

Warsztat 4 - Bazy danych

1. Typy danych w bazie danych PostgreSQL

Name	Size	Range	Comment
boolean	1 bajt	false / true	przechowuje true lub false. Może być null
smallint	2 bajty	-32768 do 32767	dla małych zakresów
integer	4 bajty	-2147483648 do 2147483647	najczęściej stosowany
bigint	8 bajty	-9223372036854775 808 do 9223372036854775 807	int o dużym zakresie
real	4 bajty	dokładność 6 cyfr po dziesiętnych	the data types real and double precision are inexact, variable-precision numeric types - czyli nie przechowuje dokładnej wartości
numeric(p,s)	zmienna	Do 131072 cyfr przed przecinkiem oraz do 16383 cyfr po przecinku	p - łączna ilość cyfr s - wartość ilość miejsc po przecinku np. (5,2) - oznacza że 3 przed przecinkiem, 2 po przecinku np. 320,12

Typy tekstowe

char(n)	ciąg znaków o długości n. W przypadku niewykorzystania pełnego zakresu reszta zostanie wypełniona spacjami
varchar(n)	tak samo jak char, lecz nie dopełnia spacjami
text	tekst o zmiennej długości

Typy czasowe:

DATE	data
TIME	czas
TIMESTAMP	data i czas
TIMESTAMP WITH TIME ZONE	data i czas z uwzględnieniem strefy

Tworzenie tabeli - DDL - Data Definition Language

Do stworzenia bazy danych musimy wykorzystać zapytanie DDL, która tworzy strukturę tabeli, określając jakie pola mają się w nim znajdować, ich rodzaj oraz inne parametry.

```
CREATE TABLE XYZ (
  ID INT PRIMARY KEY NOT NULL,
  NAME VARCHAR(20) NOT NULL
);
```

W przypadku tabel zawsze ustawiamy na nich klucz główny (primary key)

Usuwanie tabeli - DROP TABLE nazwaTabeli.

Edytowanie struktury tabeli - ALTER TABLE X ALTER COLUMN

Do zmiany struktury tabelli używamy komendę Alter table np.

```
ALTER TABLE employees
ALTER COLUMN surname DROP NOT NULL;
```

Wkładanie danych do tabeli

Wkładanie danych do tabeli wykonujemy poprzez słowo kluczowe INSERT INTO np.

```
INSERT INTO employees(id, anme, surname, age) values(1, "Michał", "B:
```

Odczyt danych z tabeli - Select

Do odczytywania danych używamy operacji Select np.

```
SELECT * FROM employees;
```

Odczytywanie wyłącznie poszczególnych kolumn:

```
SELECT
    id,
    name,
    surname
FROM employees;
```

Alias

Mechanizm aliasów służy do tego, aby kolumny które są wyświetlane tylko w widoku miały inną nazwę, którą podaliśmy np.

```
SELECT
    id AS my_id,
    name AS my_name
    surname AS my_surname
FROM employees;
```

Where

Aby narzucić warunek do wyświetlanego zapytania możemy użyć klauzuli WHERE, np.

```
SELECT *
FROM employees
WHERE name = 'Michał';
```

Łączenie warunków - operatory logiczne OR/AND

Domemy używać operatorów do łączenia warunków np.

```
SELECT *  
  FROM employees  
 WHERE name = 'Michał'  
 AND surname='Bialek';
```

Operatory

W zapytaniach SQL możemy również używać operatory matematyczne takie jak: dodawanie, odejmowanie, mnożenie, dzielenie, modulo.

Dostępne są również operatory porównania takie jak:

1. równość (=)
2. nierówność (!=)
3. różność (<>)
4. mniejszość (<)
5. większość (>)
6. mniejszość lub równość (<=)
7. większość lub równość (>=).

