Bazy Danych

Andrzej M. Borzyszkowski
Instytut Informatyki

Uniwersytetu Gdańskiego

materiały dostępne elektronicznie
http://inf.ug.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
  - programy wykorzystujące dane

Cechy systemów baz danych

- Zapewnienie:
  - spójności
  - współbieżności
  - poufności (zarządzanie dostępem)
  - niezawodności
  - wydajności
- Abstrakcja danych (niezależność od programów użytkowych)
- Języki zapytań (*query*), dziś jeden standard – SQL

Projekt bazy danych

- Model relacyjny – relacja ≈ tabela
- Diagram encji i związków
- Pojęcie klucza, główny, obcy
- Normalizacja
- SQL, definicja tabel
- Algebra relacji
- SQL, dostęp do danych

Diagram encji i związków

- Projekt bazy danych: encje, związki, charakter tych związków
- Encje posiadają swoje atrybuty, związki pomiędzy encjami też czasami można wyposażyć w atrybut
- W diagramach encji i związków warto 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

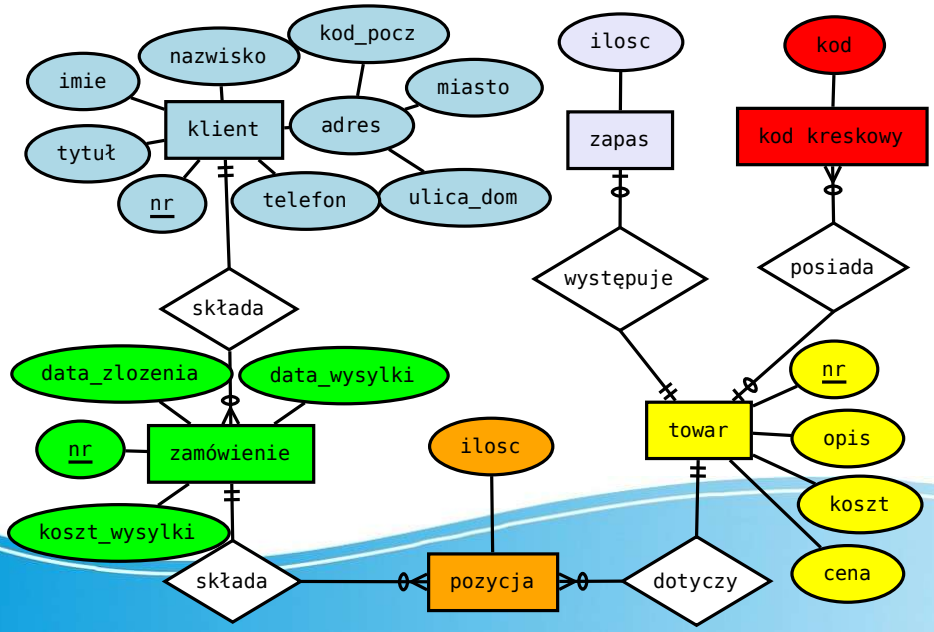
`Towar <występuje w> Zapasie`

- 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`

Diagram ER (notacja Martina)



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 – (integracja danych)
  - iloczyn kartezjański (w praktyce raczej nieużyteczny)
  - operacje teoriomnogościowe: suma, przecięcie, różnica

Klucze

- Klucz kandydujący: jednoznacznie określa encję/krotkę
  - tzn. nie może się powtórzyć (integralność klucza)
  - najczęściej jest jednym atrybutem
  - musi być minimalnym zestawem atrybutów
  - 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

Normalizacja

- 1 postać normalna – już było: komórki tabeli są pojedynczymi wartościami
- Pojęcie funkcyjnej zależności (funkcyjnego determinowania)
- Definicja klucza w nowym języku: klucz determinuje funkcyjnie wszystkie pozostałe atrybuty
- Tw Heatha: jeśli A, B i C są zbiorami atrybutów i relacja spełnia zależność funkcyjną A → B, to można/należy ją rozłożyć na rzuty na {A,B} i {A,C}
  - jeśli atrybut zależy funkcyjnie od podzbioru klucza, to naruszona jest 2 postać normalna, stosujemy rzut
  - jeśli zależy od całego klucza, ale tranzytywnie, to naruszona jest 3 postać normalna, stosujemy rzut
-

Normalizacja, c.d.

