECT
  S.imie, S.nazwisko, U.nazwa
FROM
uczelnie U LEFT OUTER JOIN studenci S
ON
 S.typ_uczel_id = U.typ_uczel_id;
```

LUB

```
SELECT
  S.imie, S.nazwisko, U.nazwa
FROM
 uczelnie U RIGHT OUTER JOIN studenci S
ON
  S.typ_uczel_id = U.typ_uczel_id;
```

LUB

```
SELECT
  S.imie, S.nazwisko, U.nazwa
FROM
studenci S RIGHT OUTER JOIN uczelnie U
ON
 S.typ_uczel_id = U.typ_uczel_id;
```

imie	nazwisko	nazwa
Artur	Nowakowski	Politechnika
Jan	Kowalski	Politechnika
Roman	Nowak	NULL
Stefan	Antkowiak	NULL
Ewa	Konieczna	NULL
Anna	Wojtasik	Akademia
Marek	Pawlak	Politechnika

pojawili się studenci, którzy nie są przypisani do żadnego typu uczelni

imie	nazwisko	nazwa
Anna	Wojtasik	Akademia
Artur	Nowakowski	Politechnika
Jan	Kowalski	Politechnika
Marek	Pawlak	Politechnika
NULL	NULL	Uniwersytet

pojawił się typ uczelni, do której nie jest przypisany żaden student

Operacje na relacjach - ZŁĄCZENIA ZEWNĘTRZNE (3/3)

```
SELECT
  S.imie, S.nazwisko, U.nazwa
FROM
  studenci S FULL OUTER JOIN uczelnie U
ON
  S.typ_uczel_id = U.typ_uczel_id;
```

FULL OUTER JOIN = LEFT OUTER JOIN + RIGHT OUTER JOIN

imie	nazwisko	nazwa
Artur	Nowakowski	Politechnika
Jan	Kowalski	Politechnika
Roman	Nowak	NULL
Stefan	Antkowiak	NULL
Ewa	Konieczna	NULL
Anna	Wojtasik	Akademia
Marek	Pawlak	Politechnika
NULL	NULL	Uniwersytet

pojawili się studenci, którzy nie są przypisani do żadnego typu uczelni

pojawił się typ uczelni, do której nie jest przypisany żaden student

Operacje na relacjach - ILOCZYN KARTEZJAŃSKI

```
SELECT
  studenci.imie, studenci.nazwisko, uczelnie.nazwa
FROM
  uczelnie, studenci;
```

imie	nazwisko	nazwa
Artur	Nowakowski	Akademia
Artur	Nowakowski	Politechnika
Artur	Nowakowski	Uniwersytet
Jan	Kowalski	Akademia
Jan	Kowalski	Politechnika
Jan	Kowalski	Uniwersytet
Roman	Nowak	Akademia
Roman	Nowak	Politechnika
Roman	Nowak	Uniwersytet
Stefan	Antkowiak	Akademia
Stefan	Antkowiak	Politechnika
Stefan	Antkowiak	Uniwersytet
Ewa	Konieczna	Akademia
Ewa	Konieczna	Politechnika
Ewa	Konieczna	Uniwersytet
Anna	Wojtasik	Akademia
Anna	Wojtasik	Politechnika
Anna	Wojtasik	Uniwersytet
Marek	Pawlak	Akademia
Marek	Pawlak	Politechnika
Marek	Pawlak	Uniwersytet

UCZELNIE: 3 rekordy
STUDENCI: 7 rekordów
wynik: 3 x 7 = 21 rekordów

stud_id	imie	nazwisko	typ_uczel_id
1	Artur	Nowakowski	P
2	Jan	Kowalski	P
3	Roman	Nowak	U
4	Stefan	Antkowiak	A
5	Ewa	Konieczna	A
6	Anna	Wojtasik	A
7	Marek	Pawlak	P

typ_uczel_id	nazwa
A	Akademia
P	Politechnika
U	Uniwersytet

dużo zdublowanych
(z reguły z reguły bez praktycznego znaczenia!) danych

Operacje na relacjach - GRUPOWANIE

STUDENCI

stud_id	imie	nazwisko	typ_uczel_id
1	Artur	Nowakowski	P
2	Jan	Kowalski	P
3	Roman	Nowak	U
4	Stefan	Antkowiak	A
5	Ewa	Konieczna	A
6	Anna	Wojtasik	A
7	Marek	Pawlak	P

```
SELECT
  COUNT(typ_uczel_id) AS ilosc, typ_uczel_id
FROM
  studenci
GROUP BY
  typ_uczel_id;
```

ilosc	typ_uczel_id
3	A
3	P
1	U

Operacje na relacjach - GRUPOWANIE

STUDENCI

stud_id	imie	nazwisko	typ_uczel_id
1	Artur	Nowakowski	P
2	Jan	Kowalski	P
3	Roman	Nowak	U
4	Stefan	Antkowiak	A
5	Ewa	Konieczna	A
6	Anna	Wojtasik	A
7	Marek	Pawlak	P

```
SELECT
  COUNT(typ_uczel_id) AS ilosc, typ_uczel_id, nazwisko
FROM
  studenci
GROUP BY
  typ_uczel_id;
```

Ta kolumna jest tutaj bez sensu, ale MySQL dopuszcza takie „dziwolągi”

ilosc	typ_uczel_id	nazwisko
3	A	Antkowiak
3	P	Nowakowski
1	U	Nowak

Operacje mnogościowe - SUMA (UNIA)

Unia pozwala na **zsumowanie zbiorów rekordów dwóch lub więcej relacji**. Warunkiem poprawności tej operacji jest zgodność liczby i typów atrybutów (kolumn) sumowanych relacji. Przykład przedstawiony poniżej sumuje zbiory studentów i pracowników okrojone do imienia i nazwiska (za pomocą projekcji), w celu uzyskania informacji o wszystkich osobach powiązanych z uczelnią:

STUDENCI

stud_id	imie	nazwisko	typ_uczel
1	Artur	Nowakowski	P
2	Jan	Kowalski	P
3	Roman	Nowak	U
4	Stefan	Antkowiak	A
5	Ewa	Konieczna	A
6	Anna	Wojtasik	A
7	Marek	Pawlak	P

PRACOWNICY

prac_id	imie	nazwisko	data_zatr
100	Marek	Kowalski	2000-10-10
101	Andrzej	Rychlik	1990-05-22
102	Wojciech	Barski	1995-12-01

```
SELECT imie, nazwisko
FROM studenci
UNION
SELECT imie, nazwisko
FROM pracownicy;
```

uwaga na brak średnika

tu średnik obowiązkowy

imie	nazwisko
Artur	Nowakowski
Jan	Kowalski
Roman	Nowak
Stefan	Antkowiak
Ewa	Konieczna
Anna	Wojtasik
Marek	Pawlak
Marek	Kowalski
Andrzej	Rychlik
Wojciech	Barski

Operacje mnogościowe - SUMA (UNIA)

W bazie MySQL takie coś jest dopuszczalne, choć faktycznie trochę bez sensu

```
SELECT imie, nazwisko
FROM studenci
UNION
SELECT imie, data_zatr
FROM pracownicy;
```

Niezgodność kolumn

STUDENCI

stud_id	imie	nazwisko	typ_uczel
1	Artur	Nowakowski	P
2	Jan	Kowalski	P
3	Roman	Nowak	U
4	Stefan	Antkowiak	A
5	Ewa	Konieczna	A
6	Anna	Wojtasik	A
7	Marek	Pawlak	P

PRACOWNICY

prac_id	imie	nazwisko	data_zatr
100	Marek	Kowalski	2000-10-10
101	Andrzej	Rychlik	1990-05-22
102	Wojciech	Barski	1995-12-01

imie	nazwisko
Artur	Nowakowski
Jan	Kowalski
Roman	Nowak
Stefan	Antkowiak
Ewa	Konieczna
Anna	Wojtasik
Marek	Pawlak
Marek	2000-10-10
Andrzej	1990-05-22
Wojciech	1995-12-01

Operacje mnogościowe - PRZECIĘCIE (PRZEKRÓJ)

Przekrój pozwala znaleźć iloczyn dwóch lub więcej zbiorów relacji tzn. takich, które **występują zarówno w jednej jak i w drugiej relacji**. Podobnie jak w przypadku unii, warunkiem poprawności tej operacji jest zgodność liczby i typów atrybutów relacji bazowych.

STUDENCI

stud_id	imie	nazwisko	typ_uczel
1	Artur	Nowakowski	P
2	Jan	Kowalski	P
3	Roman	Nowak	U
4	Stefan	Antkowiak	A
5	Ewa	Konieczna	A
6	Anna	Wojtasik	A
7	Marek	Pawlak	P

PRACOWNICY

prac_id	imie	nazwisko	data_zatr
100	Marek	Kowalski	2000-10-10
101	Andrzej	Rychlik	1990-05-22
102	Wojciech	Barski	1995-12-01

uwaga na brak średnika

```
SELECT nazwisko
FROM studenci
INTERSECT
SELECT nazwisko
FROM pracownicy;
```

nazwisko
Kowalski

Operacje mnogościowe - RÓŻNICA

Operacja obliczania różnicy dwóch relacji polega na znalezieniu wszystkich rekordów, które **występują w pierwszej relacji, ale nie występują w drugiej**.

STUDENCI

stud_id	imie	nazwisko	typ_uczel
1	Artur	Nowakowski	P
2	Jan	Kowalski	P
3	Roman	Nowak	U
4	Stefan	Antkowiak	A
5	Ewa	Konieczna	A
6	Anna	Wojtasik	A
7	Marek	Pawlak	P

PRACOWNICY

prac_id	imie	nazwisko	data_zatr
100	Marek	Kowalski	2000-10-10
101	Andrzej	Rychlik	1990-05-22
102	Wojciech	Barski	1995-12-01

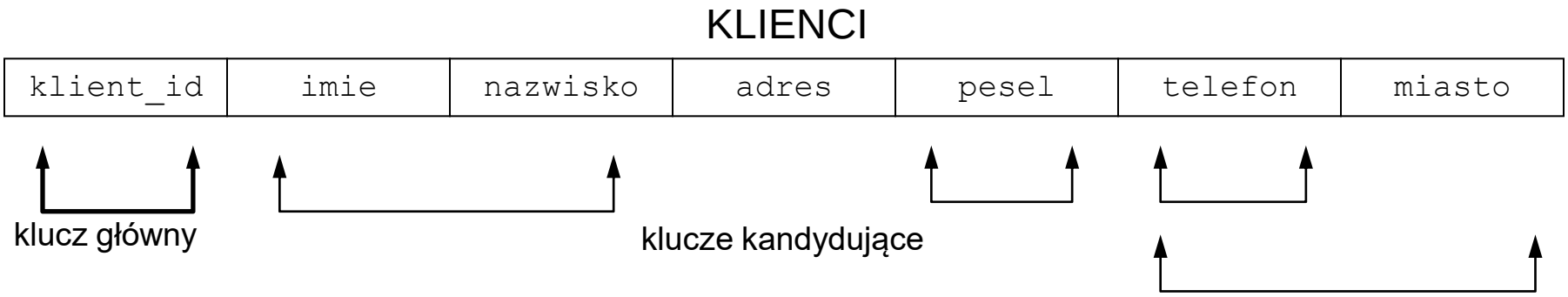
uwaga na brak średnika

```
SELECT nazwisko
FROM studenci
MINUS
SELECT nazwisko
FROM pracownicy;
```

nazwisko
Nowakowski
Nowak
Antkowiak
Konieczna
Wojtasik
Pawlak

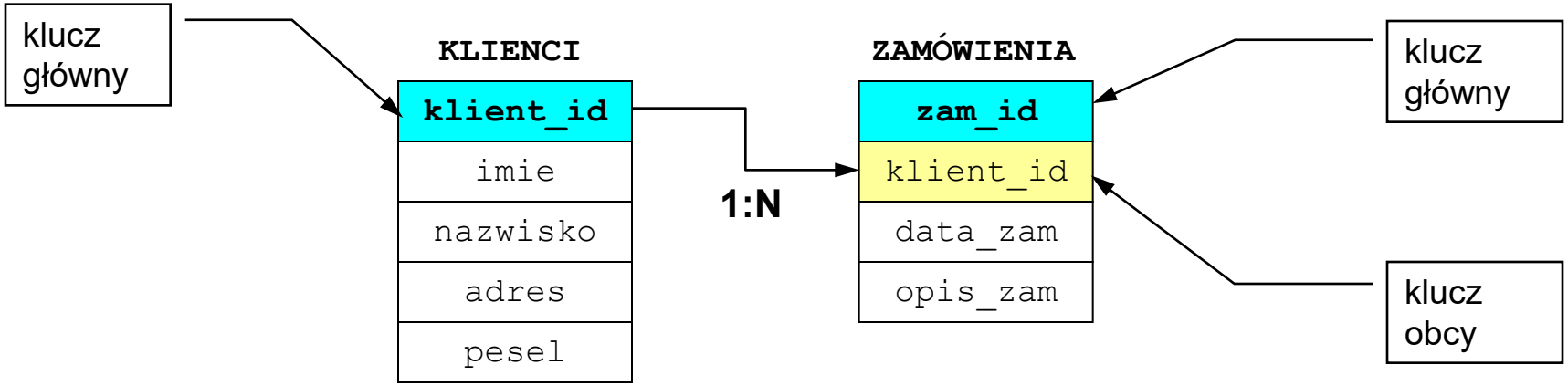
Związki między relacjami (1:1, 1:N, N:M), klucze główne i klucze obce, inne tzw. ograniczenia (ang. *constraints*) bazodanowe

Klucz główny (ang. *primary key*)



- Nadklucz (ang. *superkey*) to kolumna lub zestaw kolumn, która może być używana do jednoznacznego zidentyfikowania rekordu tabeli. Klucz główny to NAJMNIEJSZY nadklucz
- Klucz główny zapewnia unikalność poszczególnych rekordów
- Na danej tabeli można zdefiniować tylko jeden klucz główny (prosty lub złożony)
- W zdecydowanej większości przypadków powinno to być pole **numeryczne całkowite**
- Klucz główny na polu znakowym może wpływać na zmniejszenie wydajności bazy
- Klucz główny może być złożony (więcej niż jedna kolumna), jednak należy unikać zbyt wielu kolumn. Trzy kolumny wydają się być sensownym maksimum
- W powiązaniu z kluczami obcymi (patrz dalej) zapewniają wsparcie dla tzw. ograniczeń integralnościowych (ang. *integrity constraints*)
- W zasadzie każda tabela relacyjna powinna mieć zdefiniowany klucz główny

Klucz obcy (ang. *foreign key*), związki 1:1, 1:N (1/5)



- Klucz obcy reprezentuje związki między tabelami
- Kucz obcy gwarantuje, że bardzo trudno jest wprowadzić niespójność do bazy danych.Baza danych a nie programista i jego aplikacja muszą martwić się o właściwe sprawdzanie integralności bazy
- Poprawnie zaprojektowane reguły klucza obcego ułatwiają dokumentowanie relacji między tabelami
- Stosowanie kluczy obcych zwiększa obciążenie serwera (ale obecnie to żaden problem)
- Stosowanie kluczy obcych może utrudniać odzyskiwanie danych po awariach
- Mimo wszystko stosowanie kluczy obcych jest bardzo zalecane!

takim kolorem oznaczany będzie **klucz obcy**

takim kolorem oznaczany będzie **klucz główny**

Klucz obcy (ang. *foreign key*) , związki 1:1, 1:N (2/5)

KLIENCI

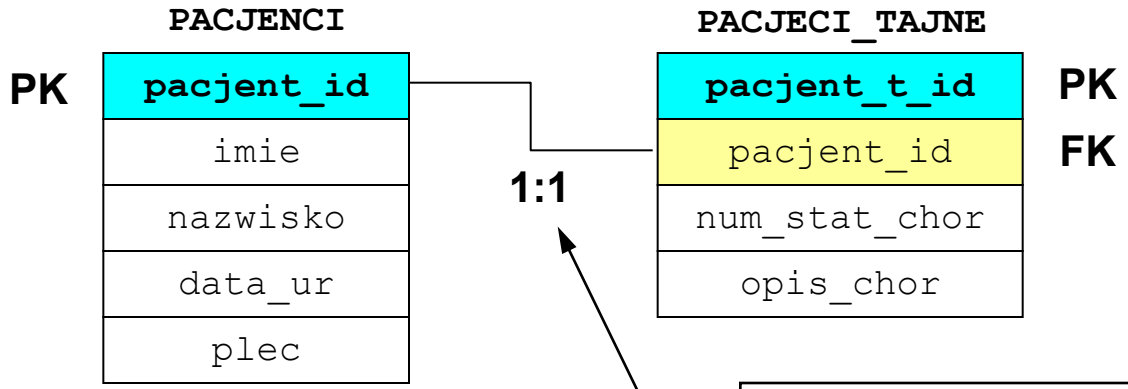
klient_id	imie	nazwisko	adres	pesel
1	Artur	Nowakowski	NULL	26016711223
2	Jan	Kowalski	Zielona Gora	10128012345
3	Roman	Nowak	Sulechow	99999999999

ZAMÓWIENIA

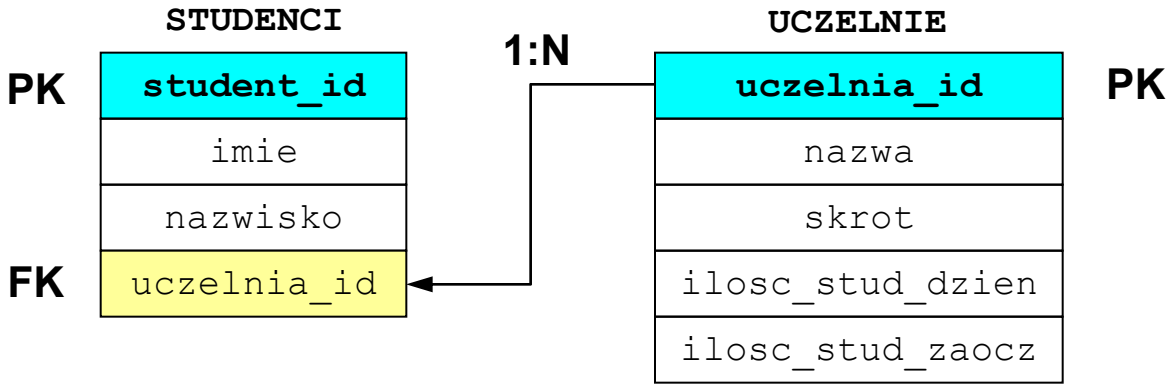
zam_id	klient_id	data_zam	opis_zam
1	1	2005-10-12	NULL
2	1	2004-07-10	NULL
3	1	1999-08-10	NULL
4	2	1998-10-13	NULL
5	3	2001-12-10	NULL
6	3	2005-08-14	NULL

w tej kolumnie NIE MOŻE pojawić się żadna wartość, która NIE WYSTĘPUJE w tabeli nadrzędnej

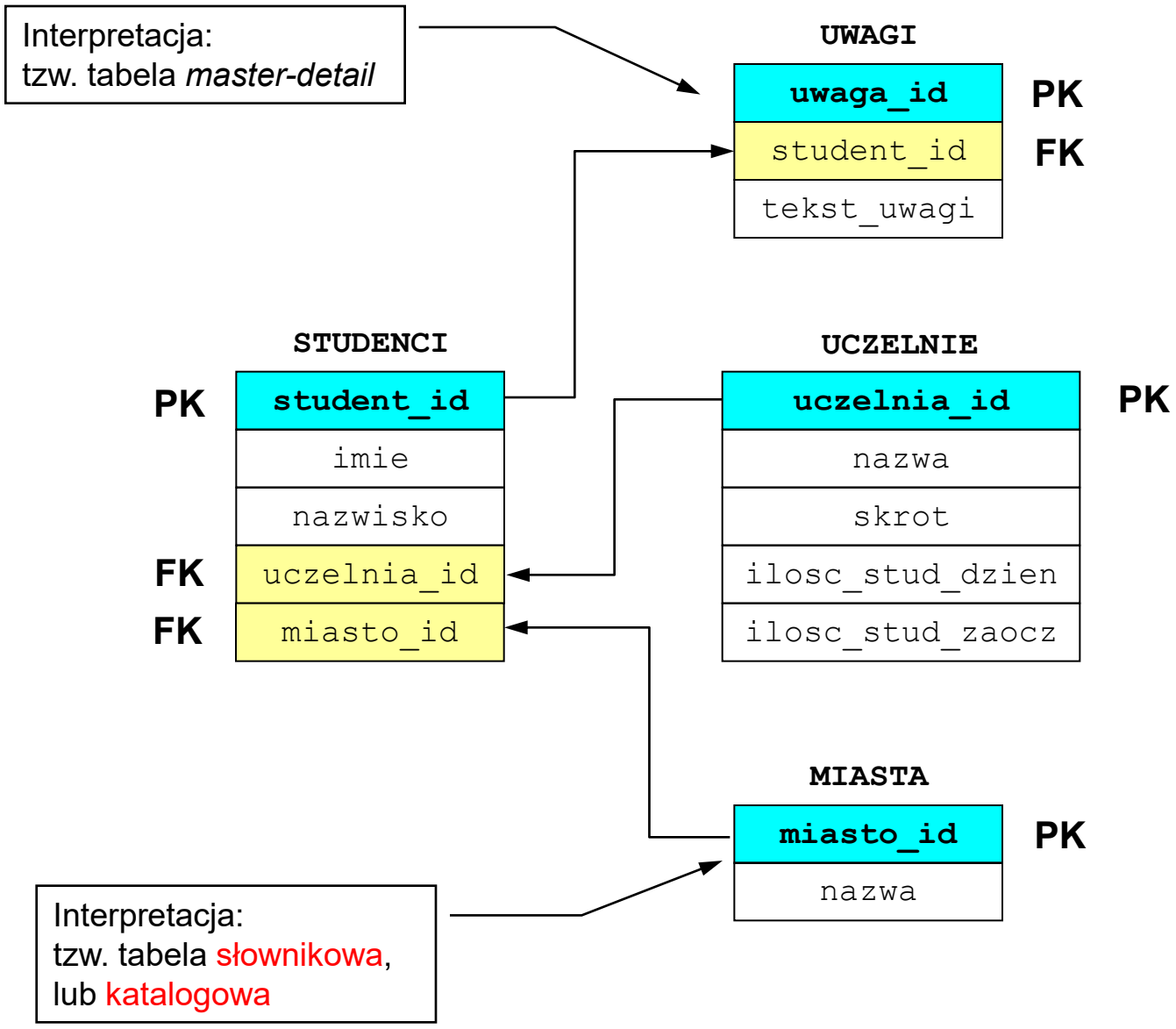
Klucz obcy (ang. *foreign key*) , związki 1:1, 1:N (3/5)



1. wymagane organicznie UNIQUE dla kolumny FK
2. w praktyce zwiazek 1:1 jest rzadko używany (łatwiej użyć tzw. widoków ang. *views* + odpowiednie uprawnienia)



Klucz obcy (ang. *foreign key*) , związki 1:1, 1:N (4/5)

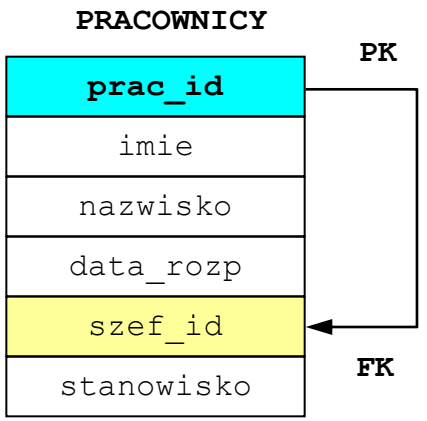


Klucz obcy (ang. *foreign key*) , związki 1:1, 1:N (5/5)

- Modelowanie podległości

tzw. self join

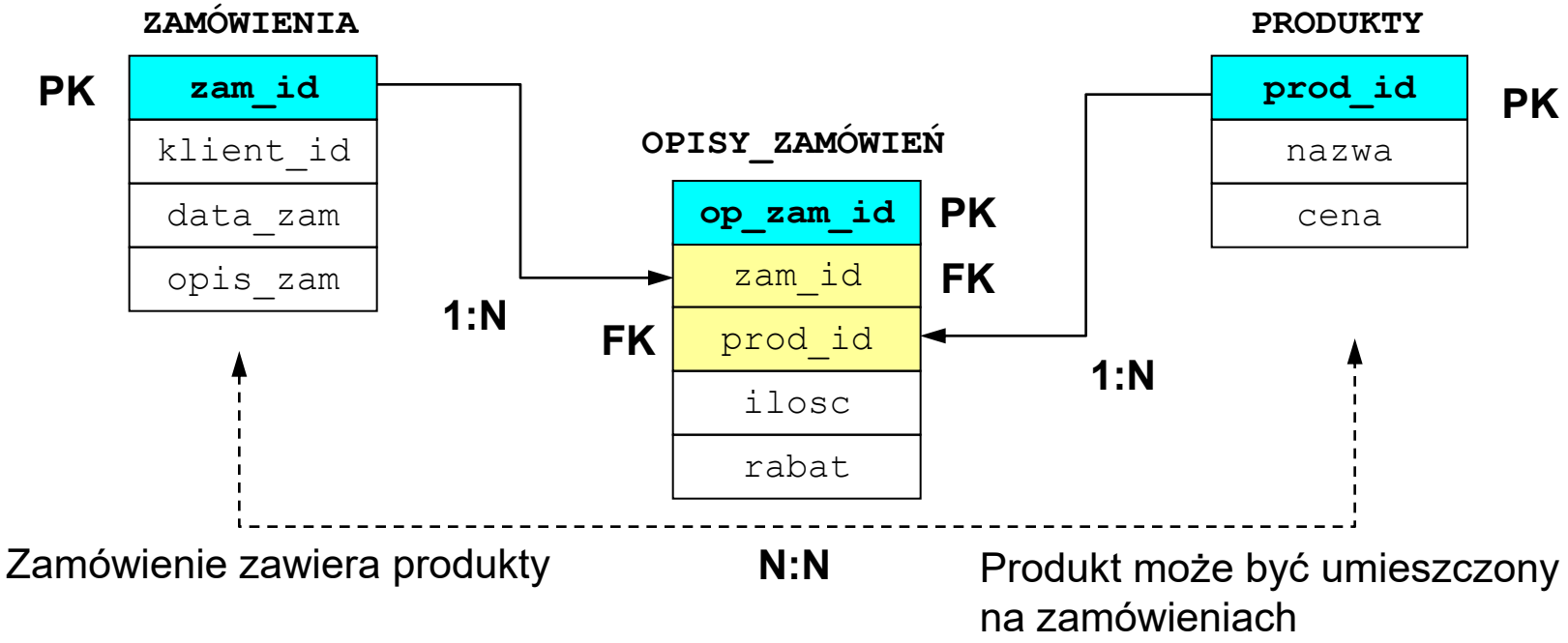
imie nazwisko	stanowisko
Marek Kowalski	**Prezes**
---Wojciech Barski	W-ce Prezes ds. Finansowych
------Edyta Zurawska	Ksiegowa
------Ewa Pytel	Windykator
------Iwona Rutkowska	Ksiegowa
---Andrzej Rychlik	W-ce Prezes ds. Rozwoju
------Jan Kowalski	Analityk
------Wojciech Nowakowski	Projektant
------Andrzej Pawlak	Projektant



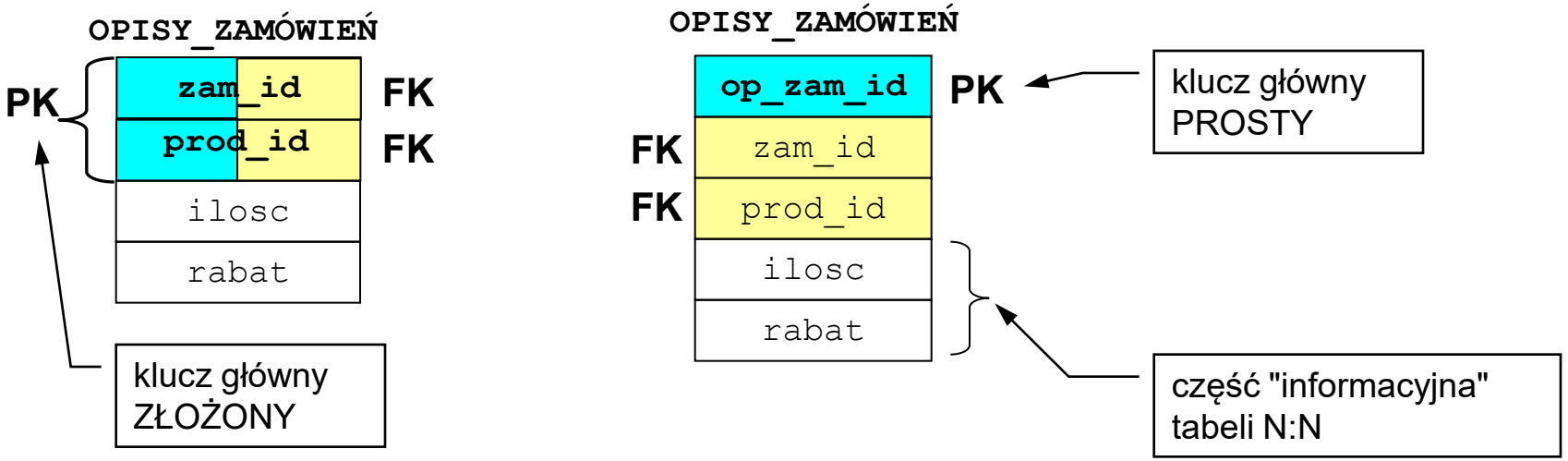
istniejące rozszerzenia SQL pozwalają uwzględniać podległości w wynikach zapytania

prac_id	imie	nazwisko	data_zatr	szef_id	stanowisko
3	Wojciech	Barski	1995-12-01	1	W-ce Prezes ds. Finansowych
1	Marek	Kowalski	2000-10-10	**NULL**	Prezes
6	Jan	Kowalski	1998-12-01	2	Analityk
5	Wojciech	Nowakowski	1995-05-11	2	Projektant
4	Andrzej	Pawlak	2000-10-10	2	Projektant
8	Ewa	Pytel	1997-03-22	3	Windykator
9	Iwona	Rutkowska	1991-05-01	3	Ksiegowa
2	Andrzej	Rychlik	1990-05-22	1	W-ce Prezes ds. Rozwoju
7	Edyta	Zurawska	2005-08-01	3	Ksiegowa

Związki N:M (1/5)



Dwie alternatywne wersje tabeli wiążącej:



Związki N:M (2/5)

Przykładowe dane

zam_id	klient_id	data_zam	opis_zam
1	1	2005-10-12	NULL
2	1	2004-07-10	NULL
3	1	1999-08-10	NULL
4	2	1998-10-13	NULL
5	3	2001-12-10	NULL
6	3	2005-08-14	NULL

op_zam_id	zam_id	prod_id	ilosc	rabat
1	1	10	10	2.50
2	1	20	7	NULL
3	1	60	1	NULL
4	2	10	12	NULL
5	2	50	22	NULL
6	3	10	10	2.00
7	3	20	100	10.00
8	3	50	2	NULL
9	3	60	5	NULL

prod_id	nazwa	cena
10	czekolada	2.40
20	piwo	3.10
30	batonik	0.90
40	chleb	1.45
50	mleko	2.10
60	jajka	0.25

Związki N:M (3/5)

