Relacyjne Bazy Danych

Andrzej M. Borzyszkowski
PJATK/ Gdańsk

materiały dostępne elektronicznie
http://szuflandia.pjwstk.edu.pl/~amb

Powtórzenie/ Podsumowanie

Co to jest i po co są bazy danych

- Przechowywanie informacji
  - trwałe
  - wiarygodne
  - informacje wzajemnie powiązane (traktowanie danych łącznie, bez zwracania uwagi na podział na pliki)
  - struktura umożliwiająca wyszukiwanie informacji
- Programy związane z bazami danych
  - system zarządzania bazą danych
  - programy narzędziowe SZBD
  - aplikacje/ interfejs użytkownika
  - różne programy wykorzystujące dane

Cechy systemów baz danych

- Zapewnienie spójności
  - współbieżności
  - poufności
  - niezawodności
- Abstrakcja danych (niezależność od programów użytkowych)

Projekt bazy danych

- Diagram encji i związków
- Model relacyjny – relacja ≈ tabela
- Klucze: główny, kandydujący, obcy
- Normalizacja
- SQL, definicja tabel
- Algebra relacji – działania na tabelach
- SQL, dostęp do danych

Diagram encji i związków

- Projekt bazy danych zaczyna się od określenia encji, związków, oraz od decyzji na temat charakteru tych związków
- Encje posiadają swoje atrybuty, związki pomiędzy encjami też czasami można wyposażyć w atrybut
- W diagramach encji i związków zaleca się używać liczby pojedynczej
  - ale tabela odpowiadająca encji będzie zawierać wiele elementów

```
Klient (nazwisko, adres, inne dane);
Towar (nazwa, kod kreskowy, wielkość zapasów,
  ceny kupna, oferowane itd);
Zamówienie (od kogo pochodzi, zestawienie
  towarów, daty wysyłki i inne, koszt wysyłki);
```


Diagram encji i związków, encje

- Pierwsza postać normalna wyklucza możliwość podania zestawienia towarów w encji zamówienie
  - potrzebna jest osobna encja dla poszczególnych pozycji każdego zamówienia
  - dopuszczając, że jeden towar może mieć wiele różnych kodów kreskowych, trzeba stworzyć osobną tabelę dla tych kodów
- Decyzja, by stworzyć osobną tabelę dla wielkości zapasów
  - można podejrzewać, że będzie systematycznie modyfikowana

```
Pozycja (jakiego zamówienia, towar, wielkość
  zamówienia, inne, np. rabat);
Kod_kreskowy (jakiego towaru, kod);
Zapas (czego, ile);
```


Diagram encji i związków, związki

- `Klient <składa> Zamówienie`
  - związek 1 do wiele (zamówienie musi pochodzić od klienta, klient może złożyć 0, 1 lub wiele zamówień)
- `Zamówienie <składa się z> Pozycje`
  - związek 1 do wiele (pozycja musi mieć określony nagłówek zamówienia, zamówienie może mieć wiele pozycji lub być nawet puste)
- `Pozycja <dotyczy> Towaru`
  - związek wiele do 1 (pozycja dotyczy towaru, nie może go nie określić, towar może wystąpić w wielu pozycjach, ale w danych zamówieniu tylko raz)
- `Towar <ma> Kod kreskowy`
  - związek 1 do wiele (dopuszczamy by towar miał wiele różnych kodów, kod kreskowy musi jednoznacznie określać towar)

Diagram encji i związków, związki

`Towar <występuje w> Zapasie`

- związek 1 do 1 (w tabeli zapasów jest najwyżej jedna pozycja dla każdego towaru)

- Uwaga: związek wieloznaczny `Zamówienie <..> Towar`, potencjalnie z dodatkowymi atrybutami np. wielkość zamówienia, został już rozłożony na dodatkową encję i dwa związki "1 do wiele".

`Zamówienie <składa się z> Pozycja <dotyczy> Towaru`

Model relacyjny

- Dane przechowywane są w tabelach
- Wierszami są pojedyncze encje, kolumnami atrybuty encji
  - atrybuty są elementarne (liczby, napisy bez struktury)
- W komórkach tabeli przechowywane są pojedyncze wartości, nigdy listy itp.
- Informacyjna zawartość bazy może być/najczęściej będzie podzielona pomiędzy wiele tabel
- Operacje na relacjach (tabelach):
  - obcięcie
  - rzut
  - złączenie, wewnętrzne i zewnętrzne – realizuje ideę integracji danych
  - iloczyn kartezjański (w praktyce tylko jako narzędzie złączenia)

