Edistynyt PostgreSQL-ohjelmointi

2020-10-02
HELSINKI

HENRI SEIJO
MISTMAP

Päivän materiaali

https://github.com/mistmap/
postgresql-training-finnish-2020-10

(tai https://bit.ly/33jhhfm)

Mistmap

- yritys käynnistyi 2019
- devaus, arkkitehtuuri, ketteröittäminen, koulutus
- avoimet järjestelmät fokuksessa

Päivän rakenne

09:00–09:15	Avaus
09:15–09:55	Ohjelmointitekniikoita
09:55–10:00	Tauko
10:00–10:40	Ohjelmointitekniikoita – harjoituksia
10:40–10:45	Tauko
10:45–11:30	Suorituskyvyn optimointi
11:30–12:30	Lounas
12:30–13:10	Suorituskyvyn optimointi – harjoituksia
13:10–13:15	Tauko
13:15–14:00	JSON
14:00–14:30	Kahvitauko
14:30–15:10	JSON – harjoituksia
15:10–15:15	Tauko
15:15–15:55	PostgreSQL:n kehitys ja Q&A
15:55–16:00	Lopetus

Ohjelmointitekniikoita

Miksi?

Liiketoimintalogiikan sijainti

Miksi ei kantaan?

Miksi kantaan?

Ekosysteemi

Kolme osaamisaluetta

Miten?

Kielet

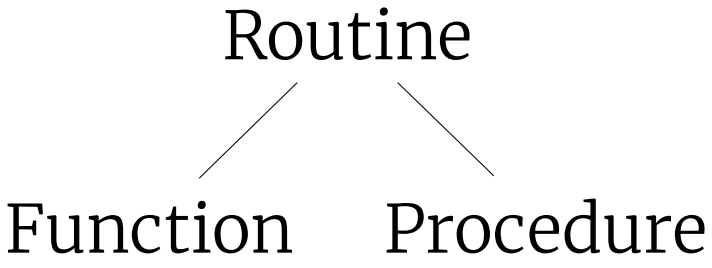
- SQL
- C
- Proseduraaliset kielet
  - PL/pgSQL
  - PL/Tcl, PL/Perl, PL/Python, PL/Java, PL/Lua, PL/R, PL/sh, PL/v8, ...

Suosi SQL:ää.

```
1  PREPARE get_top(integer) AS
2    SELECT *
3    FROM scoreboard
4    ORDER BY total_score DESC
5    LIMIT $1;
6
7  EXECUTE get_top(10);
```

Prepared statement

Epäoptimaalinen plan saattaa valikoitua generic planiksi.


Routine
Function
Procedure

```
CREATE FUNCTION calculate_circle_area(
  IN radius double precision = 1.0
)
RETURNS double precision
LANGUAGE sql
IMMUTABLE
RETURNS NULL ON NULL INPUT
PARALLEL SAFE
AS $calculate_circle_area$
  SELECT pi() * radius ^ 2;
$calculate_circle_area$;

SELECT calculate_circle_area(5);
```

Argumentit

- `IN`: syöte (oletus)
- `OUT`: tulos
- `INOUT`: päivitettävä
- `VARIADIC`: mielivaltaisen pituinen array

Paluuarvot

- perustietotyyppi: esimerkiksi `integer`
- composite type: esimerkiksi `(integer, text)`
- `SETOF/TABLE`: joukko edellisiä
- `void`: ei mitään

SQL-funktio palauttaa viimeisen lauseen tuloksen.

Funktion volatiliteetti

- IMMUTABLE
  - aina sama tulos samalla syötteellä, esimerkiksi `lower()`
  - ei lue tai muokkaa kantaa tai konfiguraatiota
- STABLE
  - sama tulos samalla syötteellä ainakin saman table scanin aikana, esimerkiksi `transaction_timestamp()`
  - ei muokkaa kantaa
- VOLATILE
  - oletus
  - tulos voi vaihtua joka kutsukerralla, esimerkiksi `random()` tai `clock_timestamp()`
  - saa muokata kantaa
  - pakollinen, jos aiheuttaa sivuefektejä, esimerkiksi lähettää sähköpostia

```
1  CREATE PROCEDURE insert_data(
2    a integer, b integer)
3  LANGUAGE plpgsql AS $insert_data$
4  BEGIN
5    CREATE TABLE t (i integer);
6    INSERT INTO t VALUES (a);
7    COMMIT;
8    INSERT INTO t VALUES (b);
9    ROLLBACK;
10 END; $insert_data$;
11
12 CALL insert_data(1, 2);
```

Function	Procedure
voi palauttaa arvon	ei palauta mitään
rajautuu DML:ään	`COMMIT` ja `ROLLBACK` käytettävissä
`SELECT func();`	`CALL proc();`
voi kutsua muun SQL:n seassa	ei voi kutsua osana muuta lausetta
`IMMUTABLE` jne. auttavat optimoijaa	ei vastaavia mahdollisuuksia

