

Wykład 8

KLASYCZNE I OBIEKTOWE MODELE DANYCH

SPIS TREŚCI

1. Wstęp.
2. Podstawowe wady dotychczasowych baz danych i systemów zarządzania.
3. Obiektowy, podstawowy model danych.
4. Obiektowy model danych.
5. Obiektowa baza danych.
6. Pojęcia związane z obiektowym modelem bazy danych.
7. Obiekt.
8. Klasa.
9. Atrybut.
11. Metoda.
12. Hermetyzacja.
13. Relacyjna a obiektowa struktura aplikacji.
14. System zarządzania bazą danych.
15. Interfejs obiektowych baz danych.
16. Obiektowy model danych.
17. Demony w obiektowej bazie danych.
18. Zasady zachowania spójności bazy.
19. Projekt języka dla obiektowej bazy danych.
20. Fazy występujące w czasie wykonywania zapytań w systemach baz danych.
21. Jak przechowywane są obiekty złożone?
22. Co stanowi zasoby, i jak ma być zapewniona spójność danych.
23. Zamknięcia stosowane dla obiektowych baz danych.
24. Strategie dostępu jednoczesnego w obiektowych bazach danych.
25. Najpopularniejsze systemy obiektowych baz danych.
26. Obiektowo - relacyjna baza danych.
27. Korzyści płynące z zastosowania obiektowości.

Generacje baz danych

★ Generacje baz danych

- systemy plików (takie jak: ISAM i VSAM),
- hierarchiczne baz danych (ISM, System 2000),
- sieciowe baz danych CODASYL (m.in. IDS, IDMS),
- relacyjnymi bazami danych.
- obiektowe bazy danych
- postrelacyjne (obiekto-relacyjne) bazy danych

★ Przejście z jednej generacji do kolejnej było zawsze wymuszane wzrostem złożoności programów użytkowych baz danych, kosztów implementowania, utrzymywania i powiększania tych programów.

★ Niewygodny dostęp do danych (przy pomocy wskaźników) oraz brak niezależności danych doprowadziły do rozwinięcia systemów czwartej generacji - relacyjnych baz danych.

★ Mimo, że obecnie relacyjne bazy danych znajdują powszechne zastosowanie na rynku, posiadają pewne ograniczenia, uniemożliwiające ich wykorzystanie do niektórych typów programów użytkowych. Spowodowane jest to faktem, iż relacyjne bazy danych (podobnie zresztą jak bazy poprzednich generacji) rozwijane były z myślą o konwencjonalnych programach użytkowych służących do obsługi magazynu, listy płac, księgowości, itp.

Modele danych

Modele danych (a w odniesieniu do konkretnej realizacji mówi się często o architekturze systemu baz danych) można rozumieć jako zbiór ogólnych zasad posługiwania się danymi. Zbiór ten obejmuje trzy główne części:

Definicje danych: zbiór reguł określających strukturę danych, tj. to co wcześniej określałem jako logiczną strukturę bazy danych (w odróżnieniu od niższego poziomu organizacji zapisu stosowanego wewnętrznie przez jądro bazy danych);

Operowanie danymi: zbiór reguł dotyczących procesu dostępu do danych i ich modyfikacji;

Integralność danych: zbiór reguł określających, które stany bazy danych są poprawne (a więc zarazem jakie operacje prowadzące do modyfikacji danych są dozwolone).

Brak jednoznacznych definicji pojęcia model danych

Model danych

Model danych (*data model*) jest pojęciem, którego ściśle zdefiniowanie stwarza problem. Jego znaczenie jest wypadkową następujących przymiotów:

- metajęzyk (pojęcia, terminologia) do mówienia o danych, o systemach baz danych i o przetwarzaniu danych;
- sposób rozumienia organizacji danych i ideologiczne lub techniczne ograniczenia w zakresie konstrukcji, organizacji i dostępu do danych;
- języki opisu i przetwarzania danych, w szczególności: diagramy struktur danych, języki opisu danych i języki zapytań;
- ogólne założenia dotyczące architektury i języków systemu bazy danych;
- ideologie lub teorie (matematyczne) dotyczące struktur danych i dostępu do danych

Główne typy (lub generacje) modeli danych

Proste modele danych.

Dane zorganizowane są w strukturę rekordów zgrupowanych w plikach. Głównymi dostępnymi operacjami są operacje na rekordach (ewentualnie na ich poszczególnych polach).

Klasyczne modele danych.

Należą do nich modele **hierarchiczne**, **sieciowe** i **relacyjne**. Modele relacyjne stanowią najbardziej popularną obecnie podstawę architektur systemów baz danych.

