Bazy Danych

Tomasz Przechlewski

luty 2020

Systemy baz danych

Systemy baz danych: dziedzina informatyki zajmująca się gromadzeniem, przechowywaniem, wyszukiwaniem i przetwarzaniem danych.

Baza danych: zbiór danych istniejący przez długi czas, często przez wiele lat. Zwykle oznacza jednak zbiór danych zarządzany przez *system zarządzania bazą danych = DBMS*.

Zasób pamiętanych danych operacyjnych wykorzystywanych przez systemy użytkowe pewnego przedsiębiorstwa (Date).

Zasoby danych w pamięci o wielkiej pojemności zapewniające dotarcie do poszczególnych elementów wg różnych kryteriów (Leksykon)

Zasób wzajemnie powiązanych danych pamiętanych bez redundancji służących jednemu lub wielu zastosowaniom w sposób optymalny dane są pamiętane w taki sposób,że są niezależne od programów przy dołączaniu i modyfikacji stosuje się wspólną metodę umożliwiającą sprawdzanie poprawności wykonywanych operacji (Martin).

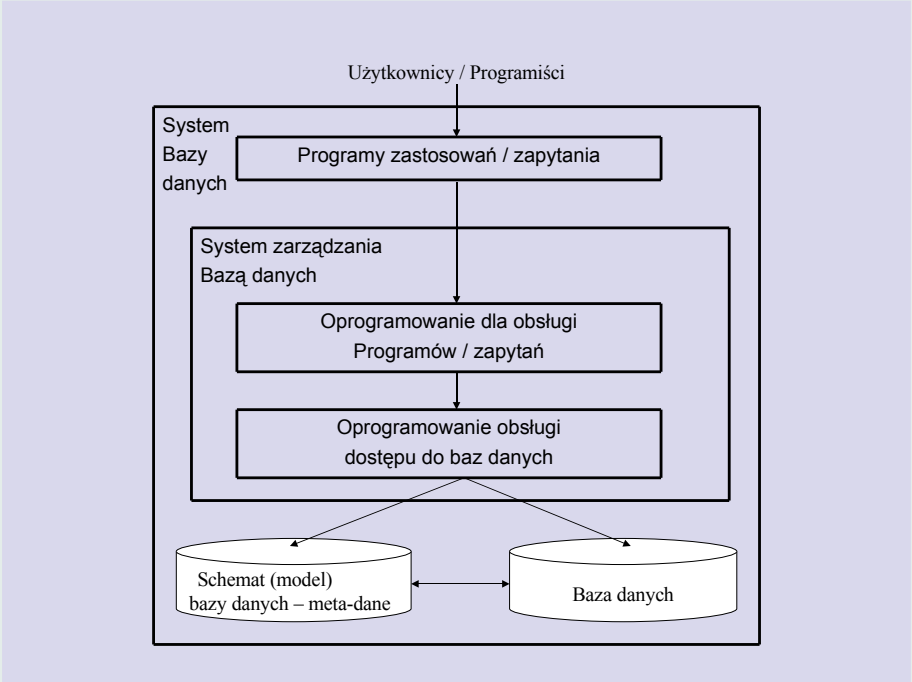
Zasób danych którego zadaniem jest reprezentowanie pewnej dziedziny przedmiotowej. — Zasób danych zapamiętanych na pamięciach zewnętrznych — Model pewnego wycinka rzeczywistości (Beynon-Davies)

System baz danych: Baza danych + System zarządzania bazą danych

Baza danych składa się ze schematu bazy danych oraz ze zbioru danych.

System zarządzania bazą danych umożliwia:

- zakładanie i usuwanie bazy,
- określenie i modyfikację *schematu bazy*,



Źródło: Beynon-Davis

- wprowadzanie danych, wyszukiwanie danych, aktualizację (modyfikowanie, usuwanie) danych. SZBD używa do w/w celów

języka baz danych: języka opisu danych (DDL) oraz języka operowania danymi (DML) lub językiem zapytań.

- przechowywanie ogromnej ilości danych (co najmniej gigabajty; wyklucza przetwarzanie w pamięci operacyjnej), przez długi czas, chroniąc je przed przypadkowym lub niepowołanym dostępem oraz umożliwia efektywny dostęp do danych z poziomu języka zapytań i operacji na danych
- sterowania jednoczesnym dostępem do danych przez wielu użytkowników, z zapewnieniem bezkolizyjności oraz ochrony danych przed uszkodzeniem.

Współbieżność, jednoczesność dostępu do danych: dane mogę być równocześnie dostępne dla wielu użytkowników.

Cechy baz danych:

Integralność: Dokładne odzwierciedlenie rzeczywistości, której baza danych jest modelem poprzez procedury badania poprawności danych przy wykonywaniu operacji na bazie danych i dokonywanie bieżących zmian

Niezależność danych: — Oddzielenie danych od procesów — Organizacja danych jest niewidoczna dla użytkowników i programów — Zmiany w części bazy danych oraz w programie nie wymagają wzajemnych zmian

— Odporność programów użytkowych na zmiany struktury pamięci i strategii dostępu — Zdolność bazy danych do stałego rozwoju bez naruszania istniejących programów użytkowych

— Logiczna (konceptualna) - globalna struktura danych może być zmieniana bez zmiany programów użytkowych — Fizyczna - rozmieszczenie fizyczne i organizacja danych mogą być zmieniane bez zmiany zarówno globalnej struktury danych jak i programów użytkowych

Minimalna redundancja

Każdy element danych jest zarejestrowany tylko raz w jednym miejscu, natomiast może być wykorzystany w wielu zastosowaniach

Redundancja niekontrolowana - szkodliwa wysokie koszty przechowywania, wielokrotne dokonywanie aktualizacji sprzeczność danych na różnych etapach aktualizacji

Redundancja kontrolowana lub minimalna skrócenie czasu dostępu, stosowanie prostszych metod adresowania duplikaty dla odtworzenia danych w razie awarii

Bezpieczeństwo danych

Dla zachowania integralności bazy danych niezbędne jest jej zabezpieczenie, ograniczenie dostępu; określić należy zbiór upoważnionych użytkowników w odniesieniu do całej lub części bazy danych

Trwałość danych

Dane zachowują aktualność przez dłuższy czas Dane trwałe- konta, studenci, pacjenci, nieruchomości, czytelnicy Dane tymczasowe - wydruk stanu konta, przegląd katalogu

Pierwsze systemy pojawiły się pod koniec lat 60-tych. Początkowo sprowadzały się do zwykłych systemów plików. System plików posiada ww. funkcje bazy, z tym, że niektóre, np. wyszukiwanie danych realizuje w sposób nieefektywny.

Przykład (pliki/katalogi):

. . .

```
Cacaki Henryk 1970 średnie operator 1,500 0
Abacki Jan 1965 wyższe dyrektor 6,000 450
Dadacka Jadwiga 1972 średnie magazynier 1,450 30
...
```

Przykład: system rezerwacji miejsc, system bankowy, system ewidencjonowania działalności przedsiębiorstwa.

Autoryzacja, kontrola dostępu: różne aspekty danych można rozpatrywać niezależnie od programów aplikacyjnych przetwarzających dane. Fizyczna niezależność od danych: niezależność od fizycznych aspektów przechowywania danych.

Spójność bazy danych, integralność: Spójność fizyczna: operacje na danych kończą się sukcesem. Spójność logiczna: baza danych jest spójna fizycznie, a jej zawartość odpowiada schematowi bazy danych i dodatkowym ograniczeniom. Integralność: logiczna spójność i zgodność ze stanem świata rzeczywistego opisywanego przez dane.

