

# Postgres arhitektura

Pristup podacima iz programskog koda  
25/26

Ishod učenja 1



# Teme

Nakon uvodnog predavanja, trebali ste kreirati račun na Supabase i istražiti osnovne funkcionalnosti kao što je povezivanje na bazu podataka i izvršavanje osnovnih upita u web sučelju

Projektni zadatak je diskutiran i objašnjen na uvodnom predavanju i formalno definiran u dokumentu objavljenom na IE

Za sva moguća pitanja stojim na raspolaganju putem poznatih medija komunikacije

# Svrha ovog materijala

Prve dvije vježbe (čitaj predavanja) osvrtnat će se na postavke Postgres baze podataka, pozadinu implementacijskih detalja kao i ponavljanje bitnih koncepata koji su obrađeni na prethodnim kolegijima:

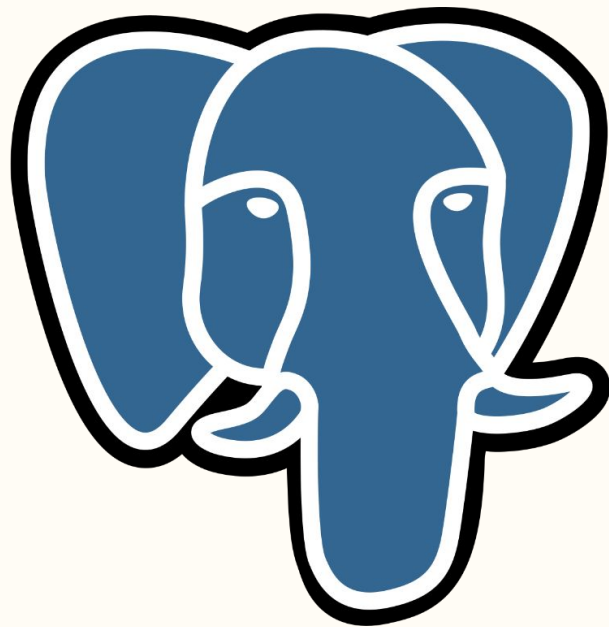
- + UBP
- + OBP

## Napomena!

Uz implementaciju programskog rješenja za prvi projekt, za projektnu obranu potrebno je razumijeti kako Postgres funkcionira "ispod haube" (naravno, temeljeno na onom što ćemo obraditi na nastavnim satima)

# Uvod u Postgres

- + Relacijska baza podataka otvorenog koda
  - pouzdana (eng. reliable) ?
  - širok ekosustav šarolikih ekstenzija
- + PostgreSQL dijalekt
  - ponešto drugačiji naspram MSSQL (T-SQL)
- + Najnovija stabilna verzija: 18
  - 2025-09-25



# Korištenje Postgresa

Pokazana će biti dva primjera postavljanja Postgres instance:

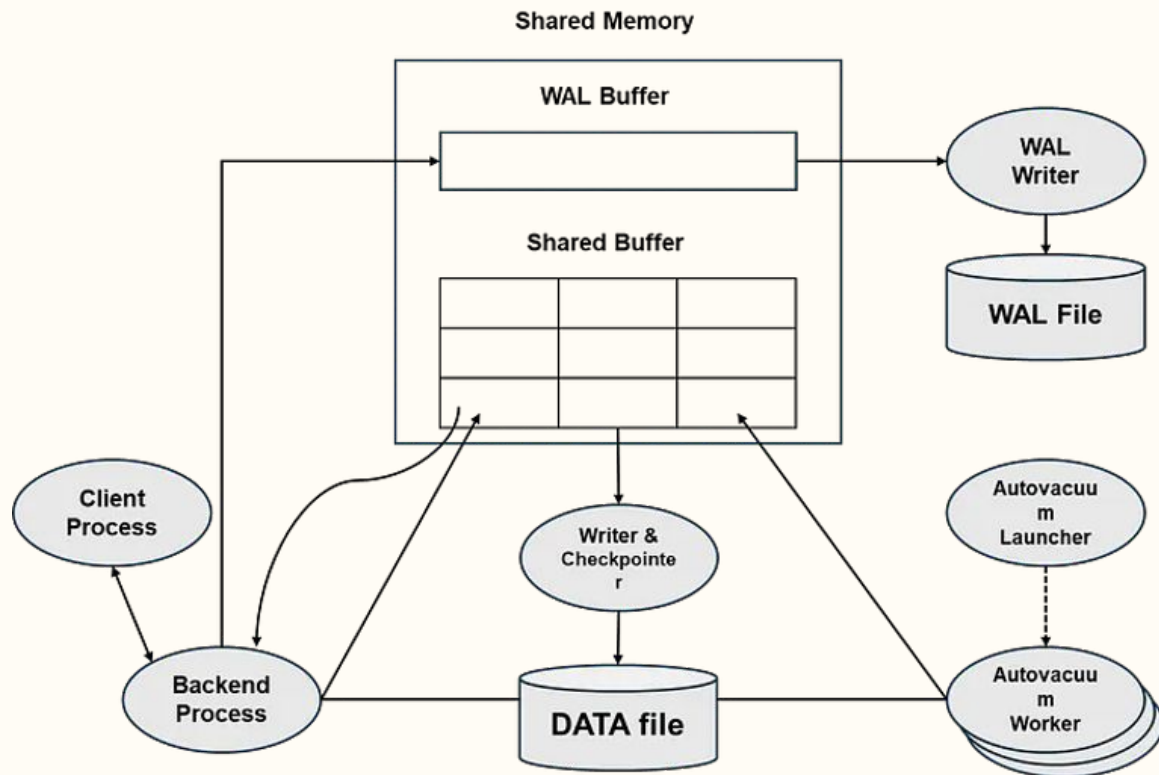
(1) **Supabase cloud**

- + besplatna, ali jako ograničena
- + dostupna udaljena instanca

(2) **Docker kontejneri !**

- + fleksibilno postavljanje instance lokalno na računalo (ili na udaljenom serveru)
- + naglasak na ovu metodu zbog raspostranjenosti korištenja Docker kontejnera

# Postgres arhitektura



# Postgres procesi

Postgres instanca sadrži četiri glavna tipa procesa:

**Postmaster**  
proces



Klijentski proces

Backend proces

Pozadinski  
proces(i)

Podsjetimo se što su uopće procesi!

# Postgres arhitektura

- + Postmaster (postgres)
  - Glavni serverski proces koji upravlja klijentskim procesima djece
- + Proces klijentska konekcije (Backend process)
  - Svaka klijentska konekcija dobije svoj vlastiti backend process (fork Postmaster procesa)

Pozadinski procesi (ključni za razumijevanje pozadine Postgresa):

WAL Writer → zapisuje stavke u Write Ahead Log

Background Writer → zapisuje dirty stranice na disk (eng. flush)



# Procesi

svaka klijentska  
konekcija dobije  
svoj backend  
proces

klijentski procesi

Postmaster (postgres)



glavni serverski  
proces koji upravlja  
klijentskim  
konekcijama

pozadinski procesi

ostali pozadinski procesi koji  
upravljaju resursima Postgres  
instance

# Pozadinski procesi

WAL writer

Zapisuje stavke u Write Ahead Log

Background writer

Zapisuje *dirty* stranice na disk (eng. *flush*)

Checkpointner

Zadržava integritet podataka tako što ih povremeno zapisuje na disk

Autovacuum launcher

Upravlja čišćenjem i uklanjanjem nekorisćenih n-torki

Archiver

Statistics collector

# Write Ahead Logging

- + izdržljivost baze podataka! (ACID)

Iz dokumentacije:

*WAL's central concept is that changes to data files (where tables and indexes reside) must be written only after those changes have been logged, that is, after WAL records describing the changes have been flushed to permanent storage. If we follow this procedure, we do not need to flush data pages to disk on every transaction commit, because we know that in the event of a crash we will be able to recover the database using the log: any changes that have not been applied to the data pages can be redone from the WAL records. (This is roll-forward recovery, also known as REDO.)*

- + reducira broj zapisa na disk i omogućuje playback svih izvršenih operacija u slučaju sistemske pogreške!

