|  | **Univerzitet u Nišu, Elektronski fakultet**  **Katedra za računarstvo** |  |
| --- | --- | --- |

**Mihajlo Veljković**

**Particionisanje podataka u PostgreSQL DBMS-u**

**Seminarski rad**

**Niš, 2023**

Sadržaj

[Uvod 3](#_Toc132727549)

[Particionisanje podataka 4](#_Toc132727550)

[Vertikalno particionisanje 4](#_Toc132727551)

[Horizontalno particionisanje 5](#_Toc132727552)

[Particionisanje u PostgreSQL DBMS-u 6](#_Toc132727553)

[Metode particionisanja 6](#_Toc132727554)

[Particionisanje po opsegu 7](#_Toc132727555)

[Particionisanje po listama 7](#_Toc132727556)

[Particionisanje po hash vrednostima 7](#_Toc132727557)

[Komponovanje metoda 8](#_Toc132727558)

[Održavanje particija 8](#_Toc132727559)

[Indeksiranje 9](#_Toc132727560)

[Orezivanje particija 9](#_Toc132727561)

[Particionisanje pomoću nasleđivanja 11](#_Toc132727562)

[Mane particionisanja 12](#_Toc132727563)

[pg\_partman ekstenzija 12](#_Toc132727564)

[Instalacija 13](#_Toc132727565)

[Kreiranje particija 13](#_Toc132727566)

[Upoređivanje performansi particionisanja 14](#_Toc132727567)

[Unos podataka 14](#_Toc132727568)

[Upiti 14](#_Toc132727569)

[Zaključak 16](#_Toc132727570)

[Literatura 17](#_Toc132727571)

# Uvod

Uz veliku potrebu za brzim razvojem aplikacija, potrebno je obratiti pažnju pri dizajniranju i implementaciji istih. Vrlo česta pojava je brzi razvoj prototipova aplikacija, a o performansama i kvalitetu koda se razmišlja tek kasnije, pa se gomila tehnički dug koji kasnije može praviti ogromne probleme i zastoje.

Dizajniranje relacionog modela podataka takođe zahteva veliku pažnju pri dizajniranju, nije uvek dovoljno skicirati šemu i prevesti model u SQL naredbe odabranog DBMS-a. Postoje optimizacije koje su moguće i pre odabira DBMS-a, a postoje i specifične optimizacije za odabrani DBMS, pa i to može uticati na izbor okruženja.

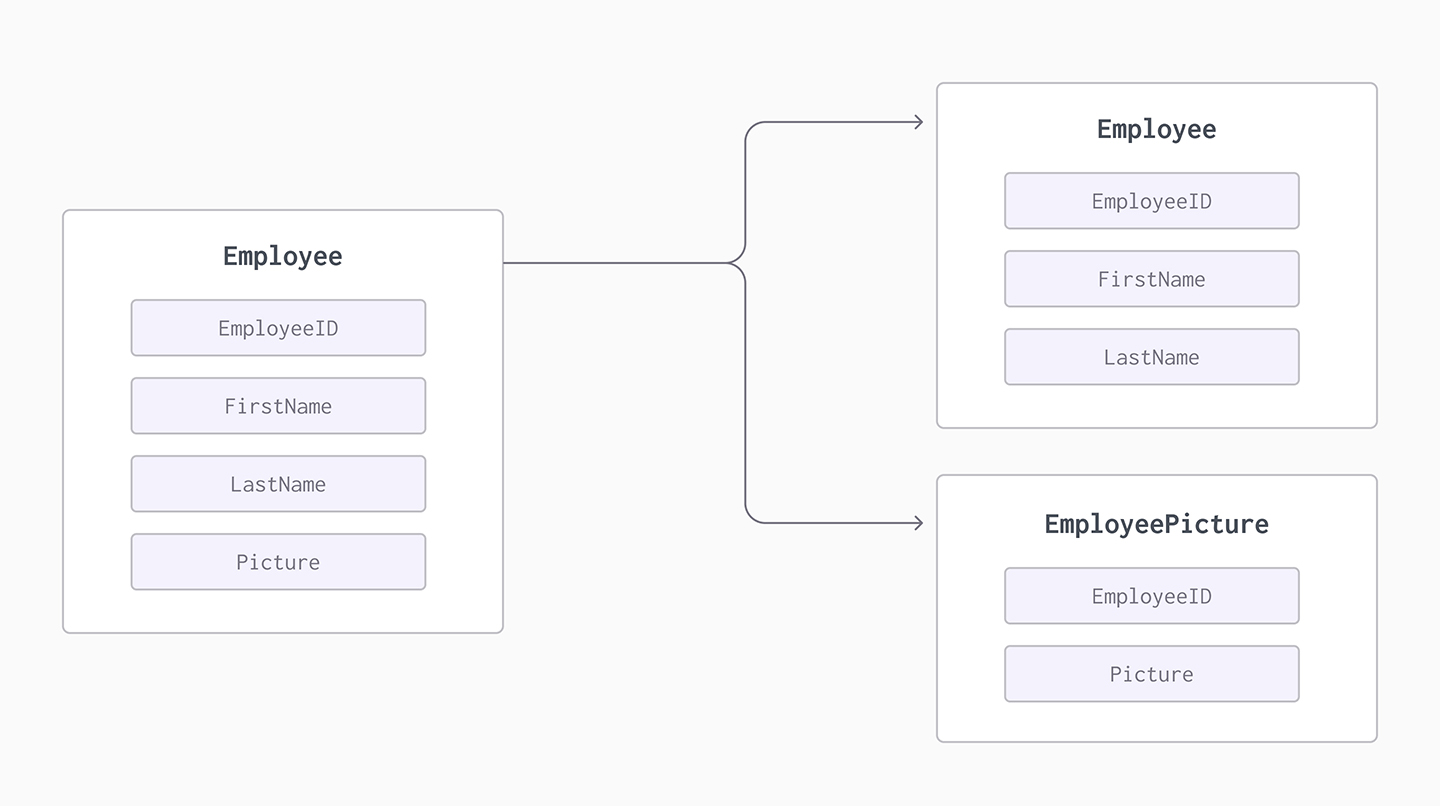
U ovom radu biće opisan koncept particionisanja podataka, prednosti i mane tog pristupa optimizacije, način na koji se particionišu podaci u PostgreSQL DBMS-u kao i neki praktični primeri.

