# Bazy danych

Zbigniew Jurkiewicz

Wydział Matematyki, Informatyki i Mechaniki
Uniwersytet Warszawski
http://mst.mimuw.edu.pl

21 stycznia 2013



Rozdział I

## Wprowadzenie



Rozdział II

## Algebra relacji



## Złączenie theta

- $R \underset{\theta}{\bowtie} S = \sigma_\theta(R \times S)$
- $\theta$ oznacza dowolny warunek na kolumny łączonych relacji, np. $A < C$.
- Złączenie theta, w którym warunek jest prostą równością pary atrybutów, nazywa się **złączeniem równościowym**.
- Pojęcie porzuconej krotki (**dangling tuple**): wiersza z jednej z relacji, do którego nie pasuje żaden wiersz z drugiej relacji.



## Złączenie theta

- Zwierzaki

| gatunek | imię | waga |
|---|---|---|
| Papuga | Kropka | 3,50 |
| Papuga | Lulu | 5,35 |
| Papuga | Hipek | 3,50 |
| Lis | Fufu | 6,35 |
| Krokodyl | Czako | 75,00 |

Gatunki

| nazwa | kontynent |
|---|---|
| Papuga | Ameryka |
| Lis | Europa |
| Krokodyl | Afryka |

- Zwierzaki $\underset{gatunek=nazwa}{\bowtie}$ Gatunki

| gatunek | imię | waga | nazwa | kontynent |
|---|---|---|---|---|
| Papuga | Kropka | 3,50 | Papuga | Ameryka |
| Papuga | Lulu | 5,35 | Papuga | Ameryka |
| Papuga | Hipek | 3,50 | Papuga | Ameryka |
| Lis | Fufu | 6,35 | Lis | Europa |
| Krokodyl | Czako | 75,00 | Krokodyl | Afryka |



## Złączenie naturalne

- Notacja: $R \bowtie S$.
- Łączone relacje muszą mieć co najmniej jedną wspólną kolumnę o tej samej nazwie.
- Warunkiem złączenia jest równość dla wszystkich par atrybutów o tych samych nazwach.
- W wyniku zostaje tylko jedna kolumna z pary kolumn o tych samych nazwach.



## Złączenie naturalne

- Zwierzaki

| gatunek | imię | waga |
|---|---|---|
| Papuga | Kropka | 3,50 |
| Papuga | Lulu | 5,35 |
| Papuga | Hipek | 3,50 |
| Lis | Fufu | 6,35 |
| Krokodyl | Czako | 75,00 |

Gatunki

| gatunek | kontynent |
|---|---|
| Papuga | Ameryka |
| Lis | Europa |
| Krokodyl | Afryka |

- Zwierzaki $\bowtie$ Gatunki

| gatunek | imię | waga | kontynent |
|---|---|---|---|
| Papuga | Kropka | 3,50 | Ameryka |
| Papuga | Lulu | 5,35 | Ameryka |
| Papuga | Hipek | 3,50 | Ameryka |
| Lis | Fufu | 6,35 | Europa |
| Krokodyl | Czako | 75,00 | Afryka |



## Przemianowanie

- Pozwala na nazywanie relacji wynikowych: $\rho_{RS(A,B,X,C,D,E)}(R \times S)$.
- Uproszczone notacja: $R1(A1, B, X, C, D, E) := (R \times S)$.

## Wyrażenia złożone

- Ponieważ jest to algebra, więc operacje można składać otrzymując wyrażenia złożone.
- Równoważność wyrażeń można wykorzystać przy optymalizacji, zastępując dane wyrażenie równoważnym mu, lecz bardziej efektywnym.



## Wyrażenia złożone: samozłączenie

- Zwierzaki

| gatunek | imię | waga |
|---|---|---|
| Papuga | Kropka | 3,50 |
| Papuga | Lulu | 5,35 |
| Papuga | Hipek | 3,50 |
| Lis | Fufu | 6,35 |
| Krokodyl | Czako | 75,00 |

- Znajdź pary zwierzaków (imiona) tego samego gatunku

$$\pi_{Z1.imie,Z2.imie}(\rho_{Z1}Zwierzaki \underset{\substack{Z1.gatunek=Z2.gatunek\wedge \\ Z1.imie<Z2.imie}}{\bowtie} \rho_{Z2}Zwierzaki)$$



## Wielozbiory

- Zgodnie z matematyczną definicją relacji jako zbioru utożsamia się jednakowe krotki (powstające np. podczas rzutowania).
- Można rozszerzyć tę algebrę na wielozbiory, dopuszczając powtórzenia.
- Powstaje jednak problem odpowiedniej semantyki dla operacji iloczynu i różnicy teoriomnogościowej.
- Intuicyjnie zdefiniowane rozszerzenia operacji na wielozbiory zastosowane do relacji dają relacje z wyjątkiem sumy, która dla dwóch relacji może dać wielozbiór.
- Przestają zachodzić niektóre prawa algebry relacji, np.

$$(R \cup S) - T = (R - T) \cup (S - T)$$

- Operator eliminacji powtórzeń $\delta(R)$.



## Grupowanie

- Operator grupowania z ewentualną agregacją

$$\gamma_{A,MIN(B)\to MinB}(R)$$

- Zauważmy, że

$$\gamma_{A_1,\ldots,A_n}(R) = \sigma(R)$$

jeśli $A_i$ to wszystkie atrybuty $R$.



## Sortowanie

- Operator sortowania $\tau_{C,B}(R)$.
- Nie jest to operator algebry relacji ani wielozbiorów, lecz ewentualnej algebry list, dlatego powinien być zewnętrznym operatorem wyrażenia!



## Złączenia zewnętrzne

- naturalne: $R \overset{\circ}{\bowtie} S$
  - jako wypełniaczy brakujących wartości w dołączonych kolumnach używa się $\perp$;
- lewostronne: $R \overset{\circ}{\bowtie}_L S$
  - brane są tylko porzucone krotki z pierwszego argumentu;
- prawostronne: $R \overset{\circ}{\bowtie}_R S$;
- wersje theta powyższych (z warunkiem u dołu).



## Złączenie zewnętrzne

- Zwierzaki

| gatunek | imię | waga |
|---|---|---|
| Papuga | Kropka | 3,50 |
| Papuga | Lulu | 5,35 |
| Papuga | Hipek | 3,50 |
| Lis | Fufu | 6,35 |
| Krokodyl | Czako | 75,00 |

Gatunki

| gatunek | kontynent |
|---|---|
| Papuga | Ameryka |
| Lis | Europa |
| Krokodyl | Afryka |
| Krowa | Europa |

- Zwierzaki $\bowtie$ Gatunki

| gatunek | imię | waga | kontynent |
|---|---|---|---|
| Papuga | Kropka | 3,50 | Ameryka |
| Papuga | Lulu | 5,35 | Ameryka |
| Papuga | Hipek | 3,50 | Ameryka |
| Lis | Fufu | 6,35 | Europa |
| Krokodyl | Czako | 75,00 | Afryka |
| Krowa | $\perp$ | $\perp$ | Europa |



## Zastosowania algebry relacji

- Zapisywanie zapytań (np. modelowanie semantyki)
- Nakładanie ograniczeń na poprawność bazy danych (**więzy**). Przykłady:

$$R \cap S = \emptyset \quad \text{(styl równościowy)}$$
$$R \cap S \subseteq \emptyset \quad \text{(styl teoriomnogościowy)}$$

- Integralność referencyjna

$$\pi_{klucz\text{-}zewnętrzny}(R) \subseteq \pi_{klucz}(S)$$
$$\pi_{klucz\text{-}zewnętrzny}(R) - \pi_{klucz}(S) = \emptyset$$

- Zależności funkcyjne

$$A \to B: \quad \sigma_{R.A=R1.A \wedge R.b \neq R1.B}(R \times \rho_{R1}(R)) = \emptyset$$

Rozdział III

# Język SQL — część 1



Rozdział IV

# Język SQL — część 2



## Wyrażenia JOIN

- Warunki łączące dla złączeń można zapisywać we frazie FROM
- Jeszcze raz imiona wszystkich zwierzaków pochodzących z Afryki.

```
SELECT imie
FROM Zwierz JOIN Gatunki
  ON (Gatunki.nazwa = Zwierz.gatunek)
WHERE kontynent = 'Afryka';
```



## Operatory złączeń

Standard SQL-92 podaje operatory złączeń do używania we frazie FROM:

| | |
|---|---|
| `T1 CROSS JOIN T2` | Iloczyn kartezjański |
| `T1 NATURAL JOIN T2` | Złączenie naturalne (równościowe po kolumnach o tych samych nazwach) |
| `T1 INNER JOIN T2` | Zwykłe złączenie |
| `T1 LEFT OUTER JOIN T2` | Złączenia zewnętrzne |
| `T1 RIGHT OUTER JOIN T2` | |
| `T1 FULL OUTER JOIN T2` | |

Po takim wyrażeniu dodatkowo podajemy

| | |
|---|---|
| `USING (`*kolumna*`, ...)` | nazwy kolumn po których łączymy |
| `ON` *warunek* | warunek ograniczający na złączenie |



## Perspektywy

W SQL relacja to tabela lub **perspektywa**.
Tworzenie perspektywy

`CREATE VIEW` *nazwa* `[(`*atrybut* `...)] AS` *zapytanie*`;`

na przykład

```
CREATE VIEW GatunkiAfryki AS
  SELECT *
  FROM Gatunki
  WHERE kontynent = 'Afryka';
```



## Perspektywy (2)

Usuwanie perspektywy

`DROP VIEW` *nazwa*`;`

Uproszczona semantyka operacyjna dla zapytań z perspektywami:

- Nazwę perspektywy we frazie `FROM` w zapytaniu zastępuje się relacjami, na podstawie których ją utworzono.
- Warunki z definicji perspektywy dołącza się do warunków zapytania.



## Modyfikacje perspektyw

Modyfikacje są dozwolone tylko dla **aktualizowalnych** perspektyw:

- zbudowanych na podstawie pojedynczej tabeli, oraz
- obejmujących wszystkie atrybuty nie posiadające wartości domyślnych.



## Modyfikacje perspektyw (2)

Warto zabronić operacji wstawiania i modyfikacji perspektywy dających wiersze, które nie bedą należeć do perspektywy, używając podczas jej tworzenia frazy `WITH CHECK OPTION`:

```
CREATE VIEW GatunkiAfryki AS
  SELECT *
  FROM Gatunki
  WHERE kontynent = 'Afryka'
  WITH CHECK OPTION;
```

## Kursory

- Kursory służą do krokowego przeglądania wyniku zapytania. Używa się ich przede wszystkim w procedurach składowanych.
- Kursor deklarujemy poleceniem DECLARE

```
DECLARE kursor_gatkon CURSOR FOR
SELECT gatunek, kontynent FROM Gatunki;
```

- Z kursora można pobierać kolejne wiersze używając polecenia

```
FETCH [ kierunek ] [ ile ]  IN | FROM  cursor
```



## Kursory (2)

- Parametr *kierunek* definiuje kierunek pobierania wierszy i może być równy
  - FORWARD — pobiera następne wiersze (zachowanie domyślne)
  - BACKWARD — pobiera poprzednie wiersze.
- Parametr *ile* określa, ile wierszy należy pobrać i może być równy
  - *n* — Liczba ze znakiem podająca liczbę wierszy do pobrania. Podanie liczby ujemnej zamienia znaczenie FORWARD i BACKWARD.
  - ALL — Wszystkie pozostałe wiersze.
  - NEXT — Równoważny podaniu 1.
  - PRIOR — Równoważny podaniu -1.



