Bazy Danych

Slajdy W2-W7

Literatura

- **Teoria baz danych**
  - Beynon-Davies P., *Systemy baz danych.* WNT
  - Date C. J. , *Wprowadzenie do systemów baz danych*. WNT
  - Henderson K., *Bazy danych w architekturze klient/serwer.* Robomatic
- **SQL**
  - Debarros A.: *Praktyczny SQL*. Wydawnictwo Naukowe PWN
  - Bowman, J., S.L. Emerson i M. Darnovsky: *Podręcznik języka SQL*. WNT, Warszawa
  - Celko J.: *Praktyki mistrza SQL. Programowanie zaawansowane*. Helion
  - Celko J.: SQL *Zaawansowane techniki programowania*. Mikom, Warszawa
  - Harrington, J.L.: *SQL dla każdego*. EDU-MIKOM
  - Ladanyi, H.: *SQL. Księga eksperta*. Helion

Literatura

- **Diagramy ERD i projektowanie; CASE**
  - **Hernandez M.: *Projektowanie baz danych dla każdego. Przewodnik krok po kroku*, Helion**
  - Connolly T.: *Systemy baz danych: Praktyczne metody projektowania implementacji i zarządzania.* Oficyna Wydawnicza READ ME
  - Barker, R.: *CASE*Method – modelowanie związków encji*. WNT
  - Jaszkiewicz A.: *Inżynieria oprogramowania*. HELION,
  - Yourdon, E.: *Współczesna analiza strukturalna.* WNT
- **ORACLE**
  - Austin Dave: *Poznaj Oracle 8*. (Prosto profesjonalnie). ZNIMIKOM
  - Rogers, U.: *Oracle. Przewodnik projektanta baz danych*. WNT
  - Wrembel, R. I W. Wieczerzycki: *Projektowanie aplikacji bazy danych Oracle*. Wydawnictwo NAKOM

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Źródła

- **Dokumentacja elektroniczna Oracle**
  - https://docs.oracle.com
  - https://docs.oracle.com/en/database/oracle/sql-developer/18.2/books.html
    - 2 Day DBA
    - Database Administrator's Guide
    - PL/SQL Language Reference
    - SQL Language Reference
    - Oracle SQL Developer User's Guide
    - Oracle SQL Developer Data Modeler User's Guide

Przykład wstępny (1)

- **System bazy danych** – to *skomputeryzowany system przechowywania rekordów*, gdzie dane przechowywane są zgodnie z tzw. *modelem danych*
  - rodzaj elektronicznej kartoteki z aktami
  - miejsce przechowywania kolekcji danych - „w plikach"
  - odzwierciedla pewien wycinek rzeczywistości

Przykład wstępny (2)

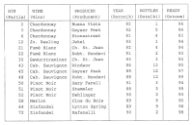
Użytkownik ma do dyspozycji **narzędzia** do przeprowadzania rozmaitych operacji na tych kolekcjach:

- Dodawanie nowych, pustych plików do bazy danych
- Wstawianie nowych danych do istniejących plików
- Odczytywanie danych z istniejących plików
- Usuwanie danych z istniejących plików
- Usuwanie plików, pustych lub nie, z bazy danych

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Przykładowa baza danych - Wina

- (z lewej) Baza złożona z tylko jednego pliku zawierającego informację na temat domowej piwniczki z winami.
- (z prawej) Przykład operacji wyszukiwania w tej bazie wraz z uzyskanymi *wynikami/danymi.*

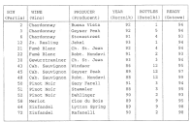


```
SELECT    WINE, WIN, PRODUCER
FROM      CELLAR
WHERE     READY < ??;
```

WINE	WIN	PRODUCER
Cab. Sauvignon	43	Windsor
Pinot Noir	51	Blandlee
Merlot	58	Clos du Bois

Baza danych - Wina

- (z prawej) Przykład operacji *wstaw, zmodyfikuj i usuń (ang. insert, update, delete)*



```
INSERT
INTO   CELLAR ( BIN, WINE, PRODUCER, YRS, BOTTLES, READY )
       VALUES ( 59, 'Pinot Noir', 'Saintsbury', 92, 1, 96 );

UPDATE CELLAR
SET    BOTTLES = 6
WHERE  BIN = 3 ;

DELETE
FROM   CELLAR
WHERE  BIN = 2 ;
```

Uwagi dla przykładu wstępnego

- Pliki w naszym przykładzie są tożsame z **tabelami** (w rzeczywistości z tabelami relacyjnymi);
- Wiersze w takiej tabeli będziemy nazywać **rekordami**;
- Operacje *SELECT, INSERT, UPDATE, DELETE* pokazane wcześniej są przykładami języka baz danych nazywanego **SQL** *(ang. Structured Query Language);*
- SQL jest stosowany w większości komercyjnych relacyjnych baz danych;
- Ciekawostka: Pierwotna wymowa *„sikłel"* obecnie *„es-kiu-el"*.

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

System zarządzania bazą danych

- **System zarządzania bazą danych**, SZBD (*ang.* ***Database Management System, DBMS***) - oprogramowanie bądź system informatyczny służący do zarządzania bazą danych.
- Funkcje oferowane przez SZBD zostaną omówione w dalszej części wykładu
- Składa się z czterech zasadniczych elementów:
  A. Dane
  B. Sprzęt
  C. System bazy\procesy bazy
  D. Użytkownicy

A. Dane

- Dostęp do danych może odbywać się **w trybie jednego użytkownika** (*single-user*) lub w trybie **z wieloma jednoczesnymi użytkownikami** (*multi-user*). W przypadku systemu z wieloma użytkownikami głównym celem SZBD jest sprawienie, że każdy z użytkowników ma wrażenie, iż tylko on korzysta z bazy.
- Dla prostoty będziemy przyjmować, że wszystkie dane przechowywane są w jednej bazie danych.
- Dane przechowywane są więc w sposób ***współdzielony*** jak i ***zintegrowany***.

Integracja – łączenie danych z wielu źródeł

Współdzielenie – mogą być wspólnie używane przez wielu użytkowników

B. Sprzęt

- Szereg rozwiązań dostępnych od ***mikrokomputerów*** do największych systemów klasy ***mainframe***.
- Najważniejsze elementy systemu to:
  - Pamięci masowe – (dyski twarde) używane do przechowywania danych, wraz z dodatkowym osprzętem m.in. kontrolery dysków
  - Procesory
  - Pamięć operacyjna

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

C. System Zarządzania Bazą Danych (1)

- Warstwa pośrednicząca między użytkownikami a fizyczną bazą danych
- Obsługuje żądania użytkowników;
- Uwalnia użytkowników od znajomości szczegółów technicznych (wyższy poziom abstrakcji jak dla języków programowania);
- **Najważniejszy składnik całego systemu bazy danych**;

C. System Zarządzania Bazą Danych (2)

Działanie:

1. Żądanie dostępu od użytkownika (np. z wykorzystaniem podjęzyka SQL)
2. SZBD przechwytuje i analizuje żądanie
3. SZBD bada kolejno dla danego użytkownika odpowiedni schemat *zewnętrzny*, *pojęciowy* i *wewnętrzny* (struktury pamięci)
4. Wykonanie przez SZBD niezbędnych operacji na bazie danych

C. System Zarządzania Bazą Danych (3)

Funkcje realizowane przez SZBD (1):

- **Definiowanie danych** – definiowanie poszczególnych schematów wew./poj./zew. i właściwych odwzorowań
- **Obróbka danych** – obsłużenie żądań użytkownika związane z *wyszukiwaniem, aktualizacją, usuwaniem* oraz *dodawaniem* danych
- Zapewnienie **mechanizmów bezpieczeństwa** i **integracji** danych

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

C. System Zarządzania Bazą Danych (4)

Funkcje realizowane przez SZBD (2):

- Słownik danych *(data dictionary)*:
  - Sam słownik można uznać za bazę danych (lecz systemu, nie użytkownika)
  - Zawiera dane o danych (zwane *metadanymi*) – tj. definicje innych obiektów w bazie
- Zapewnienie wydajności
- Kontrola współbieżności
- Mechanizmy odzyskiwania danych

D. Użytkownicy (1)

Główne grupy:

- **programiści aplikacji** – odpowiedzialni za programy wykorzystujące bazy danych; Programy przetwarzają informację na wszystkie typowe sposoby (*wyszukiwanie, dodawanie, modyfikacja, usuwanie*); aplikacje działają zazwyczaj **w trybie bezpośrednim** (*online*)

D. Użytkownicy (2)

- **użytkownicy** (ang. *end users)* – komunikują się z bazą w sposób bezpośredni za pomocą narzędzi z linii komend lub odpowiednich narzędzi klienckich (często stanowią integralną część SZBD); wykorzystują (bez)pośrednio język zapytań SQL;
- **administrator bazy danych** (ang. *database administrator*, DBA) – osoba, która jest odpowiedzialna za środowiskowy aspekt bazy danych.