```sql
DROP TABLE IF EXISTS opisy_zamowien;
DROP TABLE IF EXISTS produkty;
DROP TABLE IF EXISTS zamowienia;

CREATE TABLE produkty (
  prod_id int(11) NOT NULL PRIMARY KEY,
  nazwa varchar(50) NOT NULL,
  cena decimal(10,2) NOT NULL
) ENGINE=InnoDB;

CREATE TABLE zamowienia (
  zam_id int(11) NOT NULL PRIMARY KEY,
  klient_id varchar(30) NOT NULL,
  data_zam date NOT NULL,
  opis_zam varchar(100) DEFAULT NULL
) ENGINE=InnoDB;

CREATE TABLE opisy_zamowien (
  op_zam_id int(11) NOT NULL PRIMARY KEY,
  zam_id int(11) NOT NULL,
  prod_id int(11) NOT NULL,
  ilosc int(11) NOT NULL,
  rabat decimal(10,2) DEFAULT NULL
) ENGINE=InnoDB;
```

```sql
INSERT INTO produkty VALUES (10, 'czekolada', 2.40);
INSERT INTO produkty VALUES (20, 'piwo', 3.10);
INSERT INTO produkty VALUES (30, 'batonik', 0.90);
INSERT INTO produkty VALUES (40, 'chleb', 1.45);
INSERT INTO produkty VALUES (50, 'mleko', 2.10);
INSERT INTO produkty VALUES (60, 'jajka', 0.25);

INSERT INTO zamowienia VALUES (1, 'klient A', '2020-01-26', 'opis');
INSERT INTO zamowienia VALUES (2, 'klient B', '2020-02-21', NULL);
INSERT INTO zamowienia VALUES (3, 'klient C', '2019-07-15', NULL);
INSERT INTO zamowienia VALUES (4, 'klient D', '2018-01-26', NULL);
INSERT INTO zamowienia VALUES (5, 'klient E', '2017-11-01', NULL);
INSERT INTO zamowienia VALUES (6, 'klient F', '2016-02-02', NULL);

INSERT INTO opisy_zamowien VALUES (1001, 1, 10, 10, 2.50);
INSERT INTO opisy_zamowien VALUES (1002, 1, 20, 7, NULL);
INSERT INTO opisy_zamowien VALUES (1003, 1, 60, 1, NULL);
INSERT INTO opisy_zamowien VALUES (1004, 2, 10, 12, NULL);
INSERT INTO opisy_zamowien VALUES (1005, 2, 50, 22, NULL);
INSERT INTO opisy_zamowien VALUES (1006, 3, 10, 10, 2.01);
INSERT INTO opisy_zamowien VALUES (1007, 3, 20, 100, 10.00);
INSERT INTO opisy_zamowien VALUES (1008, 3, 50, 2, NULL);
INSERT INTO opisy_zamowien VALUES (1009, 3, 60, 5, NULL);
```

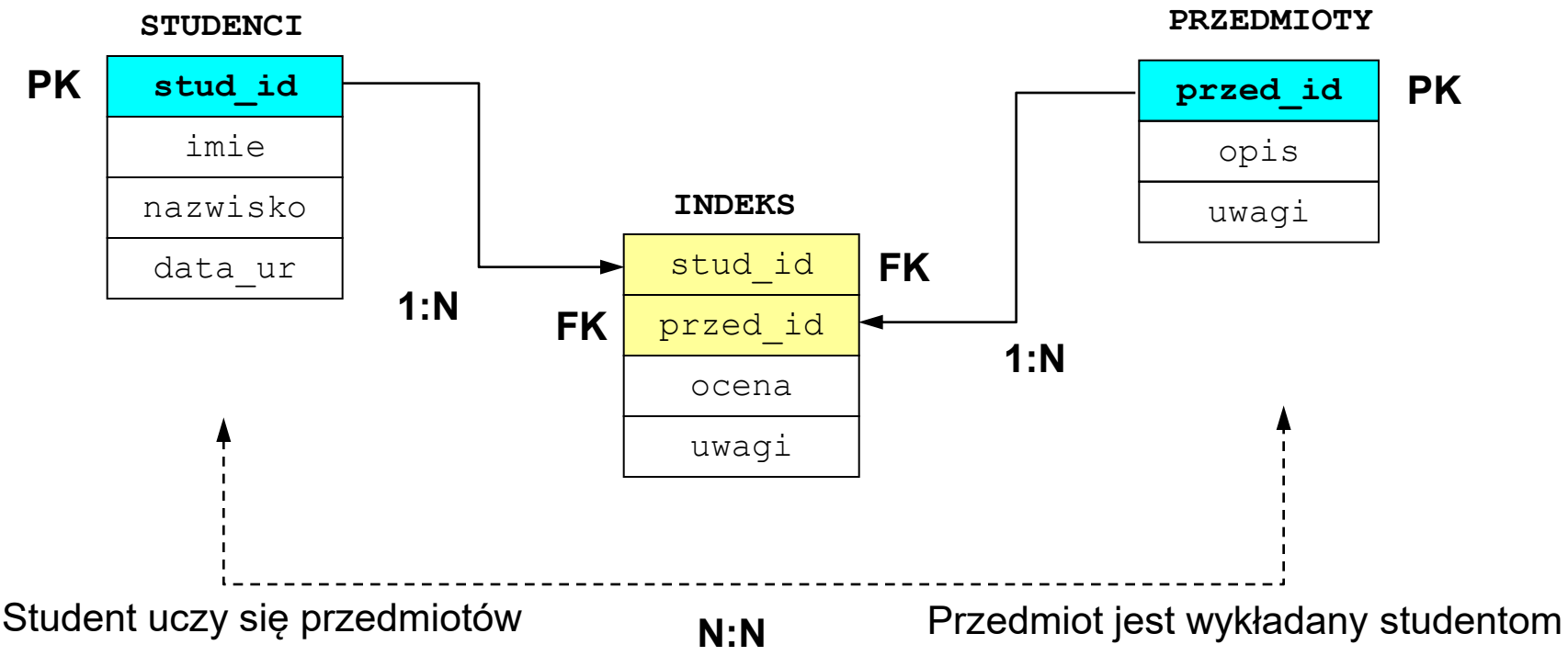
```sql
ALTER TABLE opisy_zamowien
   ADD CONSTRAINT prod_opisy_zam_fk
   FOREIGN KEY (prod_id) REFERENCES produkty (prod_id);

ALTER TABLE opisy_zamowien
   ADD CONSTRAINT zam_opisy_zam_fk
   FOREIGN KEY (zam_id) REFERENCES zamowienia (zam_id);
```

Związki N:M (4/5)

```
SELECT
  Z.zam_id,
  Z.klient_id,
  Z.data_zam,
  P.nazwa,
  P.cena,
  OZ.ilosc,
  OZ.rabat
FROM
  zamowienia AS Z,
  produkty AS P,
  opisy_zamowien AS OZ
WHERE
  Z.zam_id = OZ.zam_id AND
  P.prod_id = OZ.prod_id AND
  Z.zam_id = 1;
```