## Kursory – przykłady

Aby pobrać dwa kolejne wiersze z kursora

```
=> FETCH 2 FROM kursor_gatkon;

gatunek      kontynent
----------------------
lew          Afryka
bóbr         Europa
```



## Kursory – cofanie się

Po kursorze można się cofać

```
FETCH -1 FROM kursor_gatkon;
```

lub

```
-- Pobierz poprzedni wiersz
FETCH BACKWARD 1 FROM kursor_gatkon;
```



## Pozycjonowanie kursora

- Kursor można pozycjonować bez pobierania wierszy poleceniem

```
MOVE [ kierunek ] [ ile ]  IN | FROM  kursor
```

  Znaczenie parametrów jest takie, jak dla FETCH.
- Aby przestawić kursor o 5 wierszy do przodu

```
MOVE 5 FROM kursor_gatkon;
```

- Na zakończenie kursor należy zamknąć

```
CLOSE kursor_gatkon;
```



## Pozycjonowanie kursora (2)

- Uwaga: kursory działają tylko wewnątrz transakcji, czyli przed ich użyciem należy wykonać

```
BEGIN WORK;
```

  (o ile nie jesteśmy wewnątrz otwartej transakcji), a potem (niekoniecznie natychmiast) zamknąć transakcję

```
COMMIT WORK;
```

- Nie jest możliwa aktualizacja bieżącego wiersza kursora, trzeba używać niezależnego polecenia UPDATE.
- SQL92 nie zawiera polecenia MOVE, ale za to pozwala na absolutne pozycjonowanie kursora, co w PostgresSQL nie jest zrealizowane.



## Asercje

- Nie występują nigdzie poza standardem. Składnia:

```
CREATE ASSERTION nazwa CHECK (warunek);
```

- Przykład użycia:

```
CREATE ASSERTION DodatniaWaga
  CHECK (NOT EXISTS (SELECT * FROM Zwierz
                     WHERE waga < 0));
```



## Dziedziny

Służą do określania typów danych. Polecenie

```
CREATE DOMAIN AdresTyp AS
  VARCHAR(40) DEFAULT 'Nieznany';
```

tworzy nowy typ, którego można użyć wewnątrz CREATE TABLE

```
CREATE TABLE Studenci (
  indeks CHAR(6) PRIMARY KEY,
  imie VARCHAR(15) NOT NULL,
  nazwisko VARCHAR(15) NOT NULL,
  adres AdresTyp
);
```

Dziedzinę usuwamy poleceniem

```
DROP DOMAIN Adres;
```

## Indeksy

- Polecenie `CREATE INDEX` definiuje nowy indeks dla podanej tabeli. Pojawiło się dopiero w SQL-99.

```
CREATE [ UNIQUE ] INDEX nazwa-indeksu ON tabela
  (kolumna [, ...])
  [ WHERE warunek ]
```

- Parametr `UNIQUE` powoduje sprawdzanie duplikatów w tabeli podczas tworzenia indeksu i przy każdej modyfikacji.
- Utworzony indeks będzie oparty na kluczu powstałym przez konkatenację podanych kolumn.
- Do usuwania indeksu służy polecenie `DROP INDEX`.



## Przykłady indeksów

- Utworzymy indeks na kolumnie `kontynent` tabeli `Gatunki`

```
CREATE INDEX IndGat ON Gatunki(kontynent);
```

- Indeks może obejmować kilka kolumn.

```
CREATE INDEX IndKontChron
  ON Gatunki(kontynent,chroniony);
```

- Usuwanie indeksu:

```
DROP INDEX IndGat;
```



## Indeksy funkcyjne (Postgres)

- Istnieje też inna postać definicji indeksu:

```
CREATE [ UNIQUE ] INDEX nazwa-indeksu ON tabela
  ( funkcja( kolumna [, ... ]) )
  [ WHERE warunek ]
```

- Służy ona do definiowania **indeksów funkcyjnych**, gdzie wartością klucza indeksowego jest wynik wywołania określonej przez użytkownika *funkcji*, której parametrami są podane kolumny indeksowanej tabeli.
- Przykładowo, użycie funkcji `upper(`*kolumna*`)` pozwoli podczas indeksowania ignorować rozróżnianie dużych i małych liter.

```
CREATE INDEX test1_idx ON test1 (upper(kol1));
```

- Wartość funkcji używanej w indeksie musi zależeć jedynie od jej argumentów. Podczas jej tworzenia należy ją oznaczyć jako ustaloną (*immutable*).



## Indeksy częściowe

- Jeśli w definicji indeksu występuje klauzula `WHERE`, to powstanie **indeks częściowy**, zawierający pozycje tylko dla wierszy tabeli spełniających podany warunek.
- Na przykład w tabeli zamówień można by zdefiniować indeks tylko dla wierszy zawierających `'tak'` w kolumnie `zapłacono`.
- Wyrażenie w klauzuli `WHERE` może odwoływać się tylko do kolumn indeksowanej tabeli, nie wolno też używać podzapytań ani funkcji agregujących.



## Sekwencje

- Służą do otrzymania kolejnych wartości dla kolumn typu całkowitego.
- Tworzenie

```
CREATE SEQUENCE sekw_kat
  INCREMENT BY 1 START WITH 1;
```

- Generowanie kolejnej wartości funkcją `nextval` (jej argumentem jest **nazwa** generatora):

```
SELECT nextval('sekw_kat');
```

- Wywołanie `nextval` można też umieścić we frazie `DEFAULT` definicji kolumny w poleceniu `CREATE TABLE`.



## Sekwencje (2)

- Do otrzymania bieżącej wartości generatora sekwencji służy funkcja `curval`.

```
SELECT curval('sekw_kat');
```

zaś do ustawienia na konkretną wartość funkcja `setval`

```
SELECT setval('sekw_kat', 12);
```

- Zamiast umieszczać wywołanie `nextval` we frazie `DEFAULT`, można jako typ takiej kolumny podać `SERIAL`. Zostanie wtedy automatycznie utworzona sekwencja, a kolumna będzie w rzeczywistości typu INT4.



## Użytkownicy

Zmiana hasła użytkownika

`ALTER USER` *nazwa* `PASSWORD '`*nowe-hasło*`';`



Rozdział V

# Modelowanie danych

## Przebieg projektu

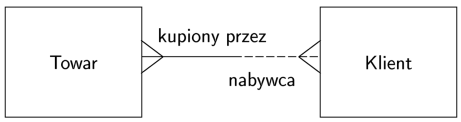
1. Uzgodnić z użytkownikami kształt przyszłego systemu.
2. Opisać w postaci specyfikacji uzgodnione wymagania.
3. Zaprojektować sposób realizacji systemu.
4. Zrealizować system i wdrożyć go.



## Diagramy związków/encji (ERD)

- Diagramy związków-encji, spopularyzowane przez Bachmana i Chena.
- Powinny zawierać związki pomiędzy danymi pamiętanymi, tzn. takimi, które nie mogą być wyprowadzone z innych danych.
- Obecnie coraz częściej używane tylko do modelowania baz danych podczas projektowania fizycznego.
- Dwa podstawowe składniki:
  - encje
  - związki
- Encje opisują (w uproszczony sposób) obiekty, wchodzące w zakres naszego zainteresowania.



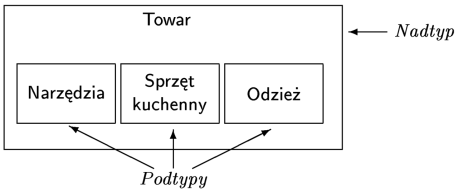



**Rysunek:** Przykładowy diagram związków encji.



## ERD — konstrukcje obiektowe

- Niektóre encje mogą odpowiadać podklasom, np. towary sprzedawane przez pewną firmę (modelowane encją `Towar`) mogą dzielić się na sprzęt, oprogramowanie i materiały pomocnicze.
- Niektóre relacyjne bazy danych (np. PostgreSQL) pozwalają reprezentować takie hierarchie tabel, jednak podklasy powinny być rozłączne.




**Rysunek:** .



## Projektowanie bazy danych

1. Atrybuty i zależności, grupowanie
2. Odwzorowanie encji/obiektów i związków w relacje/tabele
3. Normalizacja i denormalizacja.
4. Strojenie bazy, definiowanie ścieżek dostępu.



## Projektowanie bazy danych

- Projektowanie obejmuje projekt logiczny bazy danych i projekt implementacji fizycznej.
- Projektowanie logiczne bazy danych polega na zdefiniowaniu tabel, określeniu ścieżek dostępu (*access paths*) dla tabel (np. indeksów) oraz dostosowaniu indeksów do potrzeb aplikacji.
  - Warto korzystać z perspektyw — upraszcza to zwłaszcza projektowanie formularzy i raportów w wielu narzędziach.
- Projektowanie implementacji fizycznej dotyczy rozmieszczenia plików bazy danych na dyskach, planów archiwowania i odtwarzania oraz integracji z narzędziami systemu operacyjnego.



## UML: obiektowy język modelowania

- UML (*Unified Modeling Language*) został zaprojektowany przez Boocha, Jacobsena i Rumbaugh.
- W 1997 roku Object Management Group przyjęła UML 1.1 jako swój standard przemysłowy.



## Diagramy klas

- Służą do modelowania struktury systemu.
- Używane w
  - modelowaniu dziedziny systemu (modelowanie biznesowe)
  - projektowaniu (w tym bazy danych), pojawiają się wtedy klasy „techniczne"
  - inżynierii odwrotnej (*reverse engineering*)

## Diagramy klas

- Podstawowym składnikiem diagramu klas są nazwane klasy.
- Najprostsza reprezentacja klasy nie zawiera nic więcej.
- Oprócz nazwy klasa może zawierać
  - atrybuty
  - metody.



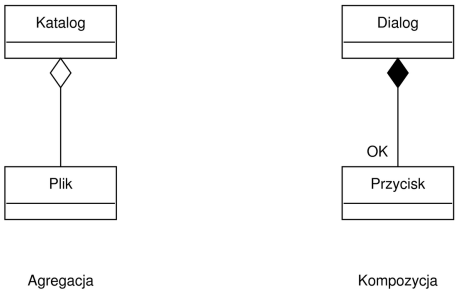
## Diagram klas: powiązania

Rodzaje powiązań:

- Asocjacja
- Agregacja i kompozycja
- Dziedziczenie
- Zależność
- *refinement* — realizacja lub uszczegółowienie



Klasa może być w relacji agregacji z wieloma klasami nadrzędnymi, natomiast w relacji kompozycji tylko z jedną nadrzędną.