Trendy w rozwoju: coraz popularniejsze (dostępne na komputery PC, wydajne systemy bazodanowe na licencji OSS: PostgreSQL, MySQL i inne), coraz więcej danych (terabajty, petabaty, odp. tysiąc lub milion gigabajtów). Pamięć trzeciego poziomu. Obliczenia równoległe.

5 głównych elementów środowiska SZBD: Sprzęt Oprogramowanie Dane Procedury Ludzie

Procedury: Instrukcje i polecenia dla tworzenia i użytkowania bd - logowanie, uruchamianie funkcji i aplikacji, inicjowanie i zamknięcie, tworzenie kopii, obsługa awarii, restrukturyzacja

Ludzie: Administratorzy danych/bazy danych, Projektanci BD, Użytkownicy

Model danych (architektura systemu baz danych) – zbiór ogólnych zasad posługiwania się danymi, obejmujący trzy główne części:

Definicja danych: zbiór reguł określających [logiczną] strukturę danych. [Oddzielenie struktury logicznej od fizycznego sposobu przechowywania danych.]

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

Integralność danych: zbiór reguł określających, które stany bazy danych są poprawne [jakie operacje prowadzące do modyfikacji danych są dozwolone].

Tsichritzis, Lochovsky:

Model danych zawiera własności wycinka świata rzeczywistego (dziedziny przedmiotowej): Własności statyczne, względnie niezmienne w czasie Własności dynamiczne, pozwalaja rejestrować zmienność świata

Operacje na bazie danych: Wprowadzania, Skreślania, Aktualizacji, Wyszukiwania (**CRUD**)

Rodzaje MD: Dwa kryteria: poziom uogólnienia i funkcjonalność;

Poziom uogólnienia: — wysokiego poziomu lub konceptualne — wdrożeniowe — niskiego poziomu lub fizyczne

model konceptualny:

Pozwala na opis dziedziny przedmiotowej w kategoriach danych i związków między nimi w sposób przystępny dla użytkownika bazy danych. Najpowszechniej stosowanymi są tu model związków encji a obecnie obiektowy

Model wdrożeniowy:

Zrozumiałe zarówno dla użytkowników jak i zbliżone do sposobu przechowywania danych w bazie. Należą do nich modele przedrelacyjne (hierarchiczny, sieciowy), relacyjny i obiektowy.

Model fizyczny:

Opisuje sposób przechowywania danych w pamięci komputerów i pamięci masowych i przedstawia formaty rekordów czy ścieżki zapisów. Typowymi pojęciami z tego zakresu są metody adresowania, pliki indeksowo-sekwencyjne, struktury łańcuchowe i pierścieniowe, pliki odwrócone czy wyszukiwanie wielokluczowe

Funkcjonalność:

Proste modele danych – modele reprezentowane przy pomocy struktury rekordów, zgrupowanych w struktury plików

Klasyczne (rekordowe) modele danych – hierarchiczne, sieciowe i relacyjne

Semantyczne modele danych – klasyczne modele zachowują podstawową orientacje opartą o rekordy, nie można odczytać znaczenia danych, ich semantyki, zawartej w modelach semantycznych.

3 główne typy (generacje) modeli danych:

Proste modele danych [pliki podzielone na rekordy i pola], **Klasyczne modele danych** [hierarchiczne, sieciowe i relacyjne], **Semantyczne modele danych** [np. obiektowe modele danych].

Relacyjny model danych: Opracowany przez E. F. Codda w latach 70–80; Od mniej więcej połowy lat 80 podstawowa architektura większości popularnych DBMS. W implementacjach teoria RMD bywa traktowana dość luźno.

Definicja danych

Oparta na jednej podstawowej strukturze danych – relacji [Pojęcie relacji można uważać za abstrakcję intuicyjnego pojęcia tabeli, zbudowanej z wierszy i kolumn, w której na przecięciu każdej kolumny z każdym wierzem występuje określona wartość]. Baza danych jest zbiorem relacji, o następujących własnościach:

Id	Nazwisko	Imię	R.ur
1245	Cacacki	Henryk	1970
2145	Abacki	Jan	1965
7191	Dadacka	Jadwiga	1972

Id	Wynagrodzenie	Kasa ZP
7191	1450	30
1245	1500	0
2145	6000	450

Każda relacja w bazie danych jest jednoznacznie określona przez swoją nazwę.

Każda kolumna w relacji ma jednoznaczną nazwę (w ramach tej relacji).

Kolumny relacji tworzą zbiór nieuporządkowany. Kolumny nazywane bywają również atrybutami.

Wszystkie wartości w danej kolumnie muszą być tego samego typu.

Wiersze relacji tworzą nieuporządkowany zbiór; w szczególności, nie ma powtarzających się wierszy.

Każde pole (przecięcie wiersza z kolumną) zawiera wartość atomową z dziedziny określonej przez kolumnę.

Każdy wiersz tabeli odpowiada jednej krotce relacji. Liczbę krotek tej relacji nazywa się jej licznością. Liczbę tę nazywa się również mocą relacji. Moc relacji zmienia się wraz z modyfikacją bazy danych

Liczba atrybutów w relacji to liczba członów tej relacji czyli stopień relacji (np. relacja n-członowa, n-arna)

Dziedzina to zbiór dopuszczalnych wartości dla jednego lub większej liczby atrybutów

Perspektywa – wynik jednej lub wielu operacji tworzących nową relację z relacji bazowych (operacje zwyczajne, zawężanie i złączania)

Każda relacja zawiera klucz główny – kolumnę (lub kolumny), której wartości jednoznacznie identyfikują wiersz (a więc w szczególności nie powtarzają się).

Do wiązania ze sobą danych przechowywanych w różnych tabelach używa się kluczy obcych. Klucz obcy to kolumna lub grupa kolumn tabeli, o wartościach z tej samej dziedziny co klucz główny tabeli z nią powiązanej.

Klucze główny i obce pozwalają reprezentować związki między krotkami

Integralność danych

Integralność encji: każda tabela musi posiadać klucz główny, a wartości klucza głównego muszą być w ramach tabeli unikalne i nie równe NULL. W szczególności, zapobiega to wystąpieniu w tabeli powtórzeń wierszy.

Integralność referencyjna: każda wartość klucza obcego może być albo równa jakiejś wartości klucza głównego występującej w tabeli powiązanej, lub (ewentualnie) NULL. Pociąga to za sobą konieczność określenia reguły postępowania w wypadku usuwania wiersza z tabeli powiązanej, co

mogłoby unieważnić niektóre wartości kluczy obcych w tabelach do niej się odnoszących. W grę wchodzą trzy postacie takiej reguły:

Restricted: usunięcie wiersza jest zabronione, dopóki nie zostaną usunięte lub odpowiednio zmodyfikowane wiersze z innych tabel, których wartości kluczy obcych stałyby się wskutek tej operacji nieważne;

Cascades: usunięcie wiersza powoduje automatyczne usunięcie z innych tabel wszystkich wierszy, dla których wartości kluczy obcych stały się nieważne;

Nullifies: nieważne wartości kluczy obcych ulegają zastąpieniu przez NULL.

W praktyce zazwyczaj jest pożądane stosowanie dalszych warunków integralności (integralność dodatkowa).

