w wykładzie wykorzystano:

1. dr inż. Piotr Macioł, wykłady
2. Lech Banachowski, SYSTEMY BAZ DANYCH, Polsko-Japońska Wyższa Szkoła Technik Komputerowych, Warszawa 2004, http://edu.pjwstk.edu.pl/wyklady/sbd/scb/index.html
3. Subieta K., Obiektowe Bazy Danych kontra Relacyjne Bazy Danych, Instytut Podstaw Informatyki PAN, Warszawa 1997

Obiektowe bazy danych

Krzysztof Regulski
WIMiIP, KISiM,
regulski@agh.edu.pl
B5, pok. 408

Wady relacyjnych baz danych

- model danych (szczególnie relacyjny) jest zbyt prosty do zamodelowania złożonych, zagnieżdżonych encji (np. w przypadku występowania zależności wielowartościowych)
- systemy baz danych nie obsługują ogólnych typów danych występujących w językach programowania

Wady relacyjnych baz danych

- model danych nie zawiera kilku często używanych pojęć semantycznych (np.: generalizacja, agregacja)
- zbyt wolne działanie systemów baz danych z programami użytkowymi wymagającymi szybkich i skomplikowanych obliczeń
- systemy baz danych nie dostarczają narzędzi do reprezentowania i zarządzania temporalnymi aspektami baz danych (m.in.: pojęciem czasu, wersjami obiektów i schematu)

Wady relacyjnych baz danych

- różnice między językami baz danych (SQL, DL/1, CODASYL DML), a językami programowania (COBOL, FORTRAN, PL/1, C++) – „niezgodność impedancji”
  - » Zespół niekorzystnych zjawisk towarzyszących formalnemu połączeniu języka zapytań (np. SQL) z uniwersalnym językiem programowania (np. C, Cobol lub PL/I) .
  - » Objawia się niezgodnościami w zakresie: składni, typów danych, semantyki, etc.
- model transakcji jest nieodpowiedni dla transakcji długotrwałych wymaganych w interakcyjnych, zespołowych środowiskach projektowania (wiązanie na czas trwania transakcji wielu relacji)

Utrudnienia w realizacji aplikacji:

- systemy projektowania wspomagane komputerowo (CAD/CAM, CASE)
- systemy języków programowania
- systemy baz wiedzy
- systemy multimedialne (operujące na obrazach, dźwiękach, dokumentach tekstowych)
- złożone systemy interfejsu użytkownika

Wymagania wobec baz piątej generacji

- reprezentowanie i posługiwanie się złożonymi, zagnieżdżonymi obiektami
- definiowanie dowolnych typów danych i operowanie nimi
- reprezentowanie i zarządzanie zmianami w bazie danych

Wymagania wobec baz piątej generacji

- reprezentowanie i operowanie pojęciami semantycznymi takimi jak hierarchia, agregacja
- podstawowe mechanizmy do wspomagania programów użytkowych wykorzystujących bazy wiedzy
- zarządzanie długotrwałymi transakcjami

Znaczenie obiektowości w bazach danych

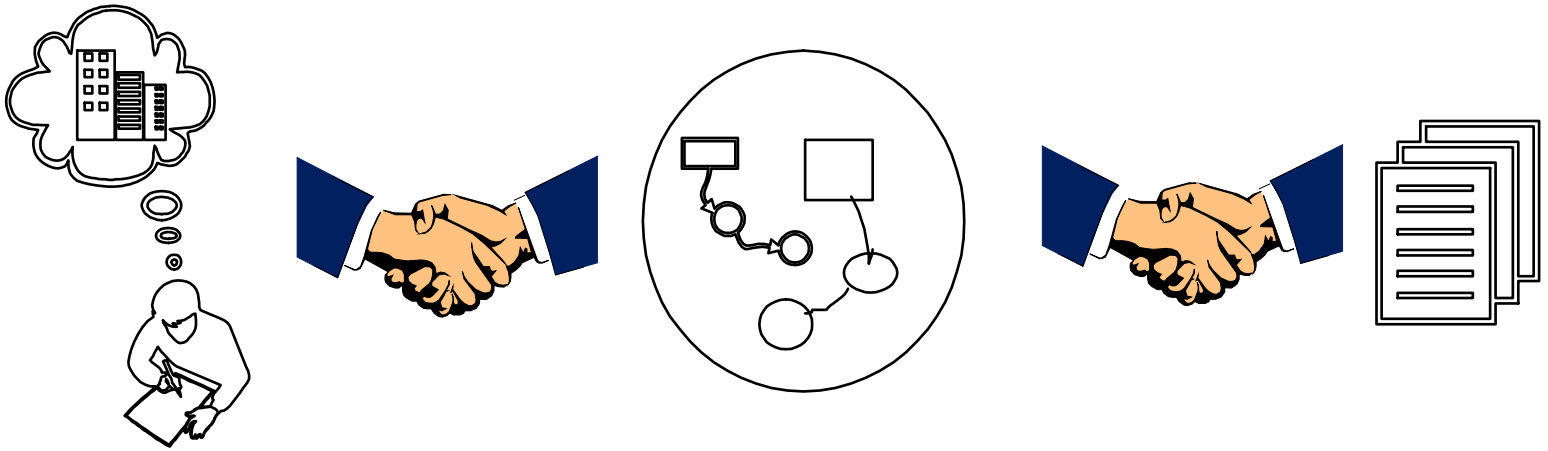
- Od lat trwają prace nad koncepcjami polegającymi na dołączaniu cech obiektowych do istniejących relacyjnych baz danych. Otrzymano w ten sposób model *obiektowo-relacyjny* i faktycznie na takim modelu jest w tej chwili oparty Standard języka SQL:1999.
- Typy obiektowe realizują zasadę abstrakcji w dwóch postaciach:
  - *abstrakcji proceduralnej* polegającej na ukryciu szczegółów złożonych algorytmów poprzez opakowanie ich w procedury i funkcje; gdy w razie potrzeby zmieniamy procedurę - nie musimy modyfikować aplikacji jej używających;
  - *abstrakcji danych* polegającej na ukryciu złożoności struktury danych przed użytkownikiem, który korzysta z tych danych; podobnie jak poprzednio, gdy w razie potrzeby zmieniamy strukturę danych – nie musimy modyfikować aplikacji jej używających.

Zalety użycia abstrakcji w bazach danych

- Ułatwione modelowanie rzeczywistych obiektów biznesowych.
- Zmniejszenie złożoności tworzenia aplikacji przez podział zadania na części. Ułatwienie dokonywania zmian. Ukrycie szczegółów implementacyjnych przed użytkownikiem. Modularność aplikacji i możliwość wielokrotnego użycia komponentów w tej samej lub w różnych aplikacjach.
- Zgrupowanie używanego kodu po stronie serwera wokół obiektów, na których kod działa. Uzyskanie większej kontroli nad kodem.
- Zastosowanie obiektowo-relacyjnego modelu danych prowadzi do zmniejszenia rozbieżności w modelach danych samej bazy danych i aplikacji bazodanowej napisanej w obiektowym języku programowania. Oba modele można oprzeć o te same pojęcia: *klasy* (*typu obiektowego*) i *instancji klasy* (*obiektu*).

Co to jest niezgodność pomiędzy modelem pojęciowym i modelem implementacyjnym?

Celem jest uzyskanie jak najmniejszej luki pomiędzy myśleniem o rzeczywistości a myśleniem o danych i procesach, które zachodzą na danych.



Mentalna percepcja świata rzeczywistego

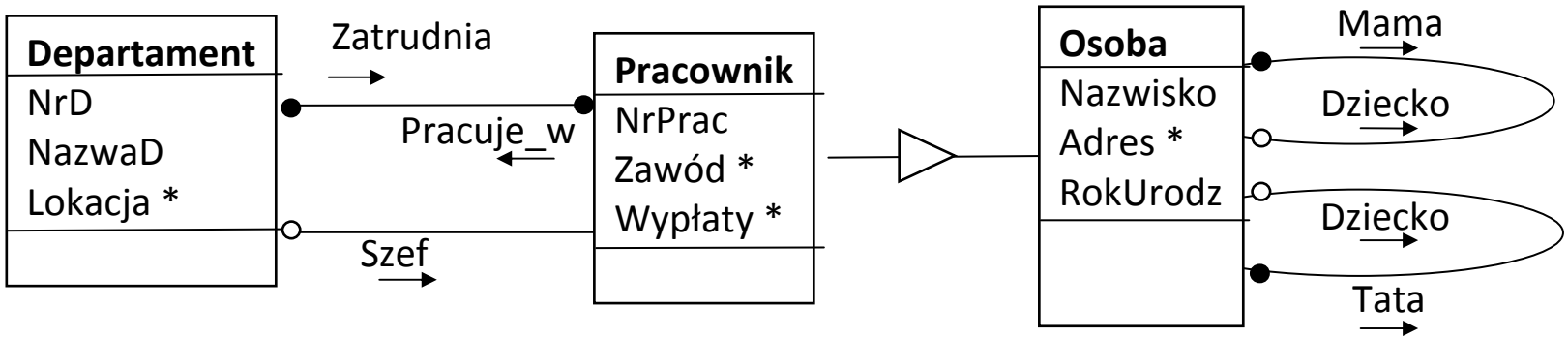
Model pojęciowy

Schemat relacyjnej struktury danych

W modelu relacyjnym model pojęciowy jest budowany w oparciu o model encja-związek. Model encja-związek stara się odwzorować świat rzeczywisty, lecz nie może być bezpośrednio zaimplementowany, gdyż relacyjna baza danych na to nie pozwala. W rezultacie,

- schemat struktury danych gubi znaczną część semantyki danych,
- użytkownik musi kojarzyć dane explicite w zdaniach SQL, co zwiększa ich złożoność i powoduje wzrost czasów wykonania.

Niezgodność modelu pojęciowego i relacyjnego



Ile schematów relacyjnych potrzeba, aby zaimplementować tę strukturę?