**Rysunek:** .

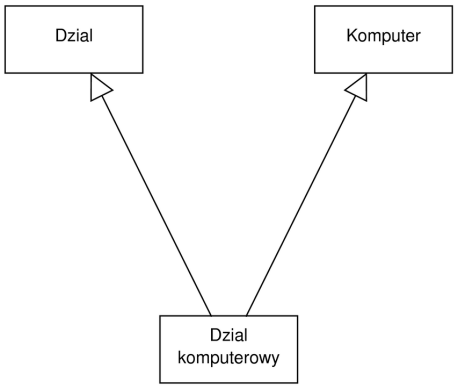


Zależność między klasami oznacza, że klasy nie mają powiązania, natomiast klasa zależna otrzymuje obiekt tej drugiej klasy np. jako parametr jednej ze swoich metod.



## Dziedziczenie

Przykład błędnego dziedziczenia:




**Rysunek:** .

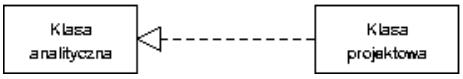


## *Refinement*

- Wiąże ze sobą dwa opisy tej samej rzeczy na różnych poziomach abstrakcji.
- W przypadku gdy wiąże typ z klasą realizującą go zwane jest **realizacją**.
- Może być użyte do modelowania różnych implementacji tej samej rzeczy.



## *Refinement*




**Rysunek:** .



## Kiedy używać agregacji?

- Czy są jakieś operacje na całości, które są automatycznie stosowane do składowych?

## Kiedy wybierać agregację, a kiedy dziedziczenie?

- Jeśli rodzaj obiektu (np. Student) może ulec zmianie (np. z dziennego na zaocznego) bez zmiany reszty atrybutów, to przy dziedziczeniu będzie to wymagało zmiany klasy obiektu!



## Atrybuty i asocjacje

Czasem mamy wątpliwości, czy w danej sytuacji użyć atrybutu czy asocjacji. Ogólna zasada jest następująca:

- Atrybuty służą do łączenia obiektów z **wartościami**. Wartość to element nie posiadający *identity*, np. liczba 1.
- Asocjacje łączą obiekty z obiektami.

Wartości zawsze mogą być zapisane bezpośrednio i odtworzone. Z obiektami jest trudniej, bo należy uwzględnić wszystkie związki.



## Przypadki użycia

- Modelu tego używa się przede wszystkim do opracowania specyfikacji wymagań.
- Dwa podstawowe składniki: aktorzy i przypadki użycia.
- Model przypadków użycia wyznacza granice systemu.
  - Najczęściej są to granice budowanego systemu (aplikacji), jednak przypadki użycia służą także do modelowania przedsiębiorstwa (tzw. **przypadki biznesowe**).
  - Nie należy jednak mieszać tych dwóch podejść w jednym modelu.



## Przypadki użycia

- Każdy przypadek to realizacja jakiejś rzeczywistej **potrzeby** użytkownika (aktora), nazwa przypadku powinna to uwzględniać, a nie opisywać czynności systemu. `Wyświetlanie wyników klasówki` to zła nazwa, lepszą będzie `Przeglądanie wyników klasówki`.
- Przypadki użycia powinny odpowiadać kolejnym fragmentom podręcznika użytkownika. Jeśli używamy pakietów do grupowania dużej liczby przypadków użycia, to każdy pakiet odpowiada rozdziałowi podręcznika.
- Scenariusze (opisy) przypadków użycia należy pisać z perspektywy użytkownika, a nie systemu.
- Nie należy tworzyć przypadków użycia zgrupowanych wokół „obiektów", np. `Przetwarzanie informacji o studentach`. Mają one zwykle za dużo aktorów i zbyt długie specyfikacje.



## Diagram stanów

- Do opisywania historii klas korzysta się z sieci przejść, zwanych też diagramami stanów. Są one uogólnionymi automatami skończonymi.
- Sieć taka składa się z wierzchołków, reprezentujących stany klasy, oraz powiązań między nimi, odpowiadających przejściu z jednego stanu do drugiego.
- Diagram stanów opisuje historię życia obiektu danej klasy.
  - Jeśli operacje na obiekcie danej klasy mogą być wykonywane w dowolnej kolejności, to diagram ten jest najprawdopodobniej zbędny.
  - Jeśli jednak klasa przedstawia (**reifikuje**) przypadek użycia, proces, zarządcę interfejsu itp., wtedy kolejnośc kroków staje się istotna. Trzeba więc określić mechanizm dostarczania i obsługi zdarzeń.



## Diagram stanów

- **Stan** stanowi abstrakcję wartości zmiennych (parametrów itp.) obiektu taką, że ich wszystkie te kombinacje dają jednakową **jakościowo** odpowiedź na zdarzenia.
- **Przejścia** etykietuje się warunkami, których spełnienie powoduje wskazaną zmianę stanu. Zamiast warunków można też stosować zdarzenia, powodujące wskazaną zmianę stanu.
- Oprócz tego z poszczególnymi przejściami mogą być związane akcje sterujące, uaktywaniające odpowiednie procesy.



## Diagram stanów

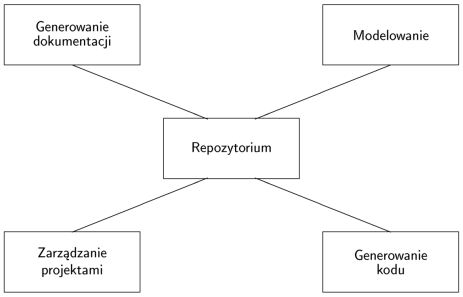
- Przy bardziej złożonym zachowaniu stosuje się sieci zagnieżdżone, zwane też diagramami Harela. W sieciach takich każde przejście dotyczące stanu złożonego dotyczy wszystkich jego podstanów wewnętrznych.
- Modelując zachowanie obiektów staramy się pokazać dwie rzeczy: zmiany stanu oraz interakcje z innymi obiektami.
- Robiąc diagramy stanów dla klas trzeba pamiętać, że klasa musi mieć atrybuty lub związki, przez zmianę których realizuje zmianę stanu. Inaczej mówiąc, stany są reprezentowane pośrednio ich wartościami.



## Narzędzia CASE

- Pakiet CASE to środowisko do modelowania i tworzenia aplikacji.
- Centralną częścią takiego pakietu jest wspólne repozytorium.
- Dzięki temu możliwa jest równoczesna wspólna praca nad wieloma projektami, z dzieleniem wspólnych fragmentów między nimi.
- Dotyczy to zarówno pojedynczych obiektów, jak i całych modeli.

## Architektura typowego narzędzia CASE




**Rysunek:** .



## Funkcje typowego narzędzia CASE

- rysowanie diagramów (obejmuje znajmość semantyki symboli i reguł poprawności)
- repozytorium modeli i ich elementów (np. gdy zmienimy nazwę klasy, powinno to być widoczne na wszystkich diagramach)
- wspieranie nawigacji po modelach
- możliwość pracy kilku użytkowników
- generowanie (szkieletu) kodu
- inżynieria odwrotna (*reverse engineering*)
- integracja z innymi narzędziami
- obsługa poziomów abstrakcji modelu
- wymiana modeli (eksport i import)



## Narzędzia CASE

- Do zarządzania repozytorium używane są zwykle systemy relacyjnych baz danych, np. SQL Anywhere (Sybase).
- Możliwa jest często inżynieria odwrotna, tzn. wciąganie do modeli analitycznych i projektowych obiektów z już istniejących programów i baz danych.



Rozdział VI

# Teoria projektowania relacyjnych baz danych



## Zależności wielowartościowe

- Tabela

  Aktorzy(nazwisko,ulica,miasto,film,rok)

  podaje adresy aktorów (mogą mieć kilka) i filmy, w których grali.
- Każdy aktor mógł grać w wielu filmach i może mieć kilka adresów.
- Ale w pojedynczym wierszu możemy zapisać tylko jeden film i jeden adres.
- Trudno będzie wtedy znajdować wszystkie filmy aktorów mieszkających w Warszawie.
- W zasadzie powinniśmy zapisać wszystkie kombinacje adresów z filmami.



## Definicja zależności wielowartościowej

**Zależność wielowartościowa**

**Zależność wielowartościowa**

$$A_1 \dots A_k \twoheadrightarrow B_1 \dots B_l$$

dla relacji $R(A_1, \dots, A_k, B_1, \dots, B_l, C_1, \dots, C_m)$ oznacza, że jeśli dwie krotki są zgodne na składowych $A_i$, to można w nich zamienić składowe $B_i$ i otrzymane krotki będą także w relacji $R$.

- Inaczej mówiąc, lewa strona każdej takiej zależności nie wyznacza pojedynczej wartości, lecz **zbiór** wartości, np.

  nazwisko $\twoheadrightarrow$ ulica,miasto
  nazwisko $\twoheadrightarrow$ film,rok



## Reguły dla zależności wielowartościowych

**Promocja**

Każda zależność funkcyjna jest zależnością wielowartościową, czyli jeśli zachodzi

$$X \rightarrow Y$$

to zachodzi także

$$X \twoheadrightarrow Y$$

- Niestety odwrotna implikacja nie jest prawdziwa!



## Reguły dla zależności wielowartościowych

**Uzupełnianie**

Jeśli dla relacji $R(X, Y, Z)$ zachodzi

$$X \twoheadrightarrow Y$$

to zachodzi także

$$X \twoheadrightarrow Z$$

## Reguły dla zależności wielowartościowych

**Przechodniość**

Jeśli dla relacji $R(X, Y, Z, V)$ zachodzą

$$X \twoheadrightarrow Y$$
$$Y \twoheadrightarrow Z$$

to zachodzi także

$$X \twoheadrightarrow Z$$



## Reguły dla zależności wielowartościowych

- Podobnie jak dla zależności funkcyjnych, nie wolno rozbijać lewej strony zależności.
- Ale dla zależności wielowartościowych nie wolno również rozbijać prawej strony!



## Czwarta postać normalna — definicja

**Nietrywialna zależność wielowartościowa**

Zależność wielowartościowa $X \twoheadrightarrow Y$ **dla relacji** $R$ jest **nietrywialna**, jeśli

- $Y \not\subseteq X$ oraz
- $X \cup Y$ nie są wszystkimi atrybutami $R$

**4NF**

Relacja $R$ jest w czwartej postaci normalnej (4NF), jeśli dla każdej nietrywialnej zależności $X \twoheadrightarrow Y$ w $R$, $X$ jest nadkluczem $R$.



## Wnioski

- Jeśli relacja jest w 4NF, to jest też w BCNF.
- Odwrotne zawieranie **nie** zachodzi.



## Rozkład do 4NF

- Podobnie jak dla BCNF, szukamy zależności $X \twoheadrightarrow Y$ naruszającej 4NF.
- Rozbijamy relację $R(X, Y, Z)$ na dwie relacje:
  $R_2(X, Y)$
  $R_1(X, Z)$
- Kończymy, gdy już nie ma takich zależności.