# Particionisanje podataka

Particionisanje podataka kod relacionih baza podataka podrazumeva deljenje tabele u zavisnosti od željene optimizacije. Moguće je uraditi vertikalno particionisanje i horizontalno particionisanje.

## Vertikalno particionisanje

Vertikalno particionisanje je najčešće kod tabela koje imaju mnogo kolona, a nisu svi podaci iz kolone uvek potrebni pri svakom upitu. Jedan primer kandidata vertikalnog particionisanja su npr. detalji korisnika. Ako u bazi čuvamo razne informacije o korisniku, uključujući i sliku ili neki drugi blob objekat, a pri upitima vrlo retko ili samo jednom učitavamo tu sliku korisnika, onda ima smisla particionisati tabelu na dve tabele: jednu sa ključem i slikom i drugu sa svim ostalim informacijama koje se kasnije mogu koristiti za druge upite i veze sa drugim tabelama. Originalnu tabelu možemo “vratiti” pomoću pogleda, gde se gubi malo na performansama, ali time se održava kompatibilnost, a sve nove upite nad particionisanim tabelama će imati brži odziv.



Slika 1.1 Primer vertikalnog particionisanja

## Horizontalno particionisanje

Horizontalno particionisanje može imati mnogo veći uticaj na performanse u zavisnosti od tipa tabele i podataka. Tabele za koje se očekuje da imaju bar po 2 GB podataka treba uzeti u obzir za neku vrstu optimizacije, najčešće particionisanje. Kod horizontalnog particionisanja podataka, dobri kandidati su tabele koje mogu logički da se podele na particije. Takođe, tabele koje imaju neku vrstu agregacije podataka (npr. po danu, mesecu, godini) su dobri kandidati za particionisanje ili tabele sa podacima koji mogu da se grupišu u celine (npr. imena po abecednom redosledu). Još jedan razlog za particije može da bude podela po geografskim regionima, čime se dobijaju i manje tabele, a moguće je i fizički instancirati server bliže nekom regionu, pa time dobiti i brži vremenski odziv za te podatke. Primer horizontalnog particionisanja može da bude podela tabele koja u sebi ima neku kolonu u vremenu, recimo datum prodaje artikla. Moguće je particionisati tabelu po mesecima i na taj način imati brzu pretragu svih prodatih artikla za dati mesec:

Tabela prodatih artikala za celu godinu

Particionisana tabela po mesecima

Slika 2.1 Horizontalno particionisanje

Još jedna prednost ove metode je mogućnost arhiviranja podataka bez ometanja funkcionalnosti baze podataka. Sa postojanjem particija, moguće je lako odabrati nepotrebne particije i odlučiti sta uraditi sa njima (u mnogim slučajevim se to vrši automatski), dok kada bi imali jednu veliku tabelu sa svim podacima, bilo bi sporo brisati neportebne podataka, a i uvek bi postojala mogućnost za neku grešku (unet pogrešan datum, izostavljen WHERE deo upita za brisanje, itd.).