Departament(NrD, NazwaD)
Lokacja(NrLokacji, NazwaLok, NrD)
Szef(NrD, NrPrac)
Pracownik(NrPrac, NrOsoby)
PracDept(NrPrac, NrD)
Zawód(IdZawodu, NazwaZawodu, NrPrac)
Wypłata (IdWypłaty,Wysokość, NrPrac)
Osoba(NrOsoby, Nazwisko, RokUrodz)
Adres(NrAdresu, Miejsce, NrOsoby)
Mama(NrOsoby, NrOsoby)
Tata(NrOsoby, NrOsoby)

Czytelna pojęciowa struktura zamieniła się na 11 nieczytelnych schematów relacji

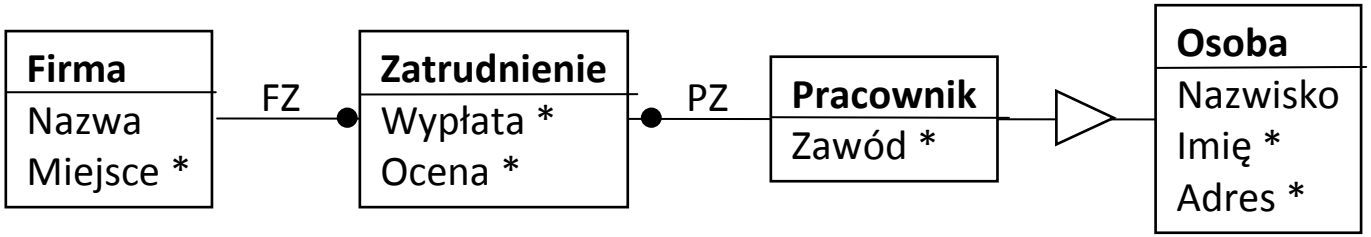
Pojawiły się nowe atrybuty - klucze

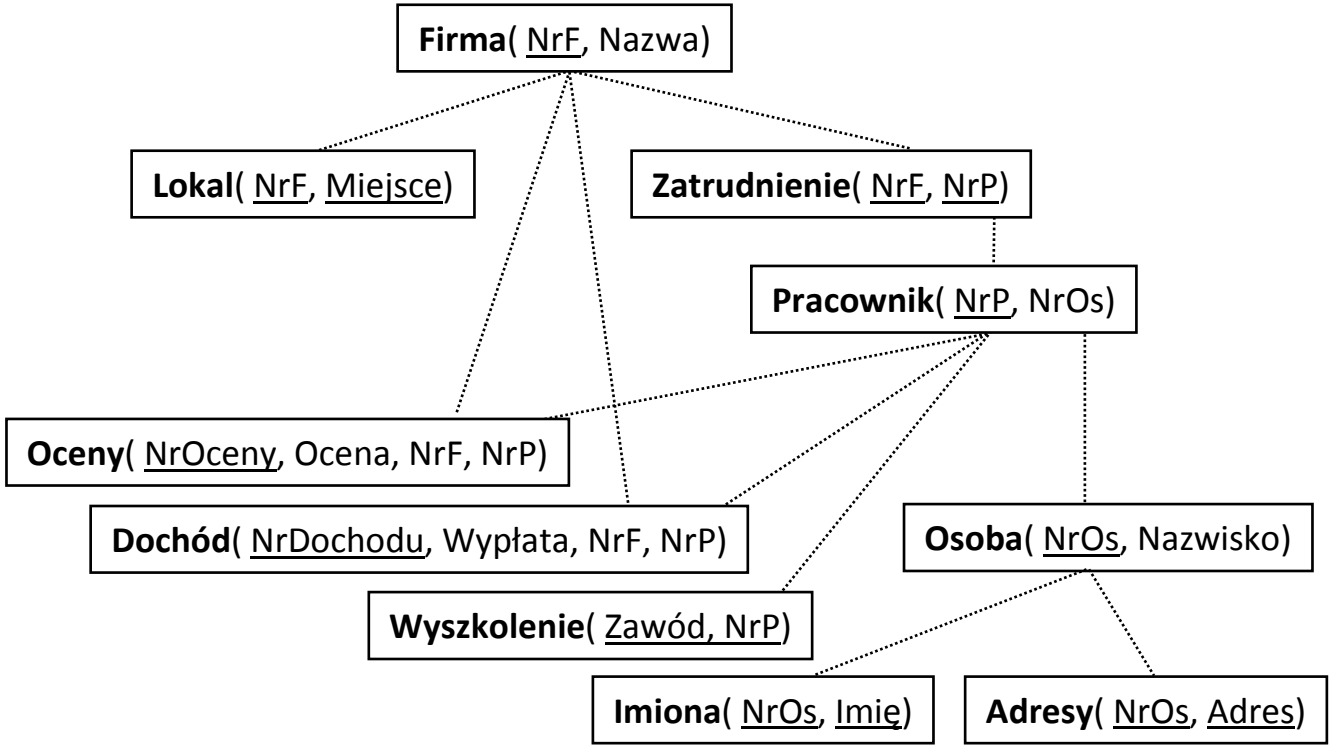
Semantyka wyrażona poprzez liczności została częściowo zgubiona

Semantyka dziedziczenia została zgubiona

Odtworzenie semantyki - użytkownik musi zrobić explicite poprzez zapytania SQL

Niezgodność modelu pojęciowego i relacyjnego





Zapytania w OQL i SQL

Podaj adresy pracowników zatrudnionych w firmach zlokalizowanych w Radomiu:

OQL:
Object Query Language – ODMG

```
select z.Adres
from Firma as x, x.FZ as y, y.PZ as z
where "Radom" in x.Miejsce
```

(26 jednostek leksykalnych)

SBQL:
Stack-Based Query Language

```
(Firma where "Radom" in Miejsce).FZ.Zatrudnienie.PZ.Pracownik.Adres
```

(17 jednostek leksykalnych)

SQL:

```
select a.Adres
from Lokal as k, Zatrudnienie as z, Pracownik as p, Osoba as s, Adresy as a
where k.Miejsce = "Radom" and k.NrF = z.NrF and z.NrP = p.NrP
        and p.NrOs = s.NrOs and s.NrOs =a.NrOs
```

(62 jednostki leksykalne)

Wady modelu relacyjnego

Z góry ustalony konstruktor typu danych (relacja), rozszerzany *ad hoc* przez wytwórców systemów relacyjnych. Brak złożonych obiektów. Informacje o pojęciach wyróżnialnych i manipulowalnych w rzeczywistości są rozproszone w krotkach wielu tablic.

Skojarzenie tych informacji następuje w zapytaniach SQL, przez co wzrasta ich złożoność oraz czas wykonania (optymalizacja zapytań tylko częściowo to rozwiązuje).

Brak wyspecjalizowanych środków do realizacji powiązań pomiędzy danymi.

Wady modelu relacyjnego

- Brak środków do przechowywania danych proceduralnych.

Wszelkie informacje wykraczające poza strukturę relacyjną (perspektywy, procedury bazy danych, BLOBy, aktywne reguły,...) są implementowane *ad hoc*.

- Brak środków hermetyzacji i modularyzacji: wykroczenie przeciwko zasadom abstrakcji i oddzielenia implementacji od specyfikacji.
- Brak uniwersalności środków dostępu do danych, powodujący konieczność zanurzenia ich w uniwersalne języki programowania, znacznie niższego poziomu; niezgodność impedancji (*impedance mismatch*). Niespełnione obietnice przetwarzania akroskopowego (*wiele-w-tym-samym-czasie*); powrót do niewygodnej techniki *jeden-w-tym-samym-czasie*.
- Niespójne mechanizmy wartości zerowych, brak wariantów, brak porządku w relacjach.

Obiekt

- konkretny lub abstrakcyjny byt posiadający nazwę, jednoznaczną identyfikację (OID), określone granice, atrybuty i inne własności oraz rodzaj struktur danych przetwarzanych przez obiektowe języki programowania oraz przechowywanych w bazie danych
- obiekt może być skojarzony z metodami lub operacjami, które na nim działają z reguły definiowanych/przechowywanych w ramach jego klasy oraz jej nadklas

Obiekt

- obiekt jest instancją klasy i posiada niepowtarzalny identyfikator
- obiekty mogą być rekurencyjnie powiązane między sobą związkami semantycznymi
- związki między obiektami reprezentowane są poprzez referencje (odwołania), które są wartościami atrybutów obiektu
- odwołanie do konkretnego obiektu w systemie jest realizowane przez wysłanie do niego komunikatu

Klasa

- byt semantyczny rozumiany jako miejsce przechowywania (specyfikacji i definicji) takich cech grupy podobnych obiektów, które są dla nich niezmienne: atrybutów, metod, ograniczeń dostępu, dozwolonych operacji na obiektach, wyjątków, itp.
- klasa stanowi wzorzec dla tworzonego obiektu i jest traktowana jako obiekt (klasowy), w celu zagwarantowania jednolitego posługiwania się komunikatami

Klasa

- z klasą mogą być związane atrybuty i metody klasowe; w atrybutach klasowych przechowywane są wartości podsumowujące wszystkie jej obiekty oraz atrybuty wspólne i atrybuty domyślne
- dwa ostatnie atrybuty są logiczną częścią każdej instancji klasy (każdego obiektu)
- w obiektowej bazie danych, dziedziną dowolnego atrybutu może być klasa, co zwiększa semantyczną spójność bazy

Klasa

- klasa (lub kilka klas), jako agregacja powiązanych ze sobą obiektów w bazie danych, stanowi cel do sformułowania zapytania
- zwykle klasy wiąże się ze sobą poprzez hierarchię (lub inną strukturę) dziedziczenia

Atrybut

- część definicji klasy, specyfikacja atrybutu polega na podaniu jego nazwy i więzów semantycznej spójności, obejmujących: dziedzinę atrybutu, itp.
- wartości atrybutów obiektu opisują jego stan
- dziedziną atrybutu może być jakakolwiek klasa ze swoim własnym zbiorem atrybutów lub klasa wartości pierwotnych (np.: integer, string)

Atrybut

- wartością atrybutu może być instancja klasy będącej jego dziedziną lub instancja dowolnej podklasy z hierarchii klas zakorzenionej w klasie stanowiącej dziedzinę atrybutu

Metoda

- procedura, funkcja lub operacja przypisana do klasy i dziedziczona przez jej podklasy
- metoda działa na stanie obiektu tej klasy (czyli operuje wartościami atrybutów)
- kod w języku programowania implementujący metodę nazywamy ciałem metody
- metoda abstrakcyjna specyfikowana jest w klasie, ale jej działanie może być przedefiniowane w dowolnej z jej podklas

Hermetyzacja

- zamknięcie pewnego zestawu bytów programistycznych w "kapsułę" (obiekt, klasę moduł, etc.) o dobrze określonych granicach.
- hermetyzacja jest podstawową techniką abstrakcji, tj. ukrycia wszelkich szczegółów danego przedmiotu lub bytu programistycznego, które na danym etapie rozpatrywania (analizy, projektowania, programowania) nie stanowią jego istotnej charakterystyki