- Dobry projekt jest w 3 postaci normalnej
  - tzn każdy atrybut niebędący częścią klucza zależy funkcyjnie tylko i bezpośrednio od klucza
- Jeszcze lepiej, jeśli po prostu każdy (postać Boyce-Codda)
  - nie zawsze da się osiągnąć podziałem tabel
  - np. zapisy studentów na lektoraty, zapisując się do lektorów mogą zapisać się dwa razy na ten sam język
  - można temu przeciwdziałać za pomocą triggerów
- 4 postać normalna: nie projektujemy zależności wielowartościowej
  - np. osobno ma być tabela zapisów na lektoraty, osobno na fakultety
  - zamiast wszystkie kombinacje w jednej tabeli

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
- Związek ,,jest'' w rozszerzonych ERD realizuje się w zależności czy generalizacja jest rozłączna i/lub kompletna

Tabele, przykład

```
create table klient (
    nr            integer     ,
    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            integer     ,
    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             integer     ,             typ atrybutu
    klient_nr      integer       not null,   atrybut
    data_zlozenia  date          not null,   musi być
    data_wysylki   date        ,             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             integer     ,
    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)

Perspektywy (widoki, *view*)

- Perspektywa jest tworzona na podstawie innych tabel/ perspektyw
  - może zawierać tylko wybrane wiersze (obcięcie)
  - lub atrybuty (rzut)
  - może zawierać atrybuty pochodne: obliczane/ zagregowane

```
CREATE VIEW towar_zysk AS
    SELECT *, cena – koszt AS zysk FROM towar
```

- Podział na danych na różne tabele w celu ułatwienia współbieżności: osobne tabele dla towarów i dla ich stanu magazynowego, perspektywa łączy jakby była to jedna tabela
- Wygoda użytkownika, zapamiętanie częstych zapytań
- Zarządzanie dostępem, użytkownik może nie mieć dostępu do całej tabeli

Dane, przykład

```
INSERT INTO zamowienie(klient_nr, data_zlozenia,
  data_wysylki, koszt_wysylki)
  VALUES(3,'13-03-2024','17-03-2024', 2.99);
```

- Brakuje atrybutu nr, jest automatycznie numerowany (o ile została zdefiniowana wartość domyślna)

```
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.

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

```
-- 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;
```

- Funkcja agregująca (jedna liczba w wyniku)

```
SELECT count(*) FROM zamowienie WHERE data_zlozenia
  BETWEEN '2025/03/01' AND '2025/03/31';
```

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

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


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
- Pytanie o nieistnienie musi używać jakiegoś zagnieżdżenia – w tabeli klient nie ma danych o jego zamówieniach

Dostęp do danych, przykład, c.d.

```
-- wypisz liczbę zamówionych towarów przez poszczególnych
  klientów
SELECT imie, nazwisko, count(*) AS towarów FROM (
   SELECT imie, nazwisko
      FROM ( klient INNER JOIN zamowienie
                ON klient.nr = zamowienie.klient_nr
           ) INNER JOIN pozycja
                ON zamowienie.nr = pozycja.zamowienie_nr
   GROUP BY klient.nr, imie, nazwisko, pozycja.towar_nr
   ) AS foo
   GROUP BY imie, nazwisko;
```

- Zagnieżdżony SELECT, tym razem wewnętrznego SELECT wynik definiuje relację użytą jako źródło danych,
  - wymagane jest użycie aliasu, nawet niepotrzebnego (SQL, Postgres nie wymaga aliasu)
  - zewnętrzny SELECT używa tutaj funkcji agregującej na zgrupowanej tabeli

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
- Klucz obcy może dopuszczać NULL, pytanie

```
SELECT * FROM towar WHERE nr NOT IN (SELECT towar_nr from
  kod_kreskowy)
```

jest niesłuszne, porównanie z NULL zwraca zawsze NULL

Zmiany danych

```
UPDATE towar SET waga=12.2 WHERE nr=7;
```

- w istniejących wierszach można zmieniać dane
  - jeśli dotyczą one kluczy kandydujących/adresatów kluczy obcych, to sprawdzana jest integralność

Zmiany schematu

```
ALTER TABLE towar ADD COLUMN waga numeric(5,2);
ALTER TABLE klient DROP CONSTRAINT klient_telefon_un
```

- istniejąca tabela może być zmieniona
  - nowy atrybut ma wartość NULL dla dotychczasowych krotek
  - więzy przestały ograniczać

```
DROP TABLE pozycja;
DROP TABLE zamowienie;
DROP TABLE towar;
```

