ALGEBRA

Pristup podacima iz programskog koda

Vježbe 01 – Postgres arhitektura

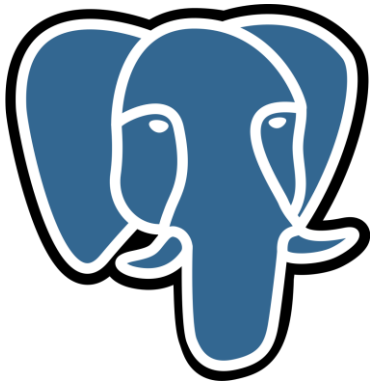
Podsjetnik

- Nakon uvodnog predavanja, trebali ste kreirati račun na Supabase i istražiti osnovne funkcionalnosti kao što je povezivanje na bazu podataka i izvršavanje osnovih 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

Uvod u Postgres

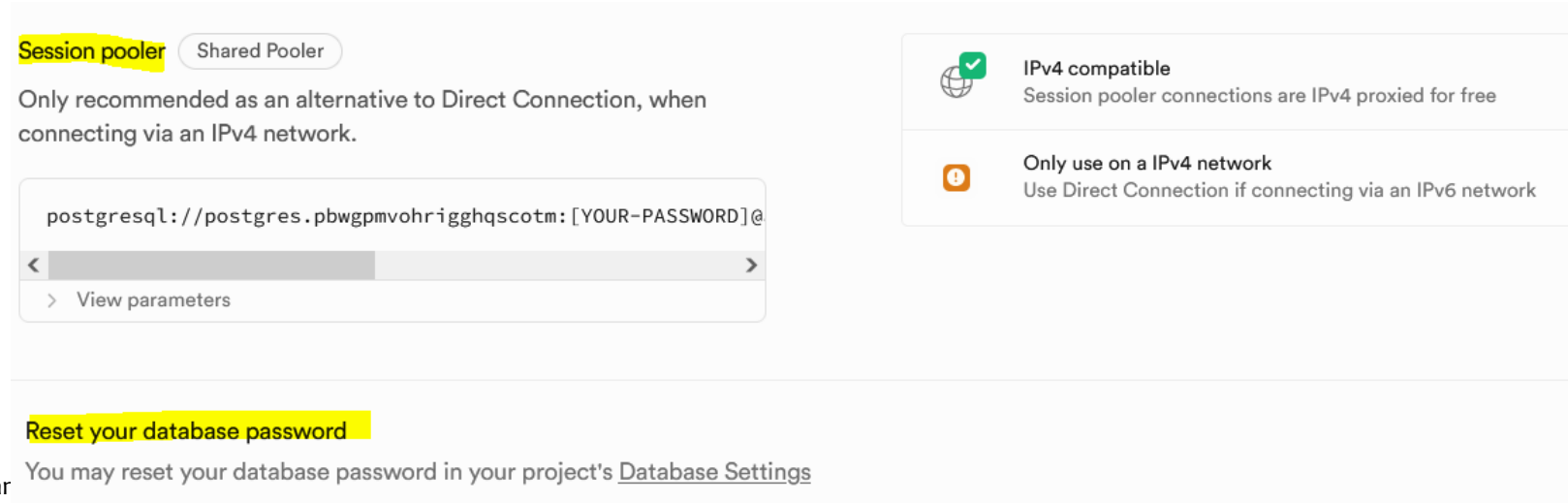
- Relacijska baza podataka otvorenog koda
  - Pouzdana (eng. *reliable*) ?
  - Širok ekosustav šarolikih ekstenzija
- PostgreSQL dijalekt SQL standarda !
  - Ponešto drugačiji spram MSSQL (T-SQL)
- Najnovija verzija: **18**
  - 2025-09-25