Dostępne są operatory logiczne:

1. OR
2. AND
3. IN - sprawdza czy wartość w jest równa jednej z podanej wartości

```
SELECT * FROM employees WHERE name IN ('Andrzej', 'Michał');
```

4. LIKE - operator podobny do String.contains(). Znak % oznacza dowolny ciąg znaku.

```
SELECT * FROM employees WHERE name LIKE '%Mi';
```

5. BETWEEN - sprawdź czy wartość znajduje się w przedziale

```
SELECT * FROM employees WHERE age BETWEEN 20 AND 50;
```

6. NOT

7. IS NULL

Sortowanie zwracanego wyniku

```
SELECT *  
FROM employees  
ORDER BY age DESC;
```

Możemy również wykonać sortowanie po kilku kolumnach do przecinka podając kolejną kolumnę i tryb sortowania

```
ORDER BY age DESC, salary desc;
```

Ograniczanie zwracanego wyniku - LIMIT

Jeżeli chcemy ograniczyć ilość zwracanych wyników, możemy użyć słowa kluczowego limit np.

```
SELECT *  
FROM employees  
ORDER BY age ASC  
LIMIT 3;
```

Unikalne wartości - DISTINCT

Słowo kluczowe DISTINCT służy do zwracania unikatowych wyników np. imiona w firmie

```
SELECT DISTINCT name FROM employees;
```

Funkcje agregujące i grupowanie

Funkcje agregujące służą do zamiany elementów w zdefiniowaną wartość liczbową. Przykładowymi funkcjami agregującymi są:

1. COUNT
2. SUM
3. AVG
4. MIN
5. MAX

Przykładowo, chcemy podliczyć ilość występujących unikalnych imion, które zaczynają się na 'M' :

```
SELECT COUNT(DISTINCT name)
FROM employees
WHERE name LIKE 'M%';
```

GROUP BY

Funkcja ta działa podobnie jak grupowanie w streamach w javie. Jeżeli chcemy uzyskać strukturę:

klucz:lista_wartości. np.

```
SELECT age, COUNT(age)
FROM employees
GROUP BY age;
```

Aktualizowanie bazy danych - UPDATE, SET, WHERE

Przykład:

```
UPDATE employees
SET salary = 10000
WHERE name='Ania' AND surname='Bogata';
```

Usuwanie rekordu w bazie danych DELETE, FROM, . additional (WHERE)

Do usuwania rekordów w bazie danych służy operacja DELETE ex.

```
DELETE
FROM EMPLOYEES
WHERE ID = 70;
```

Relacyjność baz danych

1. Na czym polega relacyjność baz danych i jak ją stosować ?

Jeżeli chcemy przechowywać jakieś dane, to informację, które są/mogłyby być wykorzystywane w kilku miejscach, powinno się przechowywać w osobnych tabelach, ponieważ każda tabela powinna odpowiadać tylko za jedną rzecz. Następnie tabele te łączymy powiązaniem, aby uzyskać wymagane dane. Wiązanie odbywa się przy pomocy kluczy.

2. Wiązanie - Do wykonania wiązania używamy zapisu:

```
//Example database:
CREATE TABLE ADDRESSES(
  ID INT NOT NULL,
  ...
  PRIMARY KEY (ID)
);
CREATE TABLE EMPLOYEES(
  ID INT NOT NULL,
  ...
```

```
ADDRESS_ID INT NOT NULL,  
PRIMARY KEY (ID),  
  
//Zapis powiązania  
    CONSTRAINT fk_address  
        FOREIGN KEY (ADDRESS_ID)  
            REFERENCES ADDRESSES (ID)  
);
```

1. Wpisujemy ograniczenie (constraint) fk_adress
2. Klucz obcy (foreign key) odwołujemy się do pola w naszej tabeli, która będzie referencją do innej tabeli.
3. Referencja (referenes) - wskazujemy, gdzie powinien klucz obcy się odnosić (referować)

Uwagi

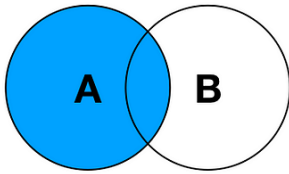
Jeżeli nasz klucz obcy jest not null, to nie można włożyć rekordu do tablicy, bez wcześniejszego utworzenia rekordu w tabeli do której się odwołujemy.