Hermetyzacja

- ortodoksyjna hermetyzacja (Smalltalk) dopuszcza tylko operacje (które można wykonać na obiekcie) określone przez metody przypisane do obiektu; bezpośredni dostęp do atrybutów obiektu jest niemożliwy
- hermetyzacja ortogonalna (C++, Eiffel) dopuszcza by dowolny atrybut obiektu (i dowolna metoda) mógł być prywatny (niedostępny z zewnątrz) lub publiczny

Agregacja

- związek pomiędzy klasami obiektów, modelujący stosunek całości do jej części (np. stosunek wydziału produkcyjnego do agregatów)
- obiekty są powiązane związkiem agregacji, jeżeli jeden z nich można uważać za część drugiego, zaś cykl i czas życia tych obiektów są jednakowe

Hierarchia klas i dziedziczenie

- klasy w systemie tworzą zakorzeniony, skierowany acyklicznie graf zwany hierarchią klas
- oznacza to, że dla klasy C klasa (lub klasy) S na niższym poziomie jest uszczegółowieniem (specjalizacją) klasy C i odwrotnie - klasa C jest uogólnieniem (generalizacją) klasy (klas) S
- klasa S dziedziczy wszystkie atrybuty i metody klasy C, mogąc jednocześnie charakteryzować się własnymi atrybutami i metodami

Hierarchia klas i dziedziczenie

- atrybuty i metody określone dla klasy C są rekurencyjnie dziedziczone przez wszystkie jej podklasy
- odziedziczone metody mogą zostać przeciążone, tzn. można zmodyfikować działanie odziedziczonej metody nie zmieniając jej nazwy

Object Database Management Group

- Pierwsza wersja standardu ODMG (Object Database Management Group) została sformułowana w 1993 roku:
- standard ODMG obejmujący definicje modelu danych
- język definicji danych ODL (Object Definition Language)
- język zapytań OQL (Object Query Language)
- definicja powiązań z językami C++ i Smalltalk

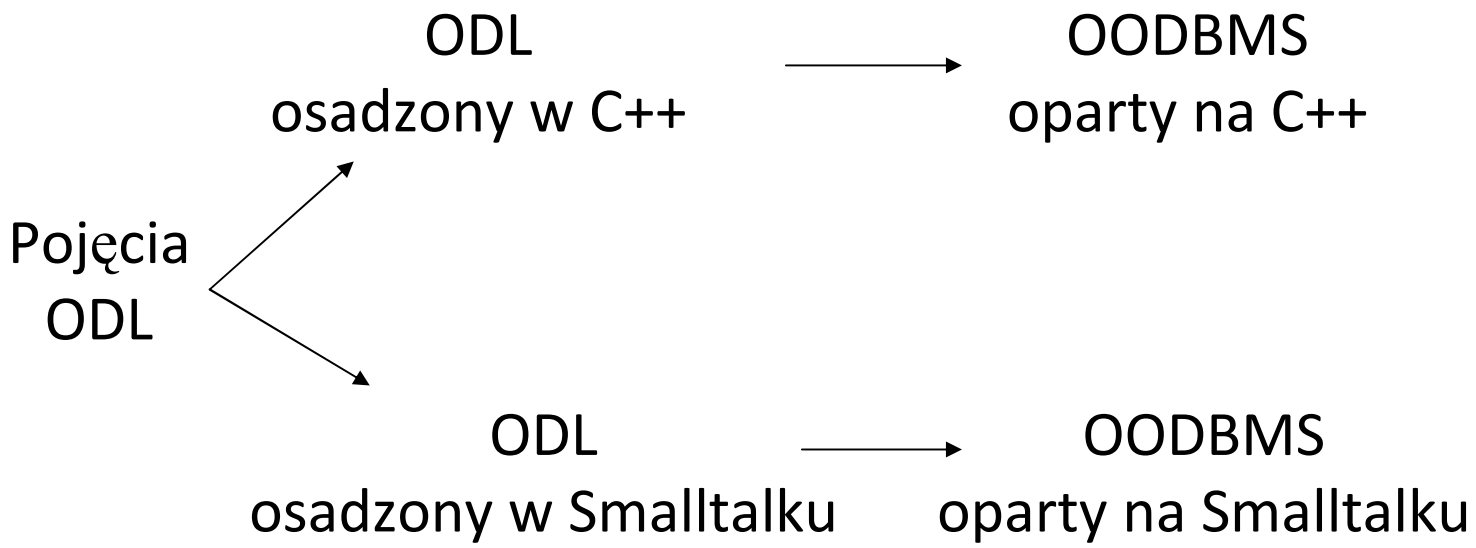
Object Database Management Group

- Podstawą modelu ODMG stał się model OMG (Object Management Group), składnia ODL jest rozszerzeniem IDL (Interface Definition Language) składnika standardu CORBA (*common object request broker architecture*)
- Standard ODMG stał się standardem producentów komercyjnych baz danych
- Członkami ODMG są zespoły: Objectivity, GemStone, Object Design, O2Technology, Poet, Versant

Język ODL

- Język definiowania obiektów (object definition language) ODL należy do klasy języków definicji danych
- ODL jest przeznaczony do specyfikowania schematu lub struktury bazy danych
- ODL musi być przetłumaczony na deklaracje w OODBMS

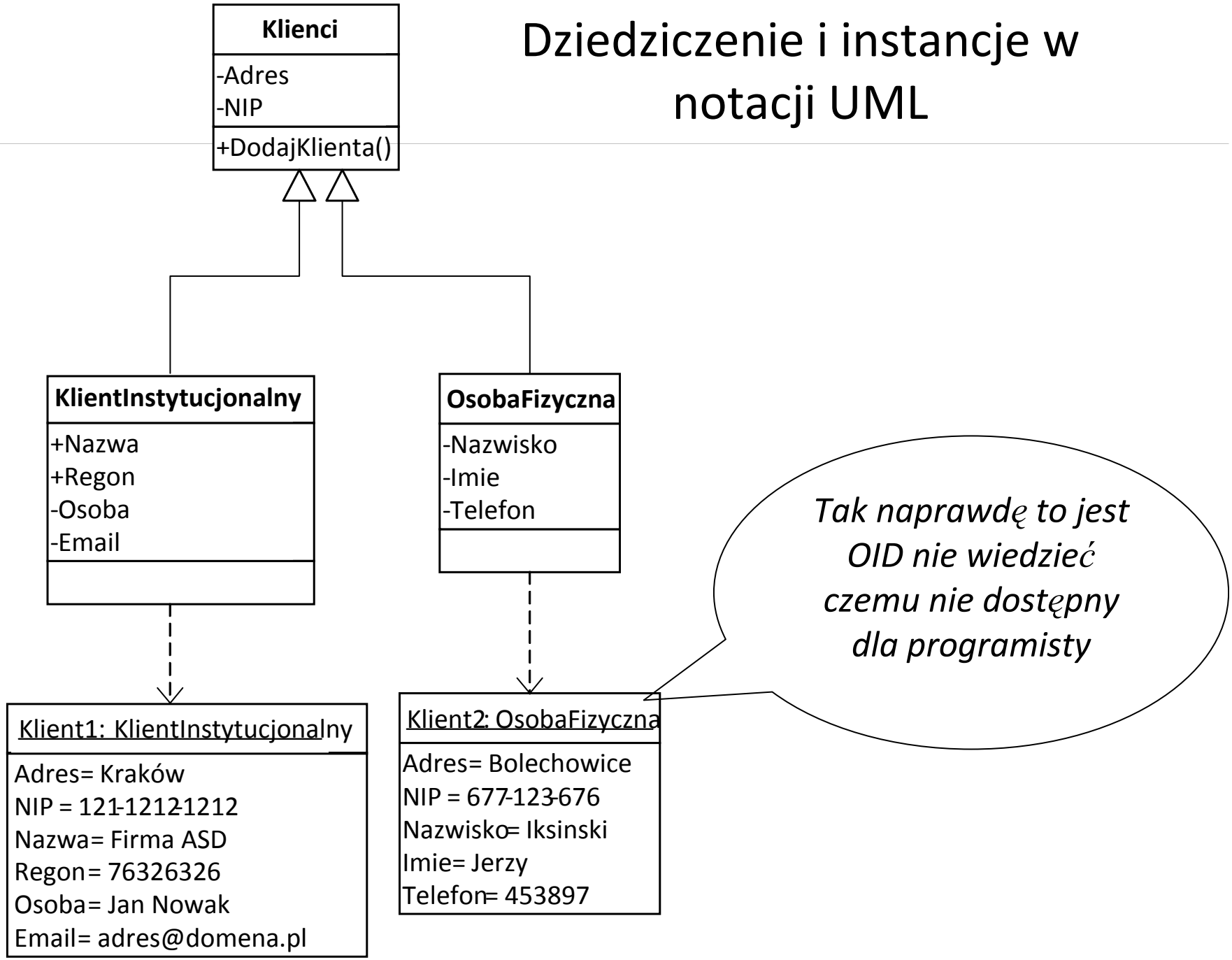
ODML → OODBMS


ODL
osadzony w C++
OODBMS
oparty na C++
Pojęcia
ODL
ODL
osadzony w Smalltalku
OODBMS
oparty na Smalltalku

UML

- Unified Modeling Langauge stanowi notację łączącą elementy trzech najpopularniejszych metod projektowania obiektowego (Rumbaugh, Booch, Jacobson)
- UML został przyjęty jako standardowa notacja dla diagramów obiektowych przez grupę OMG
- Za specyfikację języka odpowiada obecnie firma Rational Software