# Write Ahead Logging

Prilikom checkpointa, sav dirty buffer se piše na disk i čiste se stavke iz WAL-a

→ događa se periodički, moguće je postaviti putem `checkpoint_timeout` i `max_wal_size` te može biti ručno okinuto

→ moguće je postaviti `synchronous_commit` i `commit_delay`

Informacije o operacijama moguće je pronaći u sistemskim tablicama:

- + `pg_stat_bgwriter`
- + `pg_stat_wal`

# Vacuum

Prilikom izvršavanja operacije promijene ili brisanja podataka, oni bivaju označeni kao izmijenjeni dok i dalje zauzimaju mjesto na disku

→ tzv. mrtve *n-torke* (eng. *dead tuples*) su i dalje prisutne u pohrani

Vacuum proces periodički prolazi kroz datoteke (koje sadrže podatke iz tablica) te zaista oslobađa memoriju primjenjujući prethodno navedene promjene

→ analogno *soft delete* mehanizmu u web aplikacijama

Informacije se mogu pronaći u sistemskoj tablici `pg_stat_user_tables`

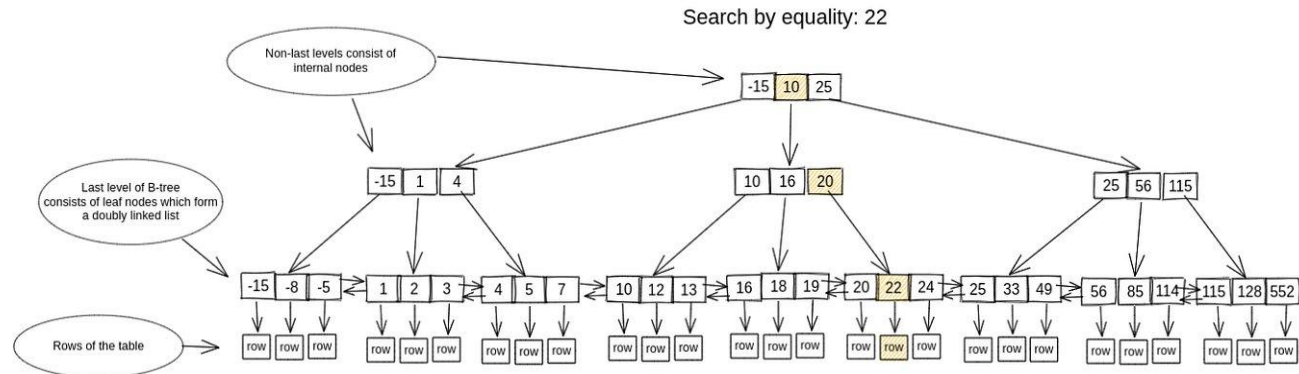
- + `last_autovacuum`
- + `last_vacuum`

# B-stabla

Lehman-Yao high-concurrency **B-Tree** algorithm

Odabir B-stabala kao pozadinske strukture pretrage ima svoje prednosti:

- ⇒ Performanse upita o jednakosti (eng. *equality*) i rasponu (eng. *range*)
- ⇒ Performanse algoritama sortiranja
- ⇒ Konzistentcija podataka



# B-stabla

**Ključevi** predstavljaju indeksirane vrijednosti, organizirane rastuće po vrijednostima u svakom čvoru

**Pokazivači** su poveznice prema čvorovima djece ili do podataka koji su sadržani u listovima (eng. leaf nodes)

Binarno pretraživanje je izvedeno na ključevima svakog čvora (nešto više o indeksima sljedeći put!)

```
CREATE INDEX index_name ON table_name (column_name);
```

# Konfiguracijske datoteke

## postgresql.conf

glavna konfiguracijska datoteka koje definira parametre za upravljanje memorijom, konekcijama, zapisima (eng. logs), WAL, upitima (shared\_buffers, work\_mem, max\_connections, logging\_collector)

## pg\_hba.conf

upravlja klijentskom autentikacijom navodeći koji se korisnici spiju spojiti s kojeg izvora, kojom metodom (md5, scram, trust, peer, etc.)

## pg\_ident.conf

mapira sistemske korisnike na PostgreSQL korisnike (autentikacijske metode poput ident.)