PL/pgSQL

Karkeasti SQL ja

- muuttujat
- ehtolauseet
- silmukat
- poikkeukset
- `RETURN`

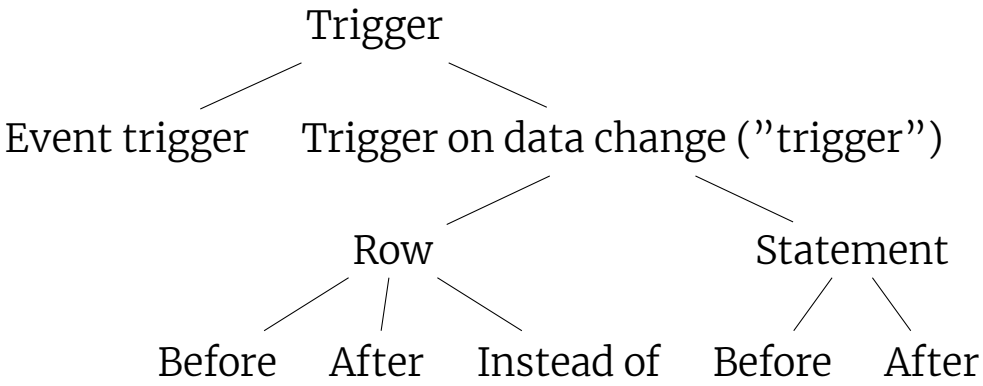
```
1  CREATE FUNCTION and_some_more(
2    IN x numeric, OUT y numeric)
3  RETURNS SETOF numeric LANGUAGE plpgsql
4  AS $and_some_more$
5  DECLARE z numeric := x + 1;
6  BEGIN
7    SELECT x INTO STRICT y;
8    RETURN NEXT;
9    SELECT z INTO STRICT y;
10   RETURN NEXT;
11   RETURN;
12 END;
13 $and_some_more$;
14
15 SELECT and_some_more(5);
```

PL/pgSQL: cursor

In fact, in a 23-year career with PostgreSQL, I have only actually found a need to use cursors twice. And I regret one of those. (Kirk Roybal, 2ndQuadrant)

Recursive CTE

```
1 WITH RECURSIVE cte(n) AS (
2   SELECT 1
3   UNION ALL
4   SELECT n + 1 FROM cte WHERE n < 4
5 )
6 SELECT * FROM cte;
```


Trigger
Event trigger
Trigger on data change (”trigger”)
Row
Statement
Before
After
Instead of
Before
After

Rakenne

Trigger luodaan kahdessa osassa:

1. Luo funktio, joka palauttaa triggerin.
2. Luo trigger, joka kytkee taulun ja funktion toisiinsa.

Triggeriä ei voi kirjoittaa SQL:llä.

Sarakkeen täydennys 1

```
1  CREATE FUNCTION add_created_at() RETURNS trigger
2  LANGUAGE plpgsql AS $add_created_at$ BEGIN
3    NEW.created_at := transaction_timestamp();
4    RETURN NEW;
5  END; $add_created_at$;
6
7  CREATE TRIGGER add_created_at
8  BEFORE INSERT ON products
9  FOR EACH ROW EXECUTE FUNCTION add_created_at();
```

Sama selvemmin

```
1  CREATE TABLE products (
2    ...
3    created_at timestamptz
4      DEFAULT transaction_timestamp() NOT NULL,
5    ...
6  );
```

Tosin käyttäjän on mahdollista korvata oletus, ellei `GRANT`illa ole evätty oikeutta sarakkeen muokkaukseen.

Sarakkeen täydennys 2

```
1  CREATE FUNCTION add_area() RETURNS trigger
2  LANGUAGE plpgsql AS $add_area$ BEGIN
3    NEW.area := pi() * NEW.radius ^ 2;
4    RETURN NEW;
5  END; $add_area$;
6
7  CREATE TRIGGER add_area
8  BEFORE INSERT OR UPDATE ON circles
9  FOR EACH ROW EXECUTE FUNCTION add_area();
```

Sama selvemmin

```
1  CREATE TABLE circles (
2    ...
3    area double precision
4      GENERATED ALWAYS AS (pi() * radius ^ 2) STORED,
5    ...
6  );
```

Arvon tarkistus

```
1  CREATE FUNCTION check_discount() RETURNS trigger
2  LANGUAGE plpgsql AS $check_discount$ BEGIN
3    IF NEW.discounted_price >= NEW.price THEN
4      RAISE EXCEPTION
5        '% must have discounted_price below price',
6        NEW.product_id;
7    END IF;
8    RETURN NEW;
9  END; $check_discount$;
10
11 CREATE TRIGGER check_discount
12 BEFORE INSERT OR UPDATE ON products
13 FOR EACH ROW EXECUTE FUNCTION check_discount();
```

Sama selvemmin

```
1  CREATE TABLE products (
2    ...
3    CONSTRAINT discounted_price_below_price
4      CHECK (discounted_price < price),
5    ...
6  );
```

Muokkausaikaleima

```
1  CREATE FUNCTION add_last_modified_at() RETURNS TRIGGER
2  LANGUAGE plpgsql AS $add_last_modified_at$ BEGIN
3    NEW.last_modified_at = transaction_timestamp();
4    RETURN NEW;
5  END; $add_last_modified_at$
6
7  CREATE TRIGGER add_last_modified_at
8  BEFORE UPDATE ON products
9  FOR EACH ROW EXECUTE FUNCTION add_last_modified_at();
```

Sama selvemmin

```
1  CREATE