Semantyczne modele danych.

Semantyka to inaczej *znaczenie*. Klasyczne modele danych nie dostarczają łatwego sposobu odczytania informacji o semantyce danych, stąd podejmuje się próby stworzenia innych modeli, uzupełniających ten brak. Przykładem częściowej realizacji tego programu są **obiektywne** modele danych.

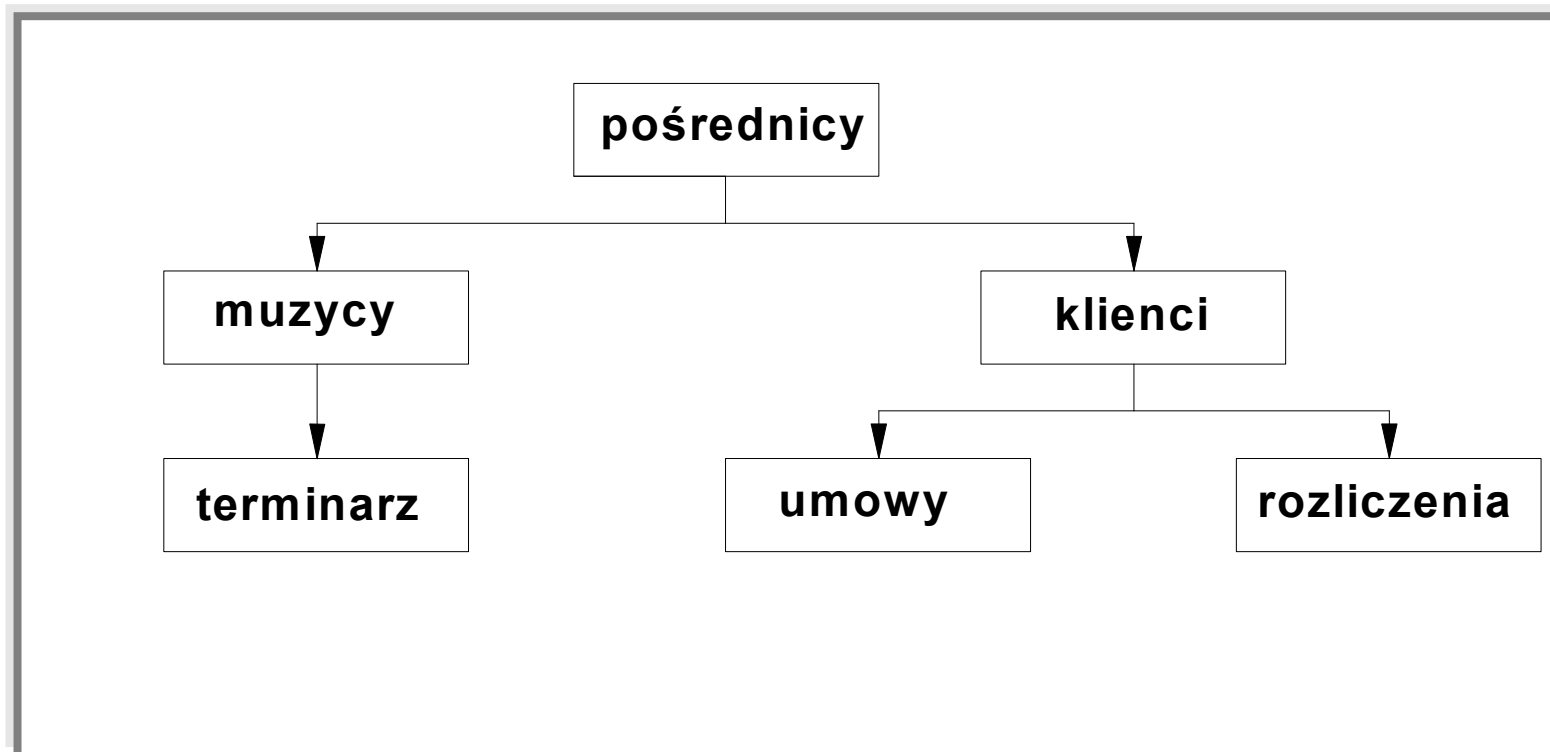
Cechy modeli klasycznych

- ★ struktura bazy danych opiera się na rekordach różnych typów o stałym formacie
- ★ każdy rekord definiowany jest poprzez stałą liczbę atrybutów
- ★ każdy atrybut jest zazwyczaj określonego rozmiaru co ułatwia implementację
- ★ nie zawierają mechanizmów bezpośredniej reprezentacji kodu w bazie danych
- ★ z modelami związane są języki zapytań umożliwiające realizację zapytań oraz modyfikacji danych
- ★ opis danych na poziomie conceptualnym oraz na poziomie wglądu
- ★ specyfikacja całościowa conceptualnej struktury bazy danych
- ★ opis implementacji bazy danych na wysokim poziomie