# Particionisanje u PostgreSQL DBMS-u

PostgreSQL DMBS od verzije 10 podržava particionasanje podataka bez dodatnih podešavanja i ekstenzija, tzv. *native* ili deklarativno particionisanje. Pre toga su se koristili okidači uz pomoć ekstenzija od kojih je najpoznatija pg\_partman.

Kreiranje tabele sa particijama zahteva dodatne korake od početka, tj. nije moguće dodati podeliti već postojeću tabelu. Moguće je dodati već postojeću tabelu kao particiju nove table, ili napraviti novu, nezavisnu tabelu od particije. Takođe je moguće napraviti stranu tabelu kao particiju. Za strane tabele u PGSQL-u se podrazumeva da nisu fizički prisutne unutar DBMS-a već da se podaci prikupljaju iz drugih izvora (drugi DBMS, druga instanca PGSQL-a, podaci sa interneta ...).

## Metode particionisanja

Tabelu je moguće podeliti na sledeće načine:

* Po opsegu – tabela je podeljena na opsege vrednosti, bilo to brojevi, datumi ili neki drugi tip podataka. Za opseg particije, donja vrednost se smatra inkluzivnom, dok gornja ne. Particija sa opsegom 1..10 u sebi neće sadržati vrednost 10.
* Po listama – tabela je podeljena tako da particije sadrže samo vrednosti specificirane pri kreiranju particije.
* Po hash vrednosti – tabela je podeljenja na particije pomoću moduo operacija i ostataka rezultata.

Moguće je i komponovati particije na takav način da sadrže više metoda, pa se time omogućava detaljnjija podela podataka.

Kao primer različitih načina particionisanja, uzećemo tabelu o prodatim artiklima.

CREATE TABLE sales (

    id int,

    product text,

    amount int,

    price int,

    sale\_date date,

    store\_id int

)

Osim ključa, u tabeli imamo ime proizvoda, količinu, cenu, datum prodaje i ključ prodavnice u kojoj je izvršena prodaja.

### Particionisanje po opsegu

U primeru prodavnice, podela po opsegu vrednosti bi bila idealna za datum prodaje. Da bi napravili tabelu, potrebno je definiciji tabele dodati ključne reči PARTITION BY i metodu particionisanja. Za opseg je ključna reč RANGE i kolona po kojoj vršimo podelu.

PARTITION BY RANGE (sale\_date);

Da bi kreirali particije, pravimo nove tabele koje su definisane kao particije.

CREATE TABLE sales\_2023\_jan PARTITION OF sales FOR VALUES FROM ('2023-01-01') TO ('2023-02-01');

Dodavanje podataka izvršavamo kao da particije ne postoje, pomoću INSERT upita:

INSERT INTO sales VALUES (1,'Some product',1,132,'2023-01-16',1);

Čitanje podataka možemo da vršimo i iz tabele *sales*, ali i iz bilo kojih particija koje napravimo (npr. *sales\_2023\_jan*).

### Particionisanje po listama

Ovu metodu možemo iskoristiti za pretragu po proizvodima. Recimo da želimo particiju u kojoj će se nalaziti samo prodaje koje su se desile na određenim proizvodima (npr. proizvodima te prodavnice). Ključna reč je LIST i kolona po kojoj se deli.

PARTITION BY LIST (product);

Particija se definiše pomoću liste vrednosti.

CREATE TABLE our\_product\_sales PARTITION OF sales FOR VALUES IN ('Our product 1', 'Our product 2', 'Our product 3');

### Particionisanje po hash vrednostima

U slučaju da imamo ukupno 5 prodavnica, možemo podeliti tabelu tako da svaka prodavnica ima svoju particiju. Moduo operacija nad kolonom *store\_id* i brojem 5 osiguraće da svaki red pripada odgovarajućoj prodavnici. Ključna reč za ovu metodu je HASH.

PARTITION BY HASH (store\_id);

Za svaku particiju treba definisati moduo i ostatak.

CREATE TABLE store\_1\_sales PARTITION OF sales FOR VALUES WITH (MODULUS 5, REMAINDER 1);

U slučaju da vrednost u kolonu *store\_id* pređe 5, vrednosti će ići u tabelu za prvu prodavnicu, pa je potrebno ponovno kreirati particije u slučaju otvaranja novih prodavnica.

### Komponovanje metoda

Ako bi hteli da imamo bolju organizaciju podataka, moguće je komponovati particije na primeru prodavnice. Postupak komponovanja je takav da na već postojeće particije samo kreiramo nove particije dodatkom ključnih reči PARTITION BY i metode.