ALGEBRA

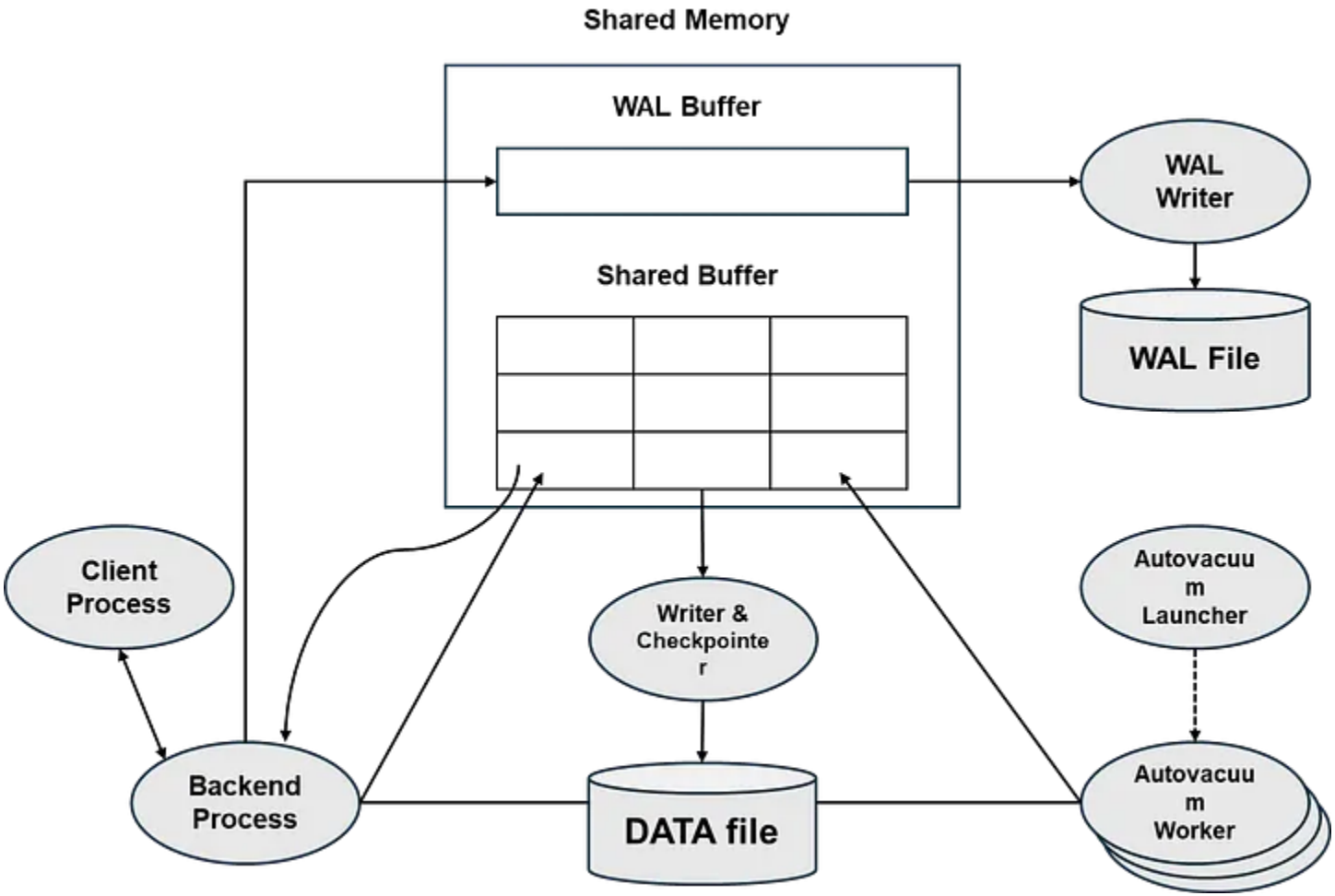
Postavljanje Postgres instance u oblaku

- Pratite korake opisane na vježbama kako biste postavili bazu na Supabase servisu (isti će biti dostupni i putem IE)
- Prilikom spajanja, koristite `Session Pooler` i postavite (resetirajte) šifru `postgres` administratorskog korisnika
  - Iskoristite podatke iz detalja konekcije kako biste se spojili na bazu podataka iz alata po izboru (Dbeaver, DataGrip – osobna preporuka, VS Code putem ekstenzija, itd.)



ALGEBRA

Postgres arhitektura



ALGEBRA

Postgres procesi

- Postgres instanca sadrži četiri glavna tipa procesa:
  - **Postmaster (Daemon) Process**
  - **Pozadinski proces**
  - Backend proces
  - Client process

  - Podsjetimo se što su uopće procesi!

Postgres arhitektura

- Postmaster (postgres)
  - Glavni serverski proces koji upravlja klijentskim *child* procesima
- Proces klijentske 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*)
  - Checkpointer → zadržava integritet podataka tako što ih povremeno zapisuje u disk
  - **Autovacuum Launcher** → upravlja čišćenjem i uklanjanjem nekorištenih n-torki
  - Archiver (ukoliko je postavljeno arhiviranje)
  - Statistics Collector (prati korištenje tablica i stupaca)

WAL

- **Write Ahead Logging**
  - Integritet podataka !
  - *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**