```
+--------+-----------+------------+-----------+------+-------+-------+
| zam_id | klient_id | data_zam   | nazwa     | cena | ilosc | rabat |
+--------+-----------+------------+-----------+------+-------+-------+
|      1 | klient A  | 2020-01-26 | czekolada | 2.40 |    10 |  2.50 |
|      1 | klient A  | 2020-01-26 | piwo      | 3.10 |     7 |  NULL |
|      1 | klient A  | 2020-01-26 | jajka     | 0.25 |     1 |  NULL |
+--------+-----------+------------+-----------+------+-------+-------+
```

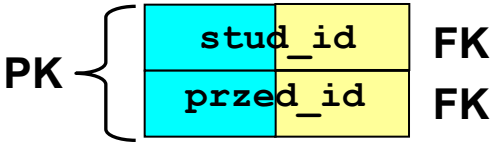
Związki N:M (5/5)

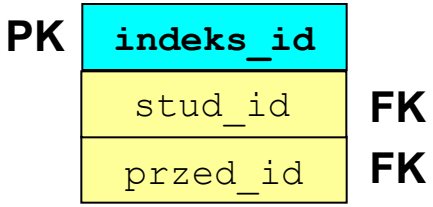


Jak ustawić klucz główny?

Rozważyć 2 alternatywne rozwiązania:

- klucz złożony (kolumny `stud_id` oraz `przed_id`)
- klucz prosty (dodatkowa kolumna, np. `pl_zaj_id`)

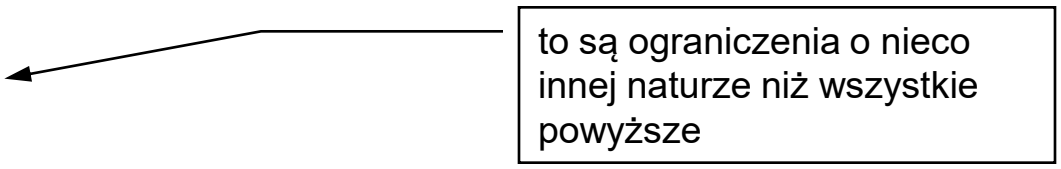




Ograniczenie (ang. constraints) bazodanowe (1/3)

- Ograniczenia pozwalają nakładać na kolumny (lub listę kolumn) dodatkowe warunki (inne niż wynikające tylko z samego typu kolumn(y)
- Ograniczenia pozwalają na wprowadzenie już na poziome bazy danych pewnej kontroli nad poprawnością wprowadzanych danych
- Dzięki temu programista jest zwolniony z samodzielnego wykonywania tej kontroli na poziomie aplikacji - zwiększa się dzięki temu poziom jej bezpieczeństwa
- Każda „dobra” baza danych wspiera typowe ograniczenia ale też definiuje swoje specyficzne
- Rodzaje ograniczeń (na przykładzie bazy MySQL)
  - → NOT NULL
  - → PRIMARY KEY (lista kolumn)
  - → DEFAULT wartość
  - → FOREIGN KEY (inaczej: REFERENCES)
  - → UNIQUE (lista kolumn)
  - → CHECK (warunek)
  - → ENUM (lista wartości)
  - → SET (lista wartości)

  - → INDEX (lista kolumn)
  - → AUTO_INCREMENT

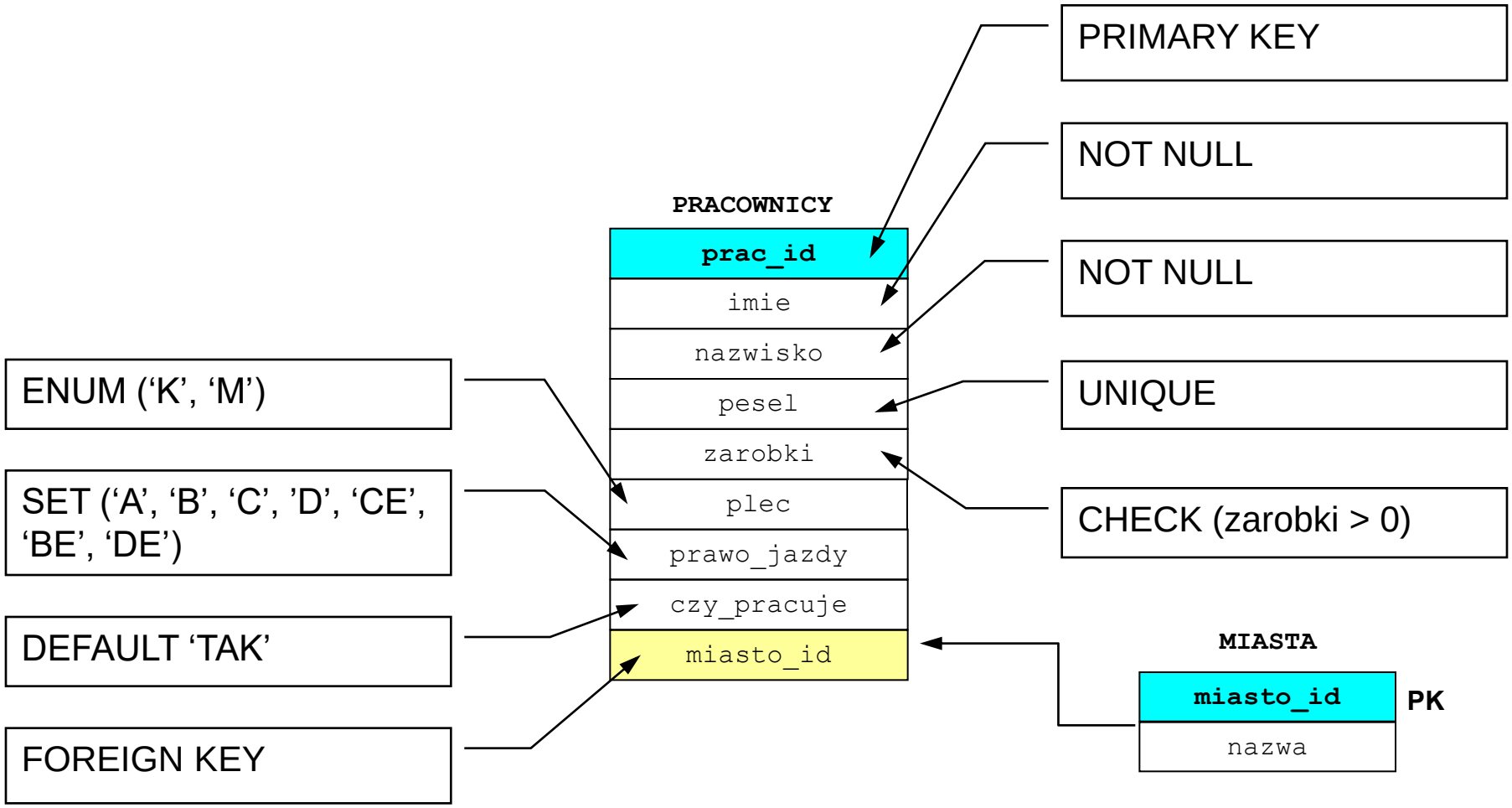


Ograniczenie (ang. constraints) bazodanowe (2/3)

- **NOT NULL** - w kolumnie nie można zapisywać wartości NULL („wartość nieznana w tym momencie”)
- **PRIMARY KEY** - każda tabela może zawierać tylko jedno takie ograniczenie. Może być zdefiniowane na poziomie jednej kolumnie (tzw. ograniczenie kolumnowe) lub na więcej niż jednej kolumnie (tzw. ograniczenie tablicowe). Zapewnia, że wszystkie wpisane wartości są unikalne i różne od NULL
- **DEFAULT** - określa domyślną wartość używaną podczas wstawiania danych w przypadku, gdy nie została jawnie podana żadna wartość dla kolumny
- **FOREIGN KEY** (**REFERENCES**) - zapewnia tzw. integralność referencyjną. Zapobiega wstawianiu błędnych rekordów w tabelach podrzędnych (po stronie „N”)
- **UNIQUE** - Zapewnia, że wszystkie wpisane wartości są unikalne. Od ograniczenia PRIMARY KEY różni się tym, że dopuszcza wpisywanie wartości NULL
- **CHECK** - pozwala na wpisywanie tylko takich wartości, które spełniają określone warunki (np. „zarobki>0”)
- **ENUM** - pozwala na wpisanie tylko jednej wartości z wcześniej zdefiniowanego zbioru
- **SET** - pozwala na wpisanie jednej lub wielu wartości z wcześniej zdefiniowanego zbioru

- **INDEX** - tworzy indeks dla podanej kolumny lub kolumn. Prawidłowo zaprojektowane indeksy w wielu przypadkach zwiększają szybkość działania bazy danych
- **AUTO_INCREMENT** - powoduje, że wartość zapisywana w kolumnie jest automatycznie powiększana (dotyczy tylko typu numerycznego całkowitego)

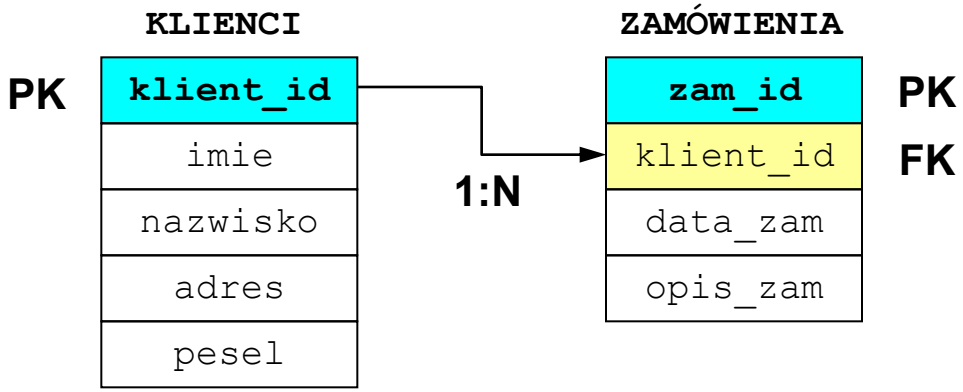
Ograniczenie (ang. constraints) bazodanowe (3/3)



Uwaga: na jednej kolumnie może być założonych kilka ograniczeń. Nic więc nie stoi na przeszkodzie, aby kolumna `pesel` miała np. trzy ograniczenia: UNIQUE, NOT NULL oraz CHECK

Różne uwagi, propozycje, sugestie (1/2)

- Nazwy tabel pisać w liczbie mnogiej a nazwy kolumn w pojedynczej
- Nazwy tabel i kolumn nie powinny być zbyt długie (maksymalnie 12-15 znaków). Można rozważyć używanie języka angielskiego (jest bardziej „skondensowany” niż język polski)
- Stosować sensowne skróty. Np. zamiast `data_zlozenia_zamowienia` lepiej będzie kolumnę nazwać po prostu `data_zam`
- Używanie w nazwach tabel i kolumn znaków podkreślnika bardzo ułatwia ich czytanie. Porównajmy: `datazam` oraz `data_zam`
- Mimo, iż niektóre systemy bazodanowe dopuszczają stosowanie w nazwach tabel i kolumn znaków narodowych (np. polskich) odradzamy taką praktykę
- Kolumnom wchodzącym w skład kluczy głównych i obcych nadawać nazwy „z ID na końcu”. Np. `uczen_id, prod_id` (ale raczej nie `id_ucznia, id_produktu`).
- Nazwy kolumn będących kluczami obcymi powinny być takie same jak odpowiadające im kolumny kluczy głównych. Bardzo ułatwia to pracę! (patrz rysunek poniżej)



Różne uwagi, propozycje, sugestie (2/2)

- Strzałka wskazuje stronę „wiele” (klucz obcy). Niektórzy projektanci stosują jednak odwrotną konwencję, tzn. strzałką oznaczana jest strona „jeden”. Jest to sprawa czysto umowna i trudno powiedzieć, która jest lepsza
- Zamiast strzałek można używać innych symboli (patrz niżej uwagi na temat modelowania związków encji)
- Starać się używać liter o jednakowej wielkości. Np. nazwy tabel pisane DUŻYMI LITERAMI a nazwy kolumn małymi

- Wydaje się, że ważniejsze od przyjętych konwencji jest ich **KONSEKWENTNE** stosowanie! Nie ma nic gorszego niż stosowanie wielu różnych (często przeciwstawnych) konwencji w ramach jednego projektu

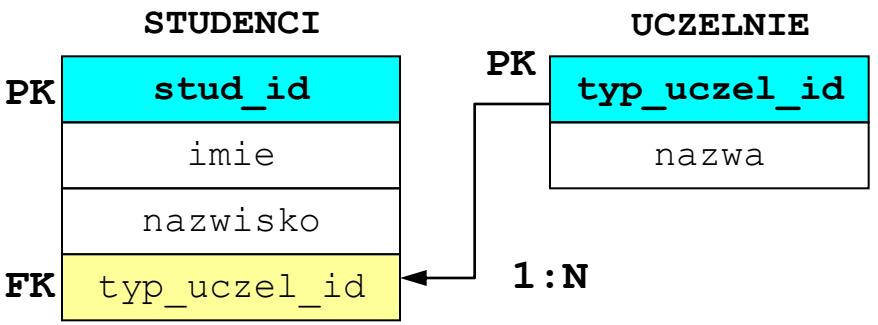
Tworzenie kluczy obcych w MySQL (skrót) (1/9)

STUDENCI

stud_id	imie	nazwisko	typ_uczel_id
1	Artur	Nowakowski	P
2	Jan	Kowalski	P
3	Roman	Nowak	U
4	Stefan	Antkowiak	A
5	Ewa	Konieczna	A
6	Anna	Wojtasik	A
7	Marek	Pawlak	P

UCZELNIE

typ_uczel_id	nazwa
A	Akademia
P	Politechnika
U	Uniwersytet



gdy jest PRIMARY KEY to NOT NULL jest w zasadzie zbędne

```
CREATE TABLE studenci (
   stud_id        INT           NOT NULL PRIMARY KEY,
   imie           VARCHAR(20)   NOT NULL,
   nazwisko       VARCHAR(30)   NOT NULL,
   typ_uczel_id   CHAR(1)
) ENGINE = InnoDB;
```

brak przecinka

obecnie tylko ten typ tabel wspiera klucze obce

LUB

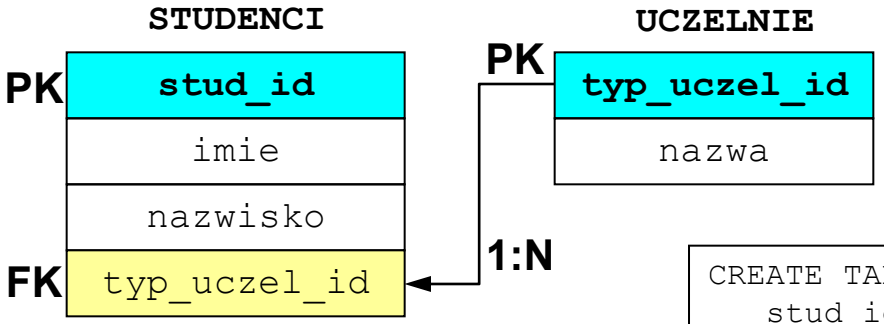
```
CREATE TABLE studenci (
   stud_id        INT           NOT NULL,
   imie           VARCHAR(20)   NOT NULL,
   nazwisko       VARCHAR(30)   NOT NULL,
   typ_uczel_id   CHAR(1),
   CONSTRAINT studenci stud id pk PRIMARY KEY (stud_id)
) ENGINE = InnoDB;
```

tu jest przecinek

LUB po prostu:
PRIMARY KEY (stud_id)

Podkreślony fragment jest opcjonalny

Tworzenie kluczy obcych w MySQL (skrót) (2/9)



krok 1

```
CREATE TABLE studenci (
   stud_id         INT           NOT NULL PRIMARY KEY,
   imie            VARCHAR(20)   NOT NULL,
   nazwisko        VARCHAR(30)   NOT NULL,
   typ_uczel_id    CHAR(1)
) ENGINE = InnoDB;
```

krok 2

```
CREATE TABLE uczelnie (
   typ_uczel_id    CHAR(1)       NOT NULL PRIMARY KEY,
   nazwa           VARCHAR(20)   NOT NULL
) ENGINE = InnoDB;
```

krok 3

zmodyfikuj tabelę `studenci` ...

... dodając do niej ograniczenie o nazwie `studenci_typ_uczel_id_fk` ...

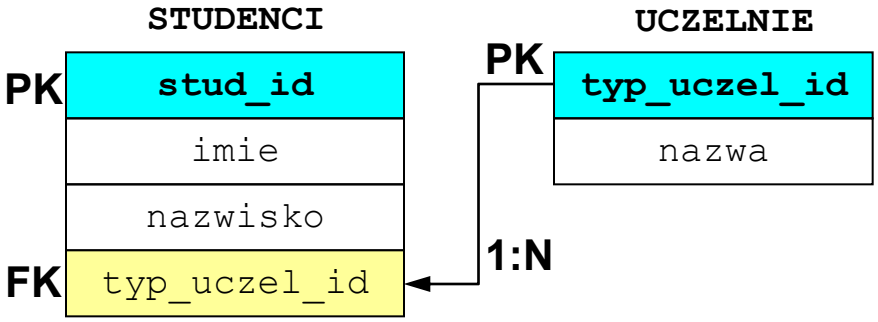
```
ALTER TABLE studenci ADD CONSTRAINT studenci_typ_uczel_id_fk
FOREIGN KEY (typ_uczel_id) REFERENCES uczelnie (typ_uczel_id);
```

... jest to ograniczenie typu klucz obcy ...

... założone na kolumnie `typ_uczel_id` ...

... odnoszące się do klucza głównego `typ_uczel_id` w tabeli `uczelnie`.

Tworzenie kluczy obcych w MySQL (skrót) (3/9)



Uwagi dotyczące stosowania kluczy obcych

- Obie tabele muszą być typu InnoDB
- Na kolumnie FK musi być założony indeks (od wersji 4.1.2, czyli bardzo już leciwej, jest on tworzony automatycznie)
- Klucz obcy musi odwoływać się do klucza PRIMARY KEY lub UNIQUE (zalecamy jednak stosowanie wyłącznie PRIMARY KEY)
- Odpowiadające sobie kolumny PK oraz FK muszą być tego samego typu
- Zalecamy, aby nazwy odpowiadających sobie kolumn PK oraz FK były takie same

Tworzenie kluczy obcych w MySQL (skrót) (4/9)

Można też tak, ale jest to mniej wygodne rozwiązanie:

```
DROP TABLE IF EXISTS studenci;
DROP TABLE IF EXISTS uczelnie;
```

nigdy nie zaszkodzi

```
CREATE TABLE uczelnie (
   typ_uczel_id   CHAR(1)      NOT NULL PRIMARY KEY,
   nazwa          VARCHAR(20)  NOT NULL
) ENGINE = InnoDB;
```

krok 1

tabele muszą być tworzone w ściśle określonej kolejności ! Najpierw UCZELNIE a potem STUDENCI

```
CREATE TABLE studenci (
   stud_id        INT          NOT NULL PRIMARY KEY,
   imie           VARCHAR(20)  NOT NULL,
   nazwisko       VARCHAR(30)  NOT NULL,
   typ_uczel_id   CHAR(1),
     FOREIGN KEY (typ_uczel_id)
     REFERENCES uczelnie (typ_uczel_id)
) ENGINE = InnoDB;
```

krok 2 + 3

LUB, gdy ograniczeniu chcemy nadać nazwę:

```
CREATE TABLE studenci (
   stud_id        INT          NOT NULL PRIMARY KEY,
   imie           VARCHAR(20)  NOT NULL,
   nazwisko       VARCHAR(30)  NOT NULL,
   typ_uczel_id   CHAR(1),
     CONSTRAINT studenci_typ_uczel_id_fk
     FOREIGN KEY (typ_uczel_id)
     REFERENCES uczelnie (typ_uczel_id)
) ENGINE = InnoDB;
```

Tworzenie kluczy obcych w MySQL (skrót) (5/9)

Składnia definicji kluczy obcych:

```
[CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
REFERENCES tbl_name (index_col_name, ...)
[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

definiuje zachowanie się danych w tabeli podrzędnej w przypadku zmiany lub wykasowania danych w tabeli nadrzędnej

- ON DELETE ... – akcja podejmowana przy próbie **wykasowania** rekordów w tabeli nadrzędnej
- ON UPDATE ... – akcja podejmowana przy próbie **modyfikacji** rekordów w tabeli nadrzędnej

- RESTRICT – nie można wykasować ani zmienić rekordów w tabeli nadrzędnej, gdy istnieją powiązane rekordy w tabeli podrzędnej
- NO ACTION – to samo co RESTRICT. Obie opcje są przyjmowane jako domyślne
- SET NULL – po wykasowaniu lub modyfikacji rekordu w tabeli nadrzędnej, w tabeli podrzędnej ustawiane są wartości NULL (uwaga: nie zadziała, gdy kolumna FK będzie miała zdefiniowane ograniczenie NOT NULL)
- CASCADE – automatycznie kasuje (lub modyfikuje) wszystkie rekordy powiązane w tabeli podrzędnej. **Opcja bardzo niebezpieczna! Stosować z rozwagą!**
- SET DEFAULT – w obecnej wersji serwera (5.0.x) opcja ta jest analizowana ale ignorowana

Tworzenie kluczy obcych w MySQL (skrót) (6/9)

Pozostała składnia:

```
ALTER TABLE yourtablename
ADD [CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
REFERENCES tbl_name (index_col_name, ...)
[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

```
ALTER TABLE yourtablename DROP FOREIGN KEY fk_symbol;
```

```
mysql> SHOW CREATE TABLE studenci;
+----------+-------------------------------------------------------------------------------
| Table    | Create Table
+----------+-------------------------------------------------------------------------------
| studenci | CREATE TABLE `studenci` (
  `stud_id` int(11) NOT NULL,
  `imie` varchar(20) NOT NULL,
  `nazwisko` varchar(30) NOT NULL,
  `typ_uczel_id` char(1) default NULL,
  PRIMARY KEY  (`stud_id`),
  KEY `studenci_typ_uczel_id_fk` (`typ_uczel_id`),
  CONSTRAINT `studenci_typ_uczel_id_fk` FOREIGN KEY (`typ_uczel_id`)
              REFERENCES `uczelnie` (`typ_uczel_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+----------+-------------------------------------------------------------------------------
```

Od wersji 4.1.2 ten indeks tworzony jest automatycznie. We wcześniejszych wersjach trzeba go było tworzyć ręcznie

Tworzenie kluczy obcych w MySQL (skrót) (7/9)

Wyświetlanie danych o ograniczeniach (na przykładzie schematu demonstracyjnego):

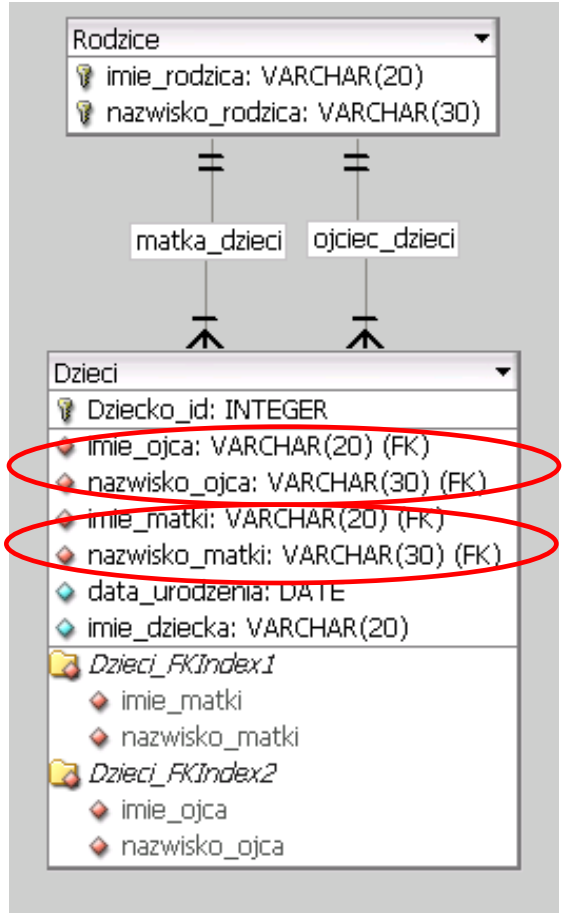
```
SELECT constraint_name, table_schema, table_name, constraint_type
FROM information_schema.table_constraints
WHERE table_schema = 'blab' ORDER BY table_name;
+--------------------------+--------------+------------+-----------------+
| constraint_name          | table_schema | table_name | constraint_type |
+--------------------------+--------------+------------+-----------------+
| customer_region_id_fk    | blab         | customer   | FOREIGN KEY     |
| sales_rep_id_fk          | blab         | customer   | FOREIGN KEY     |
| PRIMARY                  | blab         | customer   | PRIMARY KEY     |
| PRIMARY                  | blab         | dept       | PRIMARY KEY     |
| dept_region_id_fk        | blab         | dept       | FOREIGN KEY     |
| emp_manager_id_fk        | blab         | emp        | FOREIGN KEY     |
| emp_dept_id_fk           | blab         | emp        | FOREIGN KEY     |
| emp_title_fk             | blab         | emp        | FOREIGN KEY     |
| PRIMARY                  | blab         | emp        | PRIMARY KEY     |
| inventory_product_id_fk  | blab         | inventory  | FOREIGN KEY     |
| inventory_warehouse_id_fk| blab         | inventory  | FOREIGN KEY     |
| PRIMARY                  | blab         | inventory  | PRIMARY KEY     |
| item_product_id_fk       | blab         | item       | FOREIGN KEY     |
| PRIMARY                  | blab         | item       | PRIMARY KEY     |
| item_ord_id_fk           | blab         | item       | FOREIGN KEY     |
| ord_customer_id_fk       | blab         | ord        | FOREIGN KEY     |
| ord_sales_rep_id_fk      | blab         | ord        | FOREIGN KEY     |
| PRIMARY                  | blab         | ord        | PRIMARY KEY     |
| PRIMARY                  | blab         | product    | PRIMARY KEY     |
| PRIMARY                  | blab         | region     | PRIMARY KEY     |
| PRIMARY                  | blab         | title      | PRIMARY KEY     |
| warehouse_manager_id_fk  | blab         | warehouse  | FOREIGN KEY     |
| warehouse_region_id_fk   | blab         | warehouse  | FOREIGN KEY     |
| PRIMARY                  | blab         | warehouse  | PRIMARY KEY     |
+--------------------------+--------------+------------+-----------------+
```

Korzystamy tutaj ze specjalnej „systemowej" bazy danych o nazwie *information schema*. W bazie tej przechowywane s¡ różne informacje na temat innych baz. Jest to więc swego rodzaju *baza metadanych*. Pojawiła się ona w wersji 5 serwera MySQL

Ograniczenia PRIMARY KEY nie mają własnych nazw. Ich jawne definiowanie (aczkolwiek możliwe) jest więc bezcelowe

Tworzenie kluczy obcych w MySQL (skrót) (8/9)

Tworzenie kluczy złożonych




```
DROP TABLE IF EXISTS dzieci;
DROP TABLE IF EXISTS rodzice;

CREATE TABLE rodzice (
  imie_rodzica     VARCHAR(20) NOT NULL,
  nazwisko_rodzica VARCHAR(30) NOT NULL,
  PRIMARY KEY (imie_rodzica, nazwisko_rodzica)
) ENGINE = InnoDB;

CREATE TABLE dzieci (
  dziecko_id     INTEGER     NOT NULL,
  imie_ojca      VARCHAR(20)     NULL,
  nazwisko_ojca  VARCHAR(30)     NULL,
  imie_matki     VARCHAR(20)     NULL,
  nazwisko_matki VARCHAR(30)     NULL,
  data_urodzenia DATE        NOT NULL,
  imie_dziecka   VARCHAR(20) NOT NULL,
  PRIMARY KEY (dziecko_id)
) ENGINE = InnoDB;


ALTER TABLE dzieci ADD CONSTRAINT rodzice_dzieci_ojciec_fk
FOREIGN KEY (imie_ojca, nazwisko_ojca)
REFERENCES rodzice (imie_rodzica, nazwisko_rodzica);

ALTER TABLE dzieci ADD CONSTRAINT rodzice_dzieci_matka_fk
FOREIGN KEY (imie_matki, nazwisko_matki)
REFERENCES rodzice (imie_rodzica, nazwisko_rodzica);
```

```
+--------------------------+--------------+------------+-----------------+
| constraint_name          | table_schema | table_name | constraint_type |
+--------------------------+--------------+------------+-----------------+
| rodzice_dzieci_ojciec_fk | artur        | dzieci     | FOREIGN KEY     |
| rodzice_dzieci_matka_fk  | artur        | dzieci     | FOREIGN KEY     |
| PRIMARY                  | artur        | dzieci     | PRIMARY KEY     |
| PRIMARY                  | artur        | rodzice    | PRIMARY KEY     |
+--------------------------+--------------+------------+-----------------+
```

Tworzenie kluczy obcych w MySQL (skrót) (9/9)

Gdy zrezygnujemy z jawnego nazywania ograniczeń, system sam wygeneruje swoje unikalne własne (nazwy będą jednak mniej ‘samodokumentujące się’)

```
 ...

