Postgres funkcionalnosti

Pristup podacima iz programskog koda 25/26

Ishod učenja 1

Sadržaj

Strukturirani jezik upita - SQL

(1) DDL (Data Definition Language)

`CREATE`, `DROP`, `ALTER`

(2) DML (Data Modifying Language)

CRUD operacije (`INSERT`, `SELECT`, `UPDATE`, `DELETE`)

filtracija (`WHERE`) i sortiranje (`ORDER BY`)

agregatne funkcije (`AVG`, `COUNT`, `SUM`, `MIN`, `MAX`) s grupiranjem (`GROUP BY`)

SQL funkcije

```
CREATE OR REPLACE FUNCTION species_category(sp_name TEXT) RETURNS TEXT
AS $$

BEGIN

    IF sp_name ILIKE 'a%' THEN RETURN 'Group A';

    ELSIF sp_name ILIKE 'b%' THEN  RETURN 'Group B';

    ELSE RETURN 'Other';

    END IF;

END;

$$ LANGUAGE plpgsql IMMUTABLE;

-- Funkcije mogu biti IMMUTABLE, STABLE i VOLATILE (ovisno o kontekstu korištenja)
```

Podatkovni tipovi

Za izradu ORM-a, potrebno je izvesti mapiranje osnovnih Postgres podatkovnih tipova na programske podatkovne tipove, npr.

+ `varchar(255)` ⇒ `string`
+ `bigint` ⇒ `long`
+ `Timestamp [without time zone]` ⇒ `DateTime`
+ …

Dobro proučiti podatkovne tipove dostupne u Postgresu:

+ https://www.postgresql.org/docs/current/datatype.html

Postgres sekvenca

Objekt u bazi podataka koji osigurava autoinkrementalno generiranje nove vrijednosti (najčešće za primarni ključ).

Funkcije:

`nextval('seq')`
→ inkrementira i vraća sljedeću vrijednost

`currval('seq')`
→ posljednja vrijednost korištena u ovoj sesiji

`setval('seq', N)`
→ postavlja vrijednost sekvence na N

Od PG 10+, identity stupci (`GENERATED ALWAYS AS IDENTITY`) se preferiraju nad `SERIAL` tipovima.

Postgres sekvenca

Kreirana implicitno kada se koristi `SERIAL` podatkovni tip ili eksplicitno putem `CREATE SEQUENCE`

+ u pozadini je sadržan brojač u kataloškoj relaciji koji je pohranjen u `pg_sequence` sistemskom katalogu
+ jednom kada je vrijednost inkrementirana, ista je izgubljena čak i ako je transkacija poništena (eng. *rollback*) – osigurava jedinstvenost

Indeksi

Indeksi se koriste kada želimo poboljšati performanse upita pretrage i/ili sortiranja podataka iz tablice. U pozadini se stvara **B-stablo** struktura podataka (binarno stablo pretrage) koja omogućuje binarno petraživanje u vremenskoj složenosti ~`O(logn)`.

Default ponašanje je linearna pretraga (eng. *sequential scan*).

```
CREATE INDEX name ON table [USING <algorithm>] (column, <…>)
```

+ **B-Tree**
  Podržani operatori: `<`, `<=`, `=`, `>=`, `>`
+ **GIN**
  full-text search / JSONB /arrays
+ **GiST**
  geometric, full-text search
+ **BRIN**
  veliki sekvencijski podaci (primjer*: time-series*)
+ **Hash**
  usporedbe jednakosti po *hash* vrijednosti

Vrste indeksa

pojedinačni (**single index**)

klasičan indeks na jednom stupcu

parcijalni (**partial index**)

samo redovi koji zadovoljavaju dani uvjet će biti indeksirani (npr. `WHERE TYPE = 'A'`)

composite (**single index**)

indeks na više stupaca, ukoliko spora pretraga/sortiranje zahtijeva više stupaca

funkcijski(**functional index**)

podaci će biti indeksirani u ovisnosti o rezultatu funkcije (npr.
`CREATE INDEX ON lower(name))`

Problemi s indeksima

Umetanje i brisanje !!

`animal` tablica sadrži podatke o taksonomski određenjima različitih životinjskih vrsta (kingdom, phlyum, class, order, familiy, genus, species) te sadrži sljedeći indeks:

```
CREATE INDEX animal_canonical_name_idx ON animal(canonical_name)
```

Koji upiti će koristit novostvoreni indeks? (hint: koristite EXPLAIN)

```
(1)  SELECT * FROM animal WHERE canonical_name = 'bird';
(2)  SELECT * FROM animal WHERE canonical_name = 'a%';
(3)  SELECT * FROM animal WHERE canonical_name = '%a';
(4)  SELECT * FROM animal WHERE lower(canonical_name) = 'bird';
```

Transakcije

Transakcija je logička jedinica rada koja izvršava jedan ili više SQL upita kao jedinstvenu atomičku operaciju, gdje će se ili sve stavke izvršiti (`commit`) ili nijedna (`rollback`)

Primjer:

```
BEGIN;

-- operacije…

COMMIT;
```

Moguće je poništiti operacije, ukoliko dođe do pogreške - `ROLLBACK;`

Možemo stvoriti savepointe i resetirati stanje do specifičnog savepointa koristeći njegovo ime, npr. u tijelu transakcije:

```
SAVEPOINT point1;
```

te potom reset:

```
ROLLBACK point1;
```

Problemi s transakcijama

(1) Zaboravite li izvršiti `COMMIT`, transakcija zadržava lockove i blokira druge transakcije