Normalizacja

Normalizacja jest procesem identyfikowania logicznych związków między elementami danych i projektowania bazy danych która będzie reprezentować takie związki ale bez anomalii korzystania z relacji

Celem procesu normalizacji jest taki rozdział tego zbioru atrybutów na relacje aby uzyskać pewne pożądane właściwości bazy danych. Niewłaściwe zaprojektowanie schematów relacji prowadzi do: Dublowania się danych, Ich niespójności, Anomalii podczas ich aktualizacji

Podstawą procesu normalizacji jest identyfikacja logicznych związków między elementami danych. Związki te nazywa się związkami zależności lub determinowania

Dwa elementy danych A i B są w związku zależności lub determinowania jeśli pewne wartości elementu danych B zawsze występują z pewnymi wartościami elementu danych A.

W takim związku element danych A jest nazywany elementem determinującym, element danych B elementem zależnym.

Jeżeli element danych A jest determinującym elementem danych, a B zależnym elementem danych to kierunek związku jest od A do B, a nie odwrotnie.

Istnieją dwa główne typy zależności (determinowania): zależność funkcyjna (jednowartościowa) zależność niefunkcyjna (wielowartościowa)

W danej relacji R atrybut B tej relacji jest jest funkcyjnie zależny od innego jej atrybutu A wtedy i tylko wtedy, gdy każdej wartości atrybutu A odpowiada dokładnie jedna wartość atrybutu B

Pierwsza postać normalna (1NF) to relacja w której każde przecięcie wiersza i kolumny zawiera jedną i tylko jedną wartość

Relacja jest w pierwszej postaci normalnej (1NF) wtedy i tylko wtedy, gdy każdy atrybut niekluczowy jest funkcyjnie zależny od klucza głównego.

Aby przejść z 1NF do 2NF usuwa się zależności od części klucza. Oznacza to analizę tych tabel, które mają klucze złożone. Należy określić, czy atrybuty niekluczowe nie są zależne od **części** klucza złożonego.

Relacja jest w drugiej postaci normalnej (2NF) wtedy i tylko wtedy, gdy jest w 1NF i każdy atrybut niekluczowy jest w pełni funkcyjnie zależny od klucza głównego

(Nazwa-Modułu, Nr-Studenta, TypOceny, Nazwisko-Studenta, Ocena)

Nazwisko-Studenta jest funkcyjnie zależne od Nr-Studenta

Aby przejść z 2NF do 3NF usuwa się zależności przechodnie między atrybutami niekluczowymi.

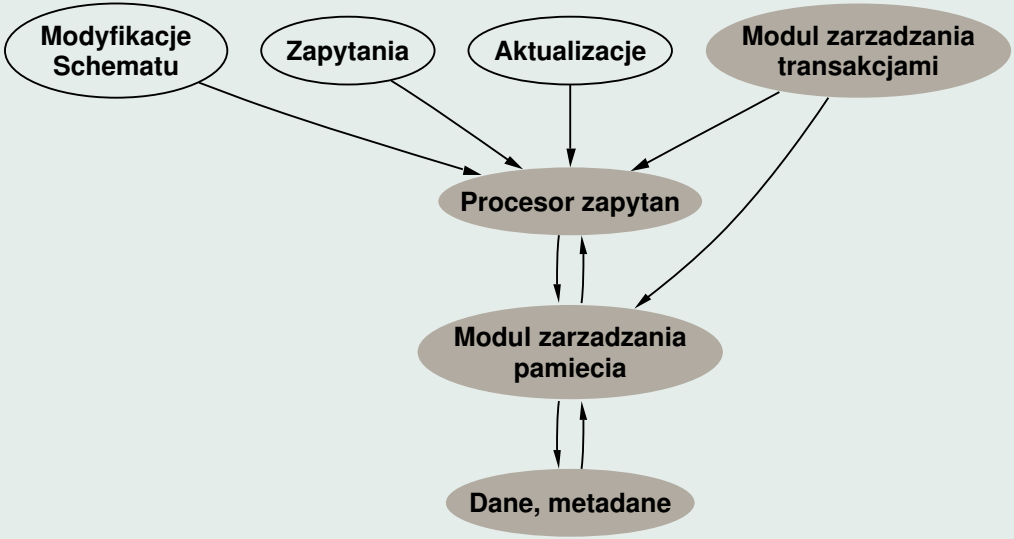
Relacja jest w trzeciej postaci normalnej (3NF) wtedy i tylko wtedy, gdy jest w 2NF i każdy niekluczowy atrybut nie jest **przechodnio zależny** od klucza głównego

(Nazwa-Modułu, Nr-pracownika, Nazwisko-pracownika)

Nazwisko-pracownika jest funkcyjnie zależne od Nr-pracownika

Architektura DBMS

Trzy rodzaje wejść do DBMS: zapytania, aktualizacje i modyfikowanie schematu (tylko administrator DBMS). Zapytania mogą być formułowane przez *interfejs zapytań bezpośrednich* (w języku SQL) lub poprzez *interfejsy programów użytkowych*.



Źródło: Ullman/Widom: Podstawowy wykład... s.26

Procesor zapytań: Obsługuje zapytania, aktualizacje danych i metadanych. Opytmalizuje sposób wykonania zadanych operacji i przekazuje je do wykonania modułowi zarządzania pamięcią.

Moduł zarządzania pamięcią Wybiera właściwe dane z pamięci, dostosowuje do wymogów modułów wyższych poziomów. W większości DBMS bezpośrednio zarządza pamięcią stałą i operacyjną (buforami).

Moduł zarządzania transakcjami: DBMS umożliwia łączenie wielu zapytań bądź modyfikacji w *transakcję*, która jest wykonywana łącznie lub *wcale*. Poprawność transakcji gwarantuje moduł zarządzania transakcjami o nast. właściwościach:

- Niepodzielność: tj. albo wszystko wykonane albo nic
- **Spójność**: baza musi odzwierciedlać stan faktyczny przykład: rezerwacja tego samego miejsca przez dwie agencje nie może się zakończyć przydzieleniem w obu przypadkach.
- **Izolacja**: wykonywanie jednej transakcji nie ma wpływu na wykonywanie innej, przykład: rezerwacja tego samego miejsca przez dwie agencje winna się zakończyć przydziałem jednego i odmową drugiego [a nie np. dwoma odmowami]
- **Trwałość**: stan bazy po transakcji nie może być zmieniony

Blokady: (*Locks*) DBMS blokuje dane, których dotyczy transakcja. Blokada może mieć różny *stopień szczegółowości*.

Logi: Moduł zarządzania transakcjami dokumentuje wszystkie operacje, tzn. rozpoczęcie *każdej* transakcji, zmiany w bazie oraz zakończenie transakcji. Log jest przechowywany w pamięci stałej [awarie] i jest ważnym czynnikiem spójności systemu.

Zatwierdzanie transakcji: kolejność wykonania transakcji: wyliczenie wartości – zapis w logu – zapis w bazie. W przypadku awarii należy porównać ze stanem bazy wyłącznie "najmłodsze" wpisy w logu.

Więzy i wyzwalacze: (*constraints*, *triggers*) Więzy to dodatkowe ograniczenia nałożone na dane. Wyzwalacze to procedury wykonywane po zajściu określonego zdarzenia (demony w syst. operacyjnych).

Trendy w rozwoju cd: **Dane multimedialne i przestrzenne**, **Architektura klient-serwer**, **Integracja danych i WWW**.