Dziedziczenie i instancje w notacji UML

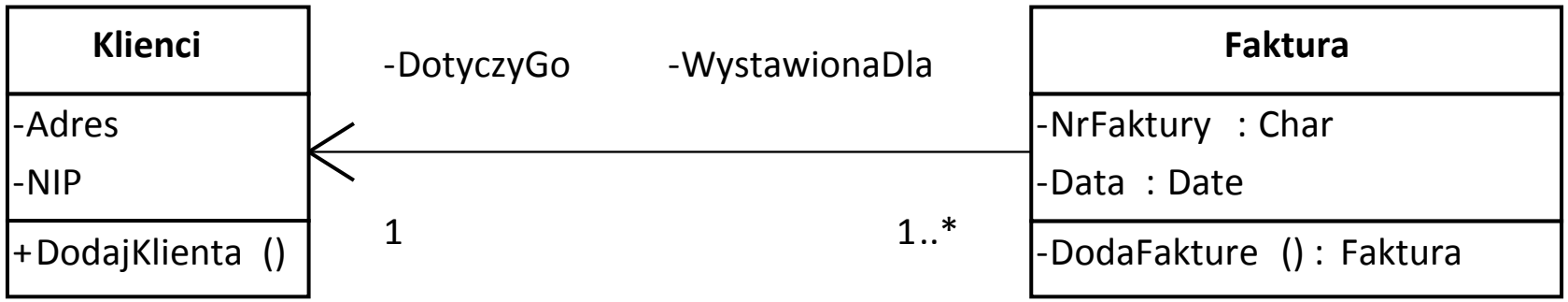


Dziedziczenie w ODL

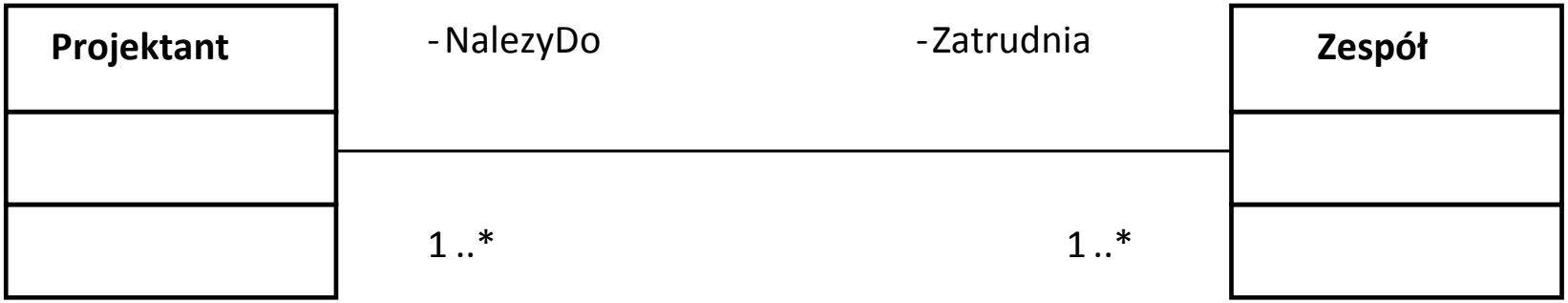
```
interface Klienci{
        attribute string Adres;
        attribute string NIP;
        DodajKlienta();
        };

interface KlientInstytucjonalny: Klienci{
    attribute string Nazwa;
    attribute string Regon;
    attribute string Osoba;
    attribute string Email;
    };
```

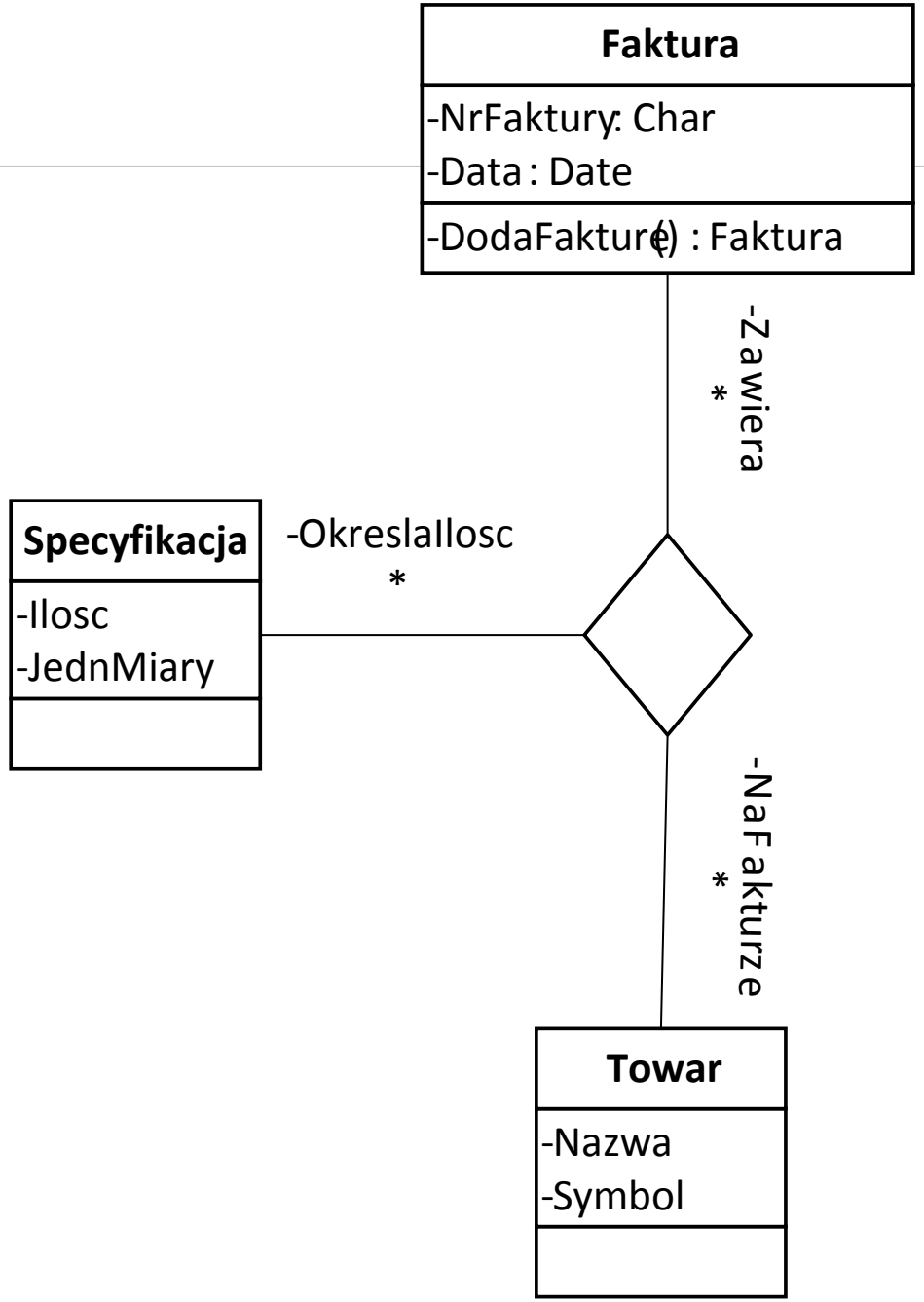
Asocjacje w notacji UML



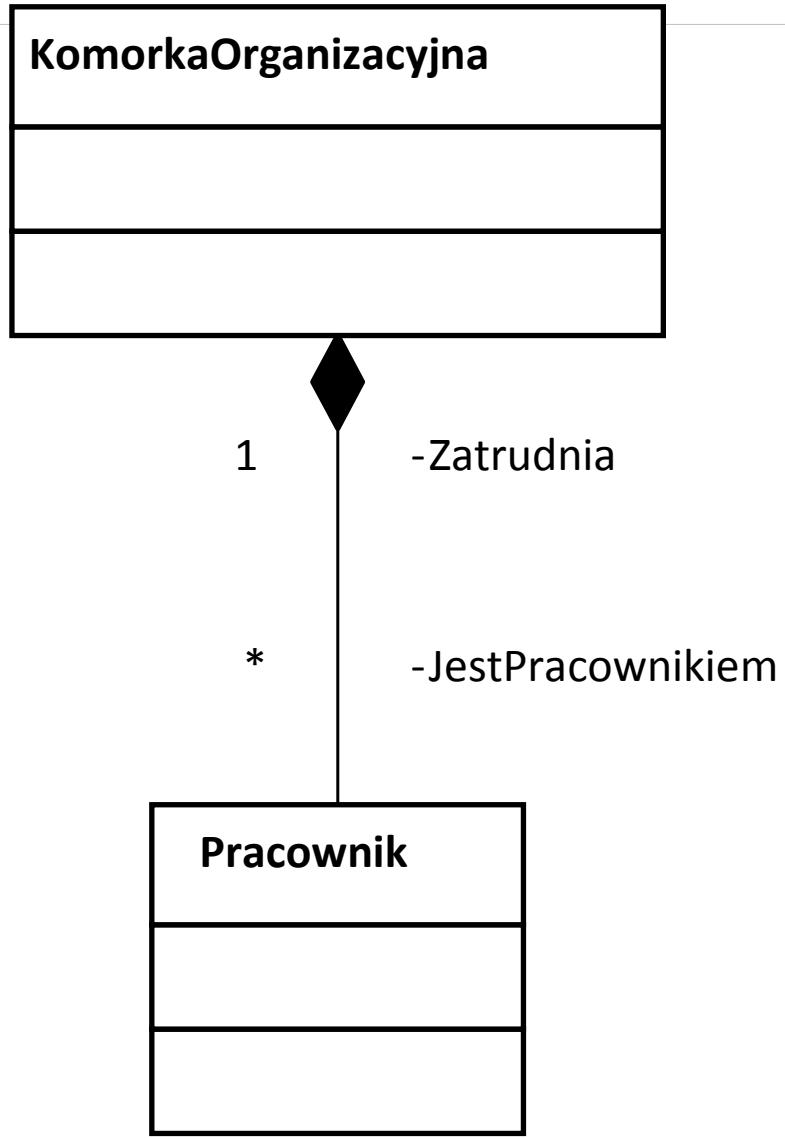
Związek wiele-do-wielu w notacji UML



Klasa skojarzona w notacji UML



Agregacja w notacji UML



Metody w ODL

- Faktura DodajFakture(NrFaktury:’123/05’, Data:20-05-2005, WystawionoDla:x)
- gdzie x jest zmienną, której wartością jest OID obiektu klasy Klienci

Dostawcy obiektowych baz danych

- Birdstep Technology, Inc. (Birdstep RDM Mobile)
- Computer Associates International, Inc. (Jasmine Object Database)
- Custom Microsystems, Inc. (XDb)
- db4objects, Inc. (db4o - database for objects - open source)
- GemStone Systems, Inc. (GemStone/S and GemStone Facets)
- GLinkNET Technology (Link.NET)
- InterSystems Corporation (Cache´)
- Jade Software Corporation (JADE)

Dostawcy obiektowych baz danych

- eXcelon (dawniej Object Design) – produkt ObjectStore
- Objectivity – produkt Objectivity/DB
- Versant
- Poet Software – produkt FastObjects
- GemStone – produkty: GemStone/J, GemStone/S
- Fresher Information Software – produkt Matisse
- Computer Association oraz Fujitsu – produkt Jasmine
- Uniwersytet Moskiewski – produkt GOODS
- Micro Database Systems – produkt Titanium

Dostawcy obiektowych baz danych

- JYD Software Engineering Pty Ltd. (JYD Object Database)
- Logic Arts, Ltd. (VOSS)
- Matisse Software, Inc. (Matisse)
- ObjectDB Software (ObjectDB for Java/JDO)
- Orient Technologies (Orient Enterprise Edition and Orient Just Edition)

Dostawcy obiektowych baz danych

- Orient Technologies (Orient Enterprise Edition and Orient Just Edition)
- The Ozone Database Project (ozone - open source)
- Progress Software Corporation (ObjectStore, PSE Pro, and ObjectStore Event Engine)
- run Software-Werkstatt GmbH (ODABA)
- Sorceforge hosted open source (Prevayler - open source)
- Sysra (EyeDB)
- Versant Corporation (Versant Developer Suite, Versant enJin) -- Versant Corporation has merged with POET Holdings (FastObjects, Inc.).