Hierarchiczne bazy danych

- ★ W hierarchicznym modelu danych (HMBD) dane mają strukturę, którą można najprościej opisać jako odwrócone drzewo.
- ★ Jedna z tabel pełni rolę „korzenia” drzewa, a pozostałe mają postać „gałęzi” biorących swój początek w korzeniu.
- ★ Relacje w HMBD są reprezentowane w kategoriach *ojciec/syn (nadrzędny/podrzędny)*. Oznacza to że *tabela-ojciec* może być powiązana w wieloma *tabelami/synami*, lecz pojedynczy *syn* może mieć tylko jednego *ojca*. Tabele te mogą być powiązane jawnie, przez *wskaźniki*, lub przez fizyczną organizację rekordów wewnątrz tabel. Aby uzyskać dostęp do danych
- ★ W modelu hierarchicznym, użytkownik zaczyna przeszukiwanie od korzenia i przedziera się przez całe drzewo danych, aż do interesującego go miejsca. Oznacza to jednak, że użytkownik musi dobrze znać strukturę bazy danych.

Hierarchiczne bazy danych



★ Baza danych pośredników:

- każdy pośrednik pracuje dla kilku muzyków i ma pewną liczbę klientów, którzy zamawiają u niego obsługę muzyczną różnych imprez.
- klient zawiera umowę z danym muzykiem przez pośrednika i u tego właśnie pośrednika uiszcza należność za usługę

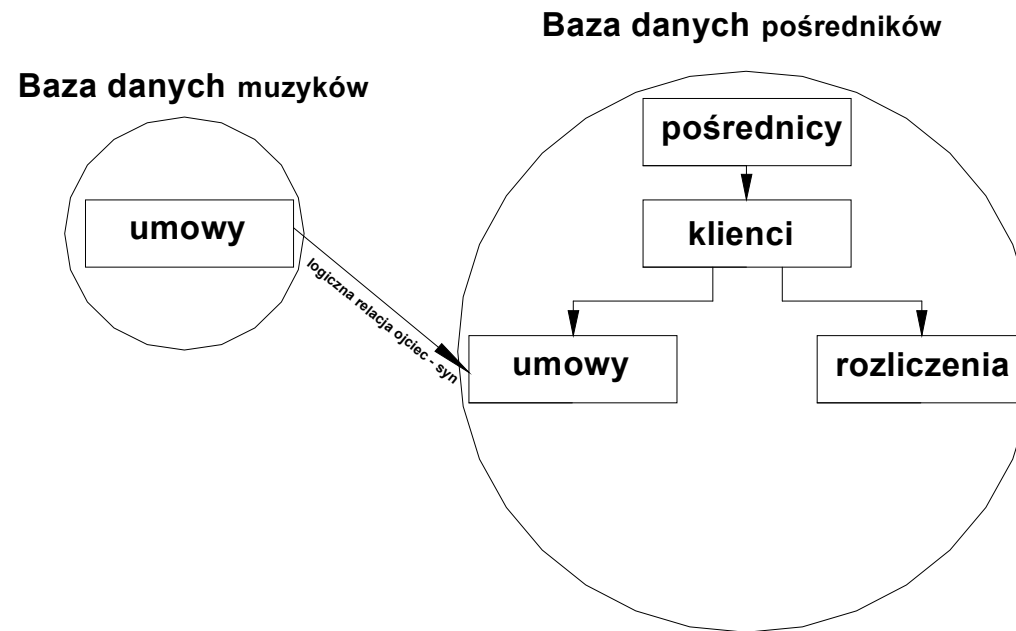
Hierarchiczne bazy danych - zalety

- potrzebne dane można szybko przywołać, ponieważ poszczególne tabele są ze sobą bezpośrednio powiązane.
- automatycznie wbudowana *integralność odwołań*. Oznacza to, że rekord w *tabeli-synu* musi być powiązany z *istniejącym* rekordem w *tabeli-ojcu*. Jeśli jeden z rekordów w *tabeli-ojcu* zostanie skasowany ulegną również wszystkie powiązane z nim rekordy w *tabelach-synów*.