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

D. Użytkownicy (3)

- **projektanci baz danych** zajmują się projektowaniem struktury logicznej bazy danych, czyli struktur modelu danych i projektowaniem struktury fizycznej bazy danych, czyli doborem parametrów fizycznego składowania danych na nośnikach. Ponadto, ich zadaniem jest przygotowanie działającej bazy danych.
- **analitycy systemowi** zajmują się analizą wymagań systemu bazy danych i aplikacji. Wynik ich pracy jest podstawą opracowania struktury logicznej (a często również fizycznej) bazy danych i jest podstawą do pracy dla programistów aplikacji.

D. Użytkownicy (4)

Inne grupy:

- Administratorzy serwerów
- Administratorzy sieci komputerowych
- Projektanci narzędzi deweloperskich

Co to jest baza danych?

- **Dane trwałe** (*persistent*) - chcemy odróżnić dane z bazy od danych *„bardziej tymczasowych"* np. instrukcje kontrolne, bloki kontroli programów (czyli dane przejściowe).
- Rozróżnienie między danymi trwałymi a tymczasowymi nie jest sztywne i zależy od kontekstu.

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Definicja bazy danych

Definicja podstawowa:

Baza danych to zbiór informacji zapisanych w ściśle określony sposób w strukturach odpowiadających założonemu *modelowi danych*.

Definicja rozszerzona:

W potocznym ujęciu BD obejmuje dane oraz program komputerowy wyspecjalizowany do gromadzenia i przetwarzania tych danych. Program taki (często zestaw programów) to "Systemem zarządzania bazą danych" (ang. *database management system*, DBMS).

Charakterystyka baz danych (1)

1. Trwałość danych

- Długi czas życia bazy danych – kilka, kilkadziesiąt, kilkaset lat
- Niezależność od działania aplikacji
- Trwałość danych jest niezależna od platformy sprzętowo-programowej

2. Rozmiar wolumenu danych

- Dane nie mieszczą się w pamięci operacyjnej – wymagana pamięć zewnętrzna (dyskowa, optyczna, taśmowa)
- Danych jest zbyt dużo dla ich liniowego przeglądania przez użytkowników

Charakterystyka baz danych (2)

3. Złożoność danych

- ***Złożoność strukturalna*** **i** ***złożoność zależności*** pomiędzy danymi - np. projekt promu kosmicznego, złożony z setek tysięcy elementów
- ***Złożoność semantyczna*** - (np. fakt przyznania kredytu jest uzależniony od spełnienia lub niespełnienia wielu wymagań przez petenta)
- ***Ograniczenia integralnościowe*** - w bazie danych pojawią się wyłącznie dane spełniające te ograniczenia

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Encje i związki (1)

Encja (*entity*) – **potocznie oznacza indywidualną, rozpoznawalną rzecz, jaka ma być reprezentowana w bazie danych;**

- Encja zawiera w sobie cechy (atrybuty) obiektu, który tworzy.
- W przypadku relacyjnych baz danych, encja jest zazwyczaj utożsamiana z tabelą w bazie danych.
- Zadanie:
  - Proszę podać przykłady encji (i atrybutów w encji)

Encje i związki (1)

Encja (*entity*) – **potocznie oznacza indywidualną, rozpoznawalną rzecz, jaka ma być reprezentowana w bazie danych;**

- Encja zawiera w sobie cechy (atrybuty) obiektu, który tworzy.
- W przypadku relacyjnych baz danych, encja jest zazwyczaj utożsamiana z tabelą w bazie danych.
- Przykłady encji (i atrybuty w encji):
  - *Pracownik* (*imię, nazwisko, PESEL*)
  - *Produkt* (*wysokość, szerokość, długość, cena*)

Związki binarne
związek <SP> łączy Sapliera – Partia

Relacje są zwierciadłowe
związki typowane np. <SP>
związki relacyjne np <PP>

Atrybuty

- **Encje** mają własności – **atrybuty**
- (*Załóżmy na razie, że*) wszystkie atrybuty są „proste" i są reprezentowane przez „proste typy" m.in. liczby, napisy, daty itd.

BIN (Partia)	WINE (Wino)	PRODUCER (Producent)	YEAR (Rocznik)	BOTTLES (Butelki)	READY (Gotowe)
2	Chardonnay	Buena Vista	92	1	94
3	Chardonnay	Geyser Peak	92	5	94
6	Chardonnay	Stonestreet	91	4	93
12	Jo. Reeling	Jekel	93	1	94
21	Fumé Blanc	Ch. St. Jean	92	4	94
22	Fumé Blanc	Robt. Mondavi	91	2	93

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Dlaczego bazy danych?

Przewaga baz nad metodami tradycyjnymi – „papierowymi"/"plikami" (np. dla dużej restauracji – baza dań/tabela win):

- **Zawartość** – nie musi przeszukiwać grubych ksiąg
- **Szybkość** – komputer potrafi wyszukiwać i zmieniać dane dużo szybciej niż człowiek
- **Mniej pracy** – brak konieczności ręcznego korygowania i dopisywania w księgach
- **Aktualność** – bieżące dane dostępne na żądanie

Wyobraźmy sobie konieczność ręcznego uzupełniania ksiąg dla wielkiej sieci restauracji.

Interakcja z bazą danych

Język SQL (ang. *Structured Query Language*)

- **interakcja** programu użytkowego (aplikacji) z bazą odbywa się za pomocą języka SQL
- **język deklaratywny** - specyfikujemy tylko co chcemy otrzymać, np. *wyświetl najstarszego pracownika*
- **ustandaryzowany** - SQL-92, SQL- 99, SQL-2003; brak jednak 100% zgodności
- Prowadzone są również prace nad alternatywnymi językami zapytań opartymi na SQL'u.
  - Przykładem takiego języka może być język ciągłych zapytań CQL lub język SQL wyposażony w możliwość przetwarzania sekwencji danych - AQuery.

Interakcja z bazą danych (2)

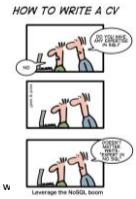
- **Aplikacje** (pod spodem nadal SQL)
  - **Formularze** - elektroniczne formularze z polami, listami, elementami wyboru umożliwiają wstawianie, modyfikowanie, usuwanie, wyszukiwanie danych
  - **Raporty** - umożliwiają prezentowanie zawartości bazy danych (teksty, wykresy, grafika)

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Technologie implementacyjne aplikacji

- **Języki 3GL**
  - np. C, C++, Visual Basic, Visual C++ - biblioteki umożliwiające zagnieżdżanie poleceń SQL w kodzie
- **Języki 4GL**
  - np. SAS 4GL, Oracle Forms – umożliwiają bezpośrednie umieszczanie poleceń SQL w kodzie aplikacji i bezpośrednią obsługę wyników poleceń SQL
- **Java, PHP, JavaScript**
  - stosowane w aplikacjach web'owych pracujących w architekturze 3-warstwowej

Nie relacyjne systemy



- Systemy nie relacyjne (nie relacyjne modele danych):
  - Dedukcyjne DBMS
  - Eksperckie DBMS
  - Obiektowe DBMS
  - Semantyczne DBMS
  - Strumieniowe DBMS
- Modele hierarchiczny i sieciowy nie są już stosowane w systemach.
- Obecnie w bazach danych najczęściej stosuje się model relacyjny, obiektowo-relacyjny lub semistrukturalny

Architektura systemu baz danych (1)

- Celem zastosowania **architektury trójwarstwowej** jest **uniezależnienie prezentacji danych** od sposobu, w jaki są one przechowywane.
- W **architekturze trójwarstwowej** ANSI/SPARC wyróżniamy trzy poziomy:
  - *Poziom zewnętrzny* – sposób w jaki poszczególni użytkownicy widzą dane;
  - *Poziom pojęciowy „pośredni"*
  - *Poziomom wewnętrzny* – fizyczny sposób przechowywania danych;

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Architektura systemu baz danych (2) Poziom zewnętrzny

- Poziom indywidualnego użytkownika (np. programista, DBA itd.);
- Użytkownik dysponuje językiem właściwym dla danego systemu (np. programista - język 4GL, inżynier baz danych - język zapytań);
- Wspólna cechą języków jest zawieranie **osadzonego** (*embeded)* **podjezyka danych** (*data sublanguage, DSL)* – tzn. podzbiór języka związany z operacjami i obiektami bazy danych.