Klucze obce wymuszają na nas stosowanie pewnej kolejności dodawania i usuwania elementów z bazy danych.

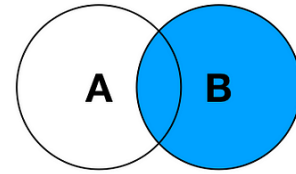
Joiny

Rozróżniamy kilka rodzajów joinów:

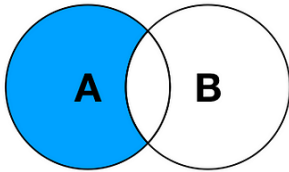
SQL JOINS



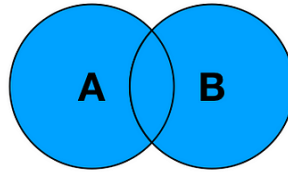
LEFT JOIN



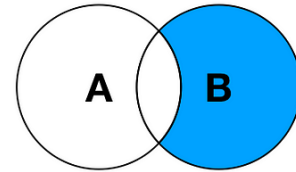
RIGHT JOIN



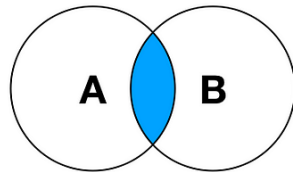
LEFT JOIN EXCLUDING
INNER JOIN



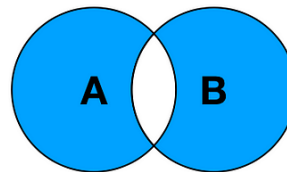
FULL OUTER JOIN



RIGHT JOIN EXCLUDING
INNER JOIN



INNER JOIN



FULL OUTER JOIN EXCLUDING
INNER JOIN

Do czego służą joiny ?

Joiny służą do łączenia tabel przy wykorzystaniu kluczy obcych.

Uwaga - przy joinowaniu mamy id tabeli pierwszej, oraz następnych tabel, dlatego odnosząc się do id musimy określić, do id z jakiej tabeli się odwołujemy. np. poprzez employees.id - przykład:

```
SELECT emp.id, name, surname, age, salary, city, street // Jakiego
FROM employees AS emp // określamy z jakiej tablicy
INNER JOIN address adr ON emp.address_id = adr.id // jaką
// +on + klucz obcy z tabeli = id z tabeli
```

Podstawowe Joiny

1. Inner join (inaczej zwykły join / przecięcie zbiorów) - bierzemy te, gdzie jest dokładne dopasowanie do siebie, czyli gdzie rekord z lewej odpowiada prawemu i vice versa.
2. Full join - zwracamy rekordy z obu tabel, nawet jeżeli ani lewa, ani prawa strona nie ma dowiązania do siebie.
3. Left join - zwrócimy wszystkie elementy z lewej strony, nawet jeżeli nie mają dowiązania z prawej, a z prawej wyświetlimy tylko te, co mają dowiązania do lewej.
4. Right join - odwrotna sytuacja do left joina.

Przykłady

1. Inner Join (przecięcie)

```
SELECT id,name,seurname,salary,city,postal_code,street
FROM employees as emp
INNER JOIN adress adr on emp.adress_id = adr.id;
```

2. Full Join (wszystko)

```
SELECT id,name,seurname,salary,city,postal_code,street
FROM employees as emp
FULL JOIN adress adr on emp.adress_id = adr.id;
```

3. Left

```
SELECT id,name,seurname,salary,city,postal_code,street
FROM employees as emp
LEFT JOIN adress adr on emp.adress_id = adr.id;
```

4. Right

```
SELECT id,name,seurname,salary,city,postal_code,street
FROM employees as emp
```

```
RIGHT JOIN address adr on emp.address_id = adr.id;
```