Hierarchiczne bazy danych - wady

- ★ **problemy z dopisywaniem danych do tabeli-syna** - jeżeli musimy zapisać w *tabeli-synu* rekord, który nie byłby powiązany z żadnym z rekordów w *tabeli-ojcu*.
 - W przykładzie nie możemy dopisać do tabeli muzyków nowej osoby, dopóki nie przypiszemy jej pośrednika w tabeli pośredników.
- ★ **nadmiarowe dane**. Wynikają z niezdolności modelu do obsługi złożonych relacji.
 - W przykładzie między klientami i muzykami istnieje relacja *wiele-do-wielu*: jeden muzyk gra dla wielu klientów, a jeden klient może zatrudniać wielu muzyków.
 - Ponieważ taki rodzaj relacji nie może zostać bezpośrednio wpisany w model HMBD, zachodzi konieczność wprowadzenia nadmiarowych danych do tabeli terminarzy i umów. Tabela terminarzy będzie zawierać dane o klientach (takich np. jak: imię, nazwisko, adres oraz numer telefonu), informujące o miejscu występu danego muzyka, lecz dane te pojawiają się również w tabeli klientów. Ponadto tabela terminarzy będzie zawierać dane o muzykach (na przykład: imię, nazwisko, telefon i gatunek muzyki), mówiące nam, którzy muzycy grają dla danego klienta. Te same dane znajdują się w tabeli muzyków. Ta nadmiarowość tworzy możliwość sytuacji, w których pewne dane zostają wprowadzone w sposób niekonsekwentny, co z kolei zaburza integralność bazy.
 - Problem nadmiarowych danych można obejść przez utworzenie osobnej hierarchicznej bazy danych dla muzyków i drugiej - dla pośredników. Nowa baza muzyków będzie składać się jedynie z tabeli muzyków, natomiast baza pośredników będzie zawierać tabele pośredników, klientów, rozliczeń i umów. Tabela terminarzy nie jest już potrzebna, ponieważ można zdefiniować *logiczną relację ojciec-syn* między tabelą umów w bazie pośredników oraz tabelą muzyków w bazie muzyków. Po wprowadzeniu takiej relacji z bazy będzie można wyczytać informacje takie jak: lista muzyków, z którymi zawarł umowy dany klient, czy terminarz występów danego muzyka. Metoda ta spełnia swoje zadanie, jeśli ją zastosujemy. Twórca bazy danych musi jednak być pewien konieczności uwzględnienia takiej relacji. W tym wypadku była ona dość oczywista, lecz wiele relacji jest znacznie bardziej skomplikowanych i potrzeba ich wprowadzenia może zostać zauważona dopiero w bardzo późnych stadiach procesu projektowania lub też - o zgrozo - po oddaniu bazy do użytku.

Hierarchiczne bazy danych

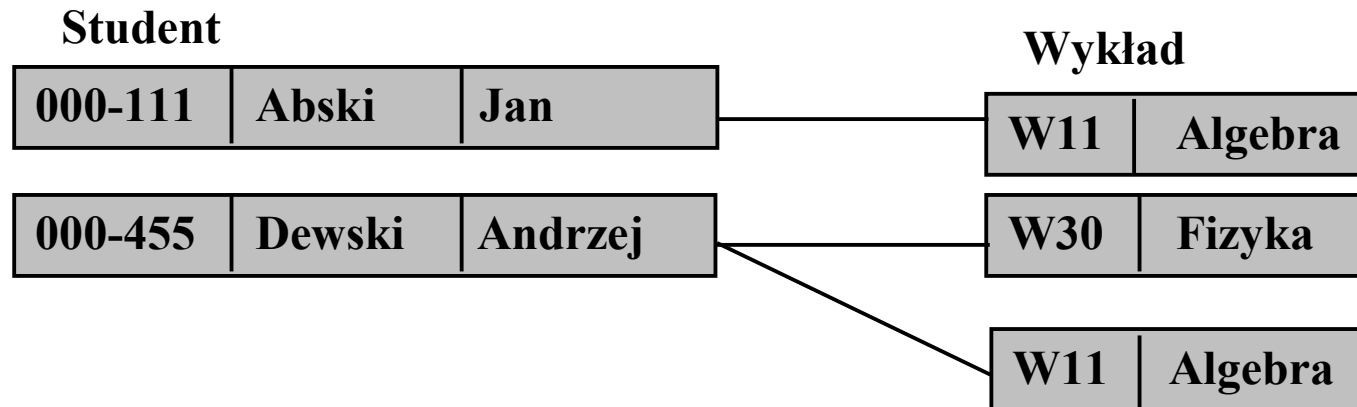


Rysunek 1.2. Wykorzystanie dwóch hierarchicznych baz danych do rozwiązania problemu relacji *wiele-do-wielu*

Hierarchiczny model bazy danych był z powodzeniem stosowany w systemach zapisu taśmowego, wykorzystywanych w komputerach typu *mainframe* do późnych lat siedemdziesiątych, i zdobył dużą popularność w firmach polegających na tych systemach. A jednak, pomimo tego, iż HMBD zapewniał szybki, bezpośredni dostęp do danych i znajdował zastosowanie w rozwiązywaniu wielu problemów, powoli narastała potrzeba wprowadzenia nowego modelu, który nie wymagałby wprowadzania takiej ilości nadmiarowych danych i złożonych relacji.