ALGEBRA

WAL

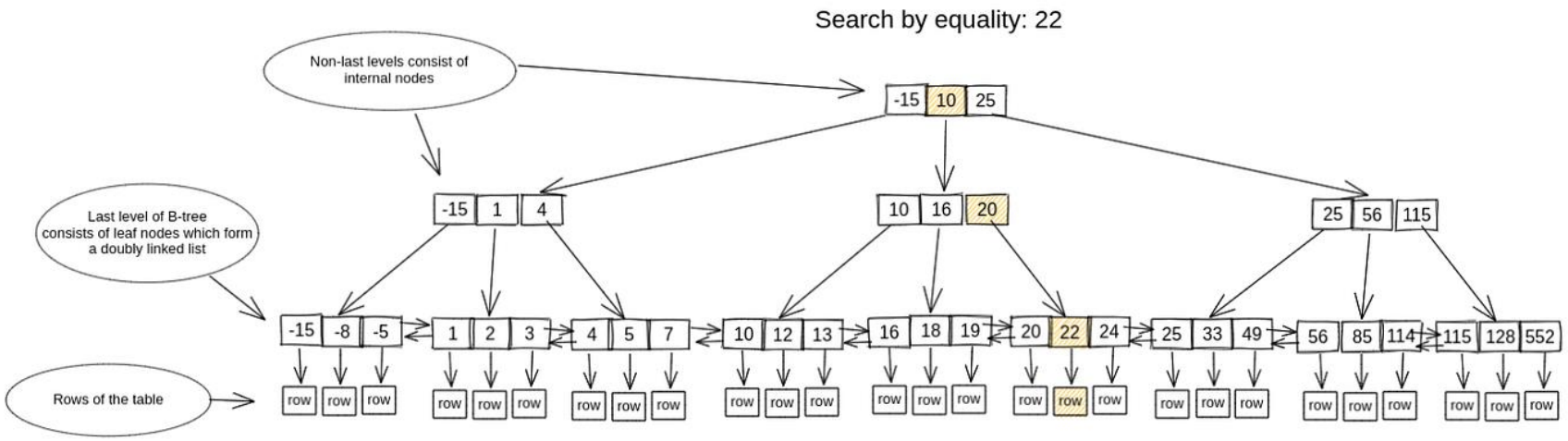
- Prilikom **checkpointa**, sav dirty buffer se piše na disk i čiste se stavke iz WAL-a
  - događa se periodički, moguće je postaviti putem `checkpoint_timeout` i `max_wal_size` te može biti ručno okinuto
  - moguće je postaviti `synchronous_commit` i `commit_delay`
- Informacije o operacijama moguće 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
  - Takozvane 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
  - Analogija s "soft delete" mehanizmom 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 ima svoje prednosti:
  - Performanse upita o jednakosti (*equality*) i rasponu (*range*) q
  - Performanse algoritama sortiranja
  - Konzistencija podataka



B-Stablo

- **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`
  - Main configuration Glavna konfiguracijska datoteka koje definira prametre 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`.)

Inicijalne baze podataka na instanci

- Nakon izvršavanja `initdb` naredbe, tri baze su stvorene: `template0`, `template1`, i `postgres`
  - Baze korištene putem SaaS (software as a service) ili rješenja u oblaku mogu sadržavati i još neke baze prilikom inicijalizacije
- `template0` i `template1` su predlošci za stvaranje korisničkih baza podataka te sadrže tablice sistemskog kataloga
- Odmah nakon izvršavanja `initdb` naredbe, dva tablična prostora su stvorena: `pg_default` i `pg_global`
  - `pg_default` smješten na **$PGDATA\base**
  - `pg_global` smješten na **$PGDATA\global**

Tablice

- Svaka tablica je asocirana s tri različite datoteke na disku:
  - Jedna koja sadrži tablične podatke, nosi ime kao i tablični OID
  - Jedna za upravljanje slobodnim prostorm, nazvana `OID_fsm`
  - 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

```
SELECT * FROM users WHERE id = 10;
```

- **Parser**
  - Semantički označava tokene u ulaznom slijedu znakova
- **Rewrite**
  - Raspisuje tokene putem pravila raspisivanja (* - obuhvati sve stupce, `users` ukoliko je view)
- **Query Planner:**
  - Sequential scan (ukoliko je tablica mala i upit ne pogađa indeks*)
  - Index scan
- **Execute**
  - Koristeći najbolji plan, izvršava upit
- **Result**
  - Vraća rezultatni skup podataka korisniku koji je izvršio upit

Primjeri

```
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);
```

Execution planner

- ANALYZE će generirati statistiku podataka u pg_stats
- Koristimo ključnu riječ EXPLAIN kako bi vidjeli plan koji je stvorio *execution planner*:

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

- 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’;
```

U sljedećoj epizodi…

- Docker postavka instance !
- Još interesantnih Postgres funkcionalnosti
- Spajanje i pristup podacima u Postgres instanci koristeći programski kod (primjeri u C# - Npgsql)