## Przykład rozkładu do 4NF

- Zaczynamy od relacji

  Aktorzy(nazwisko,ulica,miasto,film,rok)

  i zależności

  nazwisko $\twoheadrightarrow$ ulica,miasto
  nazwisko $\twoheadrightarrow$ film,rok
- W skład klucza wchodzą wszystkie kolumny tabeli.
- Obie zależności naruszają więc 4NF (bo nie są trywialne).
- Wybieramy do rozkładu pierwszą z nich.



## Przykład rozkładu do 4NF (2)

- Otrzymujemy dwie tabele

  Aktorzy1(nazwisko,ulica,miasto)
  Aktorzy2(nazwisko,film,rok)

  z tymi samymi zależnościami.
- Ponieważ obie zależności stały się trywialne, kończymy.



Rozdział VII

# Transakcje i współbieżność

## Konflikty współbieżności

- Odwiedźmy bazę danych piwiarni i zajmijmy się tabelą Sprzedaje(bar,piwo,cena).
- Przypuśćmy, że w barze „U Szwejka" sprzedaje się tylko dwa gatunki piwa: Okocim po 2,50 zł i Żywiec po 3,50 zł.
- Dzielny redaktor gazety postanowił zbadać (używając naszej bazy danych), jaka jest najwyższa i najniższa cena piwa „U Szwejka".
- W tym samym czasie szef piwiarni zdecydował, że przestaje sprzedawać dotychczasowe piwa i przerzuci się na Heineken po 4,50 zł.



## Konflikty współbieżności

- Pan redaktor wykonuje dwa następujące zapytania (po lewej stronie ich umowne nazwy)

(max)
```
SELECT MAX(cena) FROM Sprzedaje
WHERE bar = 'U Szwejka';
```

(min)
```
SELECT MIN(cena) FROM Sprzedaje
WHERE bar = 'U Szwejka';
```



## Konflikty współbieżności

- A „równocześnie" szef piwiarni wykonał dwa inne polecenia SQL

(del)
```
DELETE FROM Sprzedaje
WHERE bar = 'U Szwejka';
```

(ins)
```
INSERT INTO Sprzedaje
VALUES('U Szwejka','Heineken',4.50);
```



## Przeplecione polecenia

- Przypuśćmy, że powyższe polecenia zostały wykonane w następującej kolejności: max, del, ins, min.
- Popatrzmy na efekty:

| Ceny | Operacja | Wynik |
|---|---|---|
| {2.50,3.50} | max | 3,50 |
| {2.50,3.50} | del | |
| {} | ins | |
| {4.50} | min | 4,50 |

- A więc ostatecznie MAX(...) ¡ MIN(...)!



## Przeplecione polecenia

- Aby tego uniknąć, powinniśmy operacje poszczególnych osób pogrupować w transakcje.
- Wtedy obie operacje pana redaktora wykonają się bezpośrednio po sobie, nie wiadomo tylko, czy przed, czy po zmianie „repertuaru".



## Problem wycofywania

- Szef piwiarni po wykonaniu (bez użycia transakcji) ciągu operacji (del)(ins) postanowił wycofać drugą z nich (`ROLLBACK`)
- Jeśli redaktorowi udało się „wstrzelić" zapytanie między (ins) i `ROLLBACK`, zobaczy wartość (4,50), której nigdy nie było w bazie danych.
- Rozwiązaniem jest znowu użycie transakcji:
  - Efekty transakcji nie są widziane przez innych, dopóki transakcja nie zostanie zatwierdzona (`COMMIT`).



## Blokady

- Dla zapobiegania konfliktom używa się wewnętrznie **blokowania** dostępu do elementów danych używanych przez transakcję.
- Poziomy **ziarnistości** blokad:
  - cała baza danych,
  - pojedyncza relacja,
  - blok wierszy,
  - pojedynczy wiersz.



## Rodzaje transakcji

- **Bezpośrednie** (*direct*);
- **Konwersacyjne** — kilkakrotna wymiana informacji klient/serwer;
- **Wiązane** (*chained*) – wymagają przechowywania kontekstu;
- **Zagnieżdżone**
- **Długotrwałe**.
- **Kolejkowane** – wykonywane z opóźnieniem, np. w celu grupowania;

## Blokady w Oracle

- Blokady można zakładać na całą tabelę

```
LOCK TABLE tabela
IN [SHARE | EXCLUSIVE] MODE
[NOWAIT];
```

- `SHARE` oznacza blokadę dzieloną (tylko przeciw zmianom).
- `EXCLUSIVE` to wyłączna blokada dostępu (w celu dokonania zmian).
- `NOWAIT` chroni przed czekaniem, gdy nie można natychmiast założyć blokady.
- Zdjęcie blokad następuje przez wykonanie `COMMIT` lub `ROLLBACK`.



## Blokady w Oracle (2)

- Lepiej jednak zakładać blokady na wybrane wiersze, np. gdy transakcja odczytuje pewne wiersze, a następnie dokonuje (zwykle w nich) zmian, można użyć

```
SELECT ... FOR UPDATE [NOWAIT];
```

- Taka blokada też jest ważna do końca transakcji.



## Realizacja transakcji

- Dziennik transakcji do zapisywania wszystkich operacji.
- Rejestrowanie wycofań w dzienniku.
- Podczas odtwarzania powtarzamy tylko operacje z zatwierdzonych transakcji.



## Znaczniki czasowe (*timestamps*)

- Inne podejście, dobre gdy głównie odczyty.
- Optymistyczne, wycofanie transakcji gdy konflikt = fizycznie niemożliwy ciąg (lock wycofuje tylko gdy blokada).
- Transakcja utożsamiana z momentem startu.



## Reguły

- Najpierw łatwe zapytanie

```
| ?- blisko(bielany,marymont).
yes
```

- Teraz oczekujemy kilku odpowiedzi

```
| ?- blisko(bielany,What).
What = słodowiec ? ;

What = marymont

yes
```



## Definiowanie silni

```
factorial(0,1).

factorial(N,F) :-
   N > 0,
   N1 is N - 1,
   factorial(N1,F1),
   F is N * F1.
```



## Prolog

- Program składa się z dwóch **klauzul**.
- Pierwsza z nich to klauzula unarna: tylko **nagłówek**, bez treści. Jest to skrót następującego zapisu:

```
factorial(0,1) :- true.
```

- Klauzule unarne bez zmiennych służą do zapisywania pojedynczych **faktów**.
- Druga klauzula to reguła, ponieważ posiada niepustą **treść**.
- Treść to wszystkie literały następujące po ':-' (możemy go czytać „jeśli").
- Literały w treści oddzielamy przecinkami, przecinek pełni rolę operatora koniunkcji.



## Prolog (2)

- Klauzule mają intepretację deklaratywną
  - Klauzula pierwsza mówi: „silnia o wynosi 1".
  - Klauzula druga mówi: „silnia $N$ wynosi $F$ jeśli $N > 0$ oraz jeśli $N1$ oznacza $N - 1$, to silnia $N1$ wynosi $F1$ i $F = N * F1$".

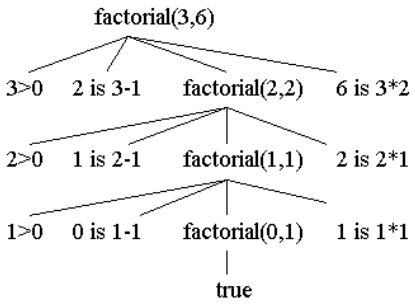
## Zapytania

Chcąc obliczyć silnię od 3 musimy podać w **zapytaniu** zmienną, na której ma znaleźć się wynik — niech to będzie W:

```
?-  factorial(3,W).
W=6
```



## Graf obliczeń




**Rysunek:** .



## Zapytania (2)

W zapytaniu nie muszą wystąpić zmienne:

```
?- factorial(3,6).
yes
?- factorial(5,2).
no
```



## Inna definicja silni

```
silnia(N,F) :- factorial(N,1,F).

factorial(0,F,F).

factorial(N,A,F) :-
    N > 0,
    A1 is N * A,
    N1 is N - 1,
    factorial(N1,A1,F).
```



## Inna definicja silni (2)

- Wprowadziliśmy drugi parametr pełniący rolę **akumulatora**.
- Dzięki temu nasza definicja ma postać iteracyjną, ponieważ wywołanie rekurencyjne jest **redukcyjne**.
- Płacimy za to zmniejszoną czytelnością.



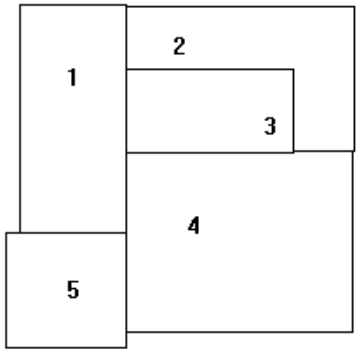
## Kolorowanie map

Zasada:

- Żadne dwa sąsiadujące regiony nie mogą mieć tego samego koloru.



## Mapa




**Rysunek:** Przykładowa mapa.



## Sąsiedztwo

Informacje o sąsiedztwie można w Prologu zapisać następująco:

```
adjacent(1,2).        adjacent(2,1).
adjacent(1,3).        adjacent(3,1).
adjacent(1,4).        adjacent(4,1).
adjacent(1,5).        adjacent(5,1).
adjacent(2,3).        adjacent(3,2).
adjacent(2,4).        adjacent(4,2).
adjacent(3,4).        adjacent(4,3).
adjacent(4,5).        adjacent(5,4).
```

## Sąsiedztwo (2)

Po załadowaniu tych faktów do interpretera można badać sąsiedztwo regionów:

```
?- adjacent(2,3).
yes
?- adjacent(5,3).
no
?- adjacent(3,R).
R = 1 ;
R = 2 ;
R = 4 ;
no
```



## Kolorowania

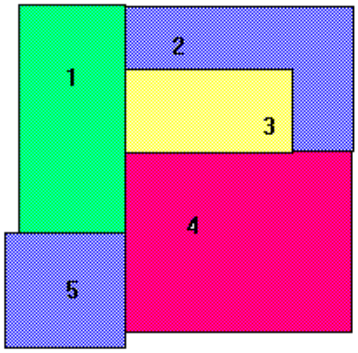
Kolorowania także możemy opisać klauzulami unarnymi.

```
color(1,red,a).     color(1,red,b).
color(2,blue,a).    color(2,blue,b).
color(3,green,a).   color(3,green,b).
color(4,yellow,a).  color(4,blue,b).
color(5,blue,a).    color(5,green,b).
```

Kolorowania oznaczyliśmy symbolami a i b.



## Kolorowania (2)




**Rysunek:** Kolorowania.



## Konflikty

Błędne kolorowanie musi zawierać konflikt — dwa sąsiednie regiony pokolorowane tym samym kolorem:

```
conflict(Coloring) :-
   adjacent(X,Y),
   color(X,Color,Coloring),
   color(Y,Color,Coloring).
```



## Konflikty (2)

Teraz możemy zbadać nasze kolorowania:

```
?- conflict(a).
no
?- conflict(b).
yes
?- conflict(Which).
Which = b
```



## Konflikty (3)

Konfliktowość mozna też zdefiniować inaczej:

```
conflict(R1,R2,Coloring) :-
   adjacent(R1,R2),
   color(R1,Color,Coloring),
   color(R2,Color,Coloring).
```



## Polimorfizm

Prolog dopuszcza równoczesne definiowanie predykatów o tej samej nazwie i różnej liczbie parametrów, wewnętrznie nazywając je 'conflict/1' i 'conflict/3'.