ALTER TABLE dzieci ADD CONSTRAINT rodzice_dzieci_ojciec_fk
FOREIGN KEY (imie_ojca, nazwisko_ojca)
REFERENCES rodzice (imie_rodzica, nazwisko_rodzica);

ALTER TABLE dzieci ADD CONSTRAINT rodzice_dzieci_matka_fk
FOREIGN KEY (imie_matki, nazwisko_matki)
REFERENCES rodzice (imie_rodzica, nazwisko_rodzica);
```

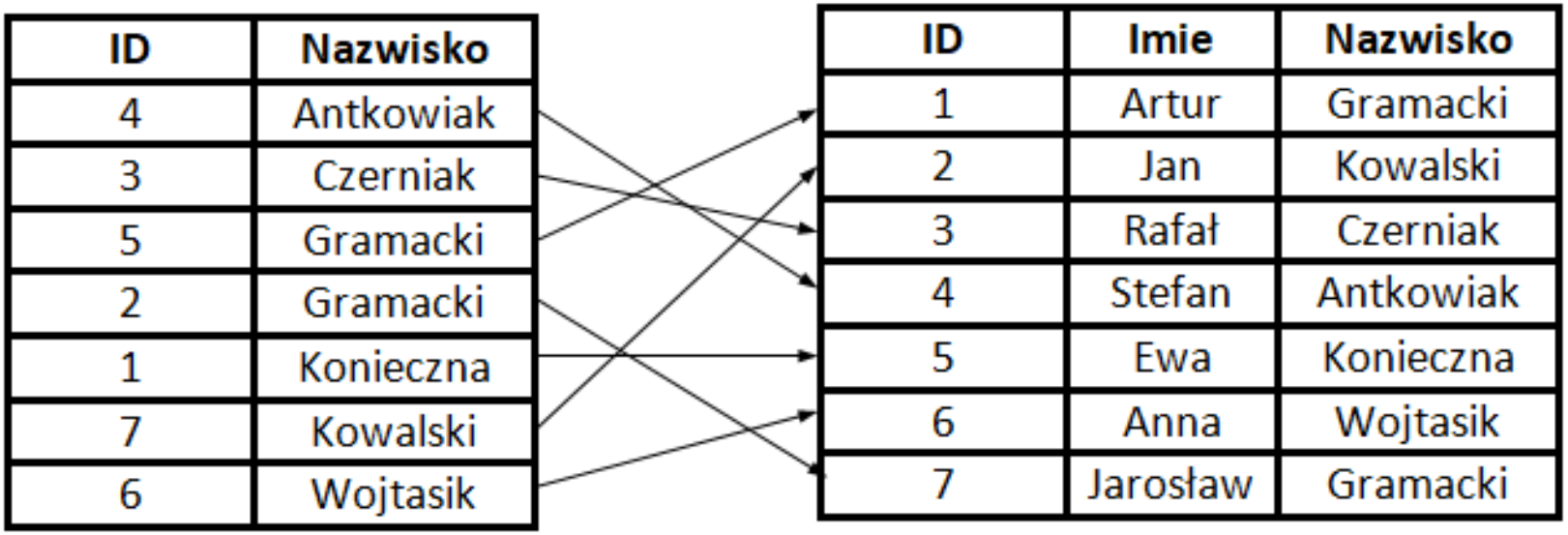
Wówczas:

```
+-----------------+--------------+------------+-----------------+
| constraint_name | table_schema | table_name | constraint_type |
+-----------------+--------------+------------+-----------------+
| dzieci_ibfk_2   | artur        | dzieci     | FOREIGN KEY     |
| dzieci_ibfk_1   | artur        | dzieci     | FOREIGN KEY     |
| PRIMARY         | artur        | dzieci     | PRIMARY KEY     |
| PRIMARY         | artur        | rodzice    | PRIMARY KEY     |
+-----------------+--------------+------------+-----------------+
```

Wygenerowane automatycznie przez serwer

Indeksy bazodanowe

- Indeksy to struktury danych na dysku umożliwiające szybki dostęp do rekordów bez konieczności skanowania całej tabeli. Indeks można wyobrazić sobie jako posortowaną kopię indeksowanej kolumny, jak na rysunku poniżej



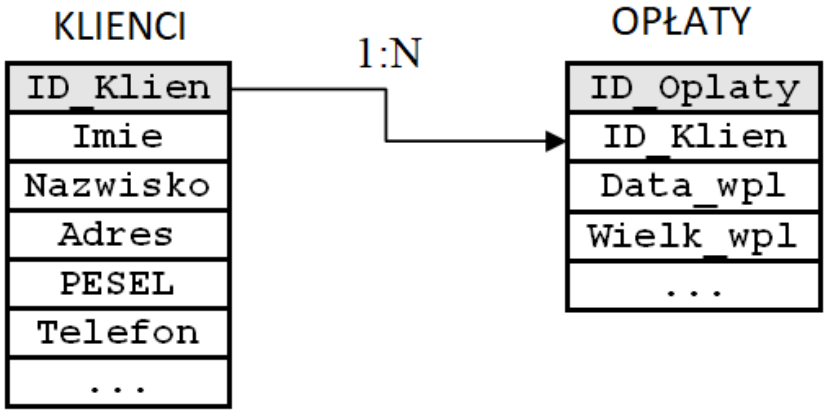
ID	Nazwisko
4	Antkowiak
3	Czerniak
5	Gramacki
2	Gramacki
1	Konieczna
7	Kowalski
6	Wojtasik

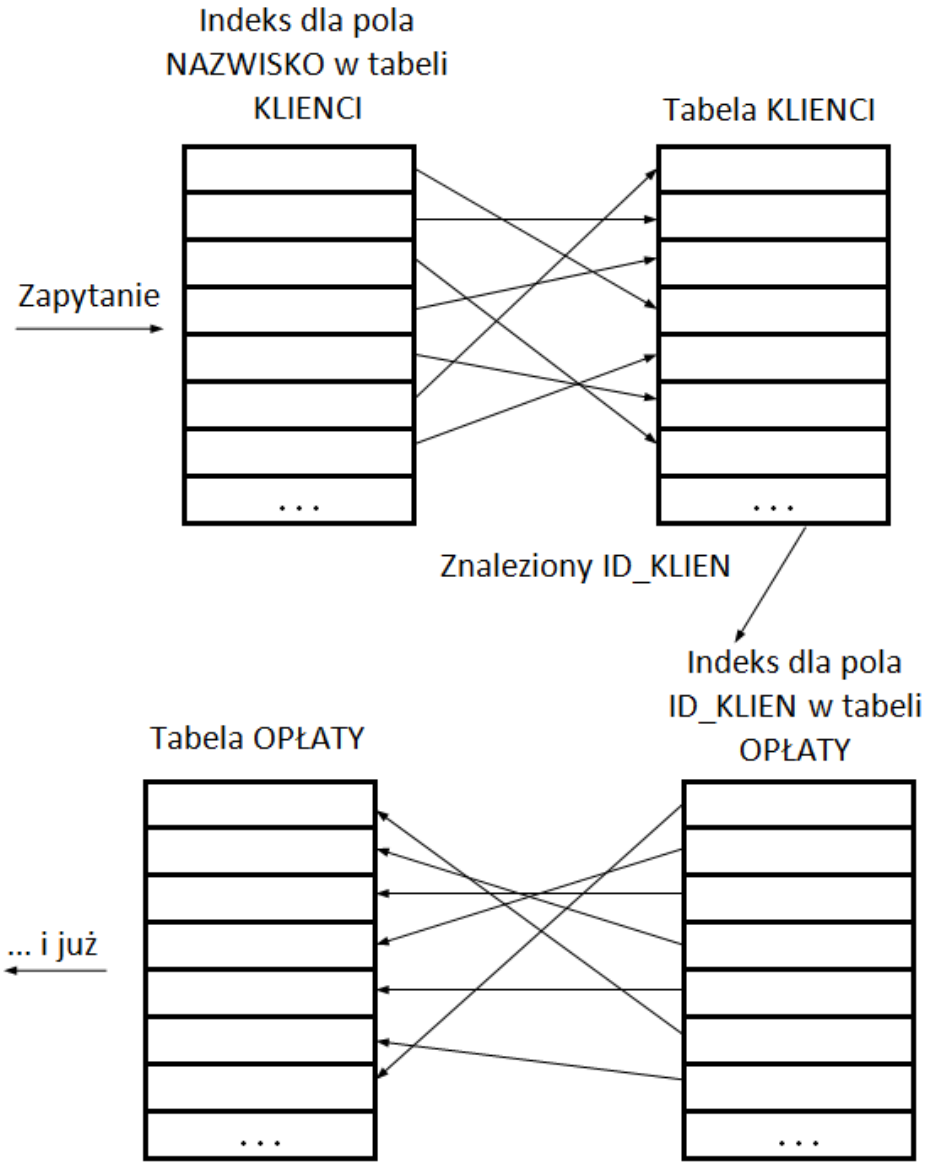
ID	Imie	Nazwisko
1	Artur	Gramacki
2	Jan	Kowalski
3	Rafał	Czerniak
4	Stefan	Antkowiak
5	Ewa	Konieczna
6	Anna	Wojtasik
7	Jarosław	Gramacki

- Indeks w bazie danych można porównać do skorowidza (zwanego często właśnie indeksu) w książce

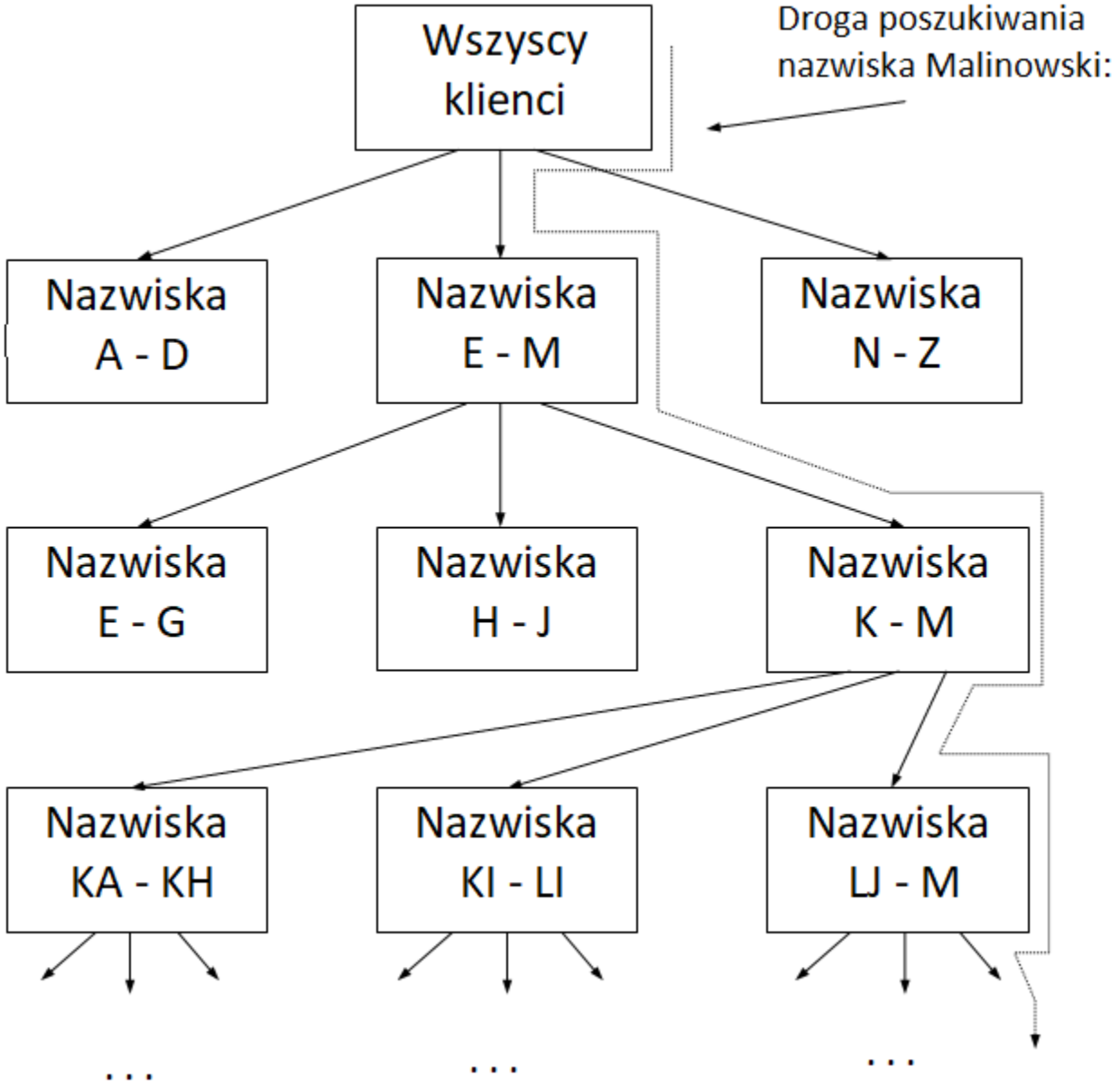
Indeksy bazodanowe

- Schemat postępowania polegający na uzyskaniu dostępu do tabel za pomocą dwóch indeksów
- Chcemy wypisać w porządku alfabetycznym informacje o opłatach wnoszonych przez klientów:



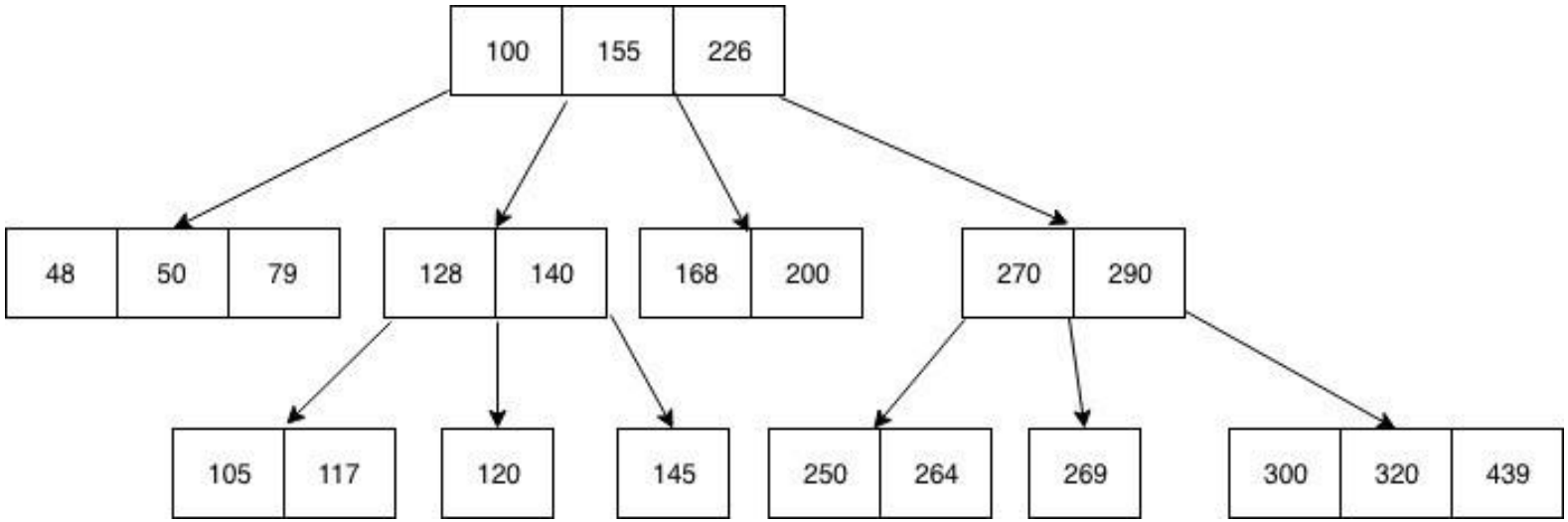


Indeksy bazodanowe



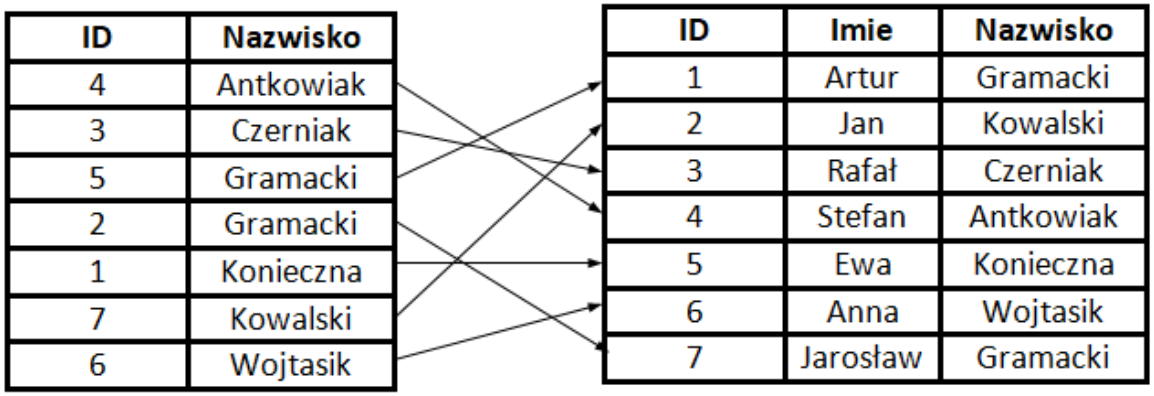
Indeksy bazodanowe

- Indeks implementowany jest najczęściej w postaci B-drzewa
  - są różne warianty B-drzew, np. B+Tree, R-Tree, R*Tree, prefix B+Tree i inne



Indeksy bazodanowe

- Nieroztropnie założone indeksy (np. ich zbyt duża liczba lub założone na niewłaściwych kolumnach) mogą negatywnie wpłynąć na działanie serwera, a w konsekwencji też na aplikacje
  - po pierwsze dlatego, że obsługa indeksów (zwykle chodzi o ich uaktualnianie) zajmuje określony czas i dla dużej ilości wprowadzanych/modyfikowanych/kasowanych danych czas ten może być znaczący
  - po drugie, indeksy zajmują określone zasoby dyskowe i może się zdarzyć, że gdy jest ich zbyt dużo zajmują one na dysku wielokrotnie więcej miejsca niż indeksowane dane!
  - trzeba więc starać się znaleźć złoty środek, co nie zawsze jest proste i oczywiste. Często właściwe rozwiązanie wymaga eksperymentowania

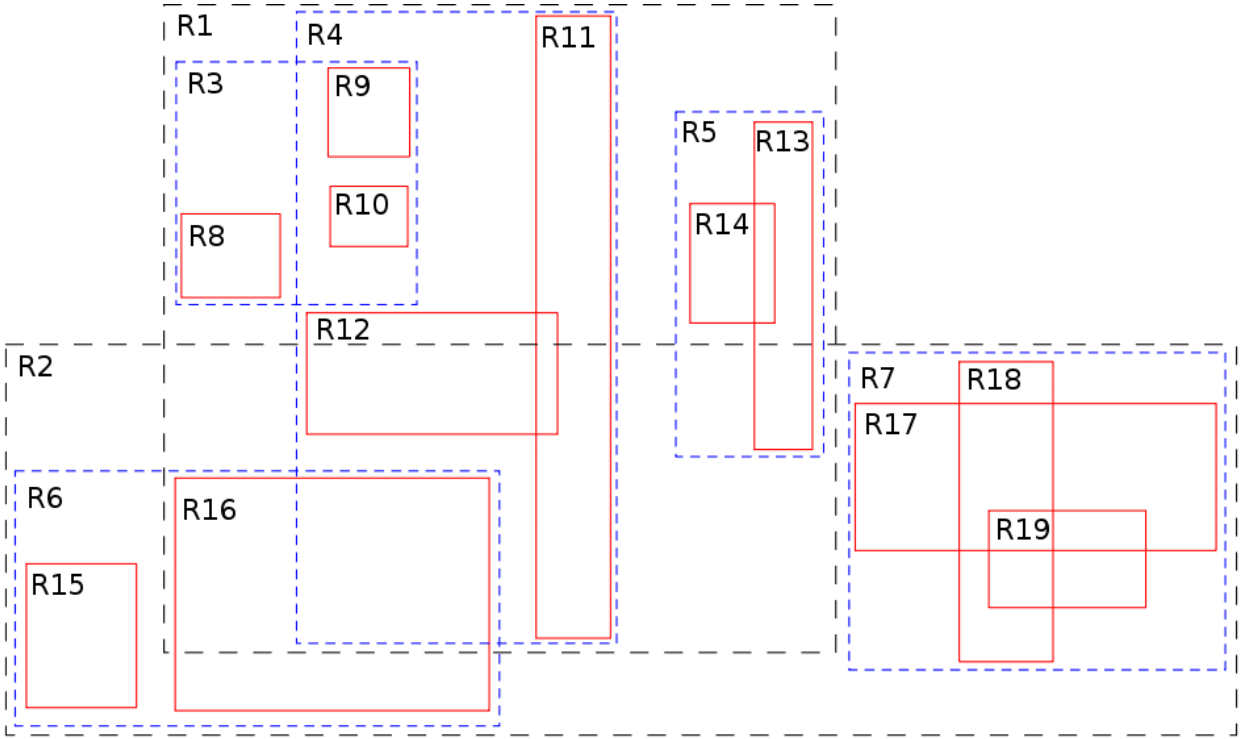


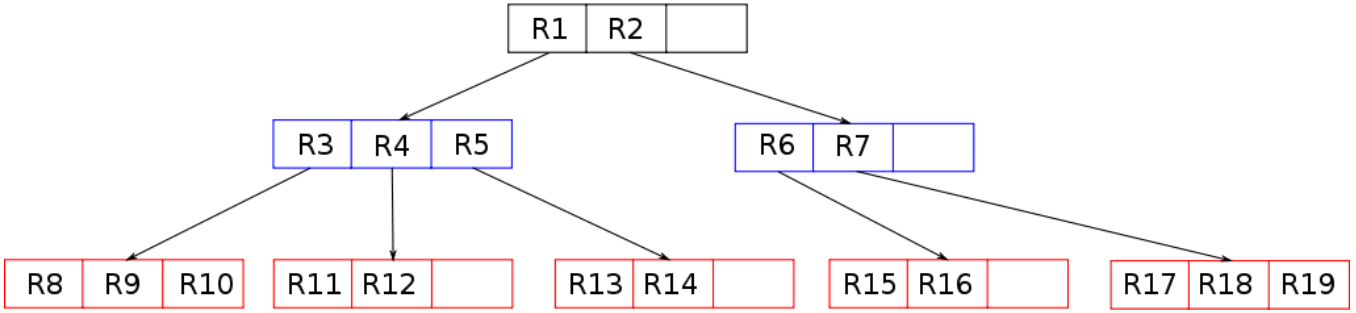
ID	Nazwisko
4	Antkowiak
3	Czerniak
5	Gramacki
2	Gramacki
1	Konieczna
7	Kowalski
6	Wojtasik

ID	Imie	Nazwisko
1	Artur	Gramacki
2	Jan	Kowalski
3	Rafał	Czerniak
4	Stefan	Antkowiak
5	Ewa	Konieczna
6	Anna	Wojtasik
7	Jarosław	Gramacki

Indeksy bazodanowe

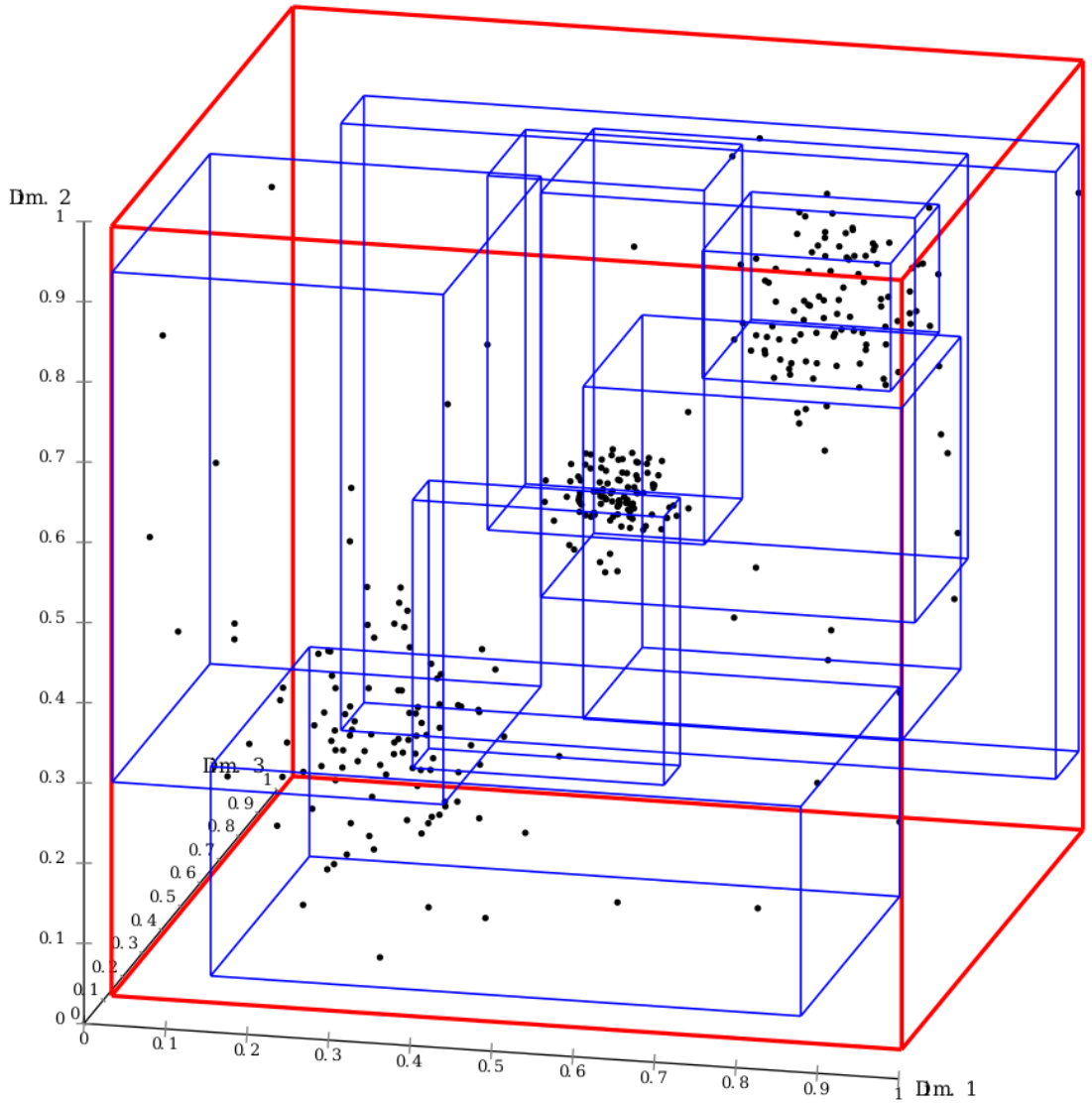
- Przykład indeksu przestrzennego





Indeksy bazodanowe

- Przykład indeksu 3D



Indeksy bazodanowe

- Jak to się robi w MySQL