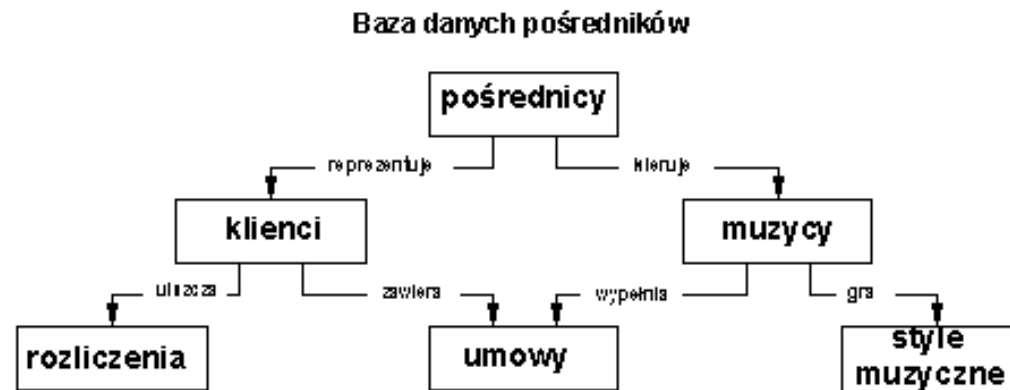
Sieciowy model danych

- ★ Sieciowy model bazy danych (SMBD) został stworzony głównie w celu rozwiązania problemów związanych z modelem hierarchicznym.
 - ★ Tak jak w HMBD, strukturę SMBD można sobie wyobrazić jako odwrócone drzewo. Różnica polega jednak na tym, że w przypadku SMBD kilka drzew może dzielić ze sobą gałęzie, a każde drzewo stanowi część ogólnej struktury bazy danych.
 - ★ Relacje w SMBD są definiowane przez *kolekcje* (ang. *set structure*).
 - ★ *Kolekcja* jest niejawną konstrukcją łączącą dwie tabele przez przypisanie jednej z nich roli *właściciela*, a drugiej - roli *członka*. (Stanowiło to znaczny postęp w porównaniu z relacjami *ojciec--syri*).
 - ★ Kolekcje umożliwiają wprowadzanie relacji *jeden-do-wielu*, co oznacza, że w obrębie danej struktury każdy rekord z *tabeli-właściciela* może być powiązany z dowolną ilością rekordów *tabeli-członka*, lecz pojedynczemu rekordowi w *tabeli-członku* może odpowiadać *tylko jeden rekord* w *tabeli-właścicielu*. Ponadto każdy rekord w *tabeli-członku* musi być powiązany z istniejącym rekordem w *tabeli-właścicielu*.
- Przykładowo, każdy student ma przyporządkowane przedmioty, które studiuje



Sieciowy model danych

- ★ **Baza danych pośredników.** W przykładzie pośrednik pracuje dla kilku muzyków i ma pewną liczbę klientów, którzy zamawiają u niego obsługę muzyczną różnych imprez i uiszczają opłaty. Każdy muzyk może zawrzeć wiele oddzielnych umów i specjalizować się w wielu różnych gatunkach muzyki.



Sieciowy model danych

- ✱ Między każdymi dwoma tabelami można zdefiniować dowolną liczbę powiązań (kolekcji), a każda tabela może uczestniczyć w wielu różnych kolekcjach.
 - Przykładowo: tabela klientów jest powiązana z tabelą opłat przez kolekcję *uiszcza* oraz z tabelą umów przez kolekcję *zawiera*. Tabela umów, oprócz tabeli klientów, jest powiązana również z tabelą muzyków przez kolekcję *wypełnia*.
- ✱ W SMBD dostęp do odpowiednich danych można uzyskać, poruszając się po kolekcjach. W przeciwieństwie do modelu hierarchicznego, gdzie poszukiwanie danych należało zacząć od podstaw, w SMBD użytkownik może rozpocząć od dowolnej tabeli, a następnie poruszać się w górę lub w dół po tabelach z nią powiązanych.

Sieciowy model danych – zalety i wady

★ Zalety:

- szybkość, z jaką można odczytywać z niego dane. Ponadto użytkownik ma możliwość tworzenia znacznie bardziej złożonych zapytań niż miało to miejsce w modelu hierarchicznym.