## Więcej o konfliktach

Możemy poznać konfliktowe regiony, a nawet ich (wspólny) kolor

```
?- conflict(R1,R2,b).
R1 = 2   R2 = 4
?- conflict(R1,R2,b),color(R1,C,b).
R1 = 2   R2 = 4   C = blue
```

## Rozpoznawanie zwierząt

```
zoolog :- hypothesize(Zwierzak),
      write('Przypuszczam, że ten zwierzak to: '),
      write(Zwierzak),
      nl,
      undo.
```



## *Unique name assumption*

**Założenie o unikalności nazw**

Indywidua (stałe) o różnych nazwach są różne. Inaczej mówiąc, nie ma aksjomatów równościowych dla stałych.



## Unikalność nazw: przykład

- W naszej przykładowej bazie

  Zwierzak(Papuga,Kropka,3.50)
  Zwierzak(Papuga,Lulu,5.35)
  Zwierzak(Papuga,Hipek,3.50)
  Zwierzak(Lis,Fufu,6.35)
  Zwierzak(Krokodyl,Czako,75.00)
- zachodzi

  Zwierzak(Krokodyl,Fufu,6.35)
- ponieważ zakładamy, że zachodzi

  Krokodyl $\neq$ Lis



## Zapytania

- Do zapisywania zapytań będziemy używać formuł zawierających zmienne.
- Ograniczymy się do formuł prostych.
- Zapytania mają wtedy postać wzorców — struktur podobnych do faktów, mogących jednak zawierać zmienne.
- Zmiennymi będą symbole poprzedzone znakami zapytania, np.

  `Ojciec(Jan,?x)`

  jest zapytaniem o wszystkie dzieci Jana.
- Odpowiedź na zapytanie składa się z bezpośrednio wyszukanych faktów pasujących do zapytania oraz z faktów wyprowadzonych przy użyciu reguł.



## Klauzule

**Klauzula**

**Klauzulą** nazywać będziemy formułę postaci

$$P_1 \wedge P_2 \wedge \cdots \wedge P_k \rightarrow N_1 \vee N_2 \vee \cdots \vee N_l$$

lub też równoważnie

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \vee N_1 \vee N_2 \vee \cdots \vee N_l$$

Wyrażenia $P_1, \ldots, P_k$ nazywać będziemy **poprzednikami** (lub **przesłankami**, zaś wyrażenia $N_1, \ldots, N_l$ **następnikami** lub **wnioskami** klauzuli.



## Rodzaje klauzul

- Klauzule bez poprzedników, zawierające tylko jeden następnik bez zmiennych

  $$\rightarrow N(a_1, \ldots, a_m).$$

  służą do reprezentowania faktów.



## Rodzaje klauzul

- Klauzule bez następników

  $$P_1 \wedge P_2 \wedge \cdots \wedge P_k \rightarrow .$$

  reprezentują negację poprzedników i mogą służyć do zapisywania ograniczeń (więzów poprawności). Stan bazy danych jest legalny, jeśli nie jest spełniona koniunkcja poprzedników, np.

  Ojciec(?x,?y) $\wedge$ Matka(?x,?y) $\rightarrow$.
- Taka klauzula mająca tylko jeden poprzednik bez zmiennych zapisuje jawnie negację faktu, np.

  Ojciec(Jan,Piotr) $\rightarrow$.

  oznacza, że Jan nie jest ojcem Piotra.



## Rodzaje klauzul

- Klauzula o pojedynczym następniku

  $$P_1 \wedge P_2 \wedge \cdots \wedge P_k \rightarrow R$$

  służy w dedukcyjnych bazach danych do definiowania relacji R.
- Jak zobaczymy dalej, pełna definicja relacji może wymagać kilku klauzul.
- W relacyjnych bazach danych klauzule takie (tak jak i następne) służą do zapisywania ograniczeń.

## Rodzaje klauzul

- Klauzule postaci
$$\rightarrow N_1 \vee N_2 \vee \cdots \vee N_k$$
wyrażają wiedzę niekompletną: wiemy, że spełniony jest jeden lub więcej następników, ale nie wiemy który.
- Przykład:
$$\rightarrow \text{Student(Jan,Matematyka)} \vee \text{Student(Jan,Biologia)}$$



## Rodzaje klauzul

- Klauzulą o pełnej postaci
$$P_1 \wedge P_2 \wedge \cdots \wedge P_k \rightarrow N_1 \vee N_2 \vee \cdots \vee N_l$$
można interpretować jako warunkowe informacje o niekompletnej wiedzy.
- Można też ich używać jako ograniczeń, np. aby zapisać, że każdy ma co najwyżej dwoje rodziców użyjemy

```
Rodzic(?p,?a) ∧ Rodzic(?q,?a) ∧ Rodzic(?r,?a)
  → (?p = ?q) ∨ (?p = ?r) ∨ (?q = ?r)
```



## Rodzaje klauzul

- Klauzula pusta, czyli klauzula bez poprzedników i następników, oznacza fałsz.
- Klauzula taka nie powinna nigdy wystąpić w niesprzecznej bazie danych.



## Dedukcyjne bazy danych

W dedukcyjnych bazach danych dopuszcza się tylko niektóre rodzaje klauzul z programowania w logice.

- Nakłada sie dodatkowe ograniczenia.
  - Brak symboli funkcyjnych.
  - Tylko jeden następnik.
    - Takie dedukcyjne bazy danych określa się jako *definite*.
  - Zmienna z konkluzji reguły musi wystąpić w jej treści.
    - Zapewnia to spełnienie założenia o zamkniętości dziedziny.
  - Jako predykaty obliczalne dozwolona tylko arytmetyka.
    - Zauważmy, że formalnie narusza to założenie o zamkniętości dziedziny.
- Jednym z takich rozszerzeń relacyjnych baz danych jest Datalog.



## Datalog

- W Datalogu klauzule zapisujemy jako reguły.
- Każdy predykat jest
  - albo ekstensjonalny: zdefiniowany wyłącznie przez fakty zapisane w bazie danych,
  - albo intensjonalny: zdefiniowany wyłącznie przez reguły.
- Reguły mają nieco zmienioną postać zapisu

*następnik* `<-` *poprzednik* `AND ...`

przy czym zarówno następnik jak i poprzedniki mają postać wzorców, na przykład

```
P(?x,?y,?z) <- Q(?x,?y,?z) AND x > 10
```



## Datalog

- Ponieważ takie reguły, podobnie jak klauzule, odpowiadają implikacjom z logiki, posiadają one zarówno interpretację logiczną jak i proceduralną.
- Na przykład regule

```
Dziadek(?x,?z) <- Ojciec(?x,?y) AND Ojciec(?y,?z)
```

odpowiada implikacja
$$(\forall x, y, z)\ \text{Ojciec}(x, y) \wedge \text{Ojciec}(y, z) \rightarrow \text{Dziadek}(x, z)$$
- Jeśli reguła nie zawiera poprzedników, to jej następnik jest zawsze spełniony.
- Podobnie jak fakty, reguły przechowuje się także w bazie danych.



## Reguły a zapytania

- Reguła pasuje do zapytania, jeśli jej konkluzja pasuje do zapytania.
- Znajdowanie odpowiedzi na zapytanie przy użyciu reguł polega na znalezieniu reguł pasujących do zapytania i dla każdej z nich wyszukanie w sposób spójny (tzn. tak, żeby wartości zmiennych były zgodne) odpowiedzi na podzapytania powstałe z przesłanek (czyli rekursja :-) i złożenie otrzymanych wyników, podstawiając otrzymane wartości zmiennych na zmienne w konkluzji.
- Ostateczną odpowiedzią na zapytanie jest lista wyszukanych faktów (dla reguł będą to konkretyzacje ich konkluzji).



## Zalety

- Pozwalają otrzymywać odpowiedzi wymagające domknięcia przechodniego.
- Przykład:
  - Mamy tabelę Zawiera(obiekt,składnik,ile) dotyczącą pojazdów (np. rowerów, sanek itp.) podającą obiekty i ich bezpośrednie składowe.
  - Chcemy otrzymać informację o wszystkich częściach wchodzących w skład obiektu 'rower'.

## Przykładowa tabela

Zawiera

| obiekt | składnik | ile |
|---|---|---|
| rower | koło | 2 |
| koło | piasta | 1 |
| koło | obręcz | 1 |
| obręcz | szprycha | 20 |
| koło | opona | 1 |
| rower | rama | 1 |
| rama | siodełko | 1 |
| rama | kierownica | 1 |



## Definicja predykatu Części

- Definicja w Datalogu

```
Czesci(?calosc,?czesc)
  <- Zawiera(?calosc,?czesc,?_).

Czesci(?calosc,?czesc)
  <- Zawiera(?calosc,?skladnik,?_)
     AND Czesci(?skladnik,?czesc).
```

- i samo zapytanie

```
Czesci('rower',?co) ?
```



## Realizacja zapytań

- W odróżnieniu od programowania w logice w Datalogu nie używa się nawracania.
- Budowa odpowiedzi odbywa się *bottom-up*: rozpoczynamy od pustej relacji `Czesci` i iteracyjnie dodajemy do niej wszystkie wiersze, które w tym momencie można otrzymać z reguł.
- Optymalizacja: w kądym kroku używam reguł tylko dla wierszy dodanych w poprzednim kroku.
- Zauważmy, że jest to typowy algorytm wyznaczania minimalnego punktu stałego (tzw. **minimalnego modelu**).
- Z otrzymanej **ekstensji** relacji wybieramy tylko wiersze pasujące do zapytania.



## Przykład obliczania odpowiedzi

- Rozpoczynam od pustej tabeli `Czesci`.
- Krok 1: z pierwszej reguły dodaję do niej wiersze `<'rower','koło'>`, `<'koło','piasta'>`, `<'koło','obręcz'>`, `<'obręcz','szprycha'>`, `<'koło','opona'>`, `<'rower','rama'>`, `<'rama','siodełko'>`, `<'rama','kierownica'>`.
- Krok 2: używając drugiej reguły dodaję wiersze ¡'rower','piasta'¿, ¡'rower','obręcz'¿, ¡'rower','opona'¿, ¡'rower','siodełko'¿, ¡'rower','kierownica'¿, ¡'koło','szprycha'¿.
- Krok 3: używając drugiej reguły dodaję wiersz ¡'rower','szprycha'¿.
- Krok 4: ponieważ żadna reguła nie produkuje już nowych wierszy, kończę iterację i wybieram wiersze ¡'rower','koło'¿, ¡'rower','rama'¿, ¡'rower','piasta'¿, ¡'rower','obręcz'¿, ¡'rower','opona'¿, ¡'rower','siodełko'¿, ¡'rower','kierownica'¿, ¡'rower','szprycha'¿.