CREATE TABLE sales\_2023\_jan PARTITION OF sales FOR VALUES FROM ('2023-01-01') TO ('2023-02-01') PARTITION BY LIST (product);

Nije ograničen broj kombinacija, ali treba napomenuti da PostgreSQL nema velikih benefita što se tiče ubrzanja rada prilikom pravljenja pod-particija, a u nekim slučajevima može čak dovesti i do gubitka performansi ili čak i pada programa zbog velikog broja operacija nad tabelama u pozadini.

## Održavanje particija

Jedan od glavnih razloga za korišćenje particija je i lakše održavanje baze podataka. Tabele sa starim podacima uglavnom ne služe ničemu, a za njihovo arhiviranje se koriste drugi alati. Nema potrebe čuvati stare i nepotrebne podatke u DBMS-u. Pomoću particija brisanje tih starih podataka se izvršava lakše i brže. Najlakši način za brisanje starih podataka je utvrđivanje opsega tih podataka i nakon toga brisanje particija u kojima su ti podaci. U primeru prodavnice, ako želimo da obrišemo podatke za prethodnu godinu, sve što treba da uradimo je da obrišemo particije (na isti način kao brisanje tabela iz DMBS-a):

DROP TABLE sales\_2022\_jan, sales\_2022\_feb, ... sales\_2022\_dec;

Brisanje podataka na ovaj način je mnogo brže jer DBMS ne mora da prolazi kroz sve podatke.

Moguće je i osloboditi particiju od glavne tabele, pa nad njom vršiti sve potrebne operacije (npr. arhiviranje), a zatim je obrisati kao običnu tabelu:

ALTER TABLE sales DETACH PARTITION sales\_2022\_jan;

Dodavanje particija je već opisano u procesu pravljenja particija i moguće je dodati nove tabele u bilo kom trenutku nakon kreiranja roditeljske tabele.

Ponekad je lakše napraviti tabelu, popuniti je podacima, proveriti podatke pa tek je onda dodati kao deo glavne tabele. Najčešći način je korišćenje CREATE TABLE LIKE sintakse koja će efektivno kopirati strukturu tabele:

CREATE TABLE sales\_2023\_may (LIKE sales INCLUDING DEFAULTS INCLUDING CONSTRAINTS);

Možemo dodati provere za vrednosti tabele pre nego što asociramo particiju kako bi izbegli proces DBMS-a koji bi proverio celu tabelu za validnost podataka:

ALTER TABLE sales\_2023\_may ADD CONSTRAINT 2023\_may CHECK ( sale\_date >= DATE '2023-05-01' AND sale\_date < DATE '2023-06-01');

Na kraju asociramo particionisanu tabelu sa roditeljskom:

ALTER TABLE sales ATTACH PARTITION sales\_2023\_may

FOR VALUES FROM ('2023-05-01') TO ('2023-06-01');

### Indeksiranje

PostgreSQL omogućava jednostavno kreiranje indeksa za sve particije tabele koristeći standarnu sintaksu za indekse:

CREATE INDEX sales\_store\_id\_idx ON sales(store\_id);

Treba obratiti pažnju na to da ovom operacijom se zaključavaju sve tabele i indeksiraju svi redovi. Kako bi izbegli zaključavanje tabela, moguće je napraviti indeks samo na glavnoj tabeli pomoću ključnih reči ON ONLY:

CREATE INDEX sales\_store\_id\_idx ON ONLY sales(store\_id);

Na ovaj način, particije se indeksiraju nezavisno i eksplicitno, a moguće je i korišćenje ključne reči CONCURRENTLY koja ne zaključava tabele pri indeksiranju. Ovo je preporučena metoda za radna okruženja gde se očekuju minimalni prekidi rada. Svaki novi indeks je potrebno asocirati sa glavnom tabelom i nakon toga je indeks glavne tabele obeležen kao validan:

CREATE INDEX CONCURRENTLY sales\_store\_id\_2023\_may\_idx ON sales\_2023\_may (store\_id);

ALTER INDEX sales\_store\_id\_idx ATTACH PARTITION sales\_store\_id\_2023\_may\_idx;

Na ovaj način je moguće dodati i UNIQUE i PRIMARY KEY pravila nad već postojećim tabelama i particijama.