(2) Transakcije koje se izvode duže vremena sprječavaju VACUUM proces koji oslobađa zauzeti prostor mrtvih n-torki

```
SELECT
     pid, usename, state, query, xact_start
FROM pg_stat_activity
WHERE state = 'active';
```

(3) Dvije transakcije koje čekaju oslobođenje locka ove druge - **deadlocks**, PostgreSQL detektira takvo stanje i odbacuje jednu transakciju

```
SELECT * FROM pg_locks;
```

Razine izolacije transakcija

Razina	Opis	Rješava problem
READ UNCOMMITTED	Ponaša se kao READ COMMITTED (Postgres ne dozvoljava dirty reads)	∅
READ COMMITTED (default)	Vidi samo one podatke koji su committed nakon svakog iskaza	sprječava prljava čitanja (eng. dirty reads)
REPEATABLE READ	Vidi snapshot od početka izvođenja transkacije	sprječava neponavljauća čitanja (eng. nonrepeatable reads)
SERIALIZABLE	Potpuno izvršavanje transakcija u seriji: najsigurnije i najsporije	sprječava sve anomalije

Usklađenost s ACID-om

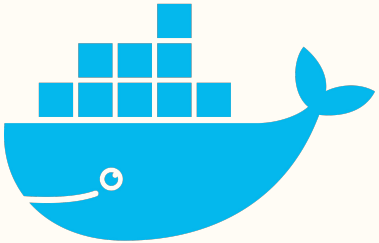
Svojstvo	Značenje	Postgres implementacija
Atomicity	Sve naredbe u transkaciji se izvrše ili se nijedna ne izvrši	putem COMMIT / ROLLBACK
Consistency	Baza podataka radi tranziciju iz jednog stanja u drugo	putem ograničenja, okidača i transkacijskog integriteta
Isolation	Paralelne transakcije ne utječu jedna na drugu	kroz MVCC (Multi-Version Concurrency Control)
Durability	Jednom kad su podaci zapisani, možemo računati da su spremljeni na disku bez obzira na nepredviđene događaje	implementirano kroz WAL (Write-Ahead Logging)

Postavljanje Postgresa

Kao što je spomenuto na prošlim vježbama, postavit ćemo Postgres kontejner putem Docker alata!

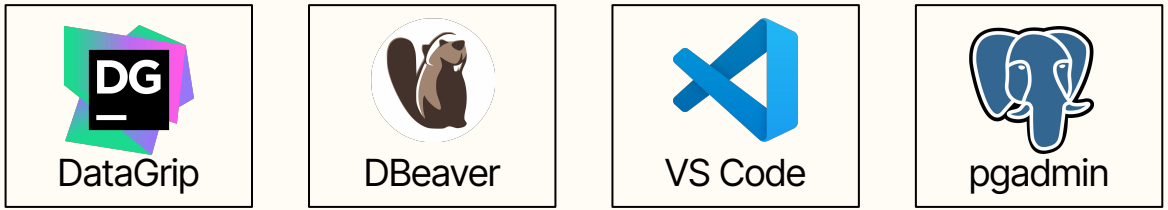
Koristit ćemo sljedeću jednostavnu naredbu:

```
docker run -d \
    --name postgres-local \
    -e POSTGRES_PASSWORD=mysecretpassword \
    postgres:18
```



Alati

Možete koristiti koji god alat želite kako biste ostvarili komunikaciju s bazom podataka i izvršavali SQL upite:



Vjerojatno je najlakše koristiti komandno-linijski alat `psql` unutar Postgres kontejnera:

```
docker run -it postgres-local psql -U postgres
```

Taj alat funkcionira kao REPL (*read-evaluation-print loop*) program, koji prvo traži upit, potom ispiše rezultat i onda ponovno traži upit.

Primjer 1

Istražite barem ove navedene sistemske tablice:

+ `pg_database`
+ `pg_stat_database`
+ `pg_stats`
+ `pg_stat_user_tables`
+ `pg_stat_activity`
+ `information_schema.tables`

Primjer 2

Izlistajte sve baze podataka i njihov sadržaj koristeći sljedeće tablice:

```
+  pg_database
    -  oid
    -  datname
+  pg_stat_database
    -  xact_commit
    -  xact_rollback
```

Primjer 3

Koristeći PPPK-Vjezbe-01-Seed.sql datoteku koja se nalazi na IE kako biste postavili podatke u bazi.

Dohvatite sljedeće podatke:

a) top 5 studenata po prosječnim brojem bodova
b) najpopularniji ispiti (po broju prijava)
c) postotak prolaznosti

Primjer 4

Umetnite podatke o dva studenta unutar transakcije, ali napravite pogrešku prilikom drugog umetanja (npr. dvostruki primarni ključ) i pokažite da se transakcija poništila (eng. *rollback*).

Sirova (eng. *raw*) konekcija

Koristit ćete je u prvom projektu, ukoliko želite implementirati vlastito ORM rješenje

U primjerima na satu koristit će se `Npgsql` - biblioteka za C# programski jezik

Primjer 5

Koristeći sirovu konekciju, napravite sljedeće primjere:

A - Ubacite novog studenta i dodijeljeni ID

B - Ažurirajte bodove i status studentove prijave ispita

C - Koristeći transakciju, obrišite jednog studenta i njegove prijave ispita

D - Korištenjem sirove konekcije (`C# + Npgsql`), dohvatite podatke o studentima i mapirajte ih na instance programske klase `Student`