Język OQL

- Obiektowy język zapytań jest odpowiednikiem SQL dla OODBMS
- OQL jest standardem OMG
- Object Query Language (OQL) bazuje na standardzie SQL wraz z instrukcją SELECT co zapewnia możliwość używania tradycyjnych konstrukcji rozszerzonych o dostęp do obiektów, identyfikatorów, dziedziczenia, gron i metod oraz śledzenie powiązań

Język OQL

- Zapytania w OQL sa traktowane jako łańcuchy przez pewien zestaw funkcji C++ lub Smalltalk
- Ciąg znaków reprezentuający zapytanie w OQL może byc parametryzowany; parametry są poprzedzane znakiem $

Język OQL

- ponieważ w języku ODL nie ma pojęcia relacji jako źródła danych konieczne jest zdefiniowanie zbioru obejmującego bieżące obiekty tej klasy
- deklaracja obejmuje słowo kluczowe *extent*, po której podaje się nazwę zasięgu
- w deklaracji zasięgu można podać klucze pozwalające identyfikować obiekty w zasięgu

Zapytania w OQL

- zapytania OQL nie wymagają specyfikacji warunków złączeń, które zdefiniowane są już w samej strukturze klas
- podobnie jak w zawansowanych wersjach SQL zmienna występująca po słowie kluczowym FROM może być dowolnym wyrażeniem oznaczającym kolekcję
  (np. może to być wyrażenie oznaczające kolejne wyrażenie typu select-from-where)

Przykład zapytania w OQL

```
SELECT a.NrFaktury, a.Data
FROM  ZakresKlientow b, b.Faktury a
WHERE b.NIP = „123-432-432”
```

- zapytanie wyświetla numery i daty faktur dla klienta, którego nr NIP jest równy „123-432-432”
- w zapytaniu nie podaje się warunków złączenia, które zapisane są w strukturach klas FAKTURA i KLIENCI

Inne elementy OQL

- Większość elementów OQL jest podobna do metod SQL, np.
  - DISTINCT,
  - podzapytania,
  - ORDER BY,
  - EXIST,
  - GROUP BY
- niektóre rozszerzają standard SQL, np.:
  - FOR ALL

Obiektowo-relacyjne bazy danych

- Standard SQL3 dopuszcza pewne elementy charakterystyczne dla koncepcji obiektowej, np.:
  - obiekty krotkowe,
  - referencje
  - abstrakcyjne typy danych

Typ wiersza

- Standard SQL3 dopuszcza definiowanie typów krotek, które w dużym uproszczeniu odpowiadają pojęciu klas obiektów
- Typy krotek mogą być deklaracjami struktur co zapewnia zagnieżdżanie struktur niedostępne w standardzie SQL2

Definiowanie typów

```
CREATE ROW TYPE Adres (
    kod_pocztowy CHAR(6),
    miasto CHAR(50),
    ulica CHAR(20)
    );

CREATE ROW TYPE Pracownik (
    nazwisko CHAR(30),
    adres Adres,
    );
```

Deklarowanie i dostęp do relacji z typem wiersza

```
CREATE TABLE Pracownicy OF TYPE Pracownik
CREATE TABLE Dostawcy OF TYPE Pracownik
CREATE TABLE Kontrahenci OF TYPE Pracownik


SELECT Pracownicy.nazwisko,
            pracownicy.adres.ulica
FROM Pracownicy
```

Referencje

- Identyczność obiektów z języków obiektowych w języku SQL3 jest zapewniona dzięki pojęciu referencji
- Typem składowej w typie wiersza może być referencja do innego typu wiersza
- Gdy traktujemy krotkę jako obiekt, wówczas referencją krotki do tego obiektu jest jego ID

Referencje

- SQL3 nie dopuszcza by referencją był zbiór obiektów
- Nie jest więc możliwe utworzenie reprezentacji związku typu wiele-do-wielu bez tabeli pomocniczej

Deklarowanie referencji

```
CREATE ROW TYPE Faktury(
  NrFaktury CHAR(20),
  DataFaktury DATE);

CREATE ROW TYPE Towary(
  NazwaTowaru CHAR(20),
  Cena FLOAT;

CREATE ROW TYPE Specyfikacja(
  Ilosc FLOAT,
  zFaktury REF(Faktury),
  Towar REF(Towary);
```

Deklarowanie referencji i dostęp do nich

```
CREATE TABLE Rachunki OF TYPE Faktury
CREATE TABLE Produkty OF TYPE Towary
CREATE TABLE Linie OF TYPE Specyfikacja

SELECT Linie→Ilosc*Towary →Cena
FROM Linie
WHERE Rachunki →DataFaktury = '2005-05-24'
```

Wnioski

- OBD zapewniają zgodność ze standardem programowania obiektowego
- ułatwiają zapis referencji (związków pomiędzy obiektami)
- przyspieszają operacje wyszukiwania
- ułatwiają projektowanie złożonych systemów dzięki enkapsulacji

Wnioski

- istotną trudnością jest zapewnienie integralności referencyjnej
- brak modelu matematycznego zapewniającego nienaruszalność standardu i zgodność aplikacji
- brak możliwości standaryzacji

Wady obiektowych baz danych

- Istnienie wskaźników w strukturach obiektowych cofa rozwój do czasów systemów sieciowych (mit!). W starych systemach sieciowych programista posługiwał się wskaźnikami fizycznymi explicite, co miało liczne wady. W systemach obiektowych wskaźniki mają charakter pojęciowych asocjacji, które poprawiają percepcję schematów baz danych, znacznie upraszczają zapytania i programy, upraszczają pielęgnację oprogramowania, redukują zapotrzebowanie na pamięć i umożliwiają znaczną poprawę czasów wykonania poprzez bezpośrednią nawigację wzdłuż wskaźników lub np. poprzez technikę przemiany wskaźników (pointer swizzling).
- Brak opracowanych mechanizmów zarządzania dużą bazą obiektów, sterowania wersjami, rejestrowania zmian, zapewnienia stabilności interfejsów.
- Duża liczba tematów związanych z obiektowością znajduje się wciąż w fazie laboratoryjnej, co powoduje, że wiele technologii jest mało stabilnych.
- Przejście na technologie obiektowe może zagrozić funkcjonowaniu obecnie działających i sprawnych systemów, które są krytyczne dla misji organizacji.

Obiektowość - potencjalne ryzyko

Technologie obiektowe są jak dotąd stosowane przez małe i średnie organizacje. Nie jest do końca pewne jak przeskalują się dla wielkich organizacji. Duża liczba tematów znajduje się ciągle w fazie laboratoryjnej. Szereg technologii jest mało stabilnych (np. metodyki projektowania).

Przejście na technologie obiektowe może zagrozić funkcjonowaniu obecnie działających i sprawnych systemów, które są krytyczne dla misji organizacji.

Zbyt mała liczba ekspertów

Nie jest jasne, jakie koszty pociągnie za sobą przejście na technologie obiektowe.

Standardy w zakresie obiektowości są niedopracowane i niestabilne. Nie wiadomo w jakim zakresie będą one pełnić swoją funkcję.

Obiektowość kontra model relacyjny
Warstwa technologii

- Sprzymierzeńcem wszystkich wdrożonych technologii baz danych, w tym relacyjnych, jest ogromna bezwładność rynku zastosowań, który niechętnie zmienia swoje preferencje, szczególnie jeżeli zostały zainwestowane duże pieniądze i czas.
- Systemy relacyjne opanowały dużą grupę „nisz ekologicznych” i można przyjąć, że pozostaną w nich przez kilka, kilkanaście, lub nawet kilkadziesiąt lat.
- Systemy obiektowe muszą poszukiwać innych nisz, które nie są zagospodarowane przez wcześniejsze technologie. Potencjalnym polem zastosowań obiektowych baz danych są multimedia, dziedziny wymagające bardziej rozbudowanych struktur danych, takie jak CAD (Computer Aided Design), CAM (Computer Aided Manufacturing), CASE, lub dziedziny implikujące nieregularne, niesformatowane struktury danych, takie jak zastosowania pełno-tekstowe, hurtownie danych, zastosowania biurowe.

Obiektowość kontra model relacyjny
Warstwa technologii

Prawie wszystkie systemy relacyjne przechodzą etap radykalnych modernizacji (m.in., a może przede wszystkim) w związku z wyzwaniem jakie stawiają technologie obiektowe.

Następuje wzmocnienie struktur danych przechowywanych w systemach relacyjnych o nowe możliwości. Rozszerzenia te wprowadzają wiele elementów obiektowości.

Wiele systemów relacyjnych jest wyposażane w możliwość współdziałania z innymi systemami (w tym nie-relacyjnymi) według standardu obiektowego OMG CORBA.

Ponad 90% programów rzekomo „zgodnych” ze standardem SQL-92 nie daje się przenieść na inne platformy SZBD, głównie ze względu na niepełność tych standardów oraz niekompatybilne rozszerzenia implementowane w poszczególnych systemach.