Architektura systemu baz danych (3) Poziom zewnętrzny c.d.

Podjęzyk danych składa się z przynajmniej dwóch podrzędnych języków:

- **języka definicji danych** *(data definitione language, DDL)* – deklarowanie obiektów BD
- **języka operowania danymi** *(data manipulation language, DML)* – przetwarzanie obiektów BD

Reasumując schemat zewnętrzny stanowi interfejs użytkownika do bazy danych. Schemat ten odwzorowuje schemat implementacyjny w schemat poprzez który użytkownik widzi bazę danych i pracuje z nią.

Architektura systemu baz danych (4) Poziom pojęciowy

- **Pojęciowy model danych** – stanowi reprezentację zawartości informacyjnej bazy danych. Stosowany format jest oderwany od fizycznego sposobu przechowywania danych;
- Pojęciowy model składa się z wystąpień **rekordów pojęciowych**; np. zbiór wystąpień rekordów pracowników, rekordów części itd.
- Pojęciowy model definiuje się za pomocą **schematu pojęciowego** stanowiącego związek wszystkich schematów zewnętrznych wraz z regułami bezpieczeństwa i integralności.

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Architektura systemu baz danych (5) Poziom wewnętrzny

- Reprezentuje niski poziom bazy danych;
- Składa się z wystąpień wielu „rekordów wewnętrznych";
- Poziom ten znajduje się o jeden stopień wyżej od poziomu fizycznego tzn. nie zajmuje się blokami lub stronami pamięci.
- **Schemat wewnętrzny** definiuje typy rekordów, precyzuje indeksy itd.

Architektura systemu baz danych (6) Zalety architektury trójwarstwowej

- **Różne perspektywy użytkowników**: architektura ANSI-SPARC pozwala stworzyć perspektywy dla różnych użytkowników dostosowane do ich potrzeb i wymagań. Każdy użytkownik powinien mieć dostęp do tych samych danych, ale na różne sposoby. Perspektywy użytkowników powinny być niezależne od siebie – zmiana jednej z nich nie powinna wpływać na pozostałe.
- **Ukrycie fizycznej implementacji**: użytkownicy nie powinni mieć dostępu do niskopoziomowych szczegółów przechowywania danych. Powinni mieć możliwość pracy z danymi bez zaprzątania sobie głowy tym, jak są one zapisane.

Architektura systemu baz danych (7) Zalety c.d.

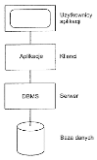
- **Zmiany w niższych warstwach**: administrator bazy danych powinien mieć możliwość zmiany sposobu przechowywania danych bez naruszania perspektyw użytkowników. Podobnie, zmiana dysku twardego w serwerze bazy danych nie powinna naruszać struktury danych w bazie.
- **Zmiany koncepcji przechowywania danych**: administrator powinien mieć możliwość zmiany koncepcji bazy danych lub jej struktury również bez naruszania perspektywy użytkownika.

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Architektura komunikacyjna

Dwie podstawowe **architektury komunikacyjne systemów baz danych** to:

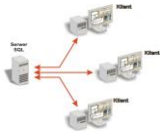
- **architektura klient-serwer**
- **architektura 3-warstwową**



Idea polega na założeniu, że interfejs użytkownika, przetwarzanie danych i składowanie danych są rozwijane w postaci osobnych modułów, zwykle na oddzielnych platformach;

Architektura klient-serwer (1)

- **Inne spojrzenie niż w architekturze ANSI/SPARC**
- Główne zadanie systemów klient-serwer to wspieranie rozwoju i wykonywania aplikacji na bazie danych;
- System bazy danych w uproszczeniu jest dwuczęściową strukturą złożoną z:
  - Serwera *(server / backend)*
  - Szeregu klientów (*client / frontend*)



Architektura klient-serwer (2)

Elementy:

- **Serwer** – wykonuje podstawowe funkcje jak definiowanie danych, obróbkę danych, zapewnienie integralności i bezpieczeństwa; serwer to synonim SZBD;
- **Klienci** – różne aplikacje korzystające z SZBD napisane przez programistów jak i aplikacje/narzędzia wbudowane w SZBD

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Przetwarzanie rozproszone (1)

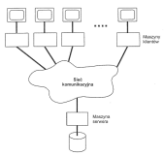
- Oznacza, w „architekturze klient-serwer” sytuację, gdzie klient i serwer **działają na różnych maszynach**;

Zalety:

- Przetwarzanie rozdzielone na różne maszyny w sposób równoległy na kliencie i serwerze;
- Serwer i klient mogą posiadać odpowiednią konfigurację dostosowaną do realizowanych zadań;

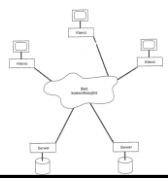
Przetwarzanie rozproszone (2)

- Jeden serwer, wielu klientów



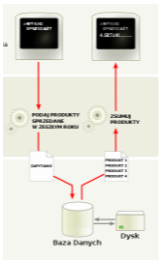
Przetwarzanie rozproszone (3)

- Wiele serwerów, wielu klientów



Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Architektura komunikacyjna - 3-warstwowa



Rola administratora bazy danych (1)

Administrator *(database administrator, DBA)*– osoba, która:

- podejmuje decyzje strategiczne i realizuje politykę dotyczącą danych w firmie,
- zapewnia konieczne wsparcie techniczne oraz
- sprawuje ogólną kontrolę systemu na poziomie technicznym.

Rola administratora bazy danych(2)

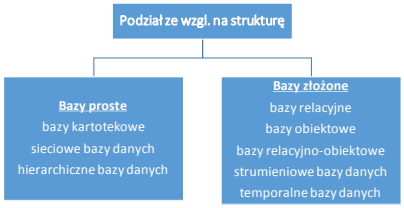
Administrator ponadto:

- **Definiowanie schematu pojęciowego** – decyduje jakie informacje przechowywać w bazie:
  - wpływ na projekt logiczny bazy a następnie na tej podstawie tworzy odpowiedni schemat pojęciowy, stosując *pojęciowy DDL;*
  - W praktycy kwestie te rzadko wyglądają tak jasno;
- **Definiowanie schematu wewnętrznego** – proces fizycznego projektu bazy danych

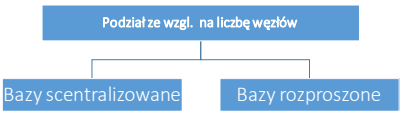
Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Podział systemów baz danych (1)

Możliwe są różne kryteria podziału systemów baz danych.