### Orezivanje particija

Opcija *enable\_partition\_pruning* omogućava optimizaciju pri upitima za sve particije kreirane deklarativno. Planer upita unapred izbacuje sve tabele koje nisu u opsegu traženih podataka i nije potrebna definicija indeksa, ograničenja ili bilo kojih dodatnih pravila. Možemo videti plan upita sa i bez optimizacija:

SET enable\_partition\_pruning = on;

EXPLAIN SELECT COUNT(\*) FROM sales WHERE sales.sale\_date >= DATE '2022-12-01';

"Aggregate  (cost=50.60..50.61 rows=1 width=8)"

"  ->  Append  (cost=0.00..48.90 rows=680 width=0)"

"        ->  Seq Scan on sales\_2022\_dec sales\_1  (cost=0.00..22.75 rows=340 width=0)"

"              Filter: (sale\_date >= '2022-12-01'::date)"

"        ->  Seq Scan on sales\_2023\_jan sales\_2  (cost=0.00..22.75 rows=340 width=0)"

"              Filter: (sale\_date >= '2022-12-01'::date)"

SET enable\_partition\_pruning = off;

EXPLAIN SELECT COUNT(\*) FROM sales WHERE sales.sale\_date >= DATE '2022-12-01';

"Aggregate  (cost=328.90..328.91 rows=1 width=8)"

"  ->  Append  (cost=0.00..317.85 rows=4420 width=0)"

"        ->  Seq Scan on sales\_2022\_jan sales\_1  (cost=0.00..22.75 rows=340 width=0)"

"              Filter: (sale\_date >= '2022-12-01'::date)"

"        ->  Seq Scan on sales\_2022\_feb sales\_2  (cost=0.00..22.75 rows=340 width=0)"

"              Filter: (sale\_date >= '2022-12-01'::date)"

"        ->  Seq Scan on sales\_2022\_mar sales\_3  (cost=0.00..22.75 rows=340 width=0)"

"              Filter: (sale\_date >= '2022-12-01'::date)"

"        ->  Seq Scan on sales\_2022\_apr sales\_4  (cost=0.00..22.75 rows=340 width=0)"

"              Filter: (sale\_date >= '2022-12-01'::date)"

"        ->  Seq Scan on sales\_2022\_may sales\_5  (cost=0.00..22.75 rows=340 width=0)"

"              Filter: (sale\_date >= '2022-12-01'::date)"

"        ->  Seq Scan on sales\_2022\_jun sales\_6  (cost=0.00..22.75 rows=340 width=0)"

"              Filter: (sale\_date >= '2022-12-01'::date)"

"        ->  Seq Scan on sales\_2022\_jul sales\_7  (cost=0.00..22.75 rows=340 width=0)"

"              Filter: (sale\_date >= '2022-12-01'::date)"

"        ->  Seq Scan on sales\_2022\_aug sales\_8  (cost=0.00..22.75 rows=340 width=0)"

"              Filter: (sale\_date >= '2022-12-01'::date)"

"        ->  Seq Scan on sales\_2022\_sep sales\_9  (cost=0.00..22.75 rows=340 width=0)"

"              Filter: (sale\_date >= '2022-12-01'::date)"

"        ->  Seq Scan on sales\_2022\_oct sales\_10  (cost=0.00..22.75 rows=340 width=0)"

"              Filter: (sale\_date >= '2022-12-01'::date)"

"        ->  Seq Scan on sales\_2022\_nov sales\_11  (cost=0.00..22.75 rows=340 width=0)"

"              Filter: (sale\_date >= '2022-12-01'::date)"

"        ->  Seq Scan on sales\_2022\_dec sales\_12  (cost=0.00..22.75 rows=340 width=0)"

"              Filter: (sale\_date >= '2022-12-01'::date)"

"        ->  Seq Scan on sales\_2023\_jan sales\_13  (cost=0.00..22.75 rows=340 width=0)"

"              Filter: (sale\_date >= '2022-12-01'::date)"

## Particionisanje pomoću nasleđivanja

Osim deklarativnog particionisanja, moguće je napraviti particije pomoću nasleđivanja. Ova metoda je korisna kada je potrebno imati dodatne kolone u particijima ili kada uslov podele nije pokriven jednom od metoda deklarativnog particionisanja. Kreiramo tabelu bez ikakvih ključnih reči o particionisanju, a za sve particije koristimo ključnu reč INHERITS i proveru za podatke:

CREATE TABLE sales\_2023\_may( CHECK ( sale\_date >= DATE '2023-05-01' AND sale\_date < DATE '2023-06-01' ) ) INHERITS(sales);

Potrebno je dodati okidač pri ubacivanju za pravilno raspoređivanje podataka:

CREATE OR REPLACE FUNCTION sales\_insert\_trigger()

RETURNS TRIGGER AS $$

BEGIN

    IF ( NEW.sale\_date >= DATE '2023-05-01' >= DATE '2006-02-01' AND

         NEW.sale\_date < DATE '2023-06-01' ) THEN

        INSERT INTO sales\_2023\_may VALUES (NEW.\*);

    ELSIF

    ...