Specjaliści powątpiewają co do implementowalności SQL3

Nie istnieje własność systemów relacyjnych, która nie mogłaby być równie dobrze zrealizowana w systemach obiektowych

Wbrew rozpowszechnianym mitom, obiektowość sprzyja wydajności m.in. poprzez technikę przemiany wskaźników (*pointer swizzling*), indeksy ścieżkowe (*path indices*), oraz poprzez nowe mechanizmy indeksacji i buforowania umożliwiającymi istotne przesunięcie przetwarzania na stronę klienta w architekturze klient-serwer.

Obiektowość kontra model relacyjny
Warstwa ideologii

Model relacyjny przegrał konfrontację z obiektowością w warstwie ideologicznej. Dzisiaj nie jest widoczny liczący się ośrodek naukowy, który aktywnie rozwijałby model relacyjny od strony ideologiczno-naukowej.

Główną wadą koncepcyjną modelu relacyjnego jest to, co miało być jego zaletą, mianowicie prostota struktur danych. Informacje o pojęciach wyróżnialnych w rzeczywistości są rozproszone w krotkach wielu tablic, co burzy związek pomiędzy pojęciowym obrazem świata, a pojęciowym obrazem struktur danych modelujących ten świat.

Model relacyjny nie zajmuje się informacją proceduralną, która często jest istotną składową obrazu pojęciowego danych. Wobec braku systematycznych środków, wszelkie informacje wykraczające poza strukturę relacyjną są implementowane ad hoc.

Ideologia relacyjna jest oparta na naiwnych poglądach co do środków przetwarzania informacji. Trzonem tych środków miały być operatory algebry relacji lub rachunku relacyjnego; niestety, okazały się one zbyt mało uniwersalne. Duża część tego przetwarzania musiała być oddelegowana do uniwersalnych języków programowania, co doprowadziło do niekorzystnego efektu niezgodności impedancji. W konsekwencji postulowane przez model relacyjny przetwarzanie makroskopowe wróciło do niesławnego przetwarzania niskiego poziomu, np. poprzez kursory w zagnieżdżonym SQL.

Obiektowość kontra model relacyjny
Warstwa teorii

Nieautentyczny, dekoracyjny charakter relacyjnych teorii. Są pseudo-naukowymi fasadami.

Zastosowanie teorii zależności funkcjonalnych, wielowartościowych i innych, sprowadza się do definicji kilku pojęć (2NF, 3NF,...) o znaczeniu marginalnym dla praktyki projektowania.

Oparcie semantyki języków zapytań takich jak SQL o algebrę relacji lub rachunek relacyjny jest pomysłem całkowicie nieudanym. Teorie te są niedostatecznie uniwersalne dla SQL.

Mitem są twierdzenia, że metody optymalizacji zapytań są konsekwencją odkrytych własności algebry relacji.

Całkowite fiasko badań nad niekompletną informacją (setki publikacji!). Rozwiązania w tym zakresie są pozbawione logiki i konsekwencji. Całkowite fiasko rozszerzeń teoretycznych takich jak uniwersalne relacje i dedukcyjne bazy danych (setki publikacji!).

W systemach „post-relacyjnych” lub „obiektowo-relacyjnych” termin „relacyjny” ma wyłącznie znaczenie tradycyjno-dekoracyjne. Systemy te, poprzez rozbudowę własności struktur danych i języków przetwarzania danych, nie są w stanie skorzystać z własności matematycznego pojęcia relacji w duchu relacyjnego modelu danych.

Twierdzenia, że model relacyjny posiada „solidne podstawy matematyczne” zaś obiektowość ich nie posiada są fałszywymi stereotypami i popisami demagogów. Zarówno systemy relacyjne, jak SQL, jak i systemy obiektowe nie mają istotnej teorii.

Częste mity przy porównaniach

- **Model relacyjny i systemy relacyjne w unikalny sposób sprzyjają implementacji rozproszonych baz danych**.
- Mit uzasadniany powierzchownymi cechami, takimi jak możliwość „łatwej” dekompozycji relacyjnej bazy danych na składowe lub dekompozycji relacji na pod-relacje. Te cechy nie są w stanie uprościć zasadniczych problemów rozproszenia danych, takich jak przezroczystość rozproszenia, rozproszone transakcje, przetwarzanie zapytań w sytuacji rozproszenia, replikacje, autonomia lokalnych baz danych, osłony, mediatory i perspektywy do spadkowych baz danych, itd.
- Nie widać powodów, dla których w obiektowych bazach danych te problemy miałyby być cięższe. Pojawienie się standardów takich CORBA, DCOM i JavaBeans sugeruje, że dalszy rozwój rozproszonych baz danych będzie odbywał się w oparciu o technologie obiektowe.

Częste mity przy porównaniach

- **Obiektowe języki zapytań nie posiadają sprawnych metod optymalizacyjnych.**
- Demagogia, z kilku powodów:

  (1) Podstawowym celem metod optymalizacyjnych w systemach relacyjnych jest kosztowna operacja złączenia. Ponieważ w systemach obiektowych złączenie jest najczęściej zmaterializowane w postaci wskaźników (asocjacji), zapotrzebowanie na tę operację jest znacznie zredukowane;

  (2) Jak wykazują testy, optymalizacja relacyjnych języków zapytań nie zawsze jest tak sprawna, jak to jest podkreślane w literaturze;

  (3) Metody oparte na przepisywaniu (rewriting), np. przesuwanie selekcji i projekcji przed złączenie, działają tak samo dla języków relacyjnych jak i obiektowych;

  (4) Nie można wskazać powodów, dla których metod sprawnych w systemach relacyjnych nie można zastosować w systemach obiektowych;

  (5) Ortogonalność i bardziej precyzyjna semantyka OQL (w stosunku do SQL) stwarza większy potencjał dla metod optymalizacyjnych;

  (6) Systemy obiektowe są o ok. 15 lat młodsze od relacyjnych, a mimo to wiele testów pokazuje, że są znacznie od nich sprawniejsze (niekiedy tysiące razy).

Częste mity przy porównaniach

- **Obiektowość jest dobra na etapie analizy i projektowania, ale jest mało przydatna dla implementacji.**
- Twierdzenie to wynika z nieprzystosowania zastosowanych narzędzi implementacyjnych (np. relacyjnych baz danych, baz dokumentów typu Lotus Notes, itp.) do pojęć i technik obiektowych.

Projektowanie bazy danych
z użyciem diagramów UML

OBIEKTOWE PROJEKTOWANIE RELACYJNEJ BAZY DANYCH

Czym jest UML?

- UML (Unified Modeling Language) – Ujednolicony Język Modelowania.
- Jest to notacja wykorzystywana do budowania modeli.
- Uzywa się jej do opisu świata obiektów w analizie obiektowej oraz programowaniu obiektowym.
- Główne zastosowanie ma w informatyce, gdzie wykorzystuje się ją
- do tworzenia modeli systemów informatycznych.
- UML jest używany wraz z jego reprezentacją graficzną.
- Oznacza to, że jego elementom są przypisane symbole wiązane ze sobą na diagramach. W UML 2.0 wyróżnia się 13 diagramów głównych i 3 abstrakcyjne.

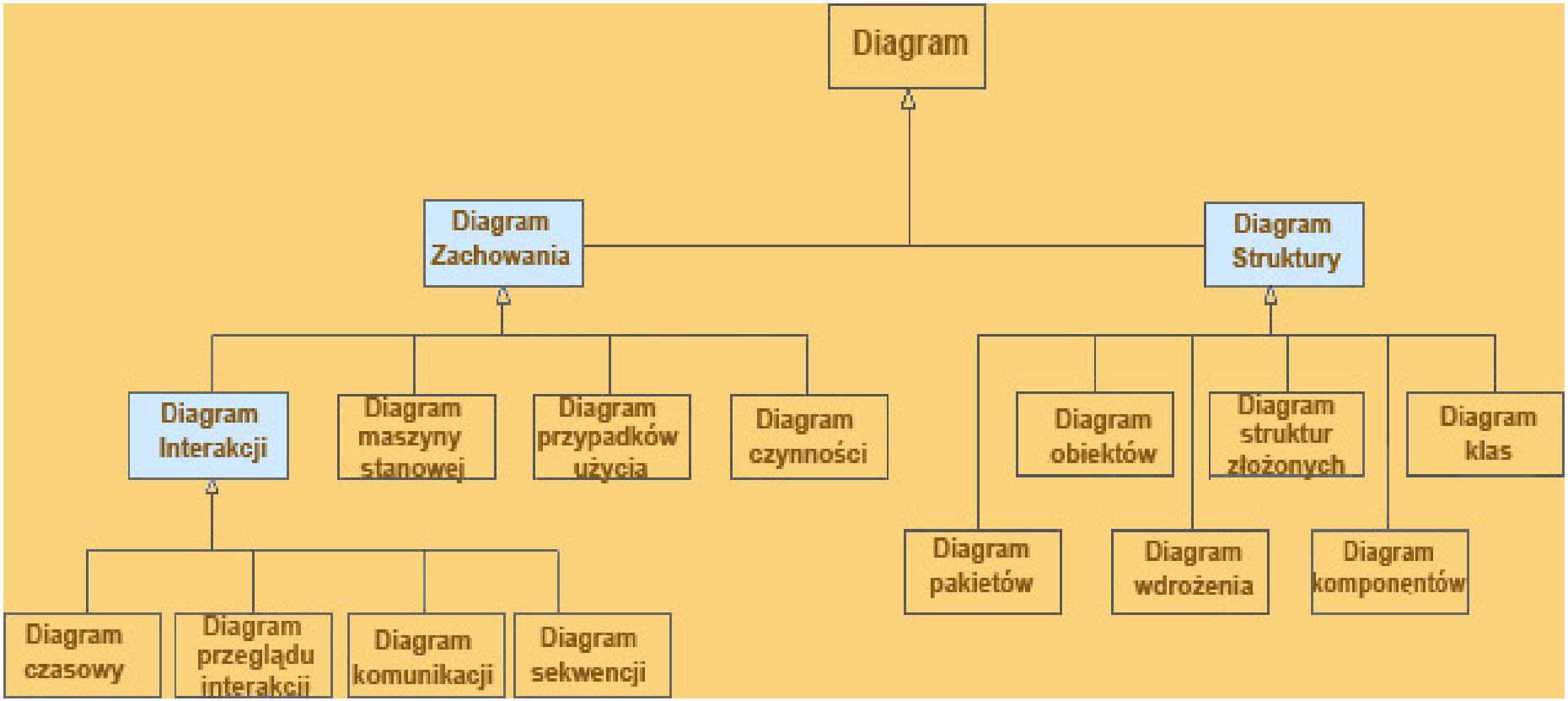


Diagram
Diagram Zachowania
Diagram Struktury
Diagram Interakcji
Diagram maszyny stanowej
Diagram przypadków użycia
Diagram czynności
Diagram obiektów
Diagram struktur złożonych
Diagram klas
Diagram pakietów
Diagram wdrożenia
Diagram komponentów
Diagram czasowy
Diagram przeglądu interakcji
Diagram komunikacji
Diagram sekwencji