Podział systemów baz danych (2)



Podział systemów baz danych (3)

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Dostępne SZBD (1)

Komercyjne bazy danych:

- **Oracle**: wersja 9i – 19c/21c w tym:
  - (Oracle Database Express Edition (XE))
- **IBM**: DB2 UDB, Informix(R) Dynamic Server
- **Microsoft**: SQL Server
- **Sybase (od 2014 SAP)**: Adaptive Server Enterprise, Adaptive Server Anywhere

Dostępne SZBD (2)

Niekomercyjne bazy danych:

- MySQL
- PostgreSQL
- FireBird
- Berkley DB
- Firebase
- MongoDB

Rys historyczny

Początek lat 60	Firmy tworzą własne systemy zarządzania danymi przeznaczone do rozwiązania konkretnych problemów	Systemy plików, brak współdzielenia, brak niezależności danych od aplikacji
Połowa lat 60	Pakiety baz danych ogólnego przeznaczenia, IDS *Integrated Data Source* firmy GEC *General Electric Company*	Charles Bachman
Druga połowa 60	SZBD IDMS dla IBM360, początek sieciowych baz danych	Goodrich, Cullinane
1969	Stworzono model CODASYL - standard sieciowych baz danych	Panowanie sieciowych baz danych
1970	Codd publikuje artykuł *A Relational Model for Large Shared Data Banks*	Powstanie koncepcji relacyjnych baz danych
	IBM opracowuje prototyp relacyjnego SZBD o nazwie System/R, który później przekształcił się w INGRES	
Lata 70 - początek 80	Intensywne prace naukowe nad relacyjnymi bazami danych, panowanie modelu sieciowego	

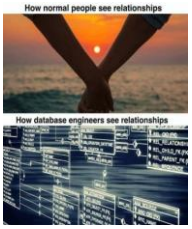
Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Rys historyczny

1983	IBM prezentuje pierwszą komercyjną relacyjną bazę danych DB2	Początek ery RBD
Lata 80 - początek 90	Rozwój osobistych komputerów przyczynia się do rozpowszechnienia RBD	
II połowa 80	Poszukiwanie nowego modelu danych, ponieważ: 1) stacje graficzne - nowe dziedziny; 2) przetwarzanie równoległe i czasu rzeczywistego; 3) popularyzacja obiektowości 4) wbudowanie przetwarzania danych do BD	Post relacyjne bazy danych
1989, Kyoto	Opublikowano *Manifest* obiektowych baz danych, co dało początek standaryzacji tych systemów	
Lata 90	Prace nad obiektowymi bazami danych, pierwsze systemy komercyjne tego typu	Dominacja relacyjnych BD
1996	ODMG-2 – Przyjęcie standardu światowego dla producentów SOBD	Popularyzacja obiektowych baz danych
Obecnie	Koegzystencja RBD oraz OBD	

Relacyjny system baz danych

- Przechowuje wszystkie **dane w relacjach** (***encjach = tabelach***).
- Każda relacja zawiera dane na konkretny temat, np. dane o klientach, pracownikach, towarach, sprzedaży itp.
- System bazy danych zarządza tymi informacjami, pozwala m.in. na szybsze ich wyszukiwanie i zorganizowanie, przydzielanie pamięci.



12 postulatów Codda


Edgara F. Codd pionier relacyjnych baz danych

0. System musi być kwalifikowany jako relacyjny, jako baza danych i jako system zarządzania

1. Postulat informacyjny - dane są reprezentowane jedynie poprzez wartości atrybutów w wierszach tabel,

2. Postulat dostępu - każda wartość w bazie danych jest dostępna poprzez podanie nazwy tabeli, atrybutu oraz wartości klucza podstawowego,.

4. Postulat dotyczący katalogu - wymaga się, aby system obsługiwał wbudowany katalog relacyjny z bieżącym dostępem dla uprawnionych użytkowników używających języka zapytań,

5. Postulat języka danych - system musi dostarczać pełnego języka przetwarzania danych, który może być używany w trybie interaktywnym jak i w obrębie programów aplikacyjnych, obsługuje operacje definiowania danych, operacje manipulowania danymi, ograniczenia związane z bezpieczeństwem i integralnością oraz operacje zarządzania transakcjami,

6. Postulat modyfikowalności perspektyw - system musi umożliwiać modyfikowanie perspektyw, o ile jest ono (modyfikowanie) semantycznie realizowalne,

7. Postulat modyfikowalności danych - system musi umożliwiać operacje modyfikacji danych, musi obsługiwać operatory INSERT, UPDATE oraz DELETE,

8. Postulat fizycznej niezależności danych - zmiany fizycznej reprezentacji danych i organizacji dostępu nie wpływają na aplikację,

9. Postulat logicznej niezależności danych - zmiany wartości w tabelach nie wpływają na aplikację,

10. Postulat niezależności więzów spójności - więzy spójności są definiowane w bazie i nie zależą od aplikacji,

11. Postulat niezależności dystrybucyjnej - działanie aplikacji nie zależy od modyfikacji i dystrybucji bazy,

12. Postulat bezpieczeństwa względem operacji niskiego poziomu - operacje niskiego poziomu nie mogą naruszać modelu relacyjnego i więzów spójności.

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Język SQL
(*ang. Structured Query Language, SQL*)

- **Strukturalny język zapytań** używany do tworzenia, modyfikowania baz danych oraz do umieszczania i pobierania danych z baz danych
- Język SQL jest językiem **deklaratywnym** (oparty na rachunku relacyjnym).
- Decyzję o sposobie przechowywania i pobrania danych pozostawia się systemowi zarządzania bazą danych (DBMS)

SQL Historia

- **Opracowany w latach 70. w firmie IBM.**
- Stał się standardem w komunikacji z serwerami relacyjnych baz danych.
- Wiele współczesnych systemów relacyjnych baz danych używa do komunikacji z użytkownikiem SQL, dlatego potocznie mówi się, że korzystanie z relacyjnych baz danych to korzystanie z SQL-a.
- **Pierwszą firmą, która włączyła SQL do swojego produktu komercyjnego był Oracle.**
- Dalsze wprowadzanie SQL-a, w produktach innych firm, wiązało się nierozłącznie z wprowadzaniem modyfikacji pierwotnego języka.
- Wkrótce utrzymanie dalszej jednolitości języka wymagało wprowadzenia standardu.

Standardy SQL (1)

- **Od 1986 SQL stał się oficjalny standardem**, wspieranym przez Międzynarodową Organizację Normalizacyjną (**ISO**) i jej członka, Amerykański Narodowy Instytut Normalizacji (**ANSI**).
- Wczesne wersje specyfikacji (**SQL86** i **SQL89**) były w dużej mierze jedynie określeniem **wspólnej płaszczyzny** łączącej różne istniejące wówczas produkty i pozostawiały wiele swobody twórcom implementacji.

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Standardy SQL (2)

- Z czasem jednak systemy komputerowe uległy integracji i rynek zaczął domagać się aplikacji oraz ich funkcji faktycznie współpracujących z wieloma różnymi bazami danych.
- Pojawiła się potrzeba określenia standardu ściślejszego.
- Mógł on jednocześnie obejmować nowe elementy, nieujęte do tej pory w języku. Tak powstał standard **SQL92**, **obowiązujący w wielu produktach** do dziś.

Standardy SQL (3)

SQL:2003 – nowy standard języka SQL. Jest to w zasadzie poprawione SQL:1999 z wyjątkiem części SQL/XML oraz kilku dodatkowych właściwości.

Zmiany wprowadzone w SQL 2003:

- Dodano nowe typy danych (BIGINT, MULTISET oraz XML)
- Usunięto typy BIT oraz BIT VARYING
- Wprowadzono rozszerzenia w sposobie wywoływania procedur
- Poszerzono instrukcję CREATE TABLE (CREATE TABLE { LIKE | AS })
- Wprowadzono instrukcję MERGE
- Wprowadzono nowy obiekt schematu – generator sekwencji
- Wprowadzono dwa nowe typy kolumn – identyfikatory oraz generowane
- Wprowadzono retrospektywne sprawdzanie więzów integralności
- Wprowadzono rozszerzenia dla OLAP w formie wbudowanych funkcji (skalarnych i agregujących)
- Wprowadzono klauzulę WINDOW

.

Standardy SQL (4)

- Prowadzone są również prace nad **alternatywnymi językami zapytań** opartymi na SQL.
- Przykładem takiego języka może być język ciągłych zapytań **CQL** lub język SQL wyposażony w możliwość przetwarzania sekwencji danych **AQuery**.
- **Obiektowy język zapytań** (Object Query Language, OQL) –według standardu ODMG (Object Data Management Group). Składnia OQL wzorowana jest na SQL.