## Rekursja w SQL

- Zapytania rekurencyjne w SQL-3 definiuje się konstrukcją

```
WITH [RECURSIVE] R(...) AS
     zapytanie używające R
  SELECT ...;
```

- Można tak zdefiniować kilka pomocniczych relacji wzajemnie rekurancyjnych.



## Przykład rekursji w SQL

- Zapiszemy nasze zapytanie z Datalogu w SQL

```
WITH RECURSIVE Czesci(calosc,czesc) AS
    (SELECT obiekt,skladnik FROM Zawiera)
    UNION
    (SELECT obiekt,czesc
     FROM Zawiera, Czesci
     WHERE skladnik = calosc)
  SELECT czesc FROM Czesci
  WHERE calosc = 'rower';
```



## Negacja

- W poprzednikach reguł Datalogu można poprzedzać predykaty negacją `NOT`.
- Taki literał jest spełniony, jeśli ta negacja jest prawdziwa dla tego predykatu w minimalnym modelu.
- Aby można to było sprawdzać, program z negacjami musi być **stratyfikowalny**: reguły predykatów intensjonalnych dzielimy na warstwy tak, by
  - Definicja predykatu nie była w wyższej warstwie, niż jego wykorzystanie w definicjach innych predykatów.
  - Negacja predykatu występowała w wyższej warstwie niż jego definicja.
- Minimalne punkty stałe wyznaczamy **kolejno** poczynając od najniższej warstwy.
- Podobne ograniczenie na poprawność rekursji z negacją obowiązuje w SQL-3.



## Typowa architektura

- Lokalne bazy danych w oddziałach obsługują OLTP.
- Nocą kopiuje się lokalne bazy danych do centralnej hurtowni danych.
- Analitycy używają hurtowni do zapytań OLAP.

## Co to jest hurtownia?

- Kolekcja rozmaitych danych
  - zorientowana tematycznie
  - przeznaczona dla menadżerów, do podejmowania decyzji
  - często kopia danych operacyjnych
  - z dodatkowymi informacjami (np. podsumowania, historia)
  - zintegrowana
  - zmienna w czasie
  - nieulotna



## Co to jest hurtownia?

- Kolekcja narzędzi do:
  - zbierania danych
  - czyszczenia i integracji danych
  - analizy i raportowania
  - monitorowania i zarządzania hurtownią



## Konieczność integracji danych

- Obecnie duże firmy (np. banki, towarzystwa ubezpieczeniowe) posiadają co najmniej kilka operacyjnych baz danych.
- Obok własnych danych operacyjnych często korzysta się również ze źródeł zewnętrznych: wyników badań rynkowych, publicznych administracyjnych baz danych itp.
- Zanim więc przystąpi się do analizy zawartej w nich informacji, należy dokonać **integracji danych** z różnych źródeł.



## Integracja danych

- Informacje ze źródłowych (najczęściej operacyjnych) baz danych są łączone w zbiorczą bazę danych, nazywaną **hurtownią danych** (*data warehouse*).
- Oddzielenie od systemów operacyjnych umożliwia operowanie również danymi historycznymi — potrzebne np. do analizy trendów.



## Hurtownie danych

- Dwa sposoby realizacji:
  - Realna zbiorcza baza danych wraz procedurami okresowego doładowywania nowymi informacjami źródłowymi.
  - Hurtownia wirtualna, oparta na koncepcji **mediatora** i *wrapper*ów.



## Wybór źródeł danych

- Idealnym rozwiązaniem przy wyborze danych do załadowania do hurtowni danych byłoby wstępne określenie wszystkich zapytań, które będą generowane przez aplikacje odpalone na hurtowni danych i ustalenie wszystkich tabel i pól zawartych w tych zapytaniach.
- Zaletą hurtowni wirtualnej jest możliwość zbadania, jakie dane źródłowe są potrzebne w używanych zapytaniach.
- Po przeanalizowaniu logów wygenerowanych przez zapytania z wszystkich aplikacji łatwiej określić schemat hurtowni.



## Łączenie danych

- Łączenie to okazja do usunięcia błędów.
- Przede wszystkim jednak dane są uspójniane.
- Przykład: pole `klient` może
  - w jednej z baz oprócz imienia i nazwiska obejmować tytuł (np. „prof.”),
  - w drugiej na tytuł jest przeznaczone osobne pole,
  - a w trzeciej imię i nazwisko trzymane są w osobnych polach, a tytułu brak.
- Czyszczenie danych obejmuje też przeformatowanie danych i eliminację duplikatów.



## Kroki budowy hurtowni

1. Analiza źródłowych danych pod względem rozmieszczenia, spisanie rodzaju i zawartości danych.
2. Projekt hurtowni danych zawierający spis dostępnych źródłowych baz danych wraz z narzędziami scalającymi.
3. Wydobywanie odpowiednich danych z baz źródłowych, przekształcenie wydobytych danych tak aby pasowały do zaprojektowanej hurtowni danych. Załadowanie przekształconych danych do hurtowni danych.

## Słownik (repozytorium) metadanych

Zawiera informacje:

- skąd pochodzą dane (z jakiej bazy danych i jakiej tabeli);
- co reprezentują, np. kolumna `salesyr` to „roczna sprzedaż brutto towaru wyrażona w złotych";
- jak dane są przechowywane (format zapisu), np. `salesyr` jako `fixed-decimal`;
- kto odpowiada za aktualizację danych (dział/rola), np. „księgowość";
- kiedy dana jest zwykle zmieniana, np. „na koniec każdego roku".



## Błędne dane

- Wartości spoza zakresu poprawności. Są one zwykle łatwe do rozpoznania.
  - Ten typ błędu zdarza się, kiedy, powiedzmy, płeć klienta, która powinna mieć wartość "F" lub "M", zawiera inną wartość, taką jak "X".
  - Odnosi się to także do wartości numerycznych, takich jak wiek. Poprawne wartości mogłyby byćzdefiniowane między 18 a 100 lat. Jeśli wiek znajduje się poza tym zakresem, to uznaje się, że wystąpił błąd.
- Błędy referencyjne. Jest to każdy błąd naruszający więzy spójności referencyjnej.
- Błędy poprawności. Są to błędy w danych niemal niemożliwe do wykrycia: wartości, które po prostu zostały wstawione niepoprawnie, ale są z zakresu poprawności. Na przykład wstawiono wiek klienta jako 26, podczas gdy powinno być 62. Błędy takie pochodzą z operacyjnych baz danych i w zasadzie tam właśnie powinny być korygowane.



## Ujednolicenie danych

Jednolitość jest problemem dla wielu przedsiębiorstw, zwłaszcza tych, które używają różnych typów narzędzi.

- Niektóre różnice dotyczą kwestii podstawowych, jak choćby kodowanie znaków.
  - W większości systemów stosuje się kod ASCII (American Standard Code for Information Interchange).
  - Są jednak wyjątki. IBM oparł wszystkie systemy „mainframe" na całkowicie odmiennym kodowaniu znaków, zwanym EBCDIC (Extended Binary Coded Decimal Interchange Code).
  - Litera „P" w kodzie ASCII ma wartość dziesiętną 80, a w EBCDIC wartość 215 (znakiem, który ma wartość 80 w EBCDIC jest „&").
- Inne różnice dotyczą dat. Większość systemów zarządzania bazami danych obsługują typ "Date", ale format przechowywania jest zależny od DBMS.



## Ujednolicenie danych

- Jeszcze subtelniejsze są różnice wewnątrz różnych aplikacji tworzonych przy użyciu tych samych narzędzi.
  - Może się to na przykład zdarzyć, że jeden z projektantów postanowi zapisywać adresy klientów w pięciu polach po 25 znaków każde, inny zaś zdecyduje się gromadzić je w jednym polu typu Varchar(100).
  - Kolor czarny w jednej aplikacji reprezentuje się napisem "BLACK", w drugiej napisem "BL", a w trzeciej "BL" oznacza kolor niebieski.
- Różnice semantyczne: w jednej bazie danych odróżnia się półciężarówki od mikrobusów, a w drugiej nie.
- Katalogi nazw i adresów pozwalają naprawić błędy przy wprowadzaniu danych i pozwolą uzupełnić brakujące informacje takie jak kod adresowy, powiat itp.
- Takie naprawianie typowych błędów wymaga jednak interwencji człowieka, który zatwierdzi poprawki programu.



## Problem wydajności i skalowalności

- Relacyjne bazy danych używają metod przyśpieszania dostępu takich jak haszowanie lub indeksy do wybrania małej liczby pożądanych rekordów bez potrzeby skanowania całej tabeli lub bazy danych.
- Takie metody przyśpieszania dostępu są wysoce efektywne w odpowiedziach na zapytania o pojedyncze pole (lub małą liczbę pól), kiedy wynikiem jest niewielka część całej tabeli.
- Przykładem takich zapytań jest: „znajdź wszystkich 25 letnich programistów".
- Odpowiedzi na takie zapytania są szybkie, gdyż może być stworzony indeks dla kolumny „wiek" lub kolumny „zawód".



## Problem wydajności i skalowalności

- Klasyczne metody dostępu sa mało pomocne w odpowiedziach na zapytania, których rezultatem jest znaczna część tabeli.
- Przykładem jest „znajdź wszystkich młodych pracowników".



## Analityczne bazy danych

- Inaczej *On-Line Analytical Processing* : OLAP
- Rosnące znaczenie: 8 mld. dol. w 1998 roku.
- Od komputerów biurkowych po olbrzymie konfiguracje:
- Wiele modnych haseł, zaklęć
  - zwijanie i rozwijanie, drążenie, MOLAP, obracanie.



## Podstawowe zagadnienia

- Co to jest analityczna baza danych?
- Modele i operacje
- Implementacja analitycznej bazy danych
- Kierunki rozwojowe

## Data Mart

- Mała hurtownia
- Obejmuje tylko część tematyki organizacji, np.
  - marketing: klienci, towary, sprzedaż
- Model dostosowany do potrzeb działu.
- Najczęściej informacja wstępnie zagregowana
  - Eliminacja zbędnych detali
  - Wybieramy pewien krytyczny poziom szczegółowości.



## Narzędzia do zapytań i analiz

- Budowanie zapytań
- Generatory raportów
  - porównania: wzrost, spadek
  - trendy,
  - grafy
- Arkusze kalkulacyjne
- Interfejs WWW
- Data Mining



## Inne operacje

- Funkcje po czasie
  - np. średnie po różnych okresach
- Atrybuty obliczane
  - np. marża = sprzedaż * stopa
- Zapytania tekstowe, np.
  - znajdź dokumenty zawierające słowa $A$ i $B$
  - uporządkuj dokumenty według częstości występowania słów $X$, $Y$ i $Z$



## Modele danych i operatory

- Modele danych
  - relacja
  - gwiazda i płatek śniegu
  - kostka: rozwinięcie idei arkusza kalkulacyjnego (tablice wielowymiarowe)
- Operatory
  - slice & dice
  - roll-up, drill down
  - pivoting
  - inne



## Wielowymiarowy model danych

- Najczęściej używa się wielowymiarowych bazy danych z uwagi na analityczny model danych o postaci kostki wielowymiarowej obejmujący:
  - **fakty** zwane też **miarami** (*measures*), np. liczba sprzedanych samochodów;
  - **wymiary** (*dimensions*), np. miesiące, regiony sprzedaży.