    ELSE

        RAISE EXCEPTION 'Date out of range.  Update the trigger function sales\_insert\_trigger!';

    END IF;

    RETURN NULL;

END;

$$

LANGUAGE plpgsql;

CREATE TRIGGER insert\_sales\_trigger

    BEFORE INSERT ON sales

    FOR EACH ROW EXECUTE FUNCTION sales\_insert\_trigger();

Okidač je potrebno ažurirati pri dodavanju novih particija. Umesto okidača, moguće je napraviti pravilo pri unosu podataka:

CREATE RULE sales\_insert\_2023\_may AS

ON INSERT TO sales WHERE

    ( sale\_date >= DATE '2023-05-01' AND sale\_date < DATE '2023-06-01' )

DO INSTEAD

    INSERT INTO sales\_2023\_may VALUES (NEW.\*);

Korišćenje pravila ima veći uticaj na performanse od okidača, ali se obračunava jednom po upitu umesto jednom po redu, pa u nekim specifičnim slučajevima sa unosom velikog broja podataka odjednom može dati bolje performanse.

Za održavanje ovakvih particija, moguće je brisati tabele ili ih otkačiti pomoću ključnih reči NO INHERIT. Moguće je i asociranje već postojećih tabela pomoću ključne reči INHERIT.

Ovaj način particionisanja se koristio pre postojanja deklarativnog i postoje određene mane pri njegovom korišćenju:

* provere za raspodelu podataka pomoću CHECK naredbi mogu da pokriju više slučajeva i postoji veći potencijal za greške
* indeksi i strani ključevi se ne propagiraju nasleđenim tabelama
* ažuriranje podataka tako da red prelazi iz jedne particije u drugu nije moguće bez dodatnih okidača
* neke komande poput VACUUM ili ANALYZE moraju da se pokrenu sa svaku particiju pojedinačno
* upiti za unos podataka koji imaju ON CONFLICT klauzule neće funkcionisati jer se ta klauzula evaluira samo nad roditeljskom tabelom
* deklarativno particionisanje je brže zbog internih optimizacija

## Mane particionisanja

Treba uzeti u obzir da particionisianje nije uvek univerzalno rešenje za veliki broj podataka i da postoje određena ograničenja:

* Kolona po kojoj se vrši podela ne sme imati NULL vrednosti.
* Primarni ključ unutar particije može postojati samo ako ključ particionisanja ne koristi funkcije ili izraze i kolone ograničenja moraju uključiti i kolone koje služe za particionisanje. Ovaj uslov postoji jer ograničenje primarnog ključa može da obuhvati samo jednu particiju, pa je jedini način da se osigura jedinstvenost ključa to da druge particije ne mogu imati istu vrednost.
* Okidači za unos podataka ne mogu promeniti u koju particiju se upisuju podaci.
* Ako je tabela particionisanja trajna, onda i sve particije moraju biti trajne, i obrnuto.
* ORM alati nemaju uvek punu podršku za particije, ili imaju manje performanse nego korišćenje upita.

## pg\_partman ekstenzija

Za lakši rad sa particijama baziranih na vremenskim ili serijskim podelama, postoji ekstenzija *pg\_partman* koja omogućava automatsko kreiranje novih particija, automatsko brisanje starih i lakše upravljanje particijama. Ekstenzija takođe poseduje svoj proces koji u pozadini proverava stanje particija i održava ih po definisanim pravilima bez potrebe za cron poslovima.

### Instalacija

Pored *pg\_partman* ekstenzije, potrebna je i *pg\_jobmon* ekstenzija za mogućnost automatskog održavanja particija. Instalacija se vrši izvršavanjem sledećeg upita:

CREATE SCHEMA partman;

CREATE EXTENSION pg\_partman SCHEMA partman;

Preporučuje se i kreiranje posebnog korisnika:

CREATE ROLE partman WITH LOGIN;

GRANT ALL ON SCHEMA partman TO partman;

GRANT ALL ON ALL TABLES IN SCHEMA partman TO partman;

GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA partman TO partman;

GRANT EXECUTE ON ALL PROCEDURES IN SCHEMA partman TO partman;  -- PG11+ only

GRANT ALL ON SCHEMA my\_partition\_schema TO partman;

GRANT TEMPORARY ON DATABASE mydb to partman; -- allow creation of temp tables to move data out of default

### Kreiranje particija

Nakon kreiranja roditeljske tabele, uzmimo za primer *sales* tabelu iz prethodnih metoda, pozivamo funkcije koje će obeležiti tabelu kao kandidata za particionisanje:

SELECT partman.create\_parent( p\_parent\_table => 'sales',

 p\_control => 'sale\_date',

 p\_type => 'native',

 p\_interval=> 'monthly',

 p\_premake => 11);