Klucze

- Klucz kandydujący: jednoznacznie określa encję/krotkę
  - tzn. nie może się powtórzyć (integralność klucza)
  - najczęściej jest jednym atrybutem
  - jeśli składa się z większej liczby, to musi być tego powód
  - jeden klucz jest zdefiniowany jako główny (PRIMARY)
  - system może sam numerować krotki
- Klucz obcy (FOREIGN KEY): jednoznacznie identyfikuje krotkę w drugiej tabeli
  - prawie zawsze jest to pojedynczy atrybut
  - prawie zawsze odnosi się do innej tabeli (ale nie musi)
  - krotka wskazywana przez klucz obcy w jednej tabeli musi istnieć w drugiej tabeli (integralność referencyjna)
  - nieoczywiste usuwanie krotek z kluczem obcym
  - ON DELETE NO ACTION/ CASCADE/ SET NULL/ SET DEFAULT

Zasady definiowania tabel

- Związek 1 do wiele powoduje zadeklarowanie klucza obcego po stronie „wiele”
- Encje będące adresatami klucza obcego na pewno będą wymagać klucza głównego, w pozostałych encjach może on być sensowny, ale niekonieczny
- Klucz główny może być automatycznie nadawany przez system, jeśli nie ma oczywistego kandydata
- Encje realizujące złożony związek wieloznaczny powodują, że zestaw kluczy obcych jest kluczem kandydującym
- Związek 1 do 1 powoduje zadeklarowanie klucza obcego będącego jednocześnie kluczem kandydującym

Tabele, przykład

```
create table klient (
    nr              serial      ,
    tytul           char(4)     ,
    imie            varchar(16) ,
    nazwisko        varchar(32)         not null,
    kod_pocztowy    char(6)             not null,
    miasto          varchar(32) ,
    ulica_dom       varchar(64) ,
    telefon         varchar(11) ,
    CONSTRAINT      klient_nr_pk PRIMARY KEY(nr)        );
create table zamowienie (
    nr              serial      ,
    klient_nr       integer             not null,
    data_zlozenia   date                not null,
    data_wysylki    date        ,
    koszt_wysylki   numeric(7,2),
    CONSTRAINT      zamowienie_nr_pk PRIMARY KEY(nr),
    CONSTRAINT      klient_fk FOREIGN KEY(klient_nr)
       REFERENCES klient(nr)
       ON UPDATE CASCADE ON DELETE CASCADE          );
```


Tabele, SQL

```
create table zamowienie (          nazwa tabeli
    nr              serial      ,  typ automatycznego numerowania
    klient_nr       integer        not null,  atrybut musi być
    data_zlozenia   date           not null,  określony
    data_wysylki    date        ,             nie musi być określony
    koszt_wysylki   numeric(7,2),             nazwa atrybutu
    CONSTRAINT      zamowienie_nr_pk PRIMARY KEY(nr),
                    nazwa warunku integralności, najczęściej
  w postaci rozwiniętej: nazwa tabeli_atrybutu_pk
                          nazwa atrybutu będącego kluczem głównym
    CONSTRAINT klient_fk FOREIGN KEY(klient_nr)
       REFERENCES klient(nr)
  nazwa tabeli i atrybutu w innej tabeli wskazywanego przez klucz obcy
       ON UPDATE CASCADE ON DELETE CASCADE          );
  określa zachowanie systemu w razie usuwania naruszającego integralność
  referencyjną
```


Tabele, przykład c.d.

```
create table towar
(
    nr              serial      ,
    opis            varchar(64)         not null,
    koszt           numeric(7,2)        not null,
    cena            numeric(7,2),
    CONSTRAINT      towar_nr_pk PRIMARY KEY(nr)
);
create table zapas
(
    towar_nr        integer not null,
    ilosc           integer not null,
    CONSTRAINT      zapas_towar_nr_pk PRIMARY KEY (towar_nr),
    CONSTRAINT      towar_nr_fk FOREIGN KEY(towar_nr)
       REFERENCES towar(nr)
       ON UPDATE CASCADE ON DELETE CASCADE          );
```

- klucz obcy i jednocześnie główny realizuje związek jedno-jednoznaczny

Tabele, przykład c.d.

```
create table pozycja
(
    zamowienie_nr integer not null,
    towar_nr      integer not null,
    ilosc         integer not null,
    CONSTRAINT    pozycja_pk
       PRIMARY KEY(zamowienie_nr, towar_nr),
    CONSTRAINT    pozycja_zamowienie_nr_fk
       FOREIGN KEY(zamowienie_nr)
       REFERENCES zamowienie(nr)
       ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT    pozycja_towar_nr_fk
       FOREIGN KEY(towar_nr)
       REFERENCES towar(nr)
       ON UPDATE CASCADE ON DELETE CASCADE
);
```