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Systemy bazodanowe używające SQL

- Apache Derby
- Caché
- DATAllegro
- DB2
- Firebird
- First SQL
- Greenplum
- HSQL
- Ingres
- Informix
- InterBase SQL
- MaxDB (dawniej SAP DB)
- Microsoft Access
- Microsoft Jet
- Microsoft SQL Server
- Mimer SQL
- MySQL
- mSQL
- OpenLink Virtuoso
- Oracle
- Oracle Rdb
- PostgreSQL
- Pervasive
- SQL/DS
- SQLite
- Sybase (Sybase Adaptive Server Enterprise, Sybase SQL Anywhere, Sybase IQ)
- Teradata

Formy SQL-a

- SQL wykorzystywany wyłącznie do **komunikacji z bazą danych**.
- Nie posiada on cech pozwalających na **tworzenie kompletnych programów**.
- Jego wykorzystanie może być trojakie i z tego względu wyróżnia się trzy formy SQL-a:
  1. SQL interakcyjny (autonomiczny)
  2. Statyczny kod SQL (Static SQL)
  3. Dynamiczny kod SQL (Dynamic SQL)

SQL interakcyjny (autonomiczny)

- Wykorzystywany jest przez użytkowników w celu bezpośredniego pobierania lub wprowadzania informacji do bazy.
- Przykładem może być zapytanie prowadzące do uzyskania zestawienia aktywności kont w miesiącu. Wynik jest wówczas przekazywany na ekran, z ewentualną opcją przekierowania go do pliku lub drukarki.

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

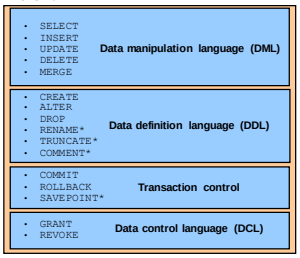
Statyczny kod SQL (Static SQL)

- Nie ulega zmianom w sensie zachowania niezmiennej treści instrukcji, które jednak zawierać mogą odwołania do zmiennych lub parametrów przekazujących wartości z lub do aplikacji.
- Statyczny SQL występuje w dwóch odmianach:
  - **Osadzony SQL** (Embedded SQL) oznacza włączenie kodu SQL do kodu źródłowego innego języka (np. Java), jedynie odwołania do bazy danych realizowane są w SQL. Do przenoszenia wartości wykorzystywane są **zmienne**.
  - **Język modułów.** W tym podejściu moduły SQL łączone są z modułami kodu w innym języku. **Moduły kodu SQL przenoszą wartości do i z parametrów**, podobnie jak to się dzieje przy wywoływaniu podprogramów w większości **języków proceduralnych**.

Dynamiczny kod SQL (Dynamic SQL)

- **Generowany jest w trakcie pracy aplikacji**. Wykorzystuje się go w miejsce podejścia statycznego, jeżeli w chwili pisania aplikacji nie jest możliwe określenie treści potrzebnych zapytań, np. powstaje ono w oparciu o decyzje użytkownika.
- Dynamiczne budowanie zapytania - bez wykorzystania dynamicznego SQL mogliśmy budować zapytania o z góry znanej konstrukcji.
  - Modyfikowalnym elementem był np. parametr wyszukiwania (np. select * from employees where employee_id= **X**), nie mogliśmy natomiast zmieniać nazwy tabeli do której się odwołujemy.

Składowe języka SQL



Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Charakterystyka języka SQL

Trzy najważniejsze składowe języka SQL:

- **języka definiowania danych** - języka **DDL** (ang. *Data Definition Language*),
- **języka sterowania danymi** - języka **DCL** (ang. *Data Control Language*)
- **język operowania na danych** – język **DML** (ang. *Data Manipulation Language*)

Dodatkowo, niektóre programy do łączenia się z silnikiem bazy danych, używają swoich własnych instrukcji, spoza standardu SQL (np. PostgreSQL).

DDL - CREATE, DROP, ALTER
DCL - GRANT, REVOKE, DENY
DML

Pisanie zapytań SQL

- Wielkość liter nie ma znaczenia, ale instrukcje SQL w obrębie zapytań tradycyjnie zapisywane są **wielkimi literami** jednak nie jest to wymóg.
- **Kod SQL** może być w jednej lub więcej linii.
- **Słowa kluczowe** nie mogą zostać skrócone lub dzielone.
- **Klauzule** są zazwyczaj umieszczone **w oddzielnych wierszach**.
- Zapytanie kończy się znakiem średnika, czyli **;**
- W celu poprawy czytelności:
  - Stosuj akapity
  - Pisz słowa kluczowe wielkimi literami

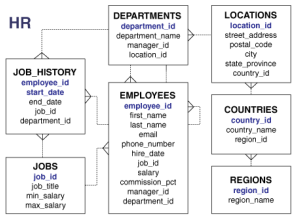
Formatowanie zapytań SQL (1)

- Przykład kodu niesformatowanego:

```
SELECT r.last_name, (SELECT MAX(YEAR(championship_date))
FROM champions AS c WHERE c.last_name = r.last_name AND
c.confirmed = 'Y') AS last_championship_year FROM riders
AS r WHERE r.last_name IN (SELECT c.last_name FROM
champions AS c WHERE YEAR(championship_date) > '2008' AND
c.confirmed = 'Y');
```

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Baza HR



- **Można wydrukować i nosić na zajęcia. Proszę przynieść na kolokwium.**

Projekcja - atrybut

Selekcja - wiersz

Łączenie - dwóch

Podstawy składni SQL SELECT

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM   table;
```

W składni:

`SELECT`	wybór jednej lub wielu kolumn
`*`	wszystkie kolumny
`DISTINCT`	ukrycie duplikatów
`column``|`*`expression`*	wybór kolumn lub wyrażenia
`alias`	nadanie wybranym kolumno nagłówków
`FROM` *`table`*	wskazanie tabeli zawierającej kolumny

+, -, *, / działa

Reguły pierwszeństwa działań artmetycznych

- **Dzielenie** i **mnożenie** ma pierwszeństwo przed dodawaniem i odejmowaniem
- **Operatory o tym samym priorytecie** są wykonywane od lewej do prawej.
- **Nawiasy** są stosowane do zmiany domyślnych reguł przetwarzania.

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Wartość NULL

null to wartość nie istniejąca

not null – ograniczenie integralnościowe

null * ω = null jak cokolwiek na null

Aliasy – Przykład

```
SELECT last_name AS name, commission_pct comm
FROM   employees;
```

NAME	COMM
King	
Kochhar	
De Haan	

...

jeśli chcemy nazwę nazwać to wadzić ją w " "

Operator konkatenacji „||”

- Łączy dane z kolumny lub ciąg znaków z danymi innej kolumny;

```
… 'Abel'||NULL||'Ellen' …
```

adw. || "cos" || adw

//concat()

w oracle AbelEllen

indziej null, zależy od implementacji

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Powtarzające się wiersze

- Domyślnie wyświetlane są wszystkie wiersze, nawet powtarzające się;
- **DISTINCT | UNIQUE** - używany wraz z instrukcją SELECT

```
SELECT DISTINCT department_id
FROM   employees;
```

DEPARTMENT_ID
10
20
50

...

Wyświetlanie struktury tabeli - Przykład

```
DESCRIBE employees
```

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

daje nam info o schemacie tabeli i ograniczeniach

Ograniczanie wierszy, które są wybrane

- Ogranicz liczbę wierszy, które są zwracane korzystając

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM   table
[WHERE condition(s)];
```

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Użycie klauzuli WHERE

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  department_id = 90 ;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

Ciągi znaków i daty w '
zgodne z formatem RR/MM/DD
lub DD.MM.YYYY

Wah ≠ WAH

Operatory porównywania

Operator	Znaczenie
=	Równy
>	Większy
>=	Większy lub równy
<	Mniejszy
<=	Mniejszy lub równy
<> alternatywnie !=, ^=	Nie równy
[NOT] BETWEEN...AND...	Pomiędzy dwiema wartościami (włącznie)
[NOT] IN(set)	Równy dowolnej wartości z listy
[NOT] LIKE	Pasuje do wzorca znakowego
IS [NOT] NULL	Jest wartością NULL