★ Wady

- ★ można uznać jednak to, że, podobnie jak w HMBD, użytkownik musi mieć dobre wyobrażenie o strukturze używanej bazy danych.
 - W przykładzie z pośrednikami na użytkownika spada ciężar pamiętania, przez które kolekcje należy przejść, aby dowiedzieć się, czy - przykładowo - należność za daną umowę została już uiszczona.
- niemożność zmiany struktury bazy danych bez ponownego tworzenia obsługujących ją programów. Jak już wiemy, relacje w SMBD są zdefiniowane za pomocą kolekcji. Danej kolekcji nie można zmienić bez modyfikowania aplikacji bazodanowej, ponieważ aplikacja ta polega na kolekcjach do wyszukiwania odpowiednich danych. Jeśli któraś kolekcja ulegnie zmianie, *wszystkie* odwołania do tej kolekcji zawarte w aplikacji obsługującej bazę danych muszą również zostać zmienione.
- ★ **Pomimo iż sieciowy model logiczny był zdecydowanym krokiem naprzód w porównaniu z modelem hierarchicznym, kilku ludzi w środowisku bazodanowym nadal uważało, że musi istnieć lepszy sposób na przechowywanie i obsługę dużych ilości danych. W miarę jak rozwijał się przemysł informatyczny, użytkownicy chcieli uzyskiwać odpowiedzi na coraz bardziej złożone zapytania. To z kolei kładło coraz większy ciężar na istniejące struktury baz danych. I tak dochodzimy do modelu relacyjnego.**

Wady klasycznych baz danych i SZBD

- ★ model danych (szczególnie relacyjny) jest zbyt prosty do zamodelowania złożonych, zagnieżdżonych encji,
- ★ systemy baz danych oferują tylko kilka prostych typów danych (integer, character), nie obsługują ogólnych typów danych występujących w językach programowania,
- ★ model danych nie zawiera często używanych pojęć semantycznych (np.: generalizacja, agregacja), a co za tym idzie SZBD nie oferuje mechanizmów do reprezentacji takich związków i zarządzania nimi (programiści muszą sami zaimplementować takie mechanizmy w aplikacjach),
- ★ zbyt wolne działanie systemów baz danych z programami użytkowymi wymagającymi szybkich i skomplikowanych obliczeń (szczególnie programami symulacyjnymi),
- ★ różnice między językami baz danych (SQL, DL/1, CODASYL DML), a językami programowania (COBOL, FORTRAN, PL/1, C++) zarówno pod względem wykorzystywanego modelu danych, jak i struktur danych,
- ★ systemy baz danych nie dostarczają narzędzi do reprezentowania i zarządzania temporalnymi aspektami baz danych (m.in.: pojęciem czasu, wersjami obiektów i schematu).

Obiektowy model danych


Obiektowy model danych (*object model, object-oriented model*) jest modelem danych, którego podstawą są pojęcia obiektowości, m.in.: obiekt, klasa, dziedziczenie, hermetyzacja.

- ★ Brak powszechnie akceptowalnych definicji modelu obiektowego
- ★ Trwają prace nad ustandaryzowaniem pojęć obiektowych w dziedzinie baz danych, prowadzone m.in.: przez ODMG (*Object Database Management Group*).
- ★ Standard zaproponowany przez ODMG stworzony został w oparciu o trzy istniejące standardy dotyczące baz danych (SQL-92), obiektów (OMG) i obiektowych języków programowania (ANSI).
- ★ Brak standardu wynika z faktu, iż rozwój podejścia obiektowego następował w trzech różnych obszarach:
 - językach programowania,
 - sztucznej inteligencji,
 - bazach danych.

Obiektowa baza danych

Obiektowa baza danych (*Object Database, Object-Oriented Database*) jest zbiorem obiektów, których zachowanie, stan i związki zostały określone zgodnie z obiekowym modelem danych.

- ★ Przyjęcie obiektowego modelu, jako podstawy bazy danych, implikuje naturalną jej rozszerzalność bez konieczności wprowadzania zmian do istniejącego systemu.
- ★ Możliwe są dwa sposoby rozszerzenia obiektowych bazy danych:
 - rozszerzanie zachowania się
 - dziedziczenie.
- ★ Zachowanie się obiektu może zostać rozszerzone przez dołączenie dodatkowych programów (metod) do już istniejących. W dowolnym momencie życia systemu można zdefiniować nowe klasy korzystając już z istniejących lub przeddefiniować istniejące.