- tabela z dwoma kluczami obcymi realizuje związek wieloznaczny (dwuargumentowy)

Dane, przykład

```
insert into zamowienie(klient_nr, data_zlozenia,
  data_wysylki, koszt_wysylki)
  values(3,'13-03-2013','17-03-2013', 2.99);
```

- brakuje atrybutu nr, jest automatycznie numerowany

```
insert into towar(opis, koszt)
  values(E'ramka do fotografii 3\'x4\'', 13.36);
```

- brakuje też atrybutu cena, jest wstawiany NULL

```
insert into pozycja values(1, 4, 1);
insert into pozycja values(1, 7, 5);
```

- atrybuty nie są wymienione, muszą być wstawione wszystkie i w kolejności wymienionej podczas tworzenia tabeli

Dostęp do danych, przykład

```
SELECT imie, nazwisko FROM klient;
```

- realizuje rzut relacji , tzn. wybór niektórych atrybutów

```
SELECT * FROM klient WHERE miasto='Sopot';
```

- realizuje obcięcie relacji , tzn. wybór niektórych krotek

```
SELECT imie, nazwisko, zamowienie.nr AS zamowienie_nr
   FROM klient, zamowienie
   WHERE klient.nr=zamowienie.klient_nr;
```

- realizuje złączenie relacji (również rzut)
  - najczęściej wspólny atrybut jest kluczem obcym jednej z relacji, wówczas jest kluczem kandydującym drugiej

```
SELECT imie, nazwisko, zamowienie.nr AS zamowienie_nr
   FROM klient INNER JOIN zamowienie
   ON klient.nr=zamowienie.klient_nr;
```

- inna/lepsza składnia na złączenie

Dostęp do danych, przykład, c.d.

```
-- wymień pary zamówień od tego samego klienta
SELECT Z1.nr AS zam1, Z2.nr AS zam2, Z2.klient_nr
FROM zamowienie AS Z1, zamowienie Z2
WHERE Z1.klient_nr=Z2.klient_nr
      AND z1.nr<z2.nr;
```

- samozłączenie – ta sama tabela, ale odczytywane są dwa różne wiersze, konieczne jest lokalne przenazwowanie (alias)

```
SELECT count(*) FROM zamowienie WHERE data_zlozenia
  BETWEEN '2015/03/01' AND '2015/03/31';
```

- funkcja agregująca (jedna liczba w wyniku)

```
SELECT nazwisko FROM klient GROUP BY nazwisko HAVING
  count (nazwisko) > 1;
```

- grupowanie podobnych wierszy w wydruku, użycie warunku dotyczącego grup – nie pojedynczych wierszy

Dostęp do danych, przykład, c.d.

```
SELECT imie, nazwisko, miasto FROM klient
   WHERE nazwisko IN (
      SELECT nazwisko FROM klient
      GROUP BY nazwisko HAVING count (nazwisko) > 1
      );
```

- zagnieżdżony SELECT, wynik jest zbiorem użytym w warunku WHERE
  - zagnieżdżenie jest nieskorelowane, tzn. najpierw można obliczyć wewnętrzny SELECT, a potem użyć wynik do obliczenia zewnętrznego SELECT-u

Dostęp do danych, przykład, c.d.

```
-- sprawdź jacy klienci nie zamówili niczego
SELECT imie, nazwisko FROM klient
   WHERE NOT EXISTS (
   SELECT *
     FROM zamowienie INNER JOIN pozycja
     ON klient.nr=zamowienie.klient_nr
     AND pozycja.zamowienie_nr=zamowienie.nr );
```

- wewnętrzny SELECT sprawdza niepustość, wynik pomijamy
  - zagnieżdżenie jest skorelowane, tzn. wewnętrzny SELECT odwołuje sie do wiersza z tabeli zewnętrznego SELECT-u – można sobie wyobrazić przeglądanie tabeli klient i obliczanie wewnętrznego SELECT-u dla każdego wiersza od nowa

Wartość nieokreślona NULL

```
INSERT INTO towar(opis, koszt, cena)
  VALUES('donica średnia', 17.12, 19.99);
INSERT INTO towar(opis, koszt, cena)
  VALUES('donica duża', 26.43, NULL);
```

- wartości mogą być jawnie definiowane jako nieokreślone
  - mogą być wynikiem opuszczenie atrybutów w INSERT

```
SELECT * FROM towar WHERE opis LIKE '%donica%' AND cena
  IS NOT NULL;
SELECT *,cena-koszt AS zysk FROM towar
  WHERE cena IS NOT NULL ORDER BY zysk DESC;
```

- nieokreśloność atrybutu można sprawdzać

Współbieżność/Transakcje

- Problemy współbieżności: utracona modyfikacja, niespójna analiza, niezatwierdzona wartość
- Cztery własności każdej transakcji: A – atomowa (atomic), C – spójna (consistency), I – odizolowana (isolated), D – trwała (durable)
- BEGIN/COMMIT (ROLLBACK): początek transakcji oraz zatwierdzenie lub wycofanie
  - narzędzia wycofania: dzienniki
- Narzędzie: blokady
  - współdzielona (odczyt) i wyłączna (zapis)
  - problem z zakleszczeniem, wiele rozwiązań
    - timeout
    - analiza grafu oczekiwań
    - zakładanie blokad w stałej kolejności
    - znaczniki czasu dla transakcji

Współbieżność/Transakcje, c.d.

- Przebiegi (wykonania)
  - możliwe przeploty operacji w transakcjach
  - ustalona kolejność operacji skonfliktowanych
  - domniemana kolejność operacji w każdej transakcji
- Poziomy izolacji
  - problemy: odczyt na brudno, niepowtarzalny, widmo,
  - w Postgresie: read commited, serializable
  - teoretycznie jeszcze: read uncommited, repeatable read
- Blokowanie dwufazowe: gwarancja szeregowalności

Procedury

- Języki: PL/pgSQL (w Postgresie) i inne
- Wywoływanie:
  - przez użytkownika
  - automatycznie podczas działania bazy (wyzwalacze)
  - przy starcie/zakończeniu połączenia a bazą
- Procedury wyzwalane, po co?
  - poprawność danych (pojedynczych, zależnych od innych)
  - śledzenie zmian, audyt, raport, zapis zmian
  - dane bieżące vs. archiwalne
  - naruszenie postaci normalnej, kopie danych, dane wynikowe
  - spowodowane ergonomią, wydajnością, specjalny format danych dla innych aplikacji

Integracja ze środowiskiem programistycznym

- Zależna od konkretnej implementacji SZBD
- Etapy: połączenie z bazą, wysłanie zapytania, odczyt wyników, rozłączenie z bazą
- Rozwiązania:
  - zanurzenie SQL w inny język
  - inny język z funkcjami bibliotecznymi
  - w języku C (biblioteka libpq), C++ (libpq++), ...
- Dostęp sieciowy via przeglądarka internetowa
  - architektura wielowarstwowa: serwer bazodanowy+Web
  - protokoły ODBC, JDBC
  - język PHP lub inny (python, ...)

Atak *SQL injection*

- Przykład PHP

```
pg_exec("SELECT * FROM lekarz WHERE peselL='$pesel';");
```

- Złośliwy użytkownik zapytany o numer pesel być może wstawi wartość '; DELETE FROM lekarz; SELECT ' i spowoduje wykonanie zapytania SQL

```
SELECT * FROM lekarz WHERE peselL=''; DELETE FROM lekarz;
  SELECT '';
```

- Możliwe szkody:
  - nieautoryzowane zmiany w zawartości bazy danych
  - dostęp do danych poufnych/masowych
  - atak DOS (denial of service, odmowa usługi) czyli przeciążenie serwera bazodanowego
  - zbadanie dokładnej struktury bazy danych
  - wykonanie poleceń systemowych

Obrona przed *SQL injection*

- Zabezpieczenie na poziomie aplikacji:
  - znaki specjalne, np. apostrofy, powinny być traktowane jako znaki specjalne, należy pilnować poprawności typów, sprawdzać postać parametrów, nie dopuszczać by zawierały pewne znaki, długość parametrów powinna być niewielka

```
SELECT * FROM lekarz WHERE peselL='\'; DELETE FROM lekarz;
SELECT \'';
```

    - nie będzie groźne i nie zwróci żadnych wyników
- Zabezpieczenie na poziomie serwera bazodanowego
  - minimalne uprawnienia użytkownika
  - mechanizm zapytań może dopuszczać parametry – należy wówczas z tego korzystać
- Zabezpieczenia na poziomie serwera aplikacji
  - analiza uprawnień użytkownika końcowego