- tabele (i dane) można też usuwać, kolejność usuwania gra rolę, najpierw tabele z kluczami obcymi
  - albo opcja CASCADE, usuwa też więzy integralności

Programowanie

- Po stronie serwera
  - konieczność rozszerzenia języka zapytań
  - rekursja
  - W Postgresie język PL/pgSQL
  - procedury wyzwalane
- Po stronie klienta
  - integracja ze środowiskami programistycznymi
  - ODBC (open database connectivity)/ JDBC (Java ....)
  - wsparcie dla innych języków
  - dostęp poprzez Web, atak SQL injection

Procedury

- 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
  - brakujące zależności funkcyjne w postaci Boyce'a-Codda
  - 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++), perl, python,
- 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, traktowane jako znaki specjalne

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

  - nie będzie groźne i nie zwróci żadnych wyników
  - poprawność typów,
  - postać parametrów, nie dopuszczać by zawierały pewne znaki, niewielka długość parametrów
- Zabezpieczenia na poziomie serwera aplikacji
  - analiza uprawnień użytkownika końcowego
  - analiza zapytań przesyłanych do bazy danych
- Zabezpieczenie na poziomie serwera bazodanowego
  - minimalne uprawnienia serwera aplikacji jako użytkownika
  - mechanizm zapytań może dopuszczać parametry – należy wówczas z tego korzystać

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

Współbieżność, transakcje, c.d.

- Przebiegi (wykonania)
  - możliwe przeploty operacji w transakcjach
  - ustalona kolejność operacji skonfliktowanych
  - domniemana kolejność operacji w każdej transakcji
  - przebiegi bezkaskadowe, ścisłe, szeregowalne
- Poziomy izolacji
  - problemy: odczyt na brudno, niepowtarzalny, widmo
  - w Postgresie: read commited, serializable
  - Inne poziomy: read uncommited, repeatable read
-

Transakcyjność, narzędzia

- Narzędzie: blokady
  - binarne
  - współdzielone (odczyt) i wyłączne (zapis)
  - problem z zakleszczeniem, wiele rozwiązań
    - timeout
    - analiza grafu oczekiwań
    - zakładanie blokad w stałej kolejności
    - znaczniki czasu dla transakcji i zakaz czekania
  - problem zagłodzenia
- Blokowanie dwufazowe: gwarancja szeregowalności

Bezpieczeństwo w bazach danych

- Użytkownicy/konta
  - klasy użytkowników z różnymi uprawnieniami
  - dostęp do bazy z uwierzytelnieniem (hasło, Kerberos, inne)
- Uprawnienia
  - uznaniowe: możliwość zmiany przez użytkownika
  - stałe: związane na stałe z kontem (stopnie tajności)
- Uprawnienia dla użytkownika
  - prawo tworzenia/usuwania innych użytkowników
  - prawo tworzenia/usuwania baz, tabel i perspektyw
- Uprawnienia dla tabel
  - rodzaje: SELECT, INSERT, DELETE, UPDATE, RULE, TRIGGER
  - perspektywy ograniczają dostęp do wybranych atrybutów

Organizacja pamięci – indeksowanie

- Rodzaje pamięci: pamięć operacyjna, SSD, dysk magnetyczny, taśma magnetyczna, pamięć optyczna
- Dane
  - rekordy o wielu polach
  - BLOB (*binary large object*) – odrębnie przechowywany
  - stała/ zmienna długość rekordów
- Organizacja pamięci
  - plik nieuporządkowany
  - plik uporządkowany (sekwencyjny) – wg klucza głównego
  - plik bezpośredni – adresy wyliczane wg funkcji mieszającej
  - B-drzewo – drzewo o dużym stopniu
- Operacje plikowe (wstawienie, znalezienie, odczytanie, modyfikacja, usunięcie rekordu)
- Indeksy – podają adres bloku dla wartości klucza

PostgreSQL

- Zarządzanie bazą danych
  - użytkownik postgres w systemie Unix, inni użytkownicy w systemie PostgreSQL,
  - polecenia powłoki Uniksa: createuser, createdb, ...
  - obsługa danych (archiwizacja, ustalenie kodowania języka)
- Tabele tymczasowe vs. perspektywy
- Typy danych: daty, różne liczbowe, binarne/BLOB, adresy, sekwencje
- Pola wielokrotne (naruszenie 1 postaci normalnej)
- Dziedziczenie tabel