Pojęcia związane z obiekowym modelem bazy danych:

Obiekty

Obiekt jest zbiorem danych i procedur. Dane są gromadzone w atrybutach obiektu. Procedury są zdefiniowane za pomocą metod obiektu. Metody są uaktywniane przez komunikaty przekazywane między obiektami.

- ★ Obiekty muszą mieć właściwość hermetyzacji. Dane i proces są zapakowane razem w ramach jednego interfejsu i udostępniane z zewnątrz w sposób kontrolowany.
- ★ Obiekt jest instancją klasy.
- ★ 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;
- ★ **Identyfikator** obiektu umożliwia jednoznaczne odwołanie do obiektu (jest niepowtarzalny w systemie)

Klasy

Klasa (class) - byt semantyczny rozumiany jako miejsce przechowywania (specyfikacji i definicji) takich cech grupy podobnych obiektów, które są dla nich niezmiennie: atrybutów, metod, ograniczeń dostępu, dozwolonych operacji na obiektach, wyjątków, itp.

- ✳ Klasa stanowi wzorzec dla tworzonego obiektu. W przypadku baz danych definiują schemat bazy danych
- ✳ W systemach obiektowych, klasa jest traktowana jako obiekt (klasowy), w celu zagwarantowania jednolitego posługiwania się komunikatami. Dlatego z klasą mogą być związane atrybuty i metody klasowe.
- ✳ W obiektowej bazie danych, dziedzina dowolnego atrybutu może być klasa, co zwiększa semantyczną spójność bazy (zwalnia od określania więzów spójności bezpośrednio w zbiorach wartości).
- ✳ Również klasa (lub kilka klas), jako agregacja powiązanych ze sobą obiektów w bazie danych, stanowi cel do sformułowania zapytania. Zwykle klasy wiążą się ze sobą poprzez hierarchię (lub inną strukturę) dziedziczenia.

Uogólnienie – mechanizm abstrakcji lub proces, zgodnie z którym jest tworzony obiekt wyższego rzędu w celu podkreślenia podobieństw pomiędzy obiektami niższego rzędu

Atrybuty

Atrybut (**attribute**) jest częścią definicji klasy, specyfikacja atrybutu polega na podaniu jego nazwy i więzów semantycznej spójności, obejmujących: dziedzinę atrybutu, jednoznaczność wartości, dopuszczalność wartości NULL, 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).
- ★ Wartością atrybutu może być instancja klasy będącej jego dziedziną lub instancja dowolnej podklasy z hierarchii klas zakorzenionej w klasie stanowiącej dziedzinę atrybutu;

Metody

Metoda (**method**) jest procedurą, funkcją lub operacją przypisaną do klasy i dziedziczoną 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 zdefiniowane w dowolnej z jej podklas
- ★ W bazach danych metody występują w czterech postaciach
 - metody tworzenia – operacje służące do tworzenia nowych instancji klasy,
 - metody usuwania – operacje służące do usuwania zbędnych obiektów,
 - metody dostępu – operacje służące do przekazywania właściwości obiektów,
 - metody przekształcania – operacje służące do przekazywania nowych obiektów uzyskanych z obiektów istniejących



Dziedziczenie

★ Typy dziedziczenia:

- dziedziczenie struktury – podklasa dziedziczy atrybuty swojej nadklasy
- dziedziczenie zachowania – podklasa dziedziczy metody swojej nadklasy

★ Rodzaje dziedziczenia:

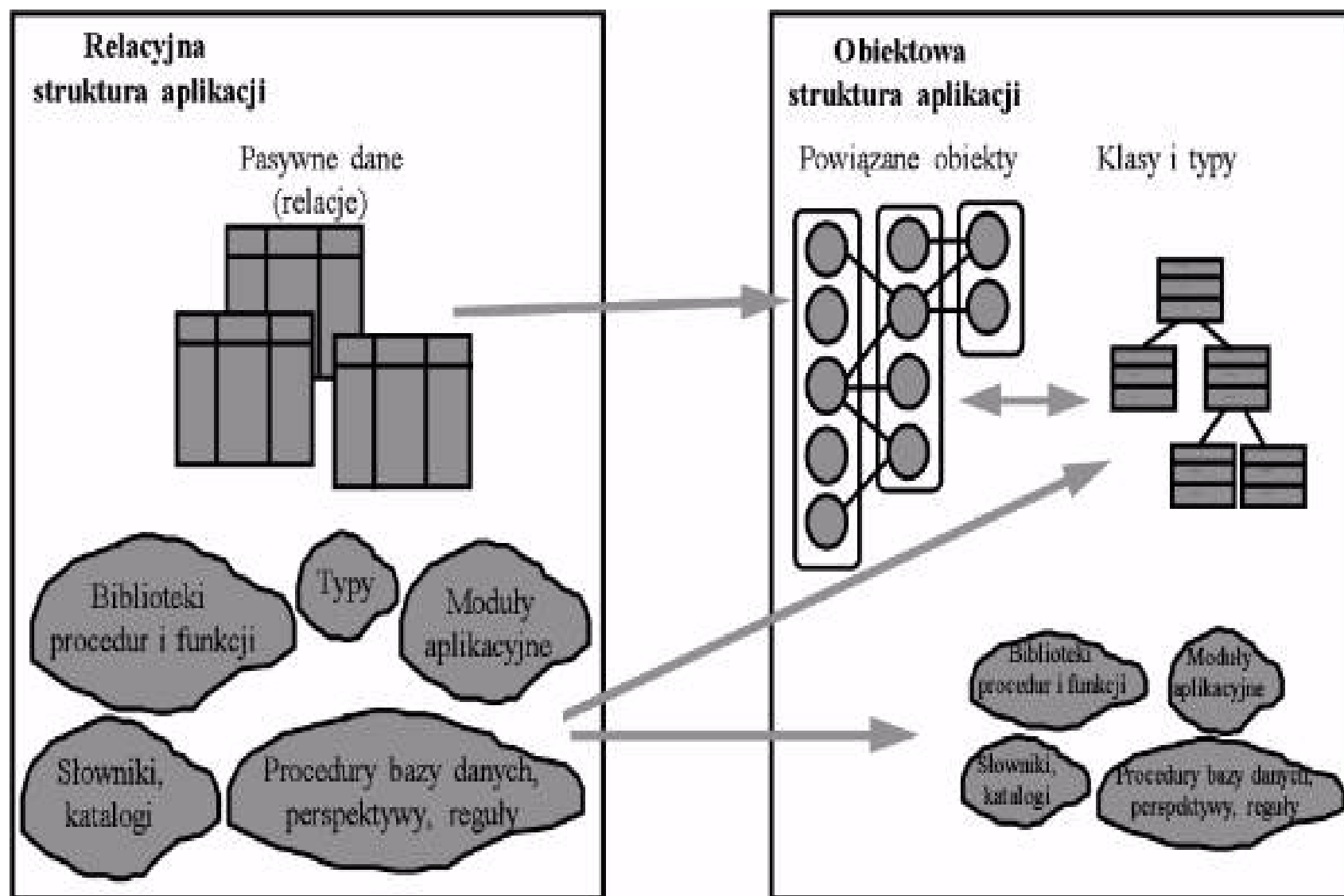
- dziedziczenie pojedyncze – klasa może być podklasą tylko jednej nadklasy
- dziedziczenie wielokrotne – klasa może dziedziczyć od więcej niż jednej nadklasy

Hermetyzacja

Hermetyzacja (encapsulation) – upakowanie razem danych i procedur w zdefiniowanym interfejsie.

- ★ 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.
- ★ W „obiektowości” wyróżnia się:
 - koncepcję **ortodoksyjnej hermetyzacji** (Smalltalk), w której wszelkie operacje, które można wykonać na obiekcie są określone przez metody przypisane do obiektu (znajdujące się w jego klasie i nadklasach). Bezpośredni dostęp do atrybutów obiektu jest niemożliwy,
 - **hermetyzacja ortogonalna** (C++), gdzie dowolny atrybut obiektu (i dowolna metoda) może być prywatny (nieдоступny z zewnątrz) lub publiczny;

Relacyjna i obiektowa struktura aplikacji



Zalety obiektowych baz danych

- ★ złożone obiekty
- ★ typy danych definiowane przez użytkownika
- ★ tożsamość obiektów (identyfikator), trwałość
- ★ hermetyzacja, hierarchia, dziedziczenie
- ★ rozszerzalność
- ★ zgodność we wszystkich fazach życia bazy i danych
- ★ metody i funkcje przechowywane wraz z danymi
- ★ nowe możliwości (wersjonowanie, rejestracja zmian, powiadamianie ...)
- ★ możliwość nowych zastosowań mniejszym kosztem (bazy multimedialne, przestrzenne, bazy aktywne...)
- ★ porównywalna wydajność (i wciąż rośnie)

Wady obiektowych baz danych

- ★ brak optymalizacji zapytań (rozumianej jak w poprzednich modelach)
- ★ niedopracowane mechanizmy zarządzania dużą bazą obiektów, sterowania wersjami, ...
- ★ mała liczba ekspertów od technik obiektowych
- ★ nie wiadomo z jakimi kosztami wiąże się migracja dużych systemów
- ★ brak dopracowanych standardów