# Sistemske baze podataka na instanci

Odmah nakon izvršavanja `initdb` naredbe, stvorena su dva tablična prostora:

- + **pg\_default**  
smješten na `$PGDATA\base`
- + **pg\_global**  
smješten na `$PGDATA\global`

Napravljene su tri sistemske baze:



postgres

template0

template1

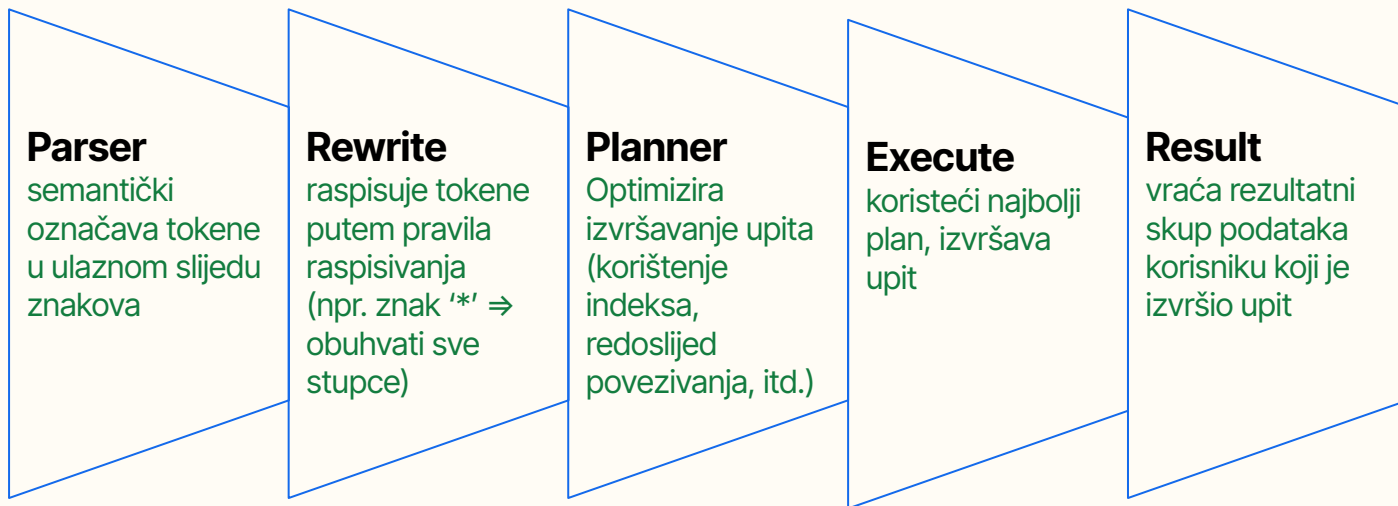
# Tablice

Svaka tablica je asocirana s tri različite datoteke na disku:

- (1) jedna koja sadrži tablične podatke, nosi ime kao i tablični OID
- (2) jedna za upravljanje slobodnim prostom, nazvana `OID_fsm`
- (3) Jedna za upravljanje vidljivošću tabličnih blokova, nazvana `OID_vm`

Indeksi nemaju vm datoteku, stoga sadrže samo dvije datoteke: `OID` i `OID_fsm`

# Primjer izvršavanja upita



# Primjer planiranja

```
CREATE TABLE animal (  
    id SERIAL PRIMARY  
    KEY,  
    heat_control TEXT  
);
```

```
INSERT INTO animal (heat_control)  
  
SELECT CASE  
    WHEN random() < 0.75 THEN 'endotherm'  
    ELSE 'ectotherm'  
  
END  
  
FROM generate_series(1, 100000);
```

# Planiranje izvršavanja

Koristimo ključnu riječ **EXPLAIN** kako bi vidjeli plan koji je stvorio execution planner:

```
EXPLAIN SELECT * FROM animal WHERE heat_control = 'ectotherm';
```

Postgres planer će ispisti plan izvršavanj upita (primjerice, koristi li indekse).

# Statistike

**ANALYZE** će generirati statistiku podataka u `pg_stats`

Možemo napraviti uvid u generiranu statistiku putem upita u sistemsku tablicu:

```
SELECT
    attname, n_distinct, most_common_vals, most_common_freqs
FROM pg_stats
WHERE tablename = 'animal';
```

# Postavljanje baze podataka

Najlakše je RDBMS (kao i ostale service) postaviti lokalno putem Dockera! Podizanje Postgres kontejnera zahtjeva instaliran Docker alat:

- (1) Na Windowsu – WSL i sve ostalo kako je opisano vodičem
- (2) Na Linux/MacOS - nativna instalacija

Detalji o Postgres imageu i svemu ostalom: [https://hub.docker.com/\\_/postgres](https://hub.docker.com/_/postgres)

Koristit ćemo sljedeću jednostavnu naredbu:

```
docker run -d \  
  --name example-01 \  
  -e POSTGRES_PASSWORD=mysecretpassword \  
  postgres:18
```