Operacje na danych

W teoretycznym opisie modelu relacyjnego operacje na danych definiuje się w terminach tzw. algebry relacyjnej. Operatory algebry relacyjnej mają

za argumenty jedną lub więcej relacji, a wynikiem ich działania zawsze jest też relacja.

Selekcja. Selekcja jest operacją jednoargumentową, określoną przez warunek dotyczący wartości kolumn danej relacji. Wynikiem jej jest relacja zawierające te wszystkie encje (wiersze) wyjściowej relacji, których atrybuty spełniają dany warunek.

Rzut. Rzut to operacja jednoargumentowa określona przez podzbiór zbioru kolumn danej relacji, dająca w wyniku tabelę składającą się z tychże kolumn wyjściowej relacji.

Iloczyn kartezjański. Argumentami są dwie relacje, wynikiem – relacja, ktorej wiersze są zbudowane ze wszystkich par wierszy relacji wyjściowych. Operacja o znaczeniu raczej teoretycznym.

Równozłączenie. Argumentami są dwie relacje, posiadające kolumny o tych samych dziedzinach np. klucz główny jednej z nich i klucz obcy drugiej. Wynikiem jest tabela otrzymana z iloczynu kartezjańskiego relacji wyjściowych poprzez selekcję za pomocą warunku równości tych „wspólnych” atrybutów.

Złączenie naturalne. Powstaje z równozłączenia dwóch tabel poprzez rzutowanie usuwające powtarzające się kolumny złączenia.

Złączenia zewnętrzne. Złączenia zewnętrzne tworzone są podobnie jak złączenie naturalne, lecz z pozostawieniem w tabeli wynikowej także wierszy, dla których nie zachodzi równość atrybutów złączenia: w przypadku złączenia lewostronnego złączenie naturalne uzupełnia się o wiersze z pierwszego argumentu nie posiadające odpowiednika (wierszu o równym atrybucie złączenia) w drugim argumencie; „brakujące" atrybuty przyjmują wartość NULL. W złączeniu prawostronnym robi się to samo, ale względem drugiego argumentu. Wreszcie złączenie obustronne obejmuje obydwie tabele wyjściowe tą samą operacją.

Suma. Suma jest operatorem działającym na dwóch zgodnych relacjach (to jest o tych samych kolumnach), produkującym relację której wiersze są sumą teoriomnogościową wierszy z relacji wyjściowych.

Przecięcie. Przecięcie znowu wymaga dwóch zgodnych tabel, wynikiem jest tabela zawierająca wiersze wspólne dla obu argumentów.

Różnica. Różnica jest określona dla dwóch zgodnych relacji i odpowiada dokładnie różnicy teoriomnogościowej zbiorów wierszy tabel wyjściowych.

SQL

Ogólne zasady składniowe:

Język SQL **nie rozróżnia małych i wielkich liter w słowach kluczowych** i nazwach (baz danych, tabel, indeksów i kolumn); często dotyczy to też wartości napisowych. Poprawne są nazwy zbudowane ze znaków alfanumerycznych, nie zaczynające się od cyfry.

Serwer MySQL może równocześnie zarządzać wieloma bazami danych, każdą z nich identyfikuje nazwa. W trakcie trwania połączenia z serwerem MySQL baza bieżąca, to taka do której domyślnie odnoszą się polecenia adresujące tabele. Odniesienia do kolumn mogą być postaci: `kolumna`, `tabela.kolumna`, `bazadanych.tabela.kolumna`, `tabela@bazadanych.kolumna`.

Każde polecenie SQL kończy się średnikiem;

Wartości napisowe podaje się tak: `"napis"`, lub tak: `'napis'`. Znaki % i _ są metaznakami, służącymi do tworzenia wzorców do porównań; oznaczają odpowiednio dowolny ciąg znaków i dowolny jeden znak. Aby zostały przekazane dosłownie, należy je poprzedzać metaznakiem \.

Definicja danych

Do utworzenia tabeli służy instrukcja `CREATE TABLE`, wymagająca podania nazwy tworzonej tabeli, nazwy każdej kolumny w tej tabeli, typu danych kolumn oraz maksymalnej długości danych w kolumnie.

Składnia polecenia (w uproszczeniu) jest następująca:

```
CREATE TABLE nazwa.tabeli
(nazwa.kolumny typ.danych[(długość) opcje],
   ... nazwa.kolumny typ.danych[(długość) opcje]) [opcje.tabeli])
```

Opcje, które mogą wystąpić po określeniu typu i długości danych to, np. `NULL`, `NOT NULL`, `PRIMARY KEY`, `UNIQUE`, `DEFAULT wartość.domyślna`, i inne.

Typy danych

Standard ISO SQL (1992) przewiduje około piętnastu typów danych, podzielonych na grupy:

- Typy napisowe (*String*): np. `CHAR(N)`, `VARCHAR(N)`. `CHAR(N)` definiuje pole napisowe o stałej długości (ew. uzupełniane spacjami),

podczas gdy `VARCHAR(N)` jest polem o zmiennej długości nie przekraczającej `N`.

- Typy liczbowe (**Numeric**): np. `INT`, `BIGINT`, `FLOAT`, `DECIMAL`. Na ogół dostępnych jest wiele różnych typów liczbowych, różniących się możliwym zakresem wartości (`INT`, `BIGINT`, `SMALLINT`, ...) i precyzją (`FLOAT`, `DOUBLE PRECISION`, ...). Typ `DECIMAL(M,D)` to liczba (ułamek) dziesiętny o ustalonej liczbie cyfr dziesiętnych w części całkowitej i ułamkowej.
- Typy daty i godziny (**Datetime**): np. `DATE`, `TIME`, `TIMESTAMP`.

Bogactwo dostępnych typów danych i możliwość określania długości należy wykorzystywać do optymalizowania definicji tabeli pod kątem zużycia miejsca i do kontroli integralności wprowadzanych (bądź wynikających z operacji na danych) wartości.

Język SQL c.d.

Definicje kolumn: opcje

Opcje mogące wystąpić w definicji kolumny w instrukcji `CREATE TABLE` dzielą się na opcje ogólne, które mogą być stosowane do (prawie) wszystkich typów kolumn, i opcje szczególne, stosujące się do niektórych (klas) typów. Niektóre z opcji wykluczają się wzajemnie, np. `PRIMARY KEY` i `NULL`.

Opcje dotyczące wszystkich typów

- `PRIMARY KEY`: określa daną kolumnę jako klucz główny tabeli. Tabela może posiadać tylko (co najwyżej) jeden klucz główny, o wartościach nie powtarzających się i różnych od `NULL`.
- `DEFAULT wartość_domyślna`: określa wartość domyślną kolumny dla nowo wprowadzanych wierszy w przypadku, gdy instrukcja tworząca nowy wiersz nie zadaje tej wartości.
- `NOT NULL` lub `NULL`: określa, czy `NULL` jest dopuszczalną wartością w tej kolumnie. Domyślnie wartość `NULL` jest dopuszczalna, za wyjątkiem kluczy (kolumn indeksowanych).
- `AUTO_INCREMENT`: jeżeli przy tworzeniu wiersza nie zada się jawnie wartości dla tej kolumny to wartością zapisaną będzie największa z wcześniej występujących w tej kolumnie powiększona o jeden.