Resources: nieinformatyk

1. DBMS - Database Management System, czyli system zarządzania bazami danych, gdzie oprócz przechowywania, dodawania oraz wyszukiwania elementów w bazie, posiada również funkcjonalności optymalizacyjne, tworzenia kopii zapasowych itp.
2. Rodzaje baz danych - główne odzielenie polega na bazy relacyjne SQL, oraz nierelacyjne bazy danych noSQL. Bazy danych noSQL dzielimy na różne rodzaje np. kary key-value, column-familiy, graph, document. każda baza danych ma swojego dostawcę (providera) i w przypadku relacyjnych ba danych mogą to być: PostgreSQL, MySQL, MSsql server, Oracle database. każdy provider ma swoją implementację, która różni się swoją specyfikacją. Natomiast dla noSQL są to mongoDB, amazon dybamoDB, redis itp.
3. Co to jest SQL (Structural Query Language) - język wykorzystywany do operacji na tabelach relacyjnych.

Elementy baz danych

1. Serwer bazy danych - jest to urządzenie, na którym funkcjonuje baza danych. Jeżeli jest to produkcyjna baza danych, to serwer który powinien być dostępny 24/7 i serwować treści dla klientów. Rozróżniamy podkategorie jak rozproszona baza danych, gdzie kilka urządzeń współpracuje ze sobą tworząc logiczną całość, bazy danych w chmurze, albo rozwiązania hybrydowe.
2. Oprogramowanie bazy danych - program zarządzające plikami bazodanowymi (DBMS). Są one wersjonowane.
3. Instancja bazy danych - Na instancję składa się pamięć RAM oraz ROM. Dane przechowywane w pamięci trwałej są przenoszone do pamięci wewnętrznej. Za przenoszenie odpowiadają procesy pracujące w tle. Instancja DB jest przypadkowa, jeżeli chcemy wiedzieć o szczegółach działania bazy danych.

Czym kierować się przy wyborze implementacji/drivera bazy db.

1. Przy wyborze konkretnej implementacji najważniejsze są kryterium funkcjonalności, możliwość łączenia danej implementacji z innymi narzędziami, finanse, znajomość przez pracowników danej bazy danych.

Co to jest normalizacja baz danych

1. Normalizacja jest to podział dużych tabel na mniejsze. Robimy to ze względu to, że brak normalizacji może powodować błędy :
 - a. anomalia wstawiania- przy dużej tabeli nie idzie wstawić np. nowego działu firmy, ponieważ tabela wymaga również podania innych not null danych
 - b. anomalia usuwania - usuwając 1 rekord możemy doprowadzić do sytuacji, gdzie zanika kompletnie informacja o innej tabeli, np. jeżeli mamy tylko 1 informatyka i go usuniemy, to usuniemy wszystkie informacje na temat istnienia działu IT w firmie.
 - c. anomalia modyfikacji, anomalia czytania, które mogą doprowadzić do niespójności danych.

W przypadku normalizacji ważna jest wiedza jakie są powiązania. np. co jest do czego powiązane:

Postać nie normalna:

IMIE	NAZWISKO	PESEL	ADRES	DZIAŁ	KIEROWNIK DZIAŁU	PENSJA
Magda	Kowalska	78052000653	ul. Kwiatowa 13/16 02-200 Wieluń	Sprzedaż	Wojciechowski Jan	4800
Jakub	Nowak	89122431862	ul. Aksamićna 115 17-682 Łuków	Sprzedaż	Wojciechowski Jan	5500
Paweł	Iksiński	98090500695	ul. Wojskowa 1/15 78-963 Katowice	IT	Renata Kadłubek	6000

1. Pierwsza postać normalna:

ID	IMIE	NAZWISKO	PESEL	ULICA	NUMER BUDYNKU	NUMER MIESZKANIA	KOD_POCZTOWY	MIEJSCOWOŚĆ	DZIAŁ	KIEROWNIK DZIAŁU	PENSJA
1	Magda	Kowalska	78052000653	Kwiatowa	13	16	02-200	Wieluń	Sprzedaż	Wojciechowski Jan	4800
2	Jakub	Nowak	89122431862	Aksamićna	115		17-682	Łuków	Sprzedaż	Wojciechowski Jan	5500
3	Paweł	Iksiński	98090500695	Wojskowa	1	15	78-963	Katowice	IT	Renata Kadłubek	6000