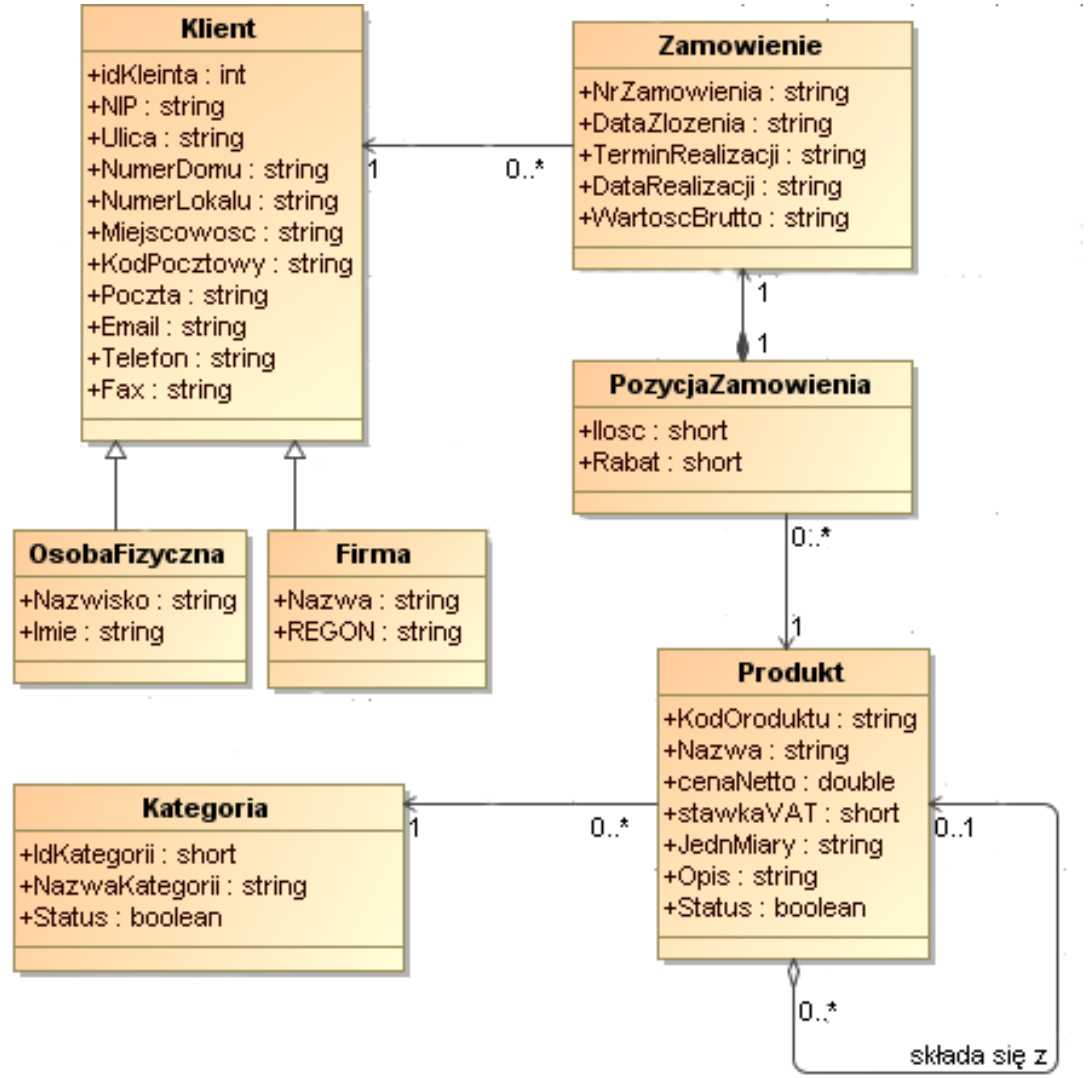
Zalety stosowania notacji UML

- Jakość rozwiązania – Ułatwione staje się zrozumienie problemu
- Przemyślane upraszczanie rzeczywistości – wysoka elastyczność i skalowalność.
- Możliwość korzystania ze sprawdzonych wzorców
- Ułatwione korzystanie z narzędzi wspomagających budowę systemów informatycznych.
- Jednolitość podejścia w obrębie całego projektu.
- Diagramy klas mają większą siłę ekspresji niż diagram ERD, gdyż udostępniają one bogatszy zestaw związków i pozwalają na modelowanie zachowania klas.

UML a Bazy Danych

- Coraz więcej aplikacji jest tworzonych zgodnie z podejściem obiektowym, natomiast brak jest obiektowych baz danych.
- Próba pogodzenia modelu obiektowego z relacyjnym.
- Projektowanie bazy danych za pomocą diagramu klas.
- Przekształcenie diagramu klas do schematu relacyjnej bazy danych.

Schemat przykładowej bazy – sklep



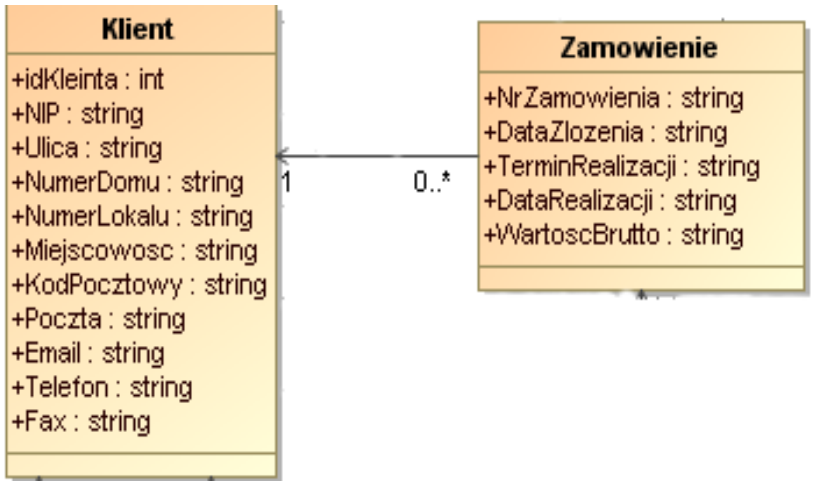
Oznaczenia liczebności

1	- dokładnie jeden
0..1	- zero lub jeden
0..*	- dowolnie wiele
1..*	- co najmniej jeden
*	- wiele
6..8	- od 6 do 8

- Na pierwszy rzut oka model UML nie różni się znacząco od schematu ERD. Obydwa zawierają:
  - Typy obiektów (klasy, encje) i powiązania między nimi;
  - Statyczne właściwości są opisane przez atrybuty;
- Diagram klas rozszerza diagram ERD m.in. o możliwość reprezentowania metod - operacji na obiektach;
- Diagram UML rozszerza zestaw związków o podane poniżej.
  - Asocjacja
  - Agregacja
  - Złożenie
  - Uogólnienie

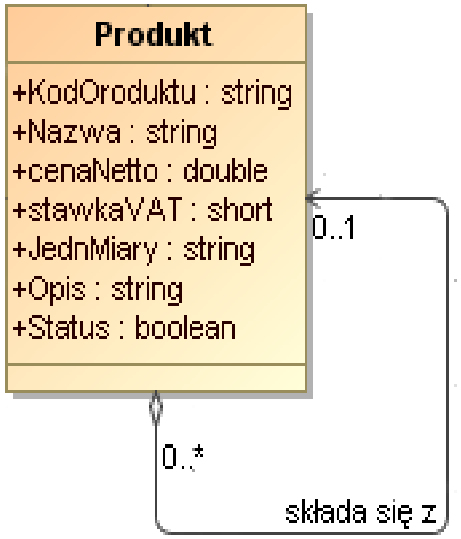
ASOCJACJA

– ASOCJACJA (association) - zwykły, równorzędny związek między dwoma klasami.



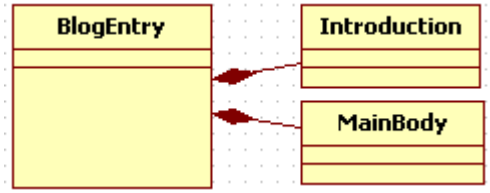
AGREGACJA

– AGREGACJA (aggregation)– specjalny rodzaj asoscjacji, który wyraża relację typu całość-część; relacja ta jest separowalna tzn. Klasa agregowana może istnieć bez klasy agrefującej.



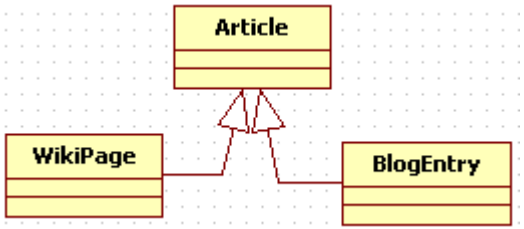
ZŁOŻENIE

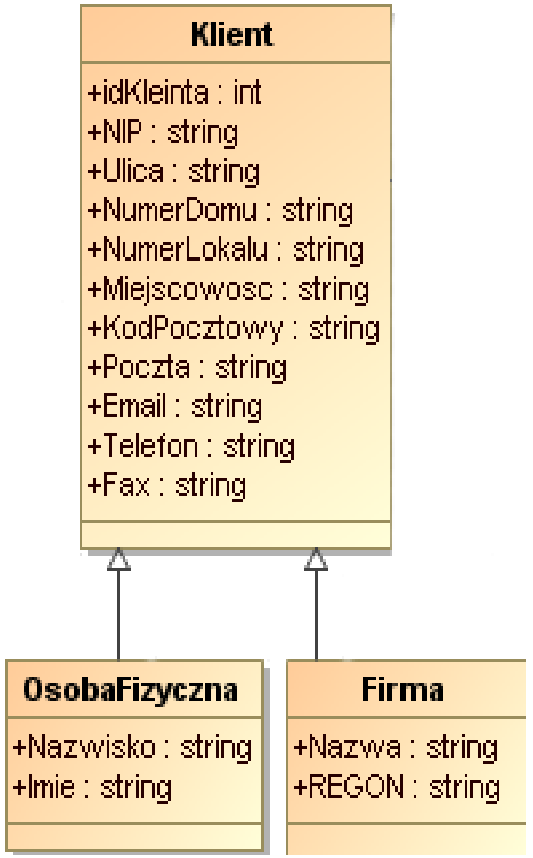
– ZŁOŻENIE (composition) – związek całość-część. Podobny do agregacji z tą różnicą, że klasa agregowana nie może istnieć bez klasy agregującej.