- **Składnia**: ... **WHERE** *expr* ***operator*** *value*

Użycie warunku BETWEEN

```
SELECT last_name, salary
FROM   employees
WHERE  salary BETWEEN 2500 AND 3500 ;
```

Dolne ograniczenie — Górne ograniczenie

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Operator IN może być stosowany z dowolnym typem danych

```
SELECT employee_id, manager_id
FROM   employees
WHERE  last_name IN ('Hartstein', 'Vargas');
```

Operator LIKE

- **Operator LIKE sprawdza, czy w danym ciągu tekstowym występuje określony wzorzec.**

```
SELECT   first_name
FROM     employees
WHERE    first_name LIKE 'S%';
```

Operator NULL

```
SELECT last_name, job_id, commission_pct
FROM   employees
WHERE  commission_pct IS NULL;
```

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Spójniki logiczne

Operator	Znacznie
AND	Zwraca TRUE jeżeli *oba* warunki są spełnione (prawdziwe)
OR	Zwraca TRUE jeżeli *dowolny* z warunków jest spełniony (prawdziwe)
NOT	Zwraca TRUE jeżeli warunek jest nie spełniony (fałsz)

- Dotychczasowe przykłady zawierały tylko jeden warunek w klauzuli **WHERE**.
- Aby użyć kilku warunków w jednej klauzuli WHERE, używamy operatorów **AND** i **OR**.

Przykłady użycia operatora NOT

Przykłady:

- ... WHERE salary **NOT** BETWEEN 10000 AND 15000
- ... WHERE last_name **NOT** LIKE '%A%'
- ... WHERE commission_pct IS **NOT** NULL

Kolejność wykonywania operacji

Priorytet	Operator
1	Operatory arytmetyczne
2	Operatory łączenia
3	Operatory porównania
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Nie równe
7	NOT warunek logiczny
8	AND warunek logiczny
9	OR warunek logiczny

Kolejność może być zmieniona z zastosowaniem nawiasów.

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Klauzula ORDER BY

```
SELECT expr
  FROM table
[WHERE condition(s)]
[ORDER BY {Column-Name|ColumnPosition|Expression}
    [ ASC | DESC ]
    [ NULLS FIRST | NULLS LAST ]]
```

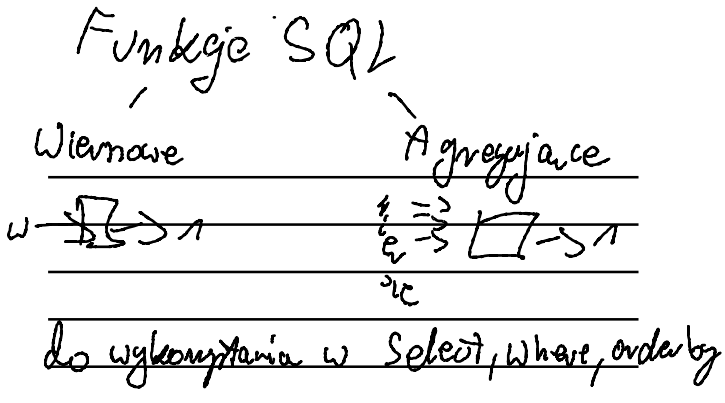
- **ORDER BY** znajduje się na ostatnim miejscu w zapytaniu SQL.
- Jaka jest kolejność sortowania dla wartości zawierających NULL gdy nie stosujemy NULLS FIRST/LAST ?

Inne przykłady dla ORDER BY

1. Sortowanie z użyciem wartości numerycznej
SELECT name, salary, bonus
FROM employee
ORDER BY salary+bonus
2. Sortowanie z użyciem funkcji
SELECT i, len
FROM measures
ORDER BY sin(i)
3. Sortowanie z wymuszonym porządkiem wartości Null
SELECT *
FROM t1
ORDER BY c1 DESC NULLS LAST

Funkcje operujące ciągami

Funkcja	Wynik
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course



Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Funkcje operujące znakami

Funkcja	Wynik
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(salary,10,'*')	*****24000
RPAD(salary, 10, '*')	24000*****
REPLACE ('JACK and JUE','J','BL')	BLACK and BLUE
TRIM('H' FROM 'HelloWorld')	elloWorld

może się trafić na bolokwium (Wyjście)

Oracle Database SQL Reference SUBSTR (1)

• Prz

{ **SUBSTR** | **SUBSTRB** | **SUBSTRC** | **SUBSTR2** | **SUBSTR4** } (char, position [, substring_length])

Dokumentacja:

https://docs.oracle.com/cd/B19306_01/server.102/b14200/functions162.htm

Oracle Database SQL Reference SUBSTR (2)

{ **SUBSTR** | **SUBSTRB** | **SUBSTRC** | **SUBSTR2** | **SUBSTR4** } (char, position [, substring_length])

- **Purpose**

The SUBSTR functions return a portion of **char**, beginning at character **position**, **substring_length** characters long. […]

- If **position** is 0, then it is treated as 1.
- If **position** is positive, then Oracle Database counts from the **beginning of char** to find the first character.
- If **position** is negative, then Oracle counts backward from the **end of char**.
- If **substring_length** is omitted, then Oracle returns all characters to the **end of char**. If **substring_length** is less than 1, then Oracle returns **null**.

char can be any of the datatypes **CHAR**, **VARCHAR2**, **NCHAR**, **NVARCHAR2**. Both **position** and **substring_length** must be of datatype **NUMBER**, or any datatype that can be implicitly converted to **NUMBER**, and must resolve to an integer. The return value is the same datatype as **char**. Floating-point numbers passed as arguments to **SUBSTR** are automatically converted to integers.

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Funkcje operujące na liczbach

- **ROUND**: Zwraca liczbę *n* zaokrągloną do *m* miejsc po przecinku.
- **TRUNC**: Zwraca liczbę *n* „obciętą" do *m* miejsc po przecinku.
- **MOD**: Zwraca resztę z dzielenia liczby *m* przez liczbę *n*.

Funkcja	Wynik
ROUND(45.926, 2)	45.93
TRUNC(45.926, 2)	45.92
MOD(1600, 300)	100

Pozostałe funkcje liczbowe

Funkcja	Wynik
abs(*n*)	wartość bezwzględna liczby *n*,
ceil(*n*)	najmniejsza liczba całkowita >= *n*,
floor(*n*)	największa liczba całkowita <= *n*,
power(*n*, *m*)	*n* podniesione do potęgi *m*,
sign(*n*)	zwraca 1 dla *n*>0, 0 dla *n* = 0 oraz -1 dla *n*< 0,
sqrt(*n*)	pierwiastek kwadratowy *n*.

Praca z Datami

- Daty w bazie Oracle są przechowywane w specjalnym wewn. formacie: ***wiek, rok, miesiąc, dzień, godzina, minuta, sekundy***.
- Domyślny format prezentowania danych (*zależny od ustawień NLS*) : DD-MON-RR lub RR/MM/DD.
  - Alternatywą jest ustawienie odpowiedniej zmiennej NLS (NLS_LANGUAGE)
  - ALTER SESSION SET NLS_LANGUAGE = 'Polish';

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date < '01-FEB-88';
```

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Operatory arytmetyczne i daty

- **Przykład 1**. Wyświetl datę jaka będzie za 100 dni.