Definicja tabeli: deklaracje dodatkowe

W ciągu definicji kolumn w instrukcji `CREATE TABLE` mogą być ponadto umieszczone dodatkowe deklaracje, służące głównie do deklarowania indeksów, w tym kluczy złożonych (indeksów obejmujących więcej niż jedną kolumnę). Są one postaci:

- `PRIMARY KEY (nazwa.kolumny.indeksowej, ...)`: określenie klucza głównego.
- `KEY [nazwa.indeksu] (nazwa.kolumny.indeksowej, ..)`: deklaruje indeksowanie ze względu na wartości z odpowiednich kolumn.
- `INDEX [nazwa.indeksu] (nazwa.kolumny.indeksowej, ...)`: synonimem dla `KEY`.
- `UNIQUE [nazwa.indeksu] (nazwa.kolumny.indeksowej, ...)`: deklaruje indeks o nie powtarzających się wartościach.

Usuwanie tabel

Tabele można usunąć za pomocą instrukcji
`DROP TABLE tabela1, ...`

Modyfikacja struktury tabel

Strukturę tabel już istniejących (i wypełnionych danymi) można modyfikować, w sensie dodawania lub usuwania kolumn i indeksów, zmiany definicji kolumn, czy wreszcie zmiany nazwy tabeli. Operacja ta jest w zasadzie bezpieczna, nawet jeżeli tabela jest w danej chwili w użyciu. Oczywiście kłopoty mogą powstać, jeżeli dane już zapisane w tabeli nie mieszczą się w nowych definicjach typów kolumn.

```
ALTER [IGNORE] TABLE nazwa.tabeli operacja1[, operacja2, ...]
```

Wprowadzanie danych: LOAD DATA

```
LOAD DATA INFILE 'plik' [ REPLACE | IGNORE ] INTO TABLE tabela
[ FIELDS [ TERMINATED BY '\t', [ OPTIONALLY ] ENCLOSED BY '' ESCAPED BY
[ LINES TERMINATED BY '\n' ] [(pole1, pole2, ...)]
```

Instrukcja INSERT

W najprostszej postaci instrukcja `INSERT` służy wprowadzeniu do tabeli pojedynczego wiersza danych:

```
INSERT INTO tabela [ (kolumna1, ...) ] VALUES (wyrażenie1, ...)
```

Jeżeli nazwy kolumn, do których wstawiamy wartości podanych wyrażeń nie zostaną podane jawnie, to wartości te zostaną wpisane do kolejnych kolumn w takim porządku, w jakim kolumny te były zdefiniowane (instrukcją `CREATE TABLE`. Pola danych w tych kolumnach, dla których nie podano wartości otrzymają wartości domyślne (zdefiniowane jawnie w instrukcji `CREATE TABLE` lub automatyczne, np. napis pusty dla pól napisowych). Szczególne zachowanie dotyczy kolumn zadeklarowanych jako `AUTO_INCREMENT` (co wspomniano wcześniej), oraz typu `TIMESTAMP` – w tym ostatnim przypadku pole kolumny otrzyma wartość odpowiadającą czasowi operacji, o ile nie podamy jawnie innej wartości (innej niż NULL).

Usuwanie wierszy: instrukcja DELETE

Do usuwania wiersza bądź wierszy z tabeli służy instrukcja `DELETE`. Specyfikacji wierszy, które mają być usunięte służy klauzula `WHERE`:
`DELETE FROM tabela WHERE warunki`

Ogólną postać wyrażeń, których można użyć w charakterze warunków omówimy nieco dalej. W najprostszym przypadku można po prostu użyć warunku żądającego, by wartość w określonej kolumnie była równa podanej stałej, np.

```
DELETE FROM spis_studentow WHERE nr_albumu = 666
```

Należy stosować apostrofów w zapisie stałych napisowych (MySQL dopuszcza również cudzysłowy). Jeżeli chcemy mieć pewność, że usunięty zostanie tylko jeden określony wiersz, to warunek powinien dotyczyć wartości klucza głównego (lub jakiegoś klucza zadeklarowanego jako UNIQUE).

Budowa wyrażeń w SQL

Wyrażenia te podlegają ewaluacji przez instrukcję `SELECT`, oraz służą do formułowania warunków podawanych w klauzuli `WHERE` instrukcji takich, jak `DELETE` czy `UPDATE`. Aby więc sprawdzić, jaki jest wynik ewaluacji danego wyrażenia, można wpisać w linii komend `mysql` instrukcję

```
SELECT wyrażenie
```

Wyrażenia logiczne

Każda wartość różna od zera i NULL odpowiada w wyrażeniu logicznym prawdzie; a więc NULL i zero reprezentują wartość „fałsz”. Złożone wyrażenia logiczne, zbudowane za pomocą operatorów logicznych, zwracają jedynkę jako reprezentację wartości prawdziwej, a zero jako reprezentację fałszu.

Operatory logiczne

NOT: logiczna negacja.

OR: logiczna alternatywa.

AND: logiczna koniunkcja.

Porównania

Jeżeli wartością któregokolwiek z argumentów porównania jest `NULL`, porównanie zwraca jako wynik `NULL` (oczywiście za wyjątkiem funkcji `ISNULL`.)

Równość oznacza się pojedynczym znakiem równości (`=`).

Nierówność można oznaczyć bądź tak: `<>`, bądź tak: `!=`.

„Mniejszy lub równy" oznacza się `<=`, „większy lub równy" oznacza się `>=`.

„Większy niż" i „mniejszy niż" oznacza się odpowiednio `>` oraz `<`.

`ISNULL(A)` zwraca jedynkę, jeżeli wartością A jest `NULL`, w przeciwnym wypadku zwraca zero.

`A BETWEEN B AND C`: wyrażenie takie jest równoważne `A>=B AND A<=C` o ile porównywane wyrażenia są tego samego typu. W przeciwnym wypadku sposób porównania wyznacza typ pierwszego argumentu (A).

Porównania napisów

Porównania napisów są nieco bardziej skomplikowane. Standard SQL definiuje pewną mocno uproszczoną wersję wyrażeń regularnych, z wykorzystaniem metaznaków % oraz _ do porównywania wyrażeń za pomocą operatora LIKE (p. poniżej); lecz ponadto MySQL umożliwia porównania z wykorzystaniem pełnej składni wyrażeń regularnych (operator REGEXP).

`wyrażenie IN (wartość1, ...)`: zwraca jedynkę jeżeli wyrażenie jest równe którejkolwiek wartości z listy podanej w nawiasach, w przeciwnym wypadku zwraca zero. Wartość wyrażenia może być numeryczna, a jej typ narzuca sposób porównania.

`wyrażenie NOT IN (wartość1, ...)`: równoważne NOT (wyrażenie IN (wartość1, ...)).

`wyrażenie1 LIKE wyrażenie2`: porównanie wzorców, w których budowie można korzystać z metaznaków % (oznaczającego dowolną liczbę – w tym zero – dowolnych znaków), oraz _ (oznaczającego dowolny pojedynczy znak). Aby we wzorcu umieścić któryś z metaznaków jako znak dosłowny, należy go poprzedzić znakiem \.

`wyrażenie1 NOT LIKE wyrażenie2`: równoważne NOT (wyrażenie1 LIKE wyrażenie2).

`wyrażenie1 REGEXP wyrażenie2`: porównanie wzorców zbudowanych zgodnie ze składnią wyrażeń regularnych.

`wyrażenie1 NOT REGEXP wyrażenie2`: równoważne NOT (wyrażenie1 REGEXP wyrażenie2.

`STRCMP(napis1, napis2)`: zwraca zero jeżeli napisy są jednakowe, minus jeden jeżeli pierwszy argument jest wcześniejszy według obowiązującego porządku sortowania, a w przeciwnym wypadku – plus jeden.