UOGÓLNIENIE

– UOGÓLNIENIE (generalization) – Związek hierarchiczny, w którym klasa nadrzędna jest bardziej ogólna od klasy podrzędnej. Odwrotna hierarchia to specjalizacja.





Przekształcenie diagramu klas do schematu relacyjnego

- Aby móc zaimplementować bazę zaprojektowaną obiektowo musimy ją sprowadzić do fizycznego modelu danych (PDM)
- Atrybuty klas stają się kolumnami, a atrybuty, które są identyfikatorami klas – kluczami głównymi (PK);
  W tabelach, które powstały z klas nie posiadających identyfikatorów należy samodzielnie wyznaczyć klucze, lub w konieczności dodać nowe kolumny;

- Jeśli klasa posiada metody, to stają się one procedurami.

Przekształcenie diagramu klas do schematu relacyjnego

- Asocjacje są zamieniane na relacje między tabelami w sposób prosty i jednoznaczny. Zależy on od liczebności związków asocjacji.
- Związki jeden do jednego (0..1-1; 1-1) są zastępowane relacjami 1:1 pomiędzy tabelami. Połączenie zachodzi pomiędzy kluczami głównymi obu tabel.
  Związki jeden do wielu (1-0..*) są zamieniane na relacje 1:N, w których klucz główny ze strony 1 jest połączony z kluczem obcym w tabeli 'wiele'. Ten klucz obcy to zazwyczaj nowa kolumna, która nie występuje w diagramie klas.
- Związki wiele do wielu (0..*-1..*) są przekształcane do relacji N:M, co wymaga utworzenia dodatkowej tabeli łączącej.

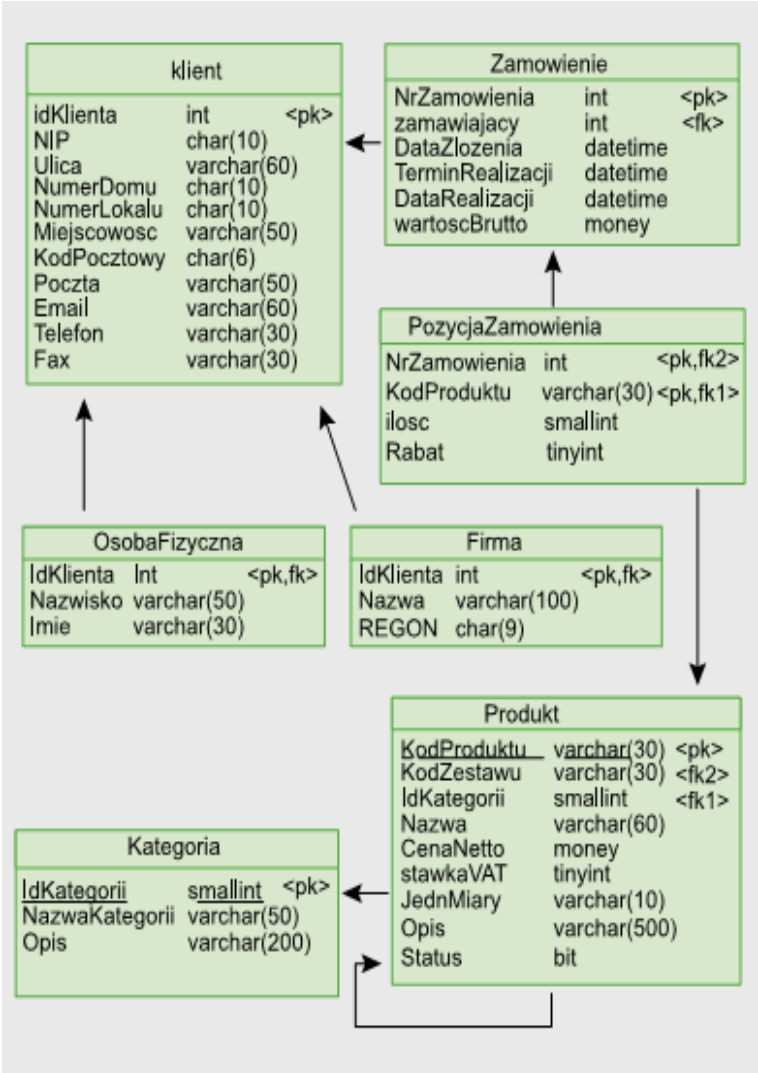
Przekształcenie diagramu klas do schematu relacyjnego

- Agregacja – przekształcana jest identycznie jak związek asocjacji o liczebności 0..1 – 1..* lub 0..1-0..* - czyli na relację 1:N (Można ustawić CASCADE UPDATE ale nie CASCADE DELETE)
- Złożenie – jest traktowane tak jak agregacja. Można ustawić CASCADE DELETE (klasa agregowana nie może istnieć bez agregującej)
- Uogólnienie – sprawia najwięcej problemów. Istnieją 3 możliwości przekształcenia uogólnienia do schematu relacyjnej bazy danych.

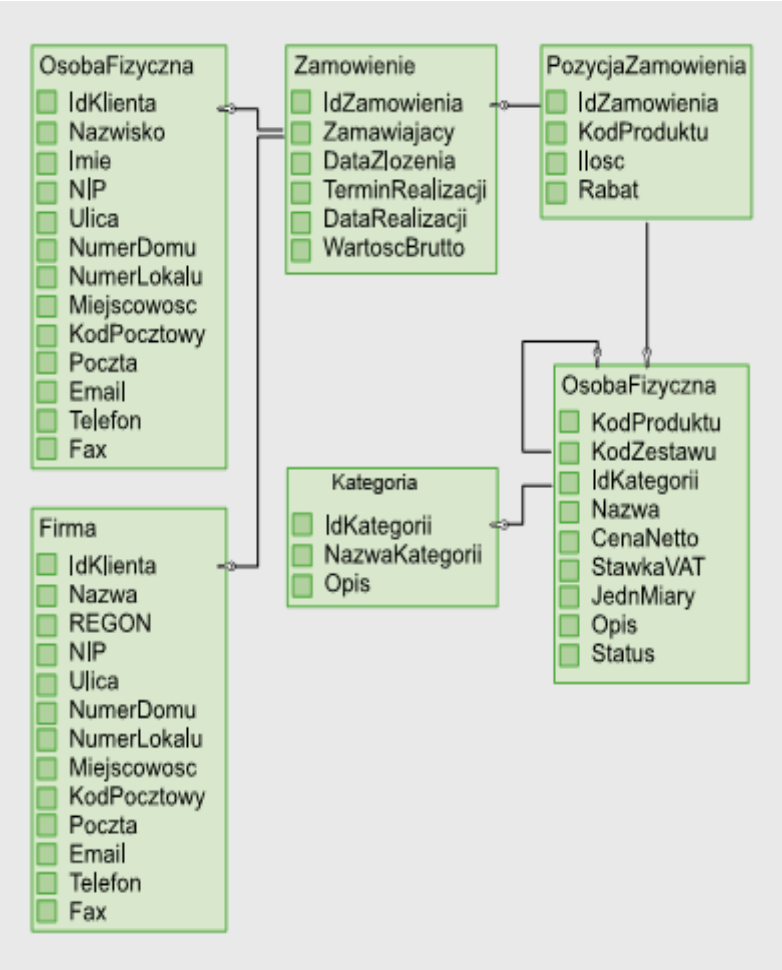
Przekształcenie diagramu klas do schematu relacyjnego

1. Tworzymy po jednej tabeli dla nadklasy i podklas i łączymy je związkami 1:1, w których klucz główny jest też kluczem obcym.
2. Tylko dla podklas tworzymy tabele i umieszczamy w nich wszystkie atrybuty z nadklasy. Należy w tym przypadku obsłużyć synchronizację między podklasami, ponieważ będą się one pokrywać.
3. Tworzymy tylko jedną tabelę, która zawiera wszystkie atrybuty z nadklasy i podklas. Tutaj problemem mogą okazać się wartości NULL, które będą się licznie pojawiały w wierszach.

Schemat bazy uzyskany zgodnie z metodą 1

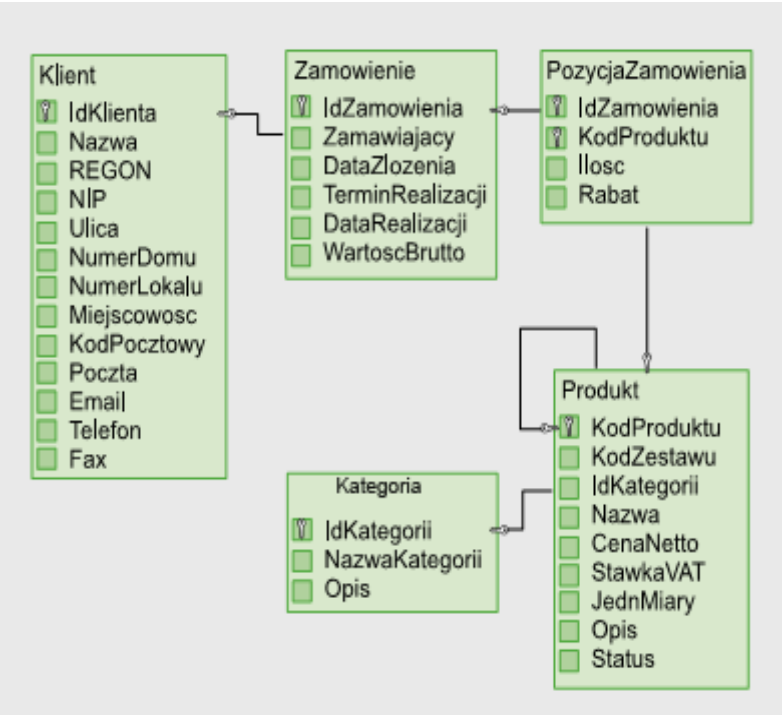


Schemat bazy uzyskany zgodnie z metodą 2



87

Schemat bazy uzyskany zgodnie z metodą 3



88