```
SELECT SYSDATE+100
FROM DUAL;
```

1/24 = 1h

d1 - d2 = wartość w dniach

Funkcje operujące na datach

Funkcja	Wynik
MONTHS_BETWEEN(date1, date2)	Liczba miesięcy pomiędzy datami
ADD_MONTHS(date, n)	Dodanie miesiąca do daty
NEXT_DAY(date, 'char')	Następny dzień od wyspecyfikowanej daty
LAST_DAY(date)	Ostatni dzień miesiąca
ROUND(date[,'fmt'])	Zaokrąglenie daty
TRUNC(date[, 'fmt'])	Obcięcie daty

Funkcje operujące na datach

Funkcja	Wynik
MONTHS_BETWEEN('01-SEP-95','11-JAN-94')	19.6774194
ADD_MONTHS('11-JAN-94',6)	'11-JUL-94'
NEXT_DAY('01-SEP-95','FRIDAY')	'08-SEP-95'
LAST_DAY('01-FEB-95')	'28-FEB-95'

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Funkcje operujące na datach

- Przyjmijmy, że `SYSDATE = '25-JUL-03'`:

Funkcja	Wynik
ROUND(SYSDATE,'MONTH')	01-AUG-03
ROUND(SYSDATE ,'YEAR')	01-JAN-04
TRUNC(SYSDATE ,'MONTH')	01-JUL-03
TRUNC(SYSDATE ,'YEAR')	01-JAN-03

Konwersja typów

Użycie TO_CHAR z liczbami (2)

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM   employees
WHERE  last_name = 'Ernst';
```

Auto:
char -> num
char -> date
num -> char
date -> char

Użycie TO_NUMBER oraz TO_DATE

- Zamienia łańcuch znaków *char* na liczbę:

```
TO_NUMBER(char[, 'format_model'])
```

- Przekształca ciąg znaków *char* w datę (DATE), zgodnie z formatem:

```
TO_DATE(char[, 'format_model'])
```

- Przykłady:
  - TO_NUMBER('100.00', '9G999D99');
  - TO_DATE('May 24, 1999', 'Month DD, YYYY'); Result: 24-MAY-99

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

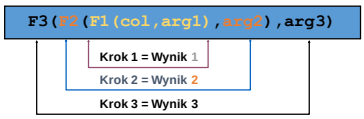
Funkcja TO_CHAR z datami

```
TO_CHAR(date, 'format_model')
```

Element	Wynik
YYYY	pełny czterocyfrowy rok
YEAR	rok słownie
MM	numer miesiąca
MONTH	pełna nazwa miesiąca
MON	trzyliterowy skrót miesiąca
DY	trzyliterowy skrót dnia
DAY	nazwa dnia tygodnia w pełnym brzmieniu, uzupełniona do 9 znaków
DD	numer dnia w miesiącu: od 1 do 31

num
9 num
0 zero
$ waluta
L float
. .
, ,

Zagnieżdżone Funkcje



Inne ważne funkcje

Poniższe funkcje działają z dowolnymi typami danych:

- `NVL (expr1, expr2)` – if e1 null then expr 2 else expr 1
- `NVL2 (expr1, expr2, expr3)` if e1 = null then e2 else e3
- `NULLIF (expr1, expr2)`
- `COALESCE (expr1, expr2, ..., exprn)`

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Typy funkcji grupujących

/agregujące

1. AVG
2. COUNT
3. MAX
4. MIN
5. STDDEV
6. SUM
7. VARIANCE

1 wynik na grupę!

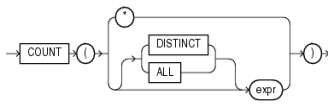
ignorują nulle

Funkcje grupujące - Składnia

```
SELECT      [column,] group_function(column), ...
FROM        table
[WHERE      condition]
[GROUP BY   column]
[ORDER BY   column];
```

Funkcja COUNT

Jak należy odczytać działanie funkcji **count?**



Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Użycie GROUP BY

```
SELECT   AVG(salary)
FROM     employees
GROUP BY department_id ;
```

AVG(SALARY)
4400
9500
3500
6400
10033.3333
19333.3333
10160
7000

Użycie GROUP BY dla wielu kolumn

```
SELECT   department_id dept_id, job_id, SUM(salary)
FROM     employees
GROUP BY department_id, job_id ;
```

Filtrowania wyników z klauzulą HAVING

- Klauzula **HAVING *warunek_logiczny*** – umożliwia **wybór grup**, spełniających warunek logiczny.
- **Warunek logiczny** może być skonstruowany jedynie **z funkcji agregujących** i/lub wyrażeń grupujących.

```
SELECT    column, group_function
FROM      table
[WHERE    condition]
[GROUP BY group_by_expression]
[HAVING   group_condition]
[ORDER BY column];
```

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Użycie HAVING

```
SELECT   department_id, MAX(salary)
FROM     employees
GROUP BY department_id
HAVING   MAX(salary)>10000 ;
```

Typy połączeń

- **Połączenie wewnętrzne**
  - Połaczenie równowartościowe,
    - Połączenie naturalne
  - Połączenie krzyżowe/Iloczyn kartezjański
- **Połączenie zewnętrzne**
  - Lewostronne
  - Prawostronne
  - Pełne
- **Połączenie zwrotne**

Złączenia w składni SQL:1999

```
SELECT   table1.column, table2.column
FROM     table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
  ON (table1.column_name = table2.column_name)]|
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)]|
[CROSS JOIN table2];
```

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Połączenia równowartościowe

```
SELECT   table1.column, table2.column
FROM     table1, table2
WHERE    table1.column_name = table2.column_name;
```

```
SELECT employees.employee_id, employees.department_id,
       departments.department_name
FROM   employees, departments
WHERE  employees.department_id = departments.department_id;
```

Połączenie naturalne (ang. *Natural Join*)

{ T1 } NATURAL JOIN { T2 }

- Połączenie jest dokonywane „**automatycznie**”;
- Dla NATURAL JOIN warunek równości **dotyczy wszystkich par atrybutów** o takich samych **nazwach**;

Połączenia równowartościowe z wykorzystaniem klauzuli ON

{ T1 } JOIN {T2} ON *search condition*

- Klauzula **JOIN** … **ON** pozwala określić **nadrzędny warunek połączenia** lub **wyspecyfikować kolumny** wg. których złączenie ma być wykonane.

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Klauzula USING

{ T1 } JOIN { T2 } USING (join column list)

- Jeżeli wiele atrybutów posiada **identyczną nazwę**, **ale różne dziedziny**, klauzula **NATURAL JOIN** może zostać zastąpiona klauzulą **USING** w celu wyspecyfikowania kolumn użytych w warunku połączenia;

Połączenia - Przykłady

```
SELECT   table1.column, table2.column
FROM     table1
NATURAL JOIN table2
-----------------------------------------------------
SELECT   table1.column, table2.column
FROM     table1
JOIN table2 USING (column_names)
-----------------------------------------------------
SELECT   table1.column, table2.column
FROM     table1
JOIN table2 ON (table1.column=table2.column)
-----------------------------------------------------
SELECT   table1.column, table2.column
FROM     table1, table2
WHERE    table1.column_name = table2.column_name;
```

Na slajdzie przedstawiono równoważne zapytania.

Złączenie zwrotne

- Ogólna **składnia** połączenia zwrotnego jest taka sama, jak każdego innego typu połączenia omawianego poprzednio. Jedyną różnicą jest tutaj **podanie tej samej nazwy relacji po obu stronach operatora definiującego połączenie**.

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Połączenia nie równowartościowe (2)

Możliwe operatory połączenia:

- nierówne (<>),
- mniejszy niż (<),
- większy niż (>),
- mniejszy lub równy (<=),
- większy lub równy (> =),
- LIKE,
- IN,
- BETWEEN … AND …

Połączenia zewnętrzne w składni SQL

```
SELECT   table1.column, table2.column
FROM     table1
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)];
```

Operatory zbiorowe

- **Składnia:**

```
SELECT command
{UNION | INTERSECT | MINUS}
SELECT command
```

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

UNION Operator (2)

- Załóżmy, że mamy dwie tabele:
  - T1 ma wiersze 1, 2 oraz 3
  - T2 ma wiersze 2, 3 oraz 4

- W wyniku usunięte zostały zduplikowane wiersze 2 oraz 3.

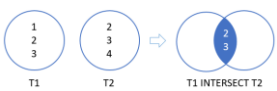
UNION ALL Operator (2)

- Załóżmy, że mamy dwie tabele:
  - T1 ma wiersze 1, 2 oraz 3
  - T2 ma wiersze 2, 3 oraz 4

- W wyniku znajdują się unikalne oraz zduplikowane wiersze 2 oraz 3.