```
CREATE TABLE indeksy (
  id  INT PRIMARY KEY AUTO_INCREMENT,
  x   INT,
  y   INT,
  z   INT
);
```

```
CREATE INDEX idx_x ON indeksy (x);
```

```
CREATE INDEX idx_y ON indeksy (y);
CREATE INDEX idx_z ON indeksy (z);
CREATE INDEX idx_xy ON indeksy (x, y);
CREATE INDEX idx_xz ON indeksy (x, z);
CREATE INDEX idx_yz ON indeksy (y, z);
CREATE INDEX idx_xyz ON indeksy (x, y, z);
```

```
ALTER TABLE indeksy DROP INDEX idx_x;
ALTER TABLE indeksy DROP INDEX idx_y;
ALTER TABLE indeksy DROP INDEX idx_z;
ALTER TABLE indeksy DROP INDEX idx_xy;
ALTER TABLE indeksy DROP INDEX idx_xz;
ALTER TABLE indeksy DROP INDEX idx_yz;
ALTER TABLE indeksy DROP INDEX idx_xyz;
```

Przykładowy projekt - założenia

Stworzyć strukturę relacyjnej bazy danych dla pewnej szkoły,która będzie umożliwiała:
Rejestrowanie uczniów tej szkoły wg. założeń:

- każdy uczeń należy do jednej (i tylko jednej) klasy a w danej klasie jest z reguły więcej niż jeden uczeń
- musi istnieć możliwość wpisania paru słów krótko opisujących daną klasę (np.: ”klasa bardzo liczna z przewagą chłopców” itp.)

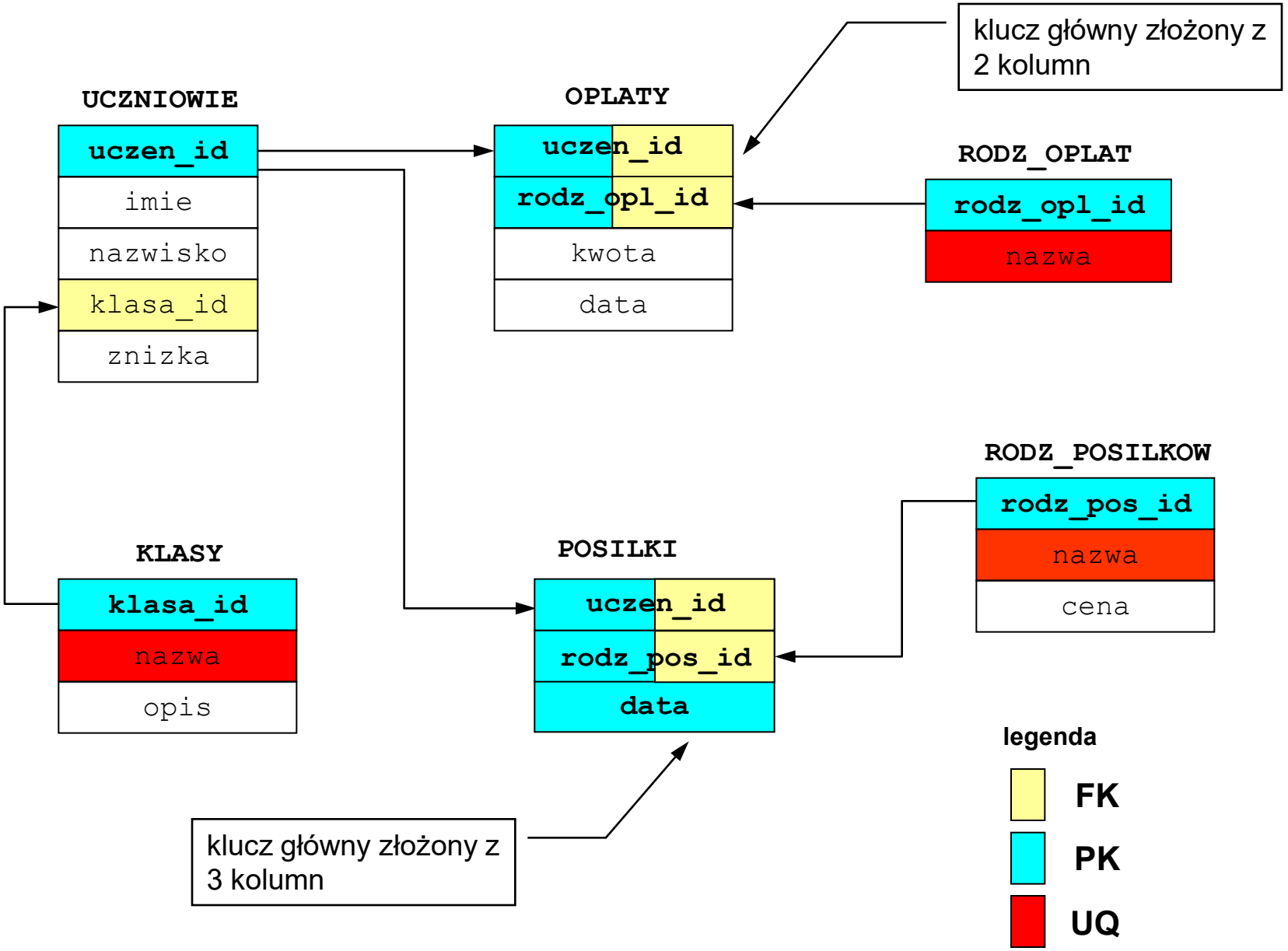
Rejestrowanie opłat wnoszonych przez poszczególnych uczniów wg. założeń:

- każdy uczeń może wnosić wiele różnych opłat (np. za ubezpieczenie, za wycieczki szkolne itp.). Opłaty nie są obowiązkowe
- dany uczeń może wnieść daną opłatę tylko raz (np. nie może zapłacić 2 razy za tą samą wycieczkę)
- wielkość opłaty (w zł) jest uzależniona od danego ucznia (np. wysokość ubezpieczenia każdy uczeń ustala indywidualnie, koszt wycieczki szkolnej jest uzależniony od dochodów rodziców itp.)

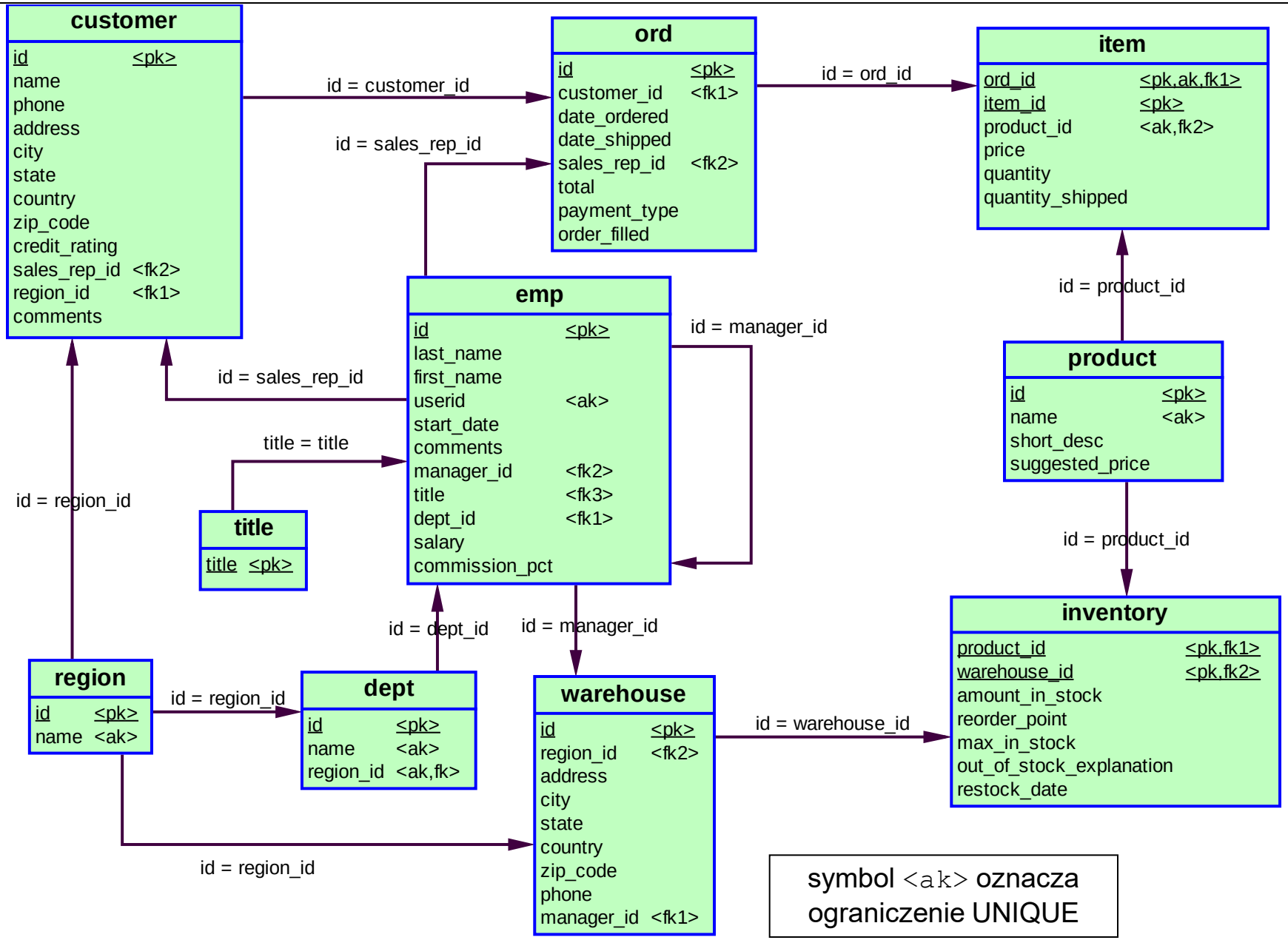
Rejestrowanie ilości i rodzajów posiłków wykupionych przez uczniów (śniadania, obiady, kolacje ew. inne) wg. założeń:

- każdy uczeń może wykupić dowolne posiłki w dowolnych dniach (tzn. posiłków nie wykupuje się w ramach tzw. miesięcznych karnetów. Można dowolnie wybrać dni)
- zakładamy, że dany uczeń może wykupić na dany dzień dany rodzaj posiłku tylko raz (tzn. wykluczamy możliwość, że np. wykupi w środę dwa obiady)
- zakładamy, że ceny posiłków są stałe i nie zmieniają się
- musi istnieć możliwość zaznaczenia faktu, że uczeń korzysta ze zniżki na posiłki (np. uczeń A ma 45% zniżki na wszystkie posiłki, uczeń B 80% zniżki a uczeń C nie korzysta ze zniżki)

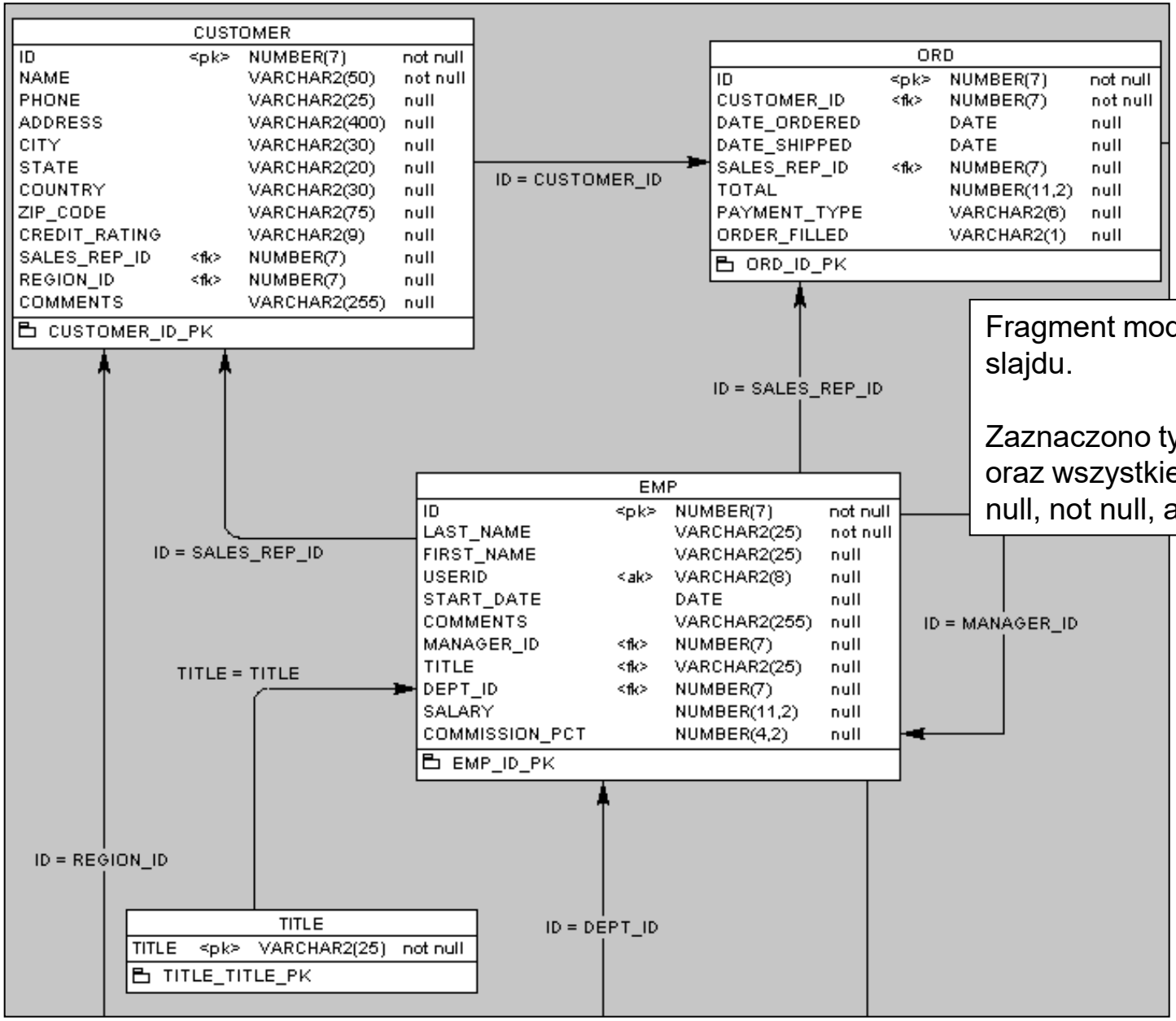
Przykładowy projekt - struktura relacyjna



Przykład bardziej złożonej struktury relacyjnej (1/4)



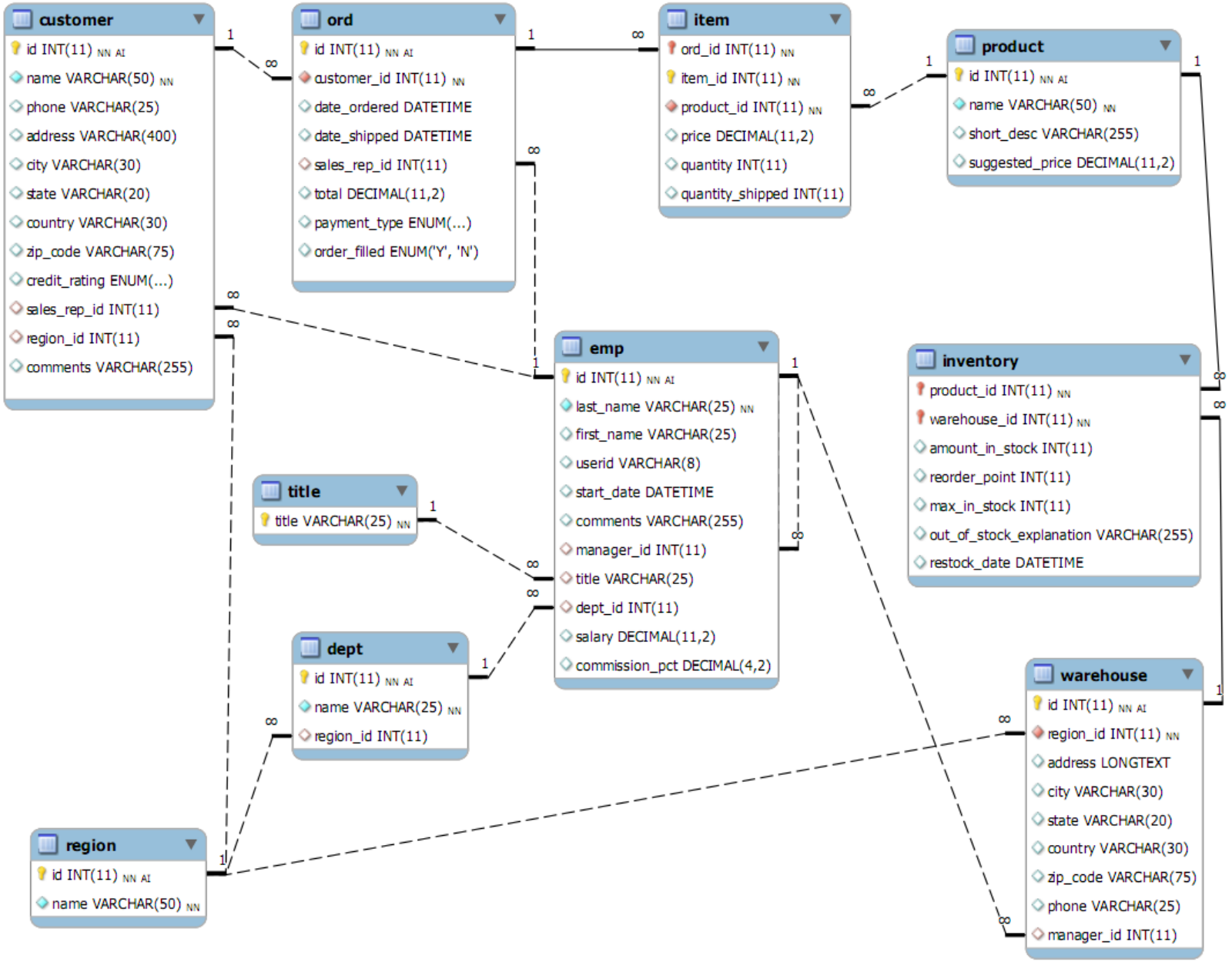
Przykład bardziej złożonej struktury relacyjnej (2/4)



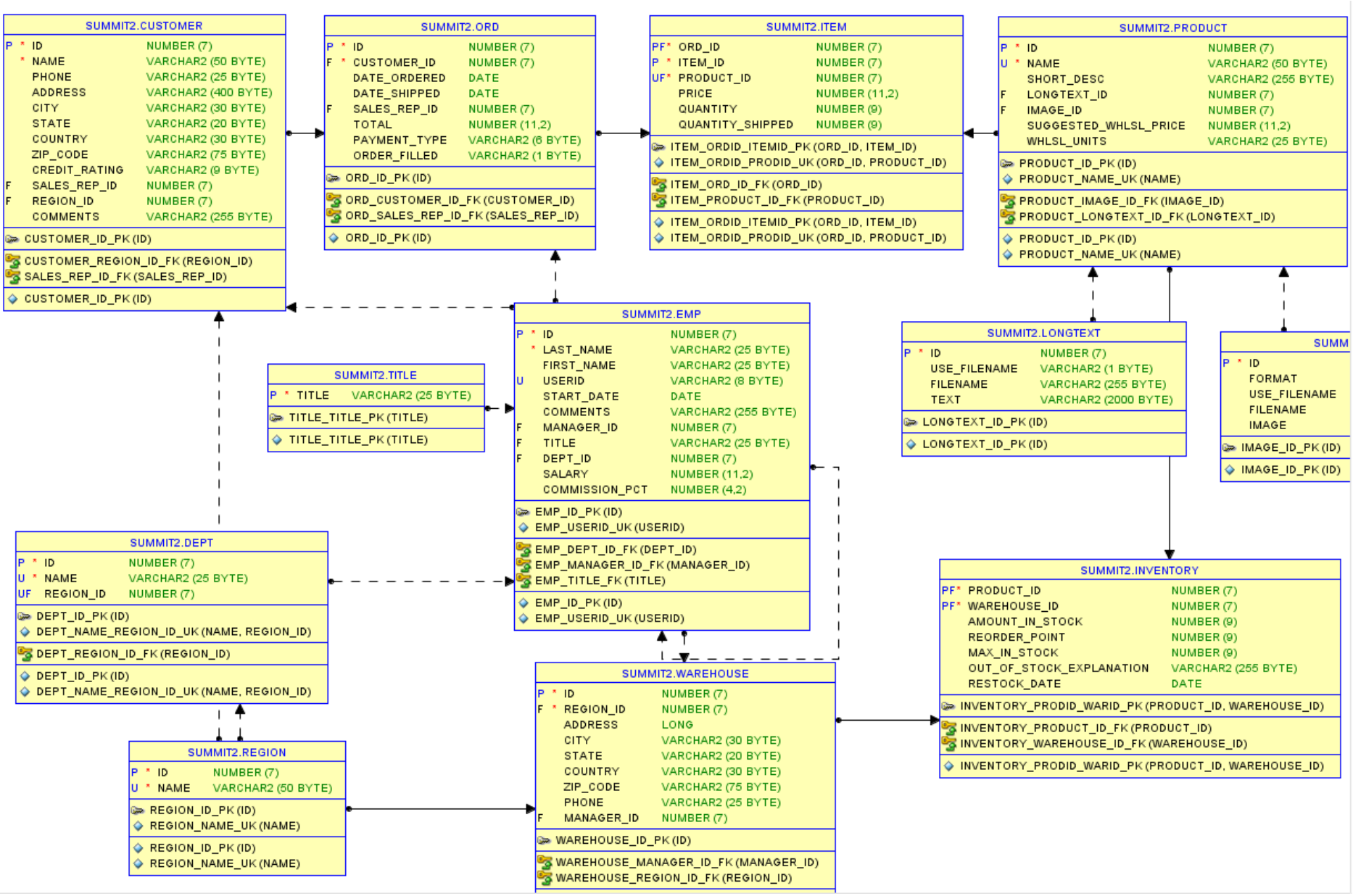
Fragment modelu z poprzedniego slajdu.

Zaznaczono typy kolumn, indeksy oraz wszystkie ograniczenia (pk, fk, null, not null, alternate keys)

Przykład bardziej złożonej struktury relacyjnej (3/4)



Przykład bardziej złożonej struktury relacyjnej (4/4)



Narzędzia wspomagające projektowanie baz danych

- PowerDesigner Data Architect
  (http://www.sybase.com/products/developmentintegration/powerdesigner)
  Oprogramowanie komercyjne + wersja Trial. Narzędzie stosunkowo złożone, ale o dużych możliwościach. Obsługuje bardzo wiele formatów baz danych.
- Toad Data Modeler
  (http://www.toadsoft.com/toaddm/toad_data_modeler.htm)
  Oprogramowanie komercyjne ale posiada nieco okrojoną wersję darmową.
- Oracle Developer Suite 10g (Oracle Designer)
  (http://www.oracle.com/technology/products/ids/index.html)
  Oprogramowanie darmowe do zastosowań nie komercyjnych. Narzędzie o bardzo dużych możliwościach ale bardzo złożone. Z założenia dla bazy danych Oracle.
- DBDesigner 4
  (http://fabforce.net/dbdesigner4/)
  Oprogramowanie GNU. Narzędzie bardzo proste, z założenia jedynie dla bazy MySQL.
- Powyższe zostało jakiś czas temu zastąpione przez MySQL Workbench
  (https://www.mysql.com/products/workbench)
- … i całkiem sporo innych

Uwagi końcowe

- projektowanie baz danych to często **bardziej sztuka niż nauka**
- prawidłowo zaprojektowana struktura relacyjna powinna spełniać warunki formalne oraz **wymogi klienta** - często jest to trudne do pogodzenia !
- teoria projektowania baz relacyjnych mówi **czego nie wolno robić**, ale nic nie mówi od czego zacząć oraz jak prawidłowo zaprojektować bazę
- przede wszystkim należy zrozumieć przedsiębiorstwo (system), które chcemy zamodelować. Tu nieocenione są **rozmowy z ludźmi**
- poznać obieg dokumentów w przedsiębiorstwie (systemie). One są najlepszym źródłem rzeczywistych wymagań klienta (choć obieg ten może być daleki od ideału - biurokracja)
- przy bardziej złożonych przedsięwzięciach opanować formalne techniki modelowania (np. wspomniane modelowanie związków encji oraz inne)
- wszelkie pomysły szkicować na papierze. Wtedy lepiej dostrzega się ew. błędy
- tworząc pierwsze wersje relacji wpisywać do nich przykładowe (ale rzeczywiste) dane. Wtedy najlepiej widać niedociągnięcia
- rozważ ew. korzyści z tzw. denormalizacji relacji (celowe zmniejszanie postaci normalnych). W wielu przypadkach może to poprawić „osiągi” aplikacji bazodanowej (niestety kosztem pewnych utrudnień)

Normalizacja relacji

Zależności funkcyjne

KLIENCI

klient_id	imie	nazwisko	adres	pesel	telefon	miasto

- Jeżeli istnieje **zależność funkcyjna** między kolumną A i kolumną B danej tabeli (relacji), oznacza to, że wartość kolumny A **determinuje (identyfikuje, wyznacza)** wartość kolumny B
- Zależność funkcyjną oznaczamy A $\rightarrow$ B
- W tabeli powyżej np. kolumna `klient_id` funkcjonalnie determinuje wszystkie pozostałe kolumny. Zależność odwrotna nie jest prawdziwa
- Bardziej formalna definicja:
  Atrybut B relacji R jest funkcyjnie zależny od atrybutu A tej relacji, jeśli zawsze każdej wartości ‘a’ atrybutu A odpowiada nie więcej niż jedna wartość ‘b’ atrybutu B. Stwierdzenie, że B jest funkcyjnie zależne od A jest równoważne stwierdzeniu, że A identyfikuje (wyznacza) B.

lub

Dla danej relacji R, w której X i Y są podzbiorami atrybutów schematu tej relacji, X wyznacza funkcyjnie Y, lub Y jest funkcyjnie zależne od X, co zapisujemy X $\rightarrow$ Y, wtedy i tylko wtedy, jeżeli dla dwóch dowolnych rekordów t_1, t_2, takich, że $t_1[X] = t_2[X]$ zachodzi zawsze $t_1[Y] = t_2[Y]$

źródło: Zbyszko Królikowski, Projektowanie schematów relacyjnych baz danych - Normalizacja, dostępne pod adresem: *www.ploug.org.pl/seminarium/05_01/pliki/2.pdf* oraz *www.ploug.org.pl/seminarium/05_01/pliki/normalizacja.pdf*

Pełne i częściowe zależności funkcyjne

- Zbiór atrybutów Y jest w **pełni funkcyjnie zależny** od zbioru atrybutów X w relacji R, jeżeli $X \rightarrow Y$ i **nie istnieje** podzbiór $X' \subset X$ taki, że $X' \rightarrow Y$.
- Zbiór atrybutów Y jest **częściowo funkcyjnie zależny** od zbioru atrybutów X w relacji R, jeżeli $X \rightarrow Y$ i **istnieje** podzbiór $X' \subset X$ taki, że $X' \rightarrow Y$.

Atrybuty wtórne

- Atrybut X w relacji R będziemy nazywać **atrybutem wtórnym**, jeżeli nie należy on do żadnego z kluczy głównych tej relacji.

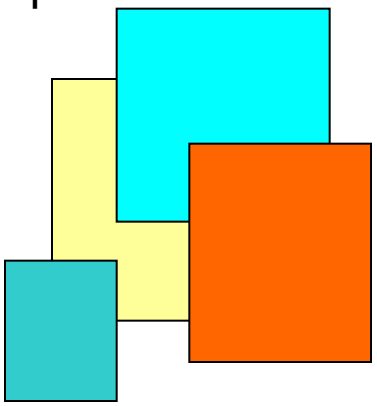
Przechodnie zależności funkcyjne

- Zbiór atrybutów Y jest w **przechodnio funkcyjnie zależny** od zbioru atrybutów X w relacji R, jeżeli $X \rightarrow Y$ i istnieje zbiór atrybutów Z, nie będący podzbiorem żadnego klucza głównego w relacji R taki, że zachodzi $X \rightarrow Z$ i $Z \rightarrow Y$.

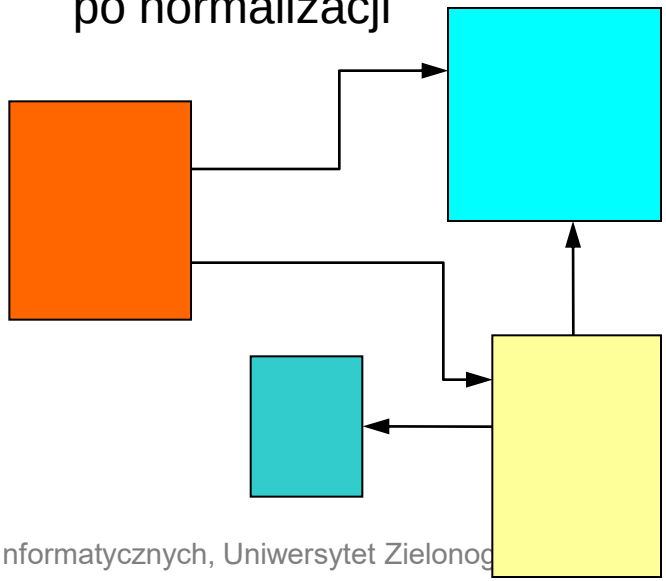
Normalizacja

- Normalizacja to proces doprowadzania relacji do odpowiednio wysokiej postaci normalnej
- Celem jest doprowadzić model relacyjny do takiego stany, w którym nie występują żadne niezamierzone anomalie. Typowe anomalie to:
  - redundancja (nadmiarowość) danych
  - anomalie przy wstawianiu danych
  - anomalie w trakcie modyfikowania danych
  - anomalie w trakcie kasowania danych
- Skutkiem wystąpienia anomalii jest często UTRATA SPÓJNOŚCI DANYCH !
- Proces normalizacji polega (mówiąc ogólnie) na dekomponowaniu relacji posiadających niepożądane cechy na mniejsze relacje, które tych niepożądanych cech nie posiadają

przed normalizacją



po normalizacji



Anomalie

prac_id	imie	nazwisko	dzial_id	nazwa_dzialu
1	Artur	Nowakowski	1	serwis
2	Jan	Kowalski	1	serwis
3	Roman	Nowak	2	księgowość
4	Stefan	Antkowiak	3	obsługa klienta
5	Ewa	Konieczna	3	obsługa klienta
6	Anna	Wojtasik	3	obsługa klienta
7	Marek	Pawlak	1	serwis

Przykładowe anomalie:

- anomalia wstawiania: nie można wstawić nazwy działu, który nie zatrudnia (na razie) żadnych pracowników
- anomalia wstawiania: wstawiając nową nazwę działu trzeba zawsze dbać o spójność z polem dzial_id
- anomalia usuwania: po wykasowaniu pracownika o numerze 3 tracimy informację, że kiedykolwiek istniał dział księgowości
- anomalia modyfikowania: zmiana nazwy działu z „serwis” na „serwis i sprzedaż” wymaga zmiany w trzech miejscach (tu: rekordach)
- redundancja danych: wielokrotnie wpisane są te same nazwy działów

Cechy poprawnego procesu normalizacji

- W trakcie normalizacji nie może dojść do utraty żadnych danych
- Wszystkie atrybuty (kolumny) występujące w relacjach przed normalizacją nie mogą zostać po normalizacji zagubione (utracone)
- Po normalizacji wszystkie zależności (funkcyjne) opisywane przez pierwotny model relacyjny muszą pozostać niezmienione

Postać nieznormalizowana

Zamówienie nr: **001**	
Klient: **Gramacki**	Identyfikator klienta: **25**
Data złożenia zamówienia: **06.10.2018**	
Opis zamówienia:	

Pozycja	Numer kat.	Ilość sztuk
Toner	**10**	**2**
Pendrive	**30**	**10**
HD	**40**	**1**

wartości nie są elementarne

zam_id	klient_id	nazwisko	data	opis_zam
001	25	Gramacki	06.10.2018	2 tonery, 10 pendrive, 1 dysk twardy
002	50	Nowak	13.07.2018	10 pendrive
003	75	Pawlak	14.08.2019	1 drukarka, 2 dyski twarde
004	100	Kowalski	20.12.2013	100 pendrive
005	125	Barski	11.11.2020	1 drukarka, 1 pamięć

dużo pustych komórek

zam_id	klient_id	nazwisko	data	poz_1	ilosc_1	poz_2	ilosc_2	poz_3	ilosc_3
001	25	Gramacki	06.10.2018	toner	2	pendrive	10	HD	1
002	50	Nowak	13.07.2018	pendrive	10				
003	75	Pawlak	14.08.2019	drukarka	1	HD	2		
004	100	Kowalski	20.12.2013	pendrive	100				
005	125	Barski	11.11.2020	drukarka	1	pamięć	1		

istnieją powtarzające się grupy

Normalizacja do 1 PN (1/2)

w tabeli są w gruncie rzeczy przechowywane informacje o dwóch rzeczach: zamówieniu jako takim oraz o szczegółach tych zamówień

ŹLE !

ZAMÓWIENIA

zam_id	data	klient_id	nazwisko	ilosc	prod_id	opis_prod
1	06.10.2018	25	Gramacki	2	10	toner
1	06.10.2018	25	Gramacki	10	30	pendrive
1	06.10.2018	25	Gramacki	1	40	HD
2	13.07.2018	50	Nowak	10	30	pendrive
3	14.08.2019	75	Pawlak	1	50	drukarka
3	14.08.2019	75	Pawlak	2	40	HD
4	20.12.2013	100	Kowalski	100	30	pendrive
5	11.11.2020	125	Barski	1	50	drukarka
5	11.11.2020	125	Barski	1	20	pamięć

zam_id, **prod_id** - klucz główny (złożony)

istnieją niekluczowe atrybuty, które są zależne tylko od części klucza głównego (`data`, `klient_id`)

Relacja jest w 1PN, gdy:

- wszystkie pola są elementarne (niepodzielne)
- nie istnieją jakiekolwiek powtarzające się grupy

ŹLE !

Normalizacja do 1 PN (2/2)

Co jest nadal źle ?

- dublowanie danych
- nie da się zarejestrować produktu, którego nikt jeszcze nie zamówił
- usuwając np. informacje o zamówieniu pierwszym tracimy informację, że kiedykolwiek był zamawiany produkt o nazwie ‘toner’

Co jest źródłem problemu ?

- istnieją atrybuty niekluczowe (czyli takie, które nie należą do klucza głównego), które są **funkcyjne zależne tylko od części klucza głównego**, np:

```
zam_id   → data
zam_id   → klient_id
```

Co należy zrobić ?

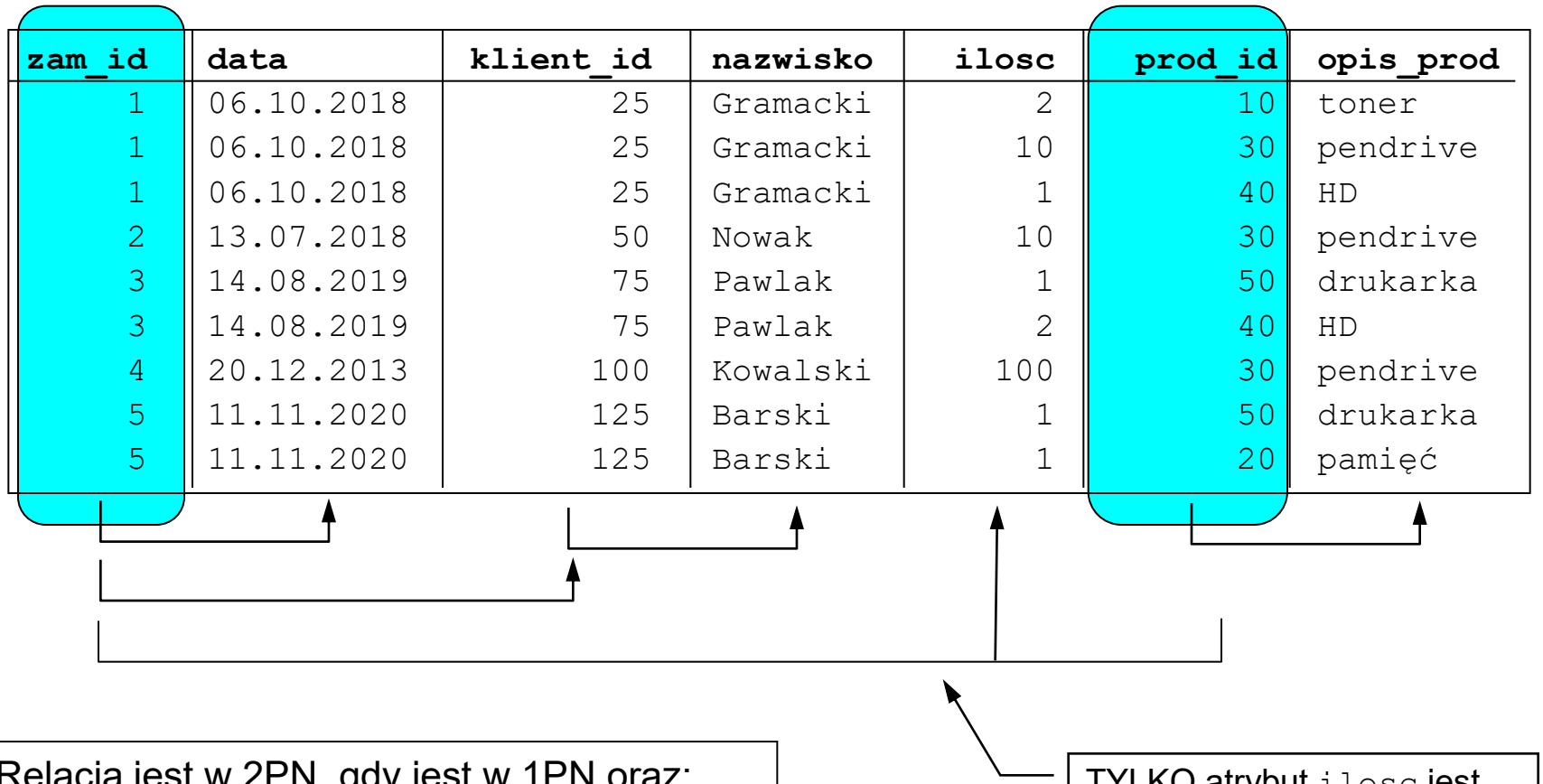
- zdekomponować relację na taki zbiór relacji, których wszystkie atrybuty niekluczowe będą zależne od całego klucza głównego

Normalizacja do 2 PN (1/3)

Zależności funkcyjne:

ZAMÓWIENIA

zam_id	data	klient_id	nazwisko	ilosc	prod_id	opis_prod
1	06.10.2018	25	Gramacki	2	10	toner
1	06.10.2018	25	Gramacki	10	30	pendrive
1	06.10.2018	25	Gramacki	1	40	HD
2	13.07.2018	50	Nowak	10	30	pendrive
3	14.08.2019	75	Pawlak	1	50	drukarka
3	14.08.2019	75	Pawlak	2	40	HD
4	20.12.2013	100	Kowalski	100	30	pendrive
5	11.11.2020	125	Barski	1	50	drukarka
5	11.11.2020	125	Barski	1	20	pamięć



Relacja jest w 2PN, gdy jest w 1PN oraz:

- każdy atrybut wtórny tej relacji jest w pełni funkcyjnie zależny od klucza głównego tej relacji

TYLKO atrybut `ilosc` jest funkcyjnie zależny od całego klucza głównego (`zam_id`, `prod_id`)

Normalizacja do 2 PN (2/3)

relacja zawiera tylko te atrybuty, które są funkcyjnie zależne od atrybutu `zam_id`

relacja zawiera tylko te atrybuty, które są funkcyjnie zależne od atrybutu `prod_id`

KLIENCI_NA_ZAMÓWIENIACH

zam_id	klient_id	nazwisko	data
1	25	Gramacki	06.10.2018
2	50	Nowak	13.07.2018
3	75	Pawlak	14.08.2019
4	100	Kowalski	20.12.2013
5	125	Barski	11.11.2020

PRODUKTY

prod_id	opis_prod
10	toner
20	pamięć
30	pendrive
40	HD
50	drukarka

przechodnia zależność funkcyjna

relacja zawiera tylko te atrybuty, które są funkcyjnie zależne od atrybutu `zam_id` oraz `prod_id`

OPISY_ZAMÓWIEŃ

zam_id	ilosc	prod_id
1	1	2
1	2	10
1	3	1
2	1	10
3	1	1
3	2	2
4	1	100
5	1	1
5	2	1

WNIOSEK:

- pierwotna **jedna** relacja będąca w 1PN, po doprowadzeniu do 2PN została rozbita na **trzy** relacje

Normalizacja do 2 PN (3/3)

Co jest nadal źle ?

- dublowanie danych (atrybut `nazwisko`)

Co jest źródłem problemu ?

- istnienie tzw. **przechodnich zależności funkcyjnych** między atrybutami, np: atrybut `nazwisko` jest przechodnio zależny funkcyjnie od atrybutu `zam_id`, gdyż atrybut `nazwisko` jest funkcyjnie zależny od atrybutu `klient_id`

Co należy zrobić ?

- zdekomponować „wadliwe” relacje na taki zbiór relacji, w których nie będą istniały przechodnie zależności funkcyjne

Relacja jest w 3PN, gdy jest w 2PN oraz:

- żaden atrybut wtórny tej relacji nie jest przechodnio zależny od klucza głównego tej relacji

Normalizacja do 3 PN

PRODUKTY

prod_id	opis_prod
10	toner
20	pamięć
30	pendrive
40	HD
50	drukarka

OPISY_ZAMÓWIEŃ

zam_id	ilosc	prod_id
1	1	2
1	2	10
1	3	1
2	1	10
3	1	1
3	2	2
4	1	100
5	1	1
5	2	1

KLIENCI_NA_ZAMÓWIENIACH

zam_id	klient_id	nazwisko	data
1	25	Gramacki	06.10.2018
2	50	Nowak	13.07.2018
3	75	Pawlak	14.08.2019
4	100	Kowalski	20.12.2013
5	125	Barski	11.11.2020

ZAMÓWIENIA

zam_id	klient_id	data
1	25	06.10.2018
2	50	13.07.2018
3	75	14.08.2019
4	100	20.12.2013
5	125	11.11.2020

KLIENCI

klient_id	nazwisko
25	Gramacki
50	Nowak
75	Pawlak
100	Kowalski
125	Barski

WNIOSEK:

- pierwotne **jedna** relacja będące w 2PN, po doprowadzeniu do 3PN została rozbita na **dwie** relacje

Normalizacja do 3 PN - kolumny wyliczane

OPISY_ZAMÓWIEŃ

zam_id	ilosc	cena_jedn	naleznosc	prod_id
1	2	5	10	10
1	10	10	100	30
1	1	5	5	40
2	10	2	20	30
3	1	4	4	50
3	2	5	10	40
4	100	6	600	30
5	1	7	7	50
5	1	8	8	60

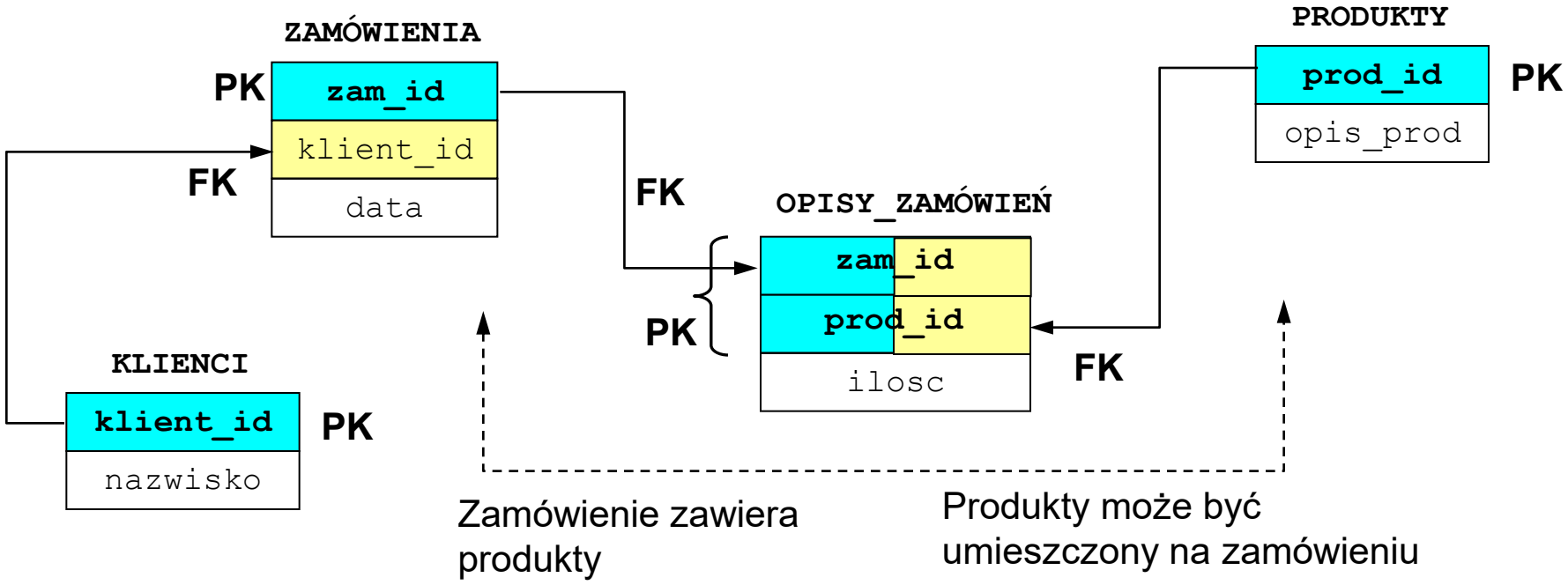
ŹLE !

ta kolumna powinna być w tabeli PRODUKTY

ŹLE !

kolumna wyliczona
(iloczyn kolumn `ilosc` oraz `cena_jedn`)

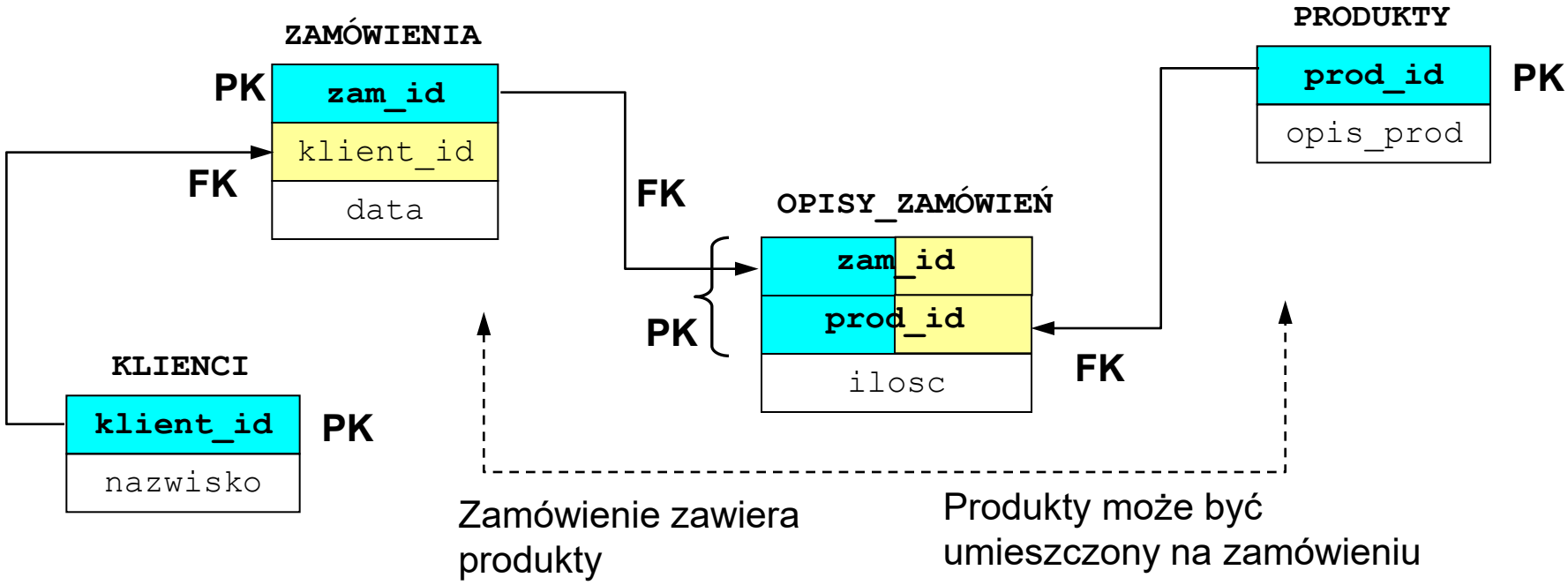
Efekt końcowy po normalizacji



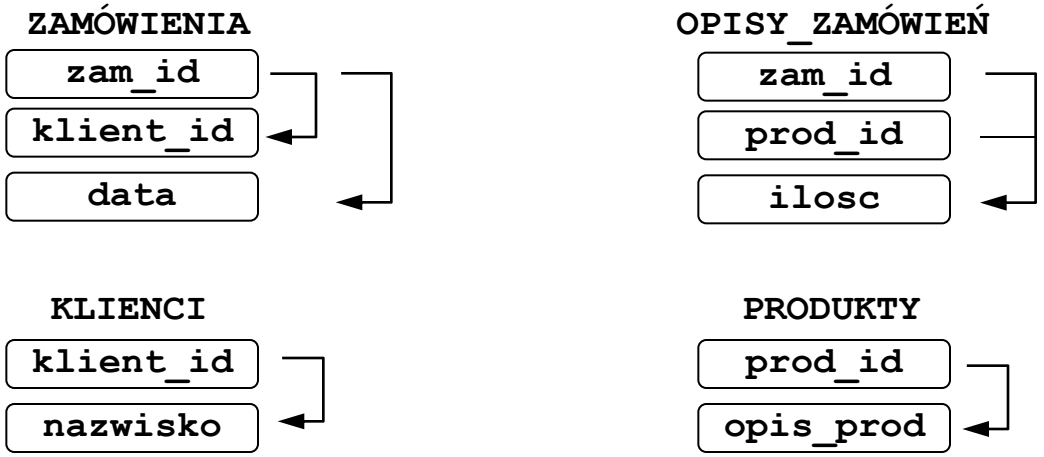
WNIOSKI:

- Do zbudowania poprawnego modelu relacyjnego często wystarczy po prostu podejście zdroworozsądkowe i odrobina doświadczenia. Formalne prowadzenie procesu normalizacji staje się wówczas „sztuką samą w sobie”
- Należy zawsze pamiętać, że doprowadzenie relacji do wyższej postaci normalnej zawsze prowadzi do podziału (dekompozycji) relacji na mniejsze relacje o pożądanych cechach
- Procedura doprowadzania relacji do wyższych postaci normalnych jest zawsze odwracalna

Efekt końcowy po normalizacji – zależności funkcyjne



Zależności funkcyjne finalnego modelu:



Normalizacja do 4 PN

Relacja jest w 3PN ale nadal widać redundancję danych. Wszystkie kolumny wchodzą w skład klucza głównego (tzw. relacje typu „all-key”)

KURSY

kurs	wykładowca	podręcznik
Informatyka	prof. Green	Podstawy informatyki
Informatyka	prof. Green	Struktury danych
Informatyka	prof. Brown	Podstawy informatyki
Informatyka	prof. Brown	Struktury danych

12 komórek

- dowolny kurs może prowadzić dowolna liczba wykładowców
- dowolny kurs może opierać się na dowolnej liczbie podręczników
- wykładowcy i podręczniki są wzajemnie niezależni

KURSY_WYKŁADOWCY

kurs	wykładowca
Informatyka	prof. Green
Informatyka	prof. Brown

KURSY_PODRĘCZNIKI

kurs	podręcznik
Informatyka	Podstawy informatyki
Informatyka	Struktury danych

8 komórek

Normalizacja do 4 PN - inny przykład

STUDENCI **45 komórek**

student	jezyk_progr	jezyk_obcy
Nowak	Java	angielski
Nowak	Java	niemiecki
Nowak	C	angielski
Nowak	C	niemiecki
Nowak	Pascal	angielski
Nowak	Pascal	niemiecki
Kowalski	PL/SQL	angielski
Kowalski	PL/SQL	niemiecki
Kowalski	PL/SQL	francuski
Malinowski	SQL	angielski
Malinowski	SQL	niemiecki
Malinowski	SQL	francuski
Malinowski	Fortran	angielski
Malinowski	Fortran	niemiecki
Malinowski	Fortran	francuski

- Każdy student może znać dowolną liczbę języków obcych
- Każdy student może znać dowolną liczbę języków programowania
- Języki obce i języki programowania są wzajemnie niezależne

STUDENCI_J_PROGR

student	jezyk_progr
Nowak	Java
Nowak	C
Nowak	Pascal
Kowalski	PL/SQL
Malinowski	SQL
Malinowski	Fortran

STUDENCI_J_OBCE

student	jezyk_obcy
Nowak	angielski
Nowak	niemiecki
Kowalski	angielski
Kowalski	niemiecki
Kowalski	francuski
Malinowski	angielski
Malinowski	niemiecki
Malinowski	francuski

28 komórek

Transakcje bazodanowe

opracowano w dużej części na podstawie: Richard Stones, Neil Matthew:
Bazy danych i MySQL. Od podstaw, Helion 2003

Czym są transakcje (1/2)

- Niemal w każdej rzeczywistej wieloużytkownikowej aplikacji bazodanowej do danych sięga **w tym samym czasie** więcej niż jeden użytkownik
- Stan idealny: z punktu widzenia użytkownika baza zachowuje się tak, jakby każdy użytkownik miał **wyłączny dostęp do zasobów serwera** i nie był zależny od innych użytkowników
- Klasyczny przykład transakcji: przelew pieniędzy z jednego konta na drugie
  - złożenie zlecenia przez klienta A
  - sprawdzenie stanu konta klienta A
  - zablokowanie kwoty przelewu na koncie klienta A
  - przesłanie kwoty przelewu na konto klienta B
  - powiększenie o kwotę przelewu salda rachunku klienta B
  - pomniejszenie o kwotę przelewu salda rachunku klienta A
- Powyższe czynności nazywane są transakcją. Gdy którakolwiek z operacji wchodzących w skład transakcji nie powiedzie się, należy **wycofać wszystkie pozostałe** (tak, jakby cała operacja nie miała miejsca)
- Czym więc jest transakcji? Jest to niepodzielny logicznie blok instrukcji. Blok instrukcji musi być **albo w całości wykonany (COMMIT), albo w całości odrzucony (ROLLBACK),**

Czym są transakcje (2/2)

- Transakcje wykonywane w jednym czasie są od siebie całkowicie niezależne
  - w idealnych warunkach każda transakcja powinna mieć wyłączny dostęp do danych
- Przykład: dwaj klienci próbują w tym **samym czasie** zarezerwować **to samo miejsce** w samolocie. Problem jaki może wystąpić polega na tym, że komputerowy system rezerwacji sprzeda obu klientom to samo miejsce. Czynności do wykonania:
  - sprawdzenie, czy miejsce jest wolne
  - jeżeli tak to poinformowanie o tym klienta
  - przyjęcie numeru karty kredytowej i obciążenie konta określoną kwotą
  - przydział biletu
  - zaznaczenie sprzedanego miejsca jako sprzedanego
- Co będzie, gdy z systemu będą korzystać **w tym samym czasie** dwaj klienci?
  - dopiero po wykonaniu ostatniego kroku system „zorientuje się”, że sprzedano dwa razy to samo miejsce
- Ja rozwiązać ten problem?
  - **izolować transakcję**, stosując wyłączny dostęp do krytycznych danych, tak aby w danej chwili miała do nich dostęp tylko jedna osoba

Własności transakcji (1/2)

- **ACID** (Atomic, Consistent, Isolated, Durable)
- **Atomowość** (Atomicity)

  Każda transakcja musi być albo wykonana w całości, albo w całości odrzucona

- **Spójność** (Consistency)

  Po zatwierdzeniu transakcji, wszystkie dane muszą być spójne ze zdefiniowanymi zasadami integralnościowymi (klucze obce, główne). Dla przykładu z bankiem, po zakończeniu transakcji salda na obu kontach muszą być prawidłowe

- **Izolacja** (Isolation)

  Każda transakcja w trakcie jej działania nie widzi zmian wprowadzanych przez inne transakcje działające współbieżnie. Jest to zazwyczaj uzależnione od poziomu izolacji, o czym będzie za chwilę. Dla przykładu z liniami lotniczymi: obaj klienci muszą odnosić wrażenie, że mają wyłączny dostęp do systemu i nikt inny nie „blokuje” im pracy. Ten warunek jest w praktyce bardzo trudny do spełnienia

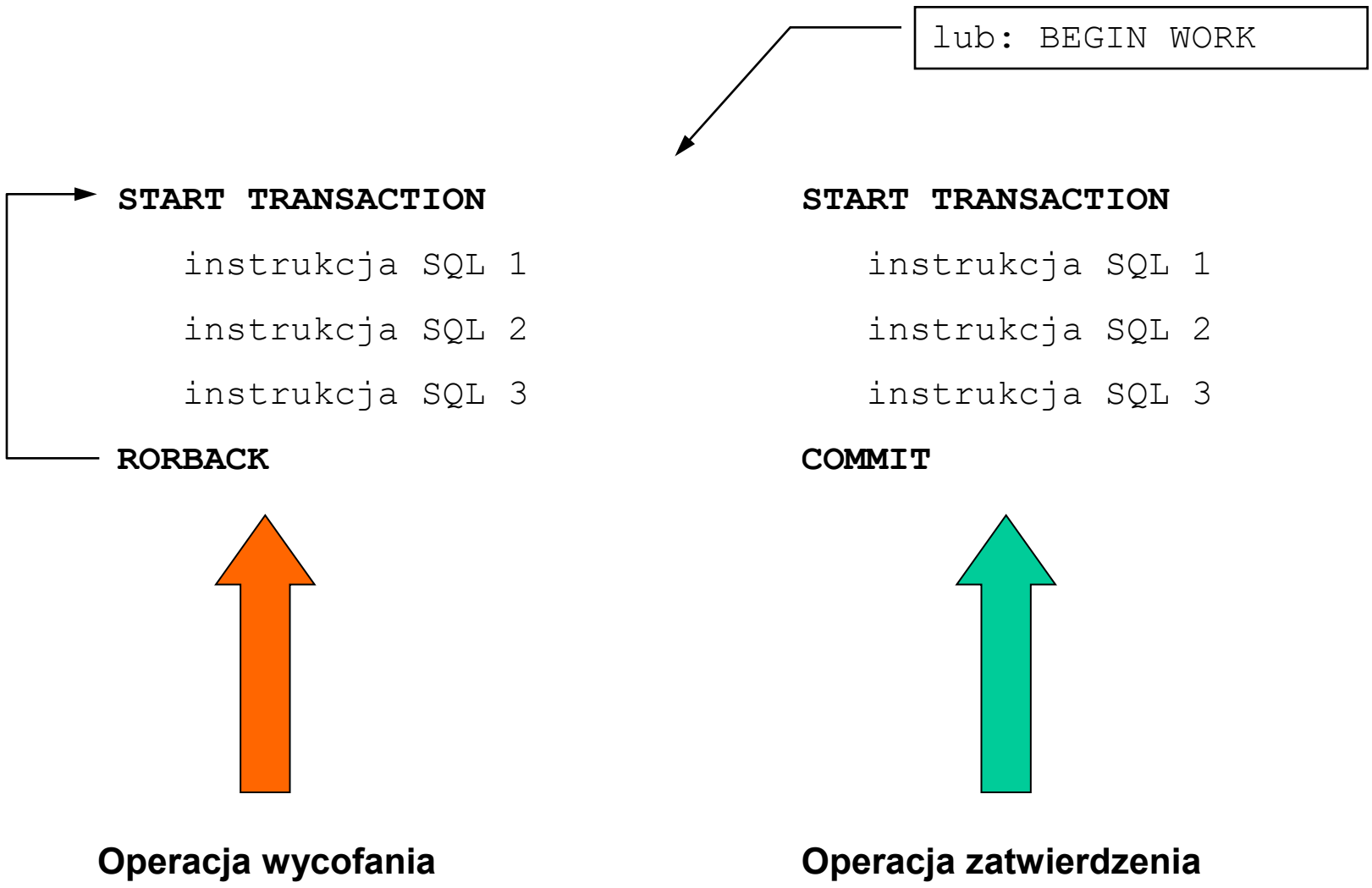
- **Trwałość** (Durability)

  Po zakończeniu transakcji jej wyniki muszą zostać trwale zapisane w bazie, nawet w przypadku pojawienia się niespodziewanej awarii systemu (np. zanik zasilania). Gdy awaria pojawi się w trakcie trwania transakcji, wszelkie nie zakończone transakcje muszą zostać anulowane

Własności transakcji (2/2)

- W systemie MySQL spełnienie wszystkich własności ACID zapewniają nam mechanizmy składowania **InnoDB** oraz **Berkeley DB**. Mechanizm **MyISAM** nie zapewnia wsparcia dla ACID
- Jakie rozwiązania techniczne wspierają właściwości ACID?
  - atomowość - transakcje
  - spójność - transakcje oraz klucze obce
  - izolacja - mechanizm umożliwiający wybór poziomu wyizolowania transakcji
  - trwałość - dziennik binarny oraz narzędzia do przywracania stanu bazy sprzed awarii (ang. *backup and recovery*)

Obsługa transakcji dla jednego użytkownika



Obsługa transakcji dla wielu użytkowników

- Poziomy izolacji (3. zasada ACID)
  - najbardziej rygorystyczne: jedno połączenie do bazy i jedna transakcja w określonym momencie. Skutek: drastyczne ograniczenie wielodostępności
- Zjawiska niepożądane
  - niespójność odczytów (Dirty Reads)

    Gdy jedna transakcja może odczytywać dane zmieniane właśnie przez inną transakcję (czyli, gdy zmiany nie zostały jeszcze zatwierdzone). Inaczej: jedna transakcja widzi niezatwierdzone zmiany innej transakcji
  - niepowtarzalność odczytów (Non-Repeatable Reads)

    W obrębie jednej transakcji kilkukrotny odczyt tego samego wiersza daje różne wartości, bo wiersz został zmieniony przez inną transakcję
  - odczyty fantomowe (Phantom Reads)

    Jest specjalnym przypadkiem Non-Repeatable Reads. Zwracany jest różny zbiór danych, bo warunek z bieżącego SELECT-u zawiera zmiany innych transakcji

Poziomy izolacji definiowane przez normę ANSI/ISO

Poziom	Niespójność odczytów	Niepowtarzalność odczytów	Fantomy
Read uncommitted	dopuszczalne	dopuszczalne	dopuszczalne
Read committed	niedopuszczalne	dopuszczalne	dopuszczalne
Repeatable read	niedopuszczalne	niedopuszczalne	dopuszczalne
Serializable	niedopuszczalne	niedopuszczalne	niedopuszczalne

- READ UNCOMMITED - w transakcjach są widoczne zmiany, które nie zostały jeszcze zatwierdzone przez inne transakcje
- READ COMMITED - widziane są tylko zmiany zatwierdzone
- REPEATABLE READS - zapewnia powtarzalność odczytu. Gdy transakcja A zlicza dane z tabeli, a transakcja B doda nowy rekord, to gdy transakcja A powtórzy swoja operację i tak otrzyma ten sam wynik co za pierwszym razem
- SERIALIZABLE - wymaga blokowania operacji czytania i pisania. Transakcje działają na "zamrożonym" stanie bazy danych

Poziomy izolacji definiowane przez normę ANSI/ISO

Uwagi

- MySQL wspiera wszystkie cztery poziomy izolacji
- MySQL domyślnie ustawia poziom `Repeatable read`
- Zaleca się tworzenie kilka małych transakcji zamiast jednej dużej
- Długo trwające transakcje mogą blokować innym użytkownikom dostęp do danych
- COMMIT trwa zwykłe bardzo szybko, ROLLBACK często trwa tak samo długo, lub nawet dłużej, niż trwała wycofywana transakcja
- Aplikacje użytkowników nie powinny tworzyć transakcji, które do ich zakończenia wymagają interakcji z użytkownikiem (może on np. zapomnieć nacisnąć guzik „OK”)

MySQL i transakcje, mechanizm składowania MyISAM

- MyISAM nie obsługuje transakcji
- Jest za to bardzo szybki
- Poszczególne instrukcje są traktowane jak transakcje. W transakcje nie można grupować kilku instrukcji
- Transakcje można emulować za pomocą blokad
  - `LOCK TABLE nazwa_tabeli [ READ | WRITE ]`

    `READ` - inni użytkownicy mogą tylko czytać tabelę

    `WRITE` - inni użytkownicy ani czytać, ani zapisywać danych. Tabela jest całkowicie blokowana dla innych użytkowników
  - `UNLOCK TABLES` (nie podaje się listy tabel. Zawsze zostają odblokowane wszystkie zablokowane wcześniej)

Bezpieczne aktualizowanie danych w tabelach MyISAM

Podniesienie pensji wybranemu pracownikowi

1. Odszukanie pracownika