## Wymiary

- Wymiary tworzą zazwyczaj hierarchie, np. dla czasu będzie to rok-kwartał-miesiąc-dzień.
- Dzięki temu możliwa jest interakcyjna zmiana poziomu szczegółowości (ziarnistości) oglądanej informacji.
- W bardziej złożonych przypadkach hierarchie mogą rozgałęziać się, np. podział na tygodnie jest niezgodny z podziałem na miesiące.



## Baza danych

- Źródłem danych jest najczęściej hurtownia danych (rzeczywista lub wirtualna).
- Bezpośrednie trzymanie w bazie danych informacji o faktach dla wszystkich poziomów szczegółowości może być kosztowne
  - Tylko dla najczęściej używanych poziomów hierarchii.
  - Pozostałe są wyliczane na bieżąco w razie potrzeby.
- Przy agregowaniu miar warto pamiętać o różnych regułach liczenia, np.
  - Wielkość sprzedaży jest na ogół sumowana
  - Temperatura lub cena będą raczej uśredniane.
- W analitycznej bazie danych trzymane są w zasadzie dane zagregowane.
- Aby obejrzeć dane szczegółowe (*drill-through*) konieczne jest sięgnięcie do hurtowni danych lub bazy operacyjnej.
- Ponieważ jest to kosztowne czasowo, taka potrzeba nie powinna występować zbyt często.



## Operacje na danych

- Przecinanie i rzutowanie (*slice and dice*)
- Zmiana poziomu szczegółowości: drążenie lub rozwijanie (*drill-down*) i zwijanie (*roll-up*),
- obracanie (*pivot*): zmienia położenie wymiaru na „wykresie".

## Podejścia do budowy bazy OLAP

1. ROLAP = „relacyjny OLAP": dopasowujemy relacyjny DBMS do schematu gwiazdy.
2. MOLAP = „wielowymiarowy OLAP": używamy specjalizowanego DBMS z modelem w rodzaju „kostka danych".



## Schemat gwiazdy

- **Schemat gwiazdy** to typowy sposób organizacji danych dla relacyjnej bazy danych dla OLAP.
- Obejmuje:
  - **Tabelę faktów**: olbrzymi zbiór faktów takich jak informacje o sprzedaży.
  - **Tabele wymiarów**: mniejsze, statyczne informacje o obiektach, których dotyczą fakty.
- Uogólnienie: model płatka śniegu.
  - Hierarchie tabel dla poszczególnych wymiarów: normalizacja wymiarów.



## Przykład schematu gwiazdy

- Chcemy gromadzić w hurtowni danych informacje o sprzedaży piwa: bar, marka piwa, piwosz, który je zakupił, dzień, godzina oraz cena.
- Tabelą faktów będzie relacja:

```
Sprzedaż(bar,piwo,piwosz,dzień,godzina,cena)
```



## Przykład, cd.

- Tabele wymiarów zawierają informacje o barach, piwach i piwoszach:

```
Bary(bar,adres,licencja)
Piwa(piwo,prod)
Piwosze(piwosz,adres,tel)
```



## Atrybuty wymiarów i atrybuty zależne

- Dwa rodzaje atrybutów w tabeli faktów:
  - **Atrybuty wymiarów**: klucze tabel wymiarów.
  - **Atrybuty zależne**: wartości wyznaczone przez atrybuty wymiarów krotki.



## Przykład: atrybut zależny

- cena jest atrybutem zależnym w przykładowej relacji Sprzedaż.
- Jest ona wyznaczona przez kombinację atrybutów wymiarów: bar, piwo, piwosz i czas (kombinacja atrybutów daty i godziny).



## Techniki ROLAP

- **Indeksy bitmapowe**: dla każdej wartości klucza indeksowego w tabeli wymiaru (np. dla każdego piwa w tabeli Piwa) tworzymy wektor bitowy podający, które krotki w tabeli faktów zawierają tę wartość.
- **Perspektywy zmaterializowane**: w hurtowni przechowujemy gotowe odpowiedzi na kilka użytecznych zapytań (perspektywy).



## Typowe zapytanie OLAP

- Zapytanie OLAP często zaczyna się od „star join": złączenia naturalnego tabeli faktów z wszystkimi lub większością tabel wymiarów.
- Przykład:

```
SELECT *
FROM Sprzedaż,Bary,Piwa,Piwosze
WHERE Sprzedaż.bar = Piwa.bar
  AND Sprzedaż.piwo = Piwa.piwo
  AND Sprzedaż.piwosz = Piwosze.piwosz;
```

## Typowe zapytanie OLAP (2)

- Rozpoczyna się złączeniem gwiaździstym.
- Wybiera interesujące krotki używając danych z tabel wymiarów.
- Grupuje po jednym lub więcej wymiarach.
- Agreguje niektóre atrybuty wyniku.



## Przykład zapytania OLAP

- Dla każdego baru w Poznaniu podaj całkowitą sprzedaż każdego piwa wytwarzanego przez browar Anheuser-Busch.
- Filtr: adres = "Poznań" i prod = "Anheuser-Busch".
- Grupowanie: po bar i piwo.
- Agregacja: Suma po cena.



## Przykład: SQL

```
SELECT bar, piwo, SUM(cena)
FROM Sprzedaż NATURAL JOIN Bary
     NATURAL JOIN Piwa
WHERE addr = 'Poznań'
  AND prod = 'Anheuser-Busch'
GROUP BY bar, piwo;
```



## Perspektywy zmaterializowane

- Bezpośrednie wykonanie naszego zapytania dla tabeli Sprzedaż i tabel wymiarów może trwać za długo.
- Jeśli utworzymy perspektywę zmaterializowaną zawierającą odpowiednie informacje, będziemy mogli znacznie szybciej podać odpowiedź.



## Przykład: Perspektywa zmaterializowana

- Jaka perspektywa mogłaby nam pomóc?
- Podstawowe wymagania:
  1. Musi łączyć co najmniej Sprzedaż, Bary i Piwa.
  2. Musi grupować co najmniej po bar i piwo.
  3. Nie musi wybierać barów w Poznaniu ani piw Anheuser-Busch.
  4. Nie musi wycinać kolumn adres ani prod.



## Przykład — c.d.

- A oto przydatna perspektywa:

```
CREATE VIEW BaPiS(bar, adres, piwo,
                  prod, sprzedaż) AS
  SELECT bar, adres, piwo, prod,
         SUM(cena) sprzedaż
  FROM Sprzedaż NATURAL JOIN Bary
       NATURAL JOIN Piwa
  GROUP BY bar, adres, piwo, prod;
```

- Ponieważ bar → adres oraz piwo → prod, jest to pozorne grupowanie, konieczne ponieważ adres i prod występują we frazie SELECT.



## Przykład — zakończenie

- Przeformułowane zapytanie z użyciem zmaterializowanej perspektwy BaPiS:

```
SELECT bar, piwo, sprzedaż
FROM BaPiS
WHERE adres = 'Poznań'
  AND prod = 'Anheuser-Busch';
```



## Aspekty materializacji

- Typ i częstość zapytań
- Czas odpowiedzi na zapytania
- Koszt pamięci
- Koszt aktualizacji

## MOLAP i kostki danych

- Klucze tabel wymiarów stają się wymiarami hiperkostki.
  - Przykład: dla danych z tabeli Sprzedaż mamy cztery wymiary: bar, piwo, piwosz i czas.
- Atrybuty zależne (np. cena) występują w punktach (kratkach) kostki.



## Marginesy

- Kostka często zawiera również agregacje (zwykle SUM) wzdłuż hiper-krawędzi kostki.
- **Marginesy** obejmują agregacje jednowymiarowe, dwuwymiarowe, ...



## Przykład: marginesy

- Nasza 4-wymiarowa kostka Sprzedaż obejmuje sumy cena dla każdego baru, każdego piwa, każdego piwosza i każdej jednostki czasu (zapewne dni).
- Zawiera też sumy cena dla wszystkich par bar-piwo, trójek bar-piwosz-dzień, ...



## Struktura kostki

- Każdy wymiar należy traktować jako mający dodatkową wartość *.
- Punkt wewnętrzny z jedną lub więcej współrzędną * zawiera agregaty po wymiarach z *.
- Przykład: Sprzedaż('Pod Żaglem', 'Bud', *, *) zawiera sumę piwa Bud wypitego w „Pod Żaglem" przez wszystkich piwoszy w dowolnym czasie.



## Rozwijanie (*drill-down*)

- *Drill-down* = „deagregacja" = rozbija agregację na jej składniki.
- Przykład: po stwierdzeniu, że „Pod Żaglem" sprzedaje się bardzo mało piw Okocim, rozbić tę sprzedaż na poszczególne gatunki piw Okocim.



## Zwijanie (roll-up)

- *Roll-up* = agregacja po jednym lub więcej wymiarach.
- Przykład: mając tabelę podającą jak wiele piwa Okocim wypija każdy piwosz w każdym barze, zwinąć ją do tabeli podającej ogólną ilość piwa Okocim wypijanego przez każdego z piwoszy.



## Roll-Up i Drill-Down

- Anheuser-Busch dla piwosz/bar

| | Jim | Bob | Mary |
|---|---|---|---|
| Joe's Bar | 45 | 33 | 30 |
| Nut-House | 50 | 36 | 42 |
| Blue Chalk | 38 | 31 | 40 |

- Zwinięcie po Bary
- A-B / piwosz

| Jim | Mary | Bob |
|---|---|---|
| 133 | 100 | 112 |



## Roll-Up i Drill-Down (2)

- Rozwinięcie po Piwa
- Piwa A-B / piwosz

| | Jim | Bob | Mary |
|---|---|---|---|
| Bud | 40 | 29 | 40 |
| M'lob | 45 | 31 | 37 |
| Bud Light | 48 | 40 | 35 |

## Zmaterializowane perspektywy kostek danych

- Dla kostek danych warto robić perspektywy zmaterializowane agregujące po jednym lub więcej wymiarach.
- Wymiary nie powinny być całkowicie agregowane — można ewentualnie grupować po atrybucie z tabeli wymiaru.



## Przykład

- Zmaterializowana perspektywa dla naszej kostki Sprzedaż mogłaby:
  1. Agregować całkowicie po piwosz.
  2. Nie agregować wcale po piwo.
  3. Agregować po czasie według tydzień.
  4. Agregować po miasto dla barów.