INTERSECT Operator

- Załóżmy, że mamy dwie tabele:
  - T1 ma wiersze 1, 2 oraz 3
  - T2 ma wiersze 2, 3 oraz 4



Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

MINUS Operator

- Załóżmy, że mamy dwie tabele:
  - T1 ma wiersze 1, 2 oraz 3
  - T2 ma wiersze 2, 3 oraz 4

Składnia podzapytania

```
SELECT    select_list
FROM      table
WHERE     expr operator
                    (SELECT       select_list
                    FROM          table);
```

Przebieg wykonywania:

1. Podzapytanie/zapytanie wewnętrzne (ang. *subquery/inner query*) wykonywane jest jednokrotnie przed zapytaniem głównym/zewnętrznym (ang. *main query/outer query*).
2. Dostarczony wynik podzapytania (zależny od danych) jest wykorzystywany do wykonania zapytania zewnętrznego.

Wykorzystanie podzapytań

```
SELECT last_name
FROM   employees
WHERE  salary >
              (SELECT salary
               FROM   employees
               WHERE  last_name = 'Abel');
```

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Operatory dla podzapytań

```
SELECT   select_list
FROM     table
WHERE    expr operator
                    (SELECT       select_list
                     FROM         table);
```

Operator
=
>
>=
<
<=
<>

Operator
IN
ANY [SOME]
ALL

Podzapytanie skorelowane

- Podzapytanie posiada referencje do kolumn tabeli zapytania nadrzędnego.

```
SELECT column1, column2, ...
FROM   table1   outer
WHERE  column1,… operator
                    (SELECT   column1,…
                     FROM     table2
                     WHERE    expr1 =
                                 outer.expr2);
```

- Można wykorzystywać dowolny operator

Podzapytanie skorelowane - Przykład

```
SELECT e.employee_id, last_name,e.job_id
FROM   employees e
WHERE  2 <= (SELECT COUNT(*)
             FROM   job_history
             WHERE  employee_id = e.employee_id);
```

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Składnia INSERT

- Dodanie nowego rekordu do tabeli:

```
INSERT INTO   table [(column [, column...])]
VALUES        (value [, value...]);
```

- **Powyższa składnia pozwala na dodanie tylko jednego wiersza;**

```
INSERT INTO   table [(column [, column...])]
VALUES        (( DEFAULT )| NULL ), [, value...]);
```

- Możemy wprowadzić wartość **NULL** / **DEFAULT**;

Wstawienie nowego rekordu

- Specyfikujemy jawnie kolumny i wstawiane wartości w klauzuli INSERT.

```
INSERT INTO departments(department_id,
       department_name, manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700);
1 row created.
```

- Dodanie nowego rekordu zawierającego wartość dla każdego atrybutu.

```
INSERT INTO departments
VALUES (70, 'Public Relations', 100, 1700);
1 row created.
```

Składnia UPDATE

- Modyfikowanie istniejących rekordów z klauzulą UPDATE:

```
UPDATE        table
SET           column = value [, column = value, ...]
[WHERE        condition];
```

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Aktualizacja krotek w relacji

- Wskaż odpowiednie rekordy w klauzuli WHERE:

```
UPDATE employees
SET    department_id = 30
WHERE department_id = 60;
2 rows updated.
```

```
UPDATE   copy_emp
SET      department_id = 30;
22 rows updated.
```

Klauzula DELETE

- Możemy usuwać krotki z relacji odpowiednio wykorzystując klauzulę DELETE:

```
DELETE [FROM]   table
[WHERE          condition];
```

Usuwanie krotek z relacji

- Wyspecyfikuj rekordy w klauzuli WHERE:

```
DELETE FROM departments
WHERE  department_name = 'Finance';
1 row deleted.
```

```
DELETE FROM departments
WHERE  department_id IN (30, 40);
2 rows deleted.
```

```
DELETE FROM   copy_emp;
22 rows deleted.
```

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Transakcja w bazie danych - definicje

Definicja 1:

Zbiór operacji na bazie danych, które stanowią w istocie pewną całość i jako takie powinny być wykonane wszystkie lub żadna z nich.

Definicja 2:

Transakcja jest sekwencją logicznie powiązanych operacji na bazie danych, która przeprowadza bazę danych z jednego stanu spójnego w inny stan spójny.

Definicja 3:

Operacja lub ciąg operacji wykonany przez jednego użytkownika lub program aplikacji odwołujący się (czytający lub modyfikujący) do zawartości bazy danych.

Transakcja (1)

- **Typy operacji** na bazie danych obejmują:
  - odczyt i zapis danych
  - akceptację (zatwierdzenie) lub wycofanie transakcji
- **Transakcja** składa się zawsze z 3 etapów:
  - rozpoczęcia
  - wykonania
  - zamknięcia

Klasyfikacja transakcji

- **Ze względu na porządek operacji:**
  - transakcja sekwencyjna
  - transakcja współbieżna
- **Ze względu na zależność operacji:**
  - transakcja zależna od danych
  - transakcja niezależna od danych
- **Ze względu na typy operacji:**
  - zapytania lub transakcja odczytu (*read only*)
  - transakcja aktualizująca - transakcja (*read/write*)

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Przetwarzanie transakcyjne

Cele:

- Bazy danych powinny zapewniać **spójność** wykonywanych operacji;
- Transakcje pozwalają na większą **kontrolę** i elastyczność w przetwarzaniu danych;
- **Chronią** przed przypadkowymi błędami;

Transakcje

- Transakcje mogą być ***niejawne*** lub ***jawne***.
- ***Transakcje niejawne*** to takie, które odbywają się bez wiedzy użytkownika. W systemach, w których ustawione jest „*autozatwierdzanie*" (*AUTOCOMMIT*), **każde zapytanie stanowi oddzielną transakcję niejawną**.
- ***Transakcja jawna*** to taka, którą użytkownik sam definiuje, czyli **określa blok instrukcji**, jaki ma się wykonać w ramach jednej transakcji.

Zalety `COMMIT` i `ROLLBACK`

- Zapewnienie **spójności** danych;
- Możliwość wykonania „**podglądu**" danych przed zatwierdzeniem trwałych zmian;
- **Grupowanie** logicznie podobnych operacji;

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Własności ACID

- Dużym problemem jest również sytuacja, w której w trakcie trwania transakcji następuje **awaria systemu**.
- Aby w wyniku działania transakcji w bazie danych nie powstał „bałagan" oraz by wszystkie mogły się wykonać poprawnie, system musi zapewnić by **transakcje spełniały** tzw. **własności ACID**:
  - **A**tomicity - Atomowość
  - **C**onsistency - Spójność
  - **I**solation - Izolacja
  - **D**urability -Trwałość

Awarie SZBD (1)

Awarie, mogą być różnego typu:

- **awarie systemu** – wynikające z wad sprzętu lub błędów oprogramowania, wpływają na pamięć operacyjną,
- **awarie nośników** – powodują utratę części pamięci operacyjnej,
- **błąd oprogramowania aplikacji** – np. logiczne błędy w programie odwołującym się do bazy danych, powodujące awarię jednej lub kilku transakcji,
- **naturalne katastrofy** –pożar,
- **nieostrożność** - przypadkowe wyłączenie komputera,
- **sabotaż**.

Przykład - Transakcja

- **Transakcja przelewu kwoty N z konta A na konto B**:

```
begin
// odejmij kwotę N z konta A;
    update konta
    SET stan = stan - N
    where id_konta = A;
// dodaj do konta B kwotę N;
    update konta
    SET stan = stan + N
    where id_konta = B;
commit;
```

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.

Zatwierdzanie danych - commit

- Dokonanie zmian:

```
DELETE FROM employees
WHERE  employee_id = 99999;
1 rows deleted.

INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
1 rows inserted.
```

- Zatwierdzenie zmian:

```
COMMIT;
Commit complete.
```

Ta prezentacja jest chroniona prawami autorskimi. Rozpowszechnianie poza domeną pwr.edu.pl jest zabronione.