* *p\_parent\_table* označava definiciju roditeljske tabele
* *p\_control* označava kolonu po kojoj se vrši particionisanje
* *p\_type* označava način kreiranja particija, moguće su vrednosti *native* i *partman*, *native* koristi deklarativni način dok *partman* koristi nasleđivanje i okidače
* *p\_interval* označava interval za particionisanje, postoje unapred napravljeni šabloni za vremenske intervale, a moguće je koristiti i cele brojeve ako se ne koristi datum za particionisanje
* *p\_premake* označava broj tabela koje će se kreirati na početku, ne uključujući prvu particiju

Ovako napravljena tabela imaće 12 particija ukupno i vršiće podelu po *sale\_date* koloni. Ukoliko želimo da automatski brišemo stare tabele, potrebno je ažurirati *part\_config* tabelu:

UPDATE partman.part\_config

SET infinite\_time\_partitions = true,

    retention = '12 months',

    retention\_keep\_table=true

WHERE parent\_table = 'sales';

Ovako konfigurisan *pg\_partman* će brisati tabele starije od godinu dana, a kreirati beskonačno novih tabela (uzimajući u obzir limit od 11 tabela koje može da kreira unapred). U okviru ove ekstenzije postoje još mnogo funkcija i opcija za upravljanje particijama, uključujući i mogućnost učitavanja podataka iz stare tabele u novu, particionisanu tabelu, a mogućnost automatskog upravljanja particijama je odlična za izolovane sisteme gde nije uvek moguće povezati se sa DBMS-om van interne mreže.

# Upoređivanje performansi particionisanja

Koristeći primer iz prethodnih poglavlja, tabela sa 100000000 podataka, jedna particionisana, jedna standardna tabela sa oko 7GB podataka možemo da uporedimo razlike u performansama.

### Unos podataka

INSERT INTO sales VALUES (generate\_series(1, 10000000),

                          'Product ' || trunc(random()\*1000),

                          trunc(random()\*10 +1),trunc(random()\*100 + random()\*10),

                            timestamp '2022-01-01 20:00:00' +

                            random() \* (timestamp '2023-01-20 20:00:00' -

                            timestamp '2022-01-01 10:00:00'),

                          random()\*5);

INSERT 0 100000000

Query returned successfully in 1 min 13 secs. -- bez particija

INSERT 0 100000000

Query returned successfully in 1 min 24 secs. -- sa particijama

Unose se nasumično generisani podaci, raspon datuma od jedne godine, sa nasumičnim ključem, cenom, količinom, a nazivi su “Product ” konkateniran sa nasumično generisanim brojem. Postoji mala razlika pri unosu podataka, particionisana tabela je sporija.

### Upiti

Za primer upita uzećemo pretragu u okviru jednog meseca i u opsegu od više meseci. Iz planera upita može se videti da tabela bez particija ima više prolaska kroz indekse i da upit traje preko 3 puta duže nego sa particijama:

-- bez particija

"Gather  (cost=1000.00..1901715.40 rows=4043910 width=31) (actual time=24.106..1889.424 rows=4012467 loops=1)"

"  Workers Planned: 2"

"  Workers Launched: 2"

"  ->  Parallel Seq Scan on sales\_big  (cost=0.00..1496324.40 rows=1684962 width=31) (actual time=56.927..1801.581 rows=1337489 loops=3)"

"        Filter: ((sale\_date >= '2022-02-03'::date) AND (sale\_date <= '2022-02-16'::date))"

"        Rows Removed by Filter: 35329178"

"Planning Time: 0.060 ms"

"JIT:"

"  Functions: 6"

"  Options: Inlining true, Optimization true, Expressions true, Deforming true"

"  Timing: Generation 0.966 ms, Inlining 94.868 ms, Optimization 49.596 ms, Emission 26.002 ms, Total 171.432 ms"

"Execution Time: 2003.729 ms"

-- sa particijama

"Bitmap Heap Scan on sales\_2022\_feb sales  (cost=54650.65..173670.12 rows=4007631 width=31) (actual time=87.417..501.921 rows=4004051 loops=1)"

"  Recheck Cond: ((sale\_date >= '2022-02-03'::date) AND (sale\_date <= '2022-02-16'::date))"

"  Heap Blocks: exact=58905"

"  ->  Bitmap Index Scan on sales\_2022\_feb\_sale\_date\_idx  (cost=0.00..53648.74 rows=4007631 width=0) (actual time=80.150..80.150 rows=4004051 loops=1)"

"        Index Cond: ((sale\_date >= '2022-02-03'::date) AND (sale\_date <= '2022-02-16'::date))"

"Planning Time: 0.089 ms"

"JIT:"

"  Functions: 2"

"  Options: Inlining false, Optimization false, Expressions true, Deforming true"

"  Timing: Generation 0.324 ms, Inlining 0.000 ms, Optimization 0.000 ms, Emission 0.000 ms, Total 0.324 ms"

"Execution Time: 601.270 ms"