1. Tabela posiada ID

2. Wszystkie kolumny są jedno wartościowe - np. adres nie jest przechowywany pod jednym kolumną, lecz jest rozdzielony na ulicę, numer domu, mieszkania, miejscowość itp.
3. Druga postać normalna

ID_PRACOWNIKA	IMIE	NAZWISKO	PESEL	ULICA	NUMER BUDYNKU	NUMER MIESZKANIA	KOD_POCZTOWY	MIEJSCOWOŚĆ	PENSJA	ID_DZIAŁU
1	Magda	Kowalska	78052000653	Kwiatowa	13	16	02-200	Wieluń	4800	1
2	Jakub	Nowak	89122431862	Aksamitna	115		17-682	Łuków	5500	1
3	Paweł	Iksiński	98090500695	Wojskowa	1	15	78-963	Katowice	6000	2
ID_DZIAŁU	NAZWA_DZIAŁU	KIEROWNIK_DZIAŁU								
1	Sprzedaż	Wojciechowski Jan								
2	IT	Renata Kadłubek								

1. Wszystkie kolumny nie kluczowe zależą od klucza głównego
4. Trzecia postać normalna

Tranzakcja

Tranzakcyjność jest to funkcjonalność, która jest wymagana do realizacji potrzeb biznesowych aplikacji. Uwzględnia ona zasadę kolejowania/dostępności do danych. Poprawnie zaimplementowana tranzakcyjność powinna zachowywać zasady ACID, czyli Atomicity, Consistency, Isolation, Durability. W zależności od stopnia izolacji transakcji możemy spodziewać się różnych negatywnych efektów współbieżności transakcji. Są to np. dirty read, phantom read, non-repeatable read.

Relacyjność bazy danych

Relacja to tabela. Co to jest relacyjność baz danych definiuje 12 zasad Edgara Codd'a.

1. Dane są przechowywane w tabeli.
2. Dostęp do danych uzyskujemy poprzez podanie tabeli, kolumny i klucza głównego
3. Istnieje wartość NULL
4. W bazie danych istnieją metadane

5. **Istnieje technologia zapytań SQL**
6. **Widoki pozwalają na operacjach DML (Data manipulation language)**
7. Oprócz czytania można też modyfikować dane
8. Fizyczna niezależność danych
9. Logiczna niezależność danych
10. **Dostępne więzy spójności (not null, fk, unique)**
11. Niezależność dystrybucyjna
12. Nie idzie ominąć reguł DB operacjami niższego poziomu.

Tworzenie bazy danych

1. Infrastruktura bazy danych (sprzęt, sieć, OS)
2. Specyfika przetwarzania (baza rozproszona/scentralizowana, OLTP/OLAP ?)
3. Implementacja RDBMS - Wybór silnika bazodanowego, instalacja i konfiguracja
4. programy BI, systemy raportujące itp.
5. Architektura i kod (tworzenie aplikacji i optymalizacji)

Modelowanie/Projektowanie bazy danych

Potrzebna jest wiedza na temat danych oraz ich połączeń, w celu doboru odpowiedniej dla nas bazy danych. Projektowanie to składa się z 3 etapów:

1. Tworzenie modelu koncepcyjnego (analiza biznesu, związku i zależności między danymi, czyli głównie biznes)
2. Tworzenie modelu logicznego (znajomość modelu relacyjnego, diagram związków encji - np. ERD, oraz normalizacja)
3. Tworzenie modelu fizycznego - transformacja na kod SQL, tworzenie kolumn technicznych itp.
4. Narzędzia przydatne do tworzenia map:
 - a. Developer Data Modeler

b. Draw.io

c. StarUML

Funkcje analityczne i funkcje agregujące i porównanie

1. Zastosowanie funkcji analitycznych jest często bardziej optymalne, ponieważ w porównaniu do funkcji agregującej, dane nie są grupowane

Jak działa WITH w sql → CTE (Common Table Expression)

Jest to bardziej złożone zapytanie SELECT. Klauzula WITH pomaga

Shopping database project