## Indeksy

- Tradycyjne techniki
  - B-drzewa, tablice haszujące, R-drzewa, gridy, ...
- Specyficzne
  - listy odwrócone
  - indeksy bitmapowe
  - indeksy złączeniowe
  - indeksy tekstowe



## Użycie list odwróconych

- Zapytanie:
  - Podaj osoby dla których wiek = 20 i imię = „Fred"
- Lista dla wiek = 20: r4, r18, r34, r35
- Lista dla imię = „Fred": r18, r52
- Odpowiedzią jest przecięcie: r18



## Użycie bitmap

- Zapytanie:
  - Podaj osoby dla których wiek = 20 i imię = „Fred"
- Mapa dla wiek = 20: 1101100000
- Mapa dla imię = „Fred": 0100000001
- Odpowiedzią jest przecięcie: 010000000000
- Dobre jeśli mało wartości danych.
- Wektory bitowe można kompresować.



## Relacyjny model danych

Podstawowe składowe:

- **relacje** odpowiadające zarówno typom encji (*entity*), jak i typom związków (*relationship*);
- **wiersze** (krotki) reprezentujące egzemplarze (wystąpienia) encji i związków;
- **kolumny** reprezentujące atrybuty;
- zbiór wartości, które można umieszczać w danej kolumnie, nazywa się jej **dziedziną**. Relacje są związane tylko wtedy, gdy mają kolumny o wspólnej dziedzinie.



## Relacyjny model danych — operacje

- Operacje na danych opisuje się:
  - algebrą relacji;
  - rachunkiem relacji (specjalizacja rachunku predykatów pierwszego rzędu).
- W praktyce używa się języka SQL łączącego obie te metody.



## Wady modelu relacyjnego

- Zorientowany na rekordy (dziedzictwo implementacyjne).
- Konieczność wcześniejszego określenia „schematu" bazy danych, dopiero potem można coś do bazy wstawiać.
- Nie można wstawiać danych nie pasujących do schematu, oficjalnie nie wolno dodawać nowych atrybutów.
- Za słaby do reprezentacji wiedzy: tylko jedna konstrukcja (tabela), atomowe atrybuty.

## Zagnieżdżony relacyjny model danych

- Dopuszcza atrybuty o wartościach zbiorowych.
- Wartości zbiorowe reprezentuje się osobnymi, wewnętrznymi tabelami.



## Obiektowy model danych

- Jedno podstawowe pojęcie do modelowania świata — **obiekt**.
- Z obiektem związany jest jego stan i zachowanie.
- Stan definiuje się wartościami własności (atrybutów) obiektu, mogą one być
  - wartościami elementarnymi (liczby, napisy) lub
  - obiektami, zawierającymi z kolei inne własności.
- Tak więc obiekty można definiować rekurencyjnie.
- Na rynku ok. 25 produktów, np. GemStone, ONTOS, ObjectStore, ENCORE.
  - Nie są to jednak jeszcze pełne i uniwersalne bazy danych.
- Dlatego wiele relacyjnych DBMS uzupełniono o możliwości obiektowe, np. Oracle.



## Własności obiektowego modelu danych

- Zachowanie obiektu opisuje się zbiorem metod — procedur operujących na stanie obiektu.
- Każdy obiekt jest jednoznacznie identyfikowany systemowym identyfikatorem (OID).
- Proste definiowanie obiektów złożonych.
- Obiekty o takich samych własnościach i zachowaniu grupuje się w klasy. Obiekt może być egzemplarzem tylko jednej klasy lub wielu.



## Własności obiektowego modelu danych (2)

- Klasy łączy się w hierarchie dziedziczenia, w których podklasa **dziedziczy** własności i metody nadklasy, dokładając swoje specyficzne.
- Niektóre modele pozwalają na przesłanianie (*overriding*) — zmianę odziedziczonej własności lub metody.



## Różnice w stosunku do modelu relacyjnego

- Atrybuty mogą być wielowartościowe.
  - Eliminuje to potrzebę używania większości złączeń równościowych.
- Możliwość definiowania związków odwrotnych.
- Nie trzeba definiować kluczy, ich rolę pełnią OIDy.
  - Eliminuje to modyfikowanie wartości klucza.
- Dwa rodzaje równości: identyczność i równość wartości.



## Różnice w stosunku do modelu relacyjnego (2)

- Złączenia używane gdy warunki w zapytaniu dotyczą porównywania wartości atrybutów. W przypadku „kluczy zewnętrznych" bezpośrednia nawigacja z użyciem OID.
- Przy ładowaniu powiązanych obiektów z bazy danych do pamięci (buforowanie) OIDy zastępuje się wskaźnikami (*pointer swizzling*), co wielokrotnie przyśpiesza nawigację.
- Wersjonowanie (długie transakcje) — tylko w niektórych OBD.



## Zapytania

Dwa tryby dostępu:

- **Nawigacyjny**: mamy OID obiektu i zaczynając od niego przechodzimy po kolejnych referencjach. Zwykle używany w programach.
- **Język zapytań** (często podobny do SQL): zapytanie opisuje zbiór obiektów. Deklaratywność.



## Zapytania

Przykład: **Znaleźć wszystkie projekty z budżetem ponad 100000 złp i mające sponsora z Warszawy**

- Relacyjnie

```
{p | (∃f)(∃a) Projekt(p) AND Firma(f) AND Adres(a)
 AND p.budżet > 100000 AND p.sponsor = f
 AND f.adres = a AND a.miasto = 'Warszawa'}
```

- Z użyciem wyrażeń ścieżkowych (co pozwala unikać zmiennych „roboczych")

```
{p | Projekt(p) AND p.budżet > 100000
  AND p.sponsor.adres.miasto = 'Warszawa'}
```

## Wersje obiektów

- Potrzebne np. w systemach CAD do eksploracji wariantów.
- Każda wersja ma własny OID, ale zachowujemy **związki wyprowadzenia** między wersjami, najczęściej tworzące hierarchię.
- Hierarchia wersji zwykle trzymana w specjalnym **obiekcie generycznym**.
- Dwa rodzaje odwołań do wersji
  - **specyficzna** (albo **statyczna**): konkretna wersja
  - **generyczna** (albo **dynamiczna**): do **domyślnej** wersji (zwykle ostatniej).



## Problemy z modelem obiektowym

- Brak optymalizacji zapytań. Główny problem to wyrażenia ścieżkowe, trudno dla nich budować indeksy.
- Brak dobrej formalizacji, ale są już algebry podobne do algebry relacji.
- Pięć typowych operacji: suma (*union*, różnica (*difference*), selekcja (*select*), generowanie (*generate*), odwzorowanie (*map*).
- Jednak brak powiązania tych operacji z niskopoziomowymi operacjami fizycznymi (takiego jak w algebrze relacji RBD), stąd ich arbitralność.



## Problemy — c.d.

- Brak uniwersalnych języków zapytań. Zwykle brak zagnieżdżonych podzapytań, funkcji agregujących, grupowania. Brak automatycznego wsparcia dla obsługi ekstensji klasy — zbioru jej egzemplarzy; użytkownik musi sam zdefiniować **kolekcję** (*collection*) i pilnować jej aktualizacji przy dodawaniu i usuwaniu obiektów.
- Kompozycyjność
  - W modelu relacyjnym wynik zapytania jest (anonimową) relacją.
  - Można go więc obrobić kolejnym zapytaniem, co umożliwia składanie zapytań.
  - W modelu obiektowym jest gorzej, obiekty ze zbioru wynikowego mogą nie należeć do żadnej klasy.
- Brak wsparcia dla redefiniowania klas (odpowiednik `ALTER` z SQL), np. dodawania lub usuwania atrybutów.
- Brak perspektyw (ale może są zbędne).



## Problemy — c.d.

- Brak sensownego mechanizmu definiowania uprawnień, nie wiadomo jaka ziarnistość: obiekt, zbiór, klasa, fragment hierarchii?
- Bardzo ograniczone więzy spójności, konieczność realizacji metodami.
- Kłopoty z współbieżnością, ręczne operowanie blokadami.
  - Problem długich transakcji (unika się ich w systemach OLTP dla relacyjnych baz danych).
  - Propozycja: wspólna publiczna baza danych + prywatne bazy danych użytkowników.



## Przykład: Orion

```
make-class 'Instytut'
  superclasses: nil
  attributes: '((obszar-badań: domain string)
                (nazwa-instytutu: domain string)
                (adres: domain Adres)
                (grupa-badawcza:
                  domain (set-of Zespół)))
```



## Orion c.d.

Uwagi:

- Można określać wartości domyślne dla atrybutów
- Wielodziedziczenie



## Zapytania w Orionie

- Brak złączeń, rzutowanie tylko na jeden atrybut lub wszystkie.
- Powód: dzięki temu wynik zawsze należy do jakiejś klasy.

```
(Instytut select :I (:I obszar-badań = 'Bazy danych'))

(Instytut select (:I ((:I obszar-badań = 'Bazy danych')
                      and (:I adres kraj = 'Włochy'))))
```



## Propozycja ODMG

```
interface Instytut
  (extent Instytuty
   key nazwa)
  {attribute string nazwa;
   attribute string obszar-badań;
   attribute Adres adres;
   relationship Zespół grupa-badawcza
   inverse Zespół::afiliacja};
```

## Zapytania w ODMG

Dozwolone wyrażenia ścieżkowe

```
select distinct x from Instytuty x
where x.obszar-badań = 'Bazy danych'
  and x.adres.kraj = 'Włochy';
```



## Implementacja

Rodzaje OID:

- **logiczne**: bez związku z adresem w pamięci zewnętrznej
- **fizyczne**: na podstawie położenia



## Logiczne OID

- Orion: ¡id-klasy,id-egzemplarza¿. Definicje atrybutów i metod trzymane w obiekcie reprezentującym klasę, więc potrzebny szybki dostęp do niego. Migracja obiektu z klasy do klasy trudnsa, bo zmienia OID.
- GemStone: informacja o klasie w samym obiekcie, a nie w OID, wymaga wczesnego pobrania obiektu.



## Fizyczne OID

- $O_2$ używa Wiss (Wisconsin Storage Subsystem), obiekt przechowywany jako rekord Wiss, OID jest identyfikatorem rekordem (RID).
  - Przy zmianie strony *forward reference*.
  - Wymagane tymczasowe OID dla obiektów utworzonych w pamięci, otrzymują trwały OID dopiero przy zatwierdzaniu (*commit*).
- Orion: w wersji rozproszonej OID dodatkowo zawierał identyfikator położenia, nie zmieniający się nawet przy migracji.



## Zagadnienia otwarte

- Czy klasy są obiektami? Inaczej mówiąc, czy schemat trzymany jest osobno?
- Jakie są dozwolone związki między klasami?
- Czy i jak jest wspierana nawigacja?
- Jakie są operacje na obiektach?
- Jak identyfikuje się obiekty?
- Jak wybiera się obiekty?
- Jaką rolę pełni klasyfikacja?
- Czy klasy obiektów mogą ewoluować?
- Czy w rozproszonym systemie sieć powinna być widoczna?
- Czy jest specjalna obsługa obiektów aktywnych?
- Czy „świat" obiektów jest zamknięty (*closed*) czy otwarty?



## Literatura


- Sikha Bagui *Achievements and Weaknesses of Object-Oriented Databases*, Journal of Object Technology, vol. 2, no. 4, July-August 2003, str. 29–41, http://www.jot.fm/issues/issue_2003_07/column2.