Za upit koji koristi veći opseg datuma, ubrzanje je procentualno manje, ali i dalje primetno, a može se videti i orezivanje pri upitu:

-- bez particija

"Seq Scan on sales\_big  (cost=0.00..2458824.96 rows=29383213 width=31) (actual time=21.606..5680.326 rows=29479624 loops=1)"

"  Filter: ((sale\_date >= '2022-02-03'::date) AND (sale\_date <= '2022-05-16'::date))"

"  Rows Removed by Filter: 80520376"

"Planning Time: 0.058 ms"

"JIT:"

"  Functions: 2"

"  Options: Inlining true, Optimization true, Expressions true, Deforming true"

"  Timing: Generation 0.320 ms, Inlining 1.796 ms, Optimization 12.239 ms, Emission 7.413 ms, Total 21.769 ms"

"Execution Time: 6411.602 ms"

-- sa particijama

"Append  (cost=0.00..915227.81 rows=29536573 width=31) (actual time=71.451..3673.099 rows=29474189 loops=1)"

"  ->  Seq Scan on sales\_2022\_feb sales\_1  (cost=0.00..179069.85 rows=7461703 width=31) (actual time=71.450..579.726 rows=7439964 loops=1)"

"        Filter: ((sale\_date >= '2022-02-03'::date) AND (sale\_date <= '2022-05-16'::date))"

"        Rows Removed by Filter: 571026"

"  ->  Seq Scan on sales\_2022\_mar sales\_2  (cost=0.00..198386.71 rows=8875181 width=31) (actual time=0.148..588.165 rows=8875181 loops=1)"

"        Filter: ((sale\_date >= '2022-02-03'::date) AND (sale\_date <= '2022-05-16'::date))"

"  ->  Seq Scan on sales\_2022\_apr sales\_3  (cost=0.00..191835.26 rows=8582084 width=31) (actual time=0.040..568.644 rows=8582084 loops=1)"

"        Filter: ((sale\_date >= '2022-02-03'::date) AND (sale\_date <= '2022-05-16'::date))"

"  ->  Seq Scan on sales\_2022\_may sales\_4  (cost=0.00..198253.12 rows=4617605 width=31) (actual time=0.051..540.226 rows=4576960 loops=1)"

"        Filter: ((sale\_date >= '2022-02-03'::date) AND (sale\_date <= '2022-05-16'::date))"

"        Rows Removed by Filter: 4292248"

"Planning Time: 0.129 ms"

"JIT:"

"  Functions: 8"

"  Options: Inlining true, Optimization true, Expressions true, Deforming true"

"  Timing: Generation 0.914 ms, Inlining 1.853 ms, Optimization 41.528 ms, Emission 27.897 ms, Total 72.193 ms"

"Execution Time: 4406.372 ms"

# Zaključak

U okviru Postgres DBMS-a postoje još veliki broj optimizacija koje u kombinaciji sa particionisanjem mogu da još više povećaju performanse, ali treba uzeti u obzir i dostupne hardverske resurse pri planiraju i implementaciji aplikacija. Kao i sa svim optimizacijama, postoje određeni kompromisi koje treba imati u vidu, pa nije ni uvek moguće primeniti sve metode. Brz razvitak tehnologija takođe ima ključnu ulogu u razvoju aplikacija, pa postoji mogućnost da će se pojaviti novi načini optimizacija, ili se pak pojave skroz drugačiji načini skladištenja i obrade podataka. Primeri u ovom radu verovatno mogu biti dodatno optimizovani i za primenu sa particijama, a i za ostale upite.

Najbitnija stavka pri planiranju i implementaciji projekta je imati inženjerski pristup procesu i uzeti u obzir trenutne i buduće probleme. Potrebno je napraviti pregled trenutnih potencijalnih rešenja i ukoliko je potrebno, napraviti plan rada za implementaciju novog. U velikom broju slučajeva su već definisani i implementirani algoritmi koji pokrivaju veliki deo domena problema, samo je potrebno modifikovati postojeća rešenja kako bi bila primenljiva nad konkretnim slučajem.

Optimizacija particionisanja se pokazala kao uspešna metoda za velike količine podataka.

# Literatura

<https://www.crunchydata.com/blog/native-partitioning-with-postgres>

<https://hevodata.com/learn/postgresql-partitions/>

<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/PostgreSQL_Partitions.html>

<https://evilmartians.com/chronicles/a-slice-of-life-table-partitioning-in-postgresql-databases>

<https://www.postgresql.org/docs/current/ddl-partitioning.html>

<https://www.singlestore.com/blog/database-sharding-vs-partitioning-whats-the-difference>

<https://github.com/pgpartman/pg_partman/blob/master/doc/pg_partman.md>