Wyrażenia arytmetyczne

Proste operatory arytmetyczne to dodawanie (+), odejmowanie (-), mnożenie (*) i dzielenie (/). Wynikiem dzielenia przez zero jest NULL).

Ponadto w MySQL dostępny jest dość bogaty zestaw funkcji matematycznych (wykładnicze, logarytmiczne, trygonometryczne itp.). W wypadku błędu (nielegalnego argumentu) zwracana jest wartość NULL. Funkcje operujące na napisach

Repertuar funkcji działających na napisach w MySQL jest również dość bogaty. Wymienimy tu tylko niektóre z nich – p. kompletny opis.

* CONCAT(X,Y, ...): zwraca złączenie napisów podanych jako argumenty.

* LENGTH(S): zwraca długość napisu (w znakach).

* LOCATE(A, B): jeżeli napis A stanowi część napisu B, zwraca pozycję początku pierwszego wystąpienia A w B. W przeciwnym wypadku zwraca zero. Synonimem jest POSITION(B IN A).

* LOCATE(A,B,C): jeżeli napis A jest częścią napisu B występującą w pozycji dalszej niż (liczba całkowita) C, zwraca pozycję tego wystąpienia.

* LEFT(napis, długość): zwraca napis składający się z długość początkowych znaków napisu.

* RIGHT(napis, długość): podobnie jak LEFT, lecz zwraca końcówkę napisu. Synonimem jest SUBSTRING(napis FROM długość).

* SUBSTRING(A,B,C): zwraca napis składający się z C znaków napisu A, począwszy od znaku w pozycji B. Akceptowana jest również składnia SUBSTRING(A FROM B FOR C).

* TRIM([[BOTH — LEADING —TRAILING] [A] FROM] B): zwraca napis uzyskany z napisu B poprzez usunięcie wystąpień pod-napisu A z początku (LEADING), końca (TRAILING) lub obu końców (BOTH). Domyślnie stosuje opcję BOTH, i jeżeli A nie podano usuwa spacje.

* REPLACE(A,B,C): zwraca napis utworzony z napisu A poprzez zastąpienie wszystkich wystąpień pod-napisu B napisem C.

* REVERSE(napis): zwraca napis z odwrotną kolejnością znaków.

Funkcje dotyczące daty i godziny

Funkcje operujące na wartościach oznaczających daty i godziny są zbyt liczne, aby tu je wszystkie wymienić. Ograniczymy się więc na razie do kilku najbardziej użytecznych; jak zwykle, pełny opis dostępny jest w dokumentacji MySQL.

* CURDATE() lub `CURRENT_DATE`: zwraca bieżącą datę w postaci YYYY-MM-DD (rok-miesiąc-dzień), lub YYYYMMDD, w pierwszym wypadku jako napis, w drugim jako liczbę całkowitą (jeżeli kontekst wymaga konwersji do typu całkowitoliczbowego).

* CURTIME() lub `CURRENT_TIME`: zwraca bieżącą godzinę w postaci HH:MM:SS (godzina:minuta:sekunda), lub HHMMSS – zależnie od kontekstu, podobnie jak poprzednia funkcja.

* NOW() lub SYSDATE() lub `CURRENT_TIMESTAMP`: zwraca bieżącą datę i godzinę w postaci napisu YYYY-MM-DD HH:MM:SS lub liczby

całkowitej YYYYMMDDHHMMSS, jeśli kontekst wymaga wartości całkowitoliczbowej.

* `DATE_FORMAT(data, format)`: konwertuje datę na napis którego format można kontrolować za pomocą napisu formatującego.

* `TIME_FORMAT(czas, format)`: funkcja analogiczna do powyższej, lecz akceptująca jedynie specyfikatory formatu dotyczące godziny, minuty i sekundy.

* `TO_DAYS(data)`: zamienia datę na liczbę całkowitą oznaczającą liczbę dni od początku roku 0. Funkcja ta (oraz następna) przydają się do obliczania liczby dni jakie upłynęły między dwiema datami.

* `FROM_DAYS(data)`: zamienia liczbę całkowitą, interpretowaną jako liczba dni od początku roku 0, na datę (napis postaci YYYY-MM-DD.

Wyrażenia warunkowe

* IFNULL(A,B): jeśli wartością A jest NULL zwraca B, w przeciwnym wypadku zwraca A.

* IF(A,B,C): jeśli wartością logiczną A jest prawda (tzn. nie zero i nie NULL) zwraca B, w przeciwnym wypadku zwraca C. Wartość A traktowana jest jako całkowitoliczbowa – aby więc stosować taką konstrukcję dla

warunku zależnego od wartości zmiennoprzecinkowej, należy w miejscu A użyć operacji porównania.

Instrukcja SELECT

Instrukcja SELECT służy głównie do pobierania danych z tabeli (tabel) na podstawie zadanych warunków. Wynikiem jej wykonania jest zawsze tabela. Składnia tej instrukcji jest złożona, dlatego omówimy ją na przykładach:

```
SELECT wyrażenie1, wyrażenie2, ...
```

Tym razem wynikiem będzie tabela o jednym wierszu i o kolumnach zawierających kolejno wartości podanych wyrażeń.

```
SELECT wyrażenie1, wyrażenie2, ... FROM tabela
```

W takiej postaci instrukcji SELECT sygnalizujemy, że dane chcemy pobierać z tabeli wymienionej po słowie kluczowym FROM. Możemy teraz, budując wyrażenia, używać nazw kolumn z tej tabeli. Kolejność kolumn w

tabeli wynikowej będzie taka, jak kolejność podanych wyrażeń. W szczególności, wyrażenia mogą być po prostu nazwami interesujących nas kolumn. Szczególny przypadek to

```
SELECT * FROM tabela
```

co spowoduje wypisanie całej tabeli.

```
SELECT wyrażenie1, wyrażenie2, ... FROM tabela WHERE warunek
```

Warunek podany po słowie kluczowym WHERE ogranicza działanie instrukcji SELECT do wierszy spełniających ten warunek. Powinien on być wyrażeniem logicznym, zbudowanym z wykorzystaniem nazw kolumn tabeli.

Złączenia tabel

Instrukcja SELECT o postaci

```
SELECT wyrażenie1, wyrażenie2, ... FROM tabela1, tabela2, ...
```

zwraca tabelę o kolumnach zawierających wartości podanych wyrażeń, obliczone dla iloczynu kartezjańskiego (pełnego złączenia) podanych

tabel. W budowie wyrażeń można korzystać z nazw kolumn z tych tabel, jeżeli jednak nazwy kolumn powtarzają się w różnych tabelach, to trzeba pamiętać o uwzględnieniu nazwy tabeli w odwołaniu do kolumny – tj. tabela.kolumna.

Oczywiście najczęściej bardziej przydatne są zapytania wykorzystujące równozłączenia tabel, a nie iloczyny kartezjańskie. Dokonuje się tego za pomocą odpowiednio dobranego warunku w klauzuli WHERE:

```
SELECT wyrażenie1, wyrażenie2, ...
  FROM tabela1, tabela2, ... WHERE warunek
```

Notacje tabela1, tabela2 i tabela1 JOIN tabela2 są równoważne.

Warunek w klauzuli WHERE powinien być wyrażeniem logicznym dotyczącym kolumn złączanych tabel, dyktuje on które wiersze iloczynu kartezjańskiego zostaną uwzględnione w złączeniu.