```
SELECT id FROM emp WHERE last_name LIKE 'Magee';
```

2. Dokonanie zmiany

```
UPDATE salary SET salary = salary + 100;
```

Problem: pomiędzy odszukaniem pracownika a zmianą jego pensji ktoś inny może dokonać zmian.

Lepsze rozwiązanie:

```
UPDATE salary SET salary = salary + 100
WHERE last_name LIKE 'Magee';
```

MySQL i transakcje, mechanizm składowania InnoDB

- InnoDB obsługuje transakcje
- Jest nieco wolniejsze niż MyISAM
- Domyślny poziom izolacji: *repeatable read*
- Wspiera tryby AUTOCOMMIT (ON / OFF)

```
mysql> SELECT @@tx_isolation;
+-----------------+
| @@tx_isolation  |
+-----------------+
| REPEATABLE-READ |
+-----------------+
```

  - `SET AUTOCOMMIT = 0` (wymaga jawnego COMMIT)

    Ważna uwaga: gdy AUTOCOMMIT jest ustawiony na 0 i gdy wykonamy polecenie BEGIN WORK (SET TRANSACTION), przed rozpoczęciem nowej transakcji MySQL wykona automatycznie COMMIT. Warto więc zawsze dla pewności, przed rozpoczęciem nowej transakcji, wykonać ROLLBACK

  - `SET AUTOCOMMIT = 1` (ustawiany domyślnie, niejawny COMMIT)

- Polecenia do obsługi transakcyjności w bazie MySQL:
  - `START TRANSACTION | BEGIN [WORK]`
  - `COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]`
  - `ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]`
  - `SET AUTOCOMMIT = {0 | 1}`
- Wspiera blokowanie wierszy. Blokada jest trzymana aż do wykonania COMMIT lub ROLLBACK

```
SELECT 0 FROM emp WHER id = 1 FOR UPDATE;
```

Zakleszczenia (ang. *deadlock*)

Zachodzi, gdy dwie różne transakcje próbują zmieniać w tym samym czasie te same dane, ale w różnej kolejności

SESJA 1	SESJA 2
`UPDATE wiersz 8`	
	`UPDATE wiersz 15`
`UPDATE wiersz 15`	
	`UPDATE wiersz 8`

`ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction`

W tym miejscu obie sesje są zablokowane, ponieważ każda z nich czeka na zdjęcie blokady

Przykłady (1/4)

```
DROP TABLE t;
CREATE TABLE t (i INT PRIMARY KEY) ENGINE = InnoDB;
insert into t values (1), (2), (3);
COMMIT;
```

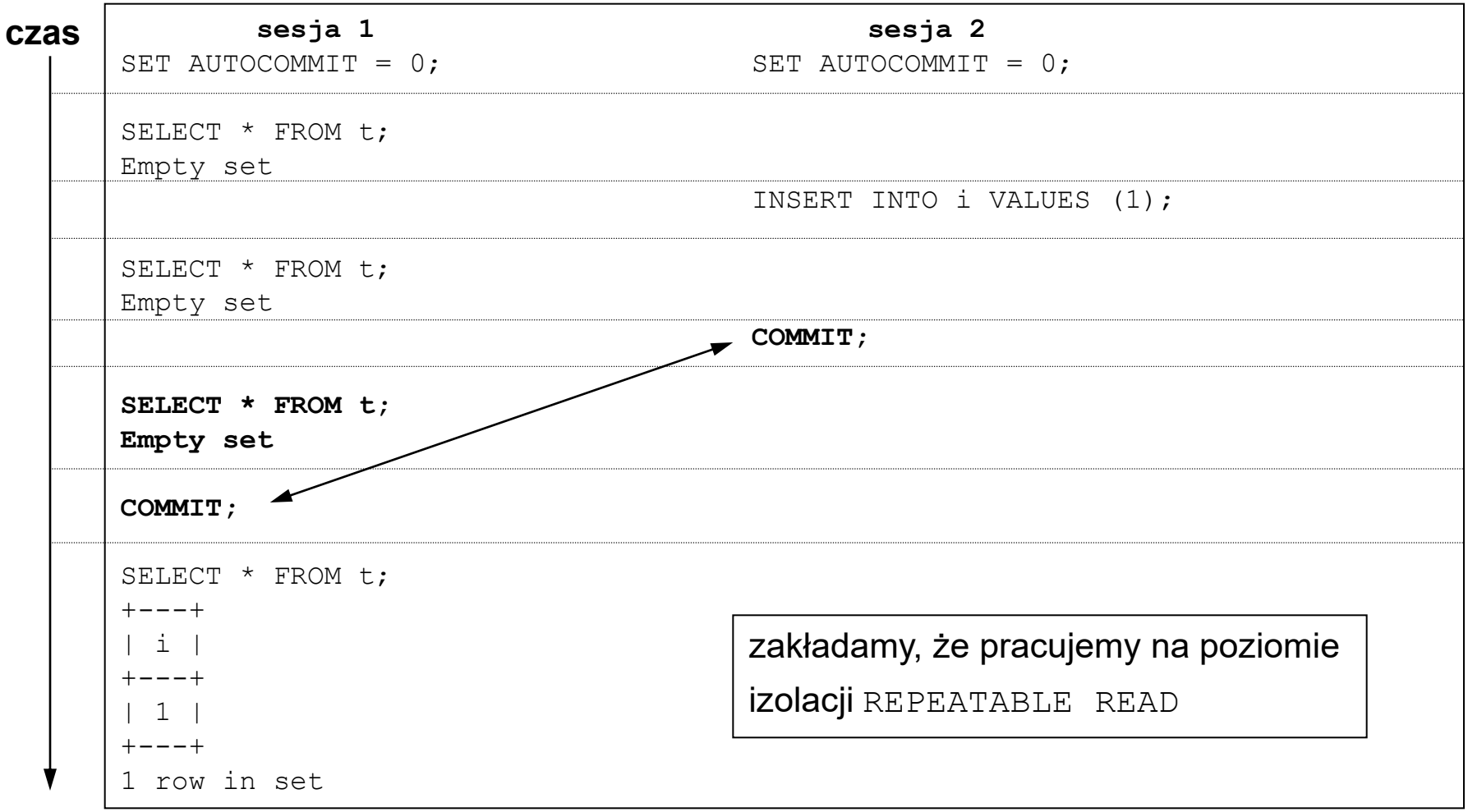
czas

sesja 1	sesja 2
SET AUTOCOMMIT = 0;	SET AUTOCOMMIT = 0;
START TRANSACTION;	START TRANSACTION;
UPDATE t SET i=10 WHERE i=1;	
	UPDATE t SET i=20 WHERE i=2;
UPDATE t SET i=20 WHERE i=2; *sesja "wisi" ...*	
sesja "puszcza"	**UPDATE t SET i=10 WHERE i=1;** *ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction*

Gdy tabela nie będzie miała klucza `PRIMARY KEY` powyższy eksperyment będzie wyglądał inaczej (jak? sprawdź sam). Zgodnie bowiem z dokumentacją podczas wykonywania polecenia `UPDATE...WHERE...` "**sets an exclusive next-key lock on every record the search encounters**"

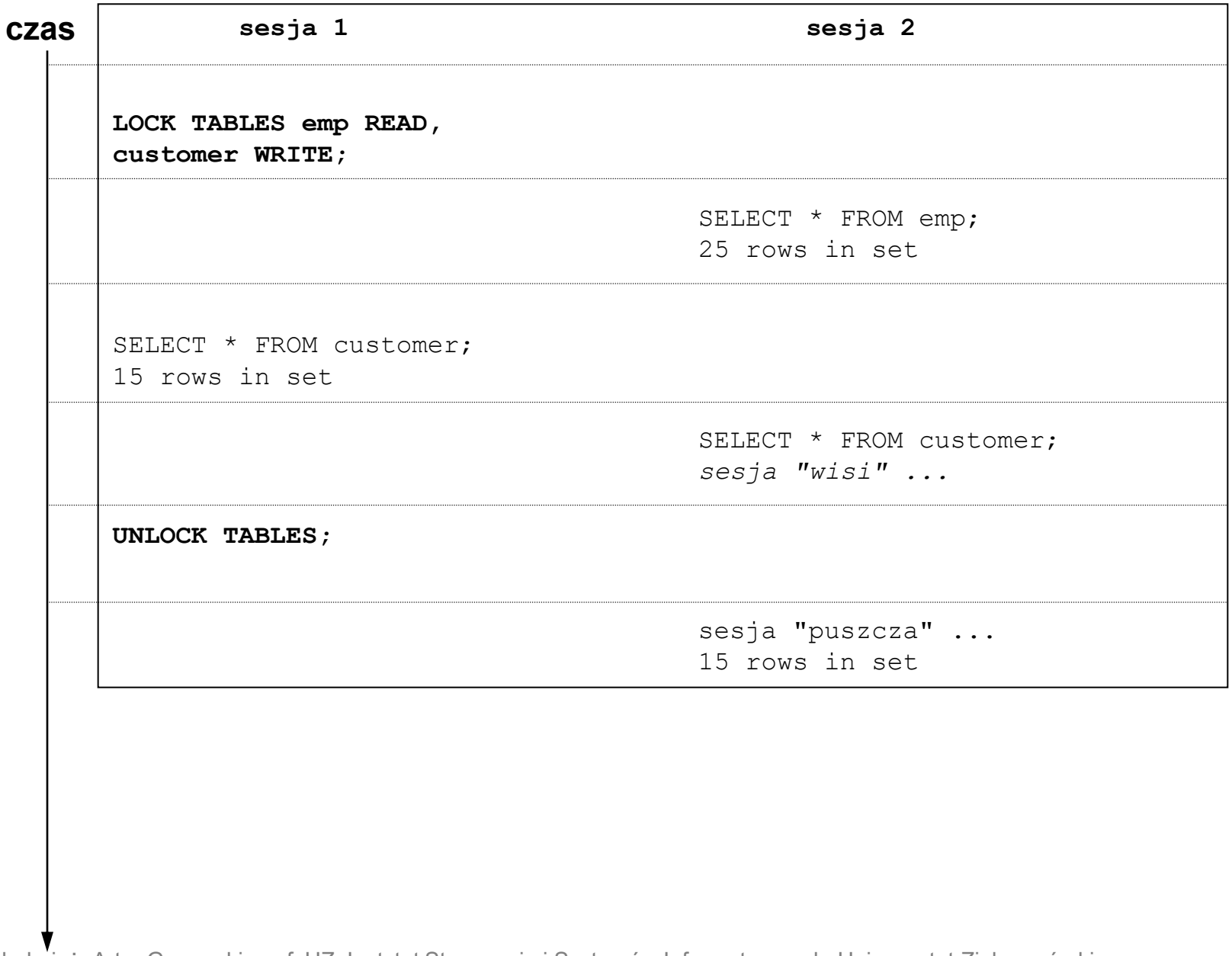
Odpowiedź: ponieważ na kolumnie używanej w klauzuli WHERE nie ma klucza, już pierwszy UPDATE (w pierwszej sesji) spowoduje zablokowanie całej tabeli. Dlatego też po wykonaniu UPDATE-u w sesji 2, sesja "zawiesza się".

Przykłady (2/4)



SELECT przypisuje naszej transakcji znacznik czasowy (ang. *timepoint*), w odniesieniu do którego nasze zapytanie „widzi” dane. Jeśli inna transakcja zmieni dane już po przypisaniu nam znacznika czasowego, nie zobaczymy zmian, chyba że wykonamy poleceniem COMMIT, które „przesunie” nam znacznik czasowy.

Przykłady (3/4)



Przykłady (4/4)

czas

sesja 1	sesja 2
SET AUTOCOMMIT = 0;	SET AUTOCOMMIT = 0;
SELECT 0 FROM emp WHERE id=10 **FOR UPDATE;** +---+ | 0 | +---+ | 0 | +---+ 1 row in set	
	UPDATE emp SET salary = 9999 WHERE id = 25; Query OK, 1 row affected
	UPDATE emp SET salary = 9999 WHERE id = 10; sesja "wisi" ...
COMMIT;	
	sesja "puszcza" ... Query OK, 1 row affected