Inny sposób złączenia tabel to złączenie zewnętrzne. MySQL implementuje dwie postacie zapisu lewostronnego złączenia zewnętrznego:

```
tabela1 LEFT [OUTER] JOIN tabela2 ON warunek
```

gdzie warunek po słowie kluczowym ON może być dowolnym wyrażeniem logicznym zbudowanym z wykorzystaniem nazw kolumn złączanych tabel, oraz

```
tabela1 LEFT [OUTER] JOIN tabela2 USING (kolumna1, kolumna2, ...)
```

Lista nazw kolumn po słowie kluczowym USING musi zawierać kolumny występujące w obu złączanych tabelach pod tymi samymi nazwami. Inaczej mówiąc, notacja powyższa jest równoważna zapisowi

```
tabela1 LEFT [OUTER] JOIN tabela2 ON tabela1.kolumna1=tabela2.kolumna1
   AND tabela1.kolumna2=tabela2.kolumna2 ...
```

Słowo kluczowe OUTER jest opcjonalne i nie ma wpływu na efekt złączenia. Przypominam, że definicja (lewostronnego) złączenia zewnętrznego oznacza, że jeżeli w „prawej" tabeli brak wiersza „pasującego" do pewnego wiersza tabeli „lewej" (tzn. spełniającego warunek złączenia), to stworzony zostanie wiersz zawierający wartość NULL we wszystkich kolumnach pochodzących z tabeli „prawej".

Dodatkowo istnieje jeszcze operacja naturalnego złączenia lewostronnego:

```
tabela1 NATURAL LEFT [OUTER] JOIN tabela2
```

będąca po prostu skrótowym zapisem złączenia z klauzulą USING zawierającą jako argument listę nazw wszystkich kolumn powtarzających się w obu złączanych tabelach.

Sortowanie wyników

Do uzyskania tabeli wynikowej instrukcji SELECT w postaci posortowanej ze względu na wartości w którejś z kolumn służy klauzula ORDER BY:

```
SELECT wyrażenie1, wyrażenie2, ... FROM złączenie ...
WHERE warunek ORDER BY kolumna [ ASC | DESC ]
```

Domyślnym porządkiem jest porządek rosnący (opcja ASC) według wartości numerycznych lub porządku sortowania wartości napisowych. Do uzyskania sortowania w porządku malejącym służy opcja DESC. Funkcje agregujące i klauzula GROUP BY

Klauzula GROUP BY w połączeniu z tzw. funkcjami agregującymi służy do uzyskania sum, średnich itp. po wierszach z tabeli (lub złączenia) spełniających warunek, formułowany jako wyrażenie analogicznie do

wyrażeń używanych w klauzuli WHERE, lecz podawany w tym przypadku po słowie kluczowym HAVING:

```
SELECT wyrażenie1, wyrażenie2, ... FROM złączenie [ WHERE warunek ]
GROUP BY kolumna-gr HAVING warunek-grupowania [ ORDER BY kolumna-sort ]
```

Zasygnalizowano tutaj właściwą kolejność wystąpienia klauzul: klauzula WHERE (o ile się pojawia) musi poprzedzać GROUP BY, która z kolei może wystąpić jedynie przed ORDER BY.

W MySQL istnieją następujące funkcje agregujące, które można wykorzystać w budowie wyrażeń w instrukcji SELECT z klauzulą GROUP BY:

COUNT(wyrażenie): zlicza wiersze dla których wyrażenie przyjmuje wartość różną od NULL;

AVG(wyrażenie): oblicza średnią wartość wyrażenia dla uwzględnionych wierszy;

MIN(wyrażenie), MAX(wyrażenie): podają odpowiednio minimalną i maksymalną wartość wyrażenia dla uwzględnionych wierszy;

SUM(wyrażenie): sumuje wyrażenie po uwzględnionych wierszach;

STD(wyrażenie) lub STDDEV(wyrażenie): oblicza standardowe odchylenie wyrażenia;

`BIT_OR(wyrażenie)`, `BIT_AND(wyrażenie)`: alternatywa i odpowiednio koniunkcja bitów wyrażenia.

Aliasy nazw kolumn i wyrażeń

Jako argumenty klauzul ORDER BY i GROUP BY nie mogą być użyte wyrażenia złożone. Można tu użyć jedynie nazw kolumn, lub – jeżeli chcemy grupować lub sortować według wartości wyrażeń złożonych – aliasy tych wyrażeń. Do stworzenia aliasu dla wyrażenia podanego w instrukcji SELECT stosuje się słowo kluczowe AS, w sposób następujący:

```
SELECT wyrażenie1, wyrażenie2 AS alias FROM złączenie ORDER BY alias
```

Alias nadany wyrażeniu staje się nazwą odpowiedniej kolumny tabeli wynikowej (zamiast dosłownej postaci tego wyrażenia). Wyprowadzanie wyników zapytania do pliku

CREATE VIEW

```
CREATE VIEW nazwa-perspektywy [(lista-kolumn)] AS select_statement
```

Base tables and views share the same namespace within a database, so a database cannot contain a base table and a view that have the same name.

Views must have unique column names with no duplicates, just like base tables. By default, the names of the columns retrieved by the SELECT statement are used for the view column names. To define explicit names for the view columns, the optional column-list clause can be given as a list of comma-separated identifiers. The number of names in column-list must be the same as the number of columns retrieved by the SELECT statement.

Columns retrieved by the SELECT statement can be simple references to table columns. They can also be expressions that use functions, constant values, operators, and so forth.

Unqualified table or view names in the SELECT statement are interpreted with respect to the default database. A view can refer to tables or views in other databases by qualifying the table or view name with the proper database name.

A view can be created from many kinds of SELECT statements. It can refer to base tables or other views. It can use joins, UNION, and subqueries. The SELECT need not even refer to any tables. The following example defines a view that selects two columns from another table, as well as an expression calculated from those columns.

Some views are updatable. That is, you can use them in statements such as UPDATE, DELETE, or INSERT to update the contents of the underlying table.

Przykłady

```
CREATE TABLE SUPPLIER (SNO INTEGER,
     SNAME VARCHAR(20), CITY  VARCHAR(20));
CREATE TABLE PART (PNO INTEGER,
     PNAME VARCHAR(20), PRICE DECIMAL(6 ,2));
CREATE TABLE SELLS (SNO INTEGER, PNO INTEGER);

SELECT * FROM PART;        SELECT * FROM SELLS;
```

pno	pname	price
1	szprycha	0.50
2	felga Rigida	40.00
3	felga Vuelta	45.00
4	piasta przód	36.00
5	piasta tył	120.00
6	sztyca	25.00
7	łańcuch	32.00
8	tryb	80.00
9	widelec Chro-Mo	90.00
10	widelec Alu	300.00

sno	pno
1	1
1	2
2	3
2	4
2	5
3	6
4	7
4	8
5	9
5	10

```
SELECT * FROM SUPPLIER
```

sno	sname	city
1	Kowalski s.c	Gdańsk
2	Brząkała s.c	Sopot
3	Cacacki s.c	Gdańsk
4	Dadacki s.c	Gdynia

```
     5 | Malinowski s.c | Gdynia

SELECT PNAME, PRICE FROM PART  SELECT * FROM PART WHERE PRICE > 99;
WHERE PRICE > 99;
                                pno |    pname    | price
    pname    | price       -----+-------------+--------
-------------+--------          5 | piasta tył  | 120.00
 piasta tył  | 120.00          10 | widelec Alu | 300.00
 widelec Alu | 300.00

SELECT PNAME, PRICE FROM PART WHERE PRICE > 99 AND PRICE < 200;

 -- Wypisz nazwy wszystkich dostawców
  i części, które dostarczają: --
SELECT S.SNAME, P.PNAME, S.CITY
 FROM SUPPLIER S, PART P, SELLS SE
 WHERE S.SNO = SE.SNO AND P.PNO = SE.PNO;

 -- wypisz wszystkie części z Gdańska: --
SELECT PNAME AS Część
FROM PART, SUPPLIER, SELLS
```

```
WHERE CITY = 'Gdańsk'
AND SELLS.PNO=PART.PNO AND SUPPLIER.SNO = SELLS.SNO ;

   Część
 -------------
 szprycha
 felga Rigida
 sztyca

-- Wypisz miasta z których pochodzą dostawy: --
SELECT DISTINCT CITY FROM SUPPLIER;

-- Wypisz wszystkich dostawców i liczbę części: --

SELECT S.SNO, S.SNAME, COUNT(SE.PNO)
 FROM SUPPLIER S, SELLS SE
 WHERE S.SNO = SE.SNO
 GROUP BY S.SNO, S.SNAME;

  sno |     sname      | count
 -----+----------------+------
```

```
    1 | Kowalski s.c   |     2
    2 | Brząkała s.c   |     3
    3 | Cacacki s.c    |     1
    4 | Dadacki s.c    |     2
    5 | Malinowski s.c |     2

-- Wypisz dostawców, którzy dostarczają 0 lub 1 część: --
-- Having pozwala na testowanie wartości zagregowanych --
SELECT S.SNO, S.SNAME, COUNT(SE.PNO)
   FROM SUPPLIER S, SELLS SE
   WHERE S.SNO = SE.SNO
   GROUP BY S.SNO, S.SNAME
   HAVING COUNT(SE.PNO) < 2 -- wyrażenie typu prawda/fałasz --
   -- musi być argumentem GROUP
    lub być argumentem funkcji agregującej --

 sno |    sname    | count
-----+-------------+-------
   3 | Cacacki s.c |     1

-- Ta sama tablica w różnych rolach --
```

```
-- Wypisz dostawców z tego samego miasta
   co dostawca o numerze 5: --
SELECT S2.SNAME
FROM SUPPLIER S1, SUPPLIER S2
WHERE S1.SNO = 5 AND S1.CITY = S2.CITY

      sname
 ---------------
  Dadacki s.c
  Malinowski s.c

-- Podzapytania: (zamiast IN może być =) --
SELECT * FROM SUPPLIER WHERE city IN
  ( SELECT city FROM SUPPLIER WHERE sno= '5')

-- Podzapytania. Wyświetl wszystkie części
   droższe niż 'tryb': --
SELECT * FROM PART
WHERE PRICE > (SELECT PRICE FROM PART WHERE PNAME='tryb');
 pno |      pname      | price
-----+-----------------+-------
```

```
   5 | piasta tył      | 120.00
   9 | widelec Chro-Mo |  90.00
  10 | widelec Alu     | 300.00

-- W standardzie języka SQL występują następujące typy predykatów
-- w klauzuli WHERE: porównania; IN (lista); BETWEEN(zakres);
-- LIKE 'wzorzec'; NULL test. Przykłady

SELECT * FROM part WHERE pname IN('tryb','sztyca');
SELECT * FROM PART where price between 40 and 200;
SELECT * FROM PART where pname like '%felga%';
-- podkreślnik = dowolny znak, % = ciąg znaków (także zerowy)
 pno |    pname     | price
-----+--------------+------
   2 | felga Rigida | 40.00
   3 | felga Vuelta | 45.00

DELETE FROM PART
WHERE PNAME NOT like '%felga%';
 pno |    pname     | price
-----+--------------+------
   2 | felga Rigida | 40.00
   3 | felga Vuelta | 45.00
```

```
UPDATE PART SET PRICE=33
WHERE PNAME='felga Vuelta';
```

```
 pno |    pname     | price
-----+--------------+------
   2 | felga Rigida | 40.00
   3 | felga Vuelta | 33.00
```

```
GRANT { SELECT | INSERT | UPDATE | DELETE } ON { TABLE | VIEW }
TO { PUBLIC | GROUP group | username }

GRANT SELECT ON PART TO ruda;
GRANT ALL ON SUPPLIER TO PUBLIC;
```

Hurtownie danych

Typy systemów wspomagania zarządzania w przedsiębiorstwie: transakcyjne, informacyjne (MIS), wspomagające podejmowanie decyzji (DSS), informowania kierownictwa (EIS – Executive Information Systems).

Systemy DSS = działają w oparciu o dane zebrane w przedsiębiorstwie; systemy EIS wykorzystują dane zewnętrzne.

Hurtownia danych [wg W. Immona] zbiór danych wspomagający podejmowanie decyzji: uporządkowany tematycznie (*subject-oriented*), zintegrowany (*integrated*), zawierający wymiar czasowy (*time-variant*), nieulotny (*nonvolatile*, niezmienne).

Przyczyny tworzenia hurtowni: zła jakość danych, niewystarczający stopień zintegrowania. Hurtownie danych = scalanie danych operacyjnych firmy + danych z zewnątrz. Internet, integracja. Przetwarzanie OLTP oraz OLAP.

Wielowymiarowy model danych (*data cube*), zwykle opracowany pod kątem wybranego **zagadnienia**, jak. np. sprzedaż czy koszty. Tablice wymiarów, tablice faktów: Przykład:

```
time       [sales]*       item       branch     location
----       =======        ----       ------     --------
time_key   time_key       item_key   branch_key loc_key
day        item_key...    name       name       address
dow        dollars_sold   type       type       phone?
month      units_sold     price?                representative?

* facts
```

Pozyskiwanie wiedzy z baz danych

Najstarszą dyscypliną naukową zajmującą się metodami uczenia się z doświadczenia i podejmowania decyzji jest *statystyka*.

Idea pozyskiwania wiedzy z baz danych powstała w obszarze technologii baz danych w izolacji od innych obszarów badawczych.

W początkowym okresie prace prowadziły do *rozszerzenia* systemów zarządzania bazami danych o możliwość wnioskowania z danych. Rozwiązania problemu poszukuje się w obszarze sztucznej inteligencji i uczenia maszynowego (*machine learning*).

Współcześnie można określić pozyskiwanie wiedzy jako wydobywanie użytecznej informacji z danych przy wykorzystaniu metodologii wykorzystywanej w takich dziedzinach wiedzy jak: bazy danych, sztuczna inteligencja, uczenia maszynowe oraz statystyka.

Proces pozyskiwania wiedzy z baz danych zwykle dzieli się na cztery podstawowe grupy zadań:

- **wybór danych** zadanie to obejmuje także czyszczenie danych, usuwanie danych błędnych, powtarzających się i niekompletnych

- **transformacja danych** przygotowanie danych „dla potrzeb" stosowanego algorytmu „drążenia danych", np. zmiana wartości nominalnych na numeryczne
- **drążenie danych** Podstawowe klasy zadań: tworzenie modeli prognostycznych, segmentacja bazy (*clustering*), wykrywanie odchyleń (*outlier detection*)
- **interpretacja wyników**

Odkrywanie wiedzy z baz danych jest procesem interakcyjnym, interaktywnym i w znacznej mierze opartym na *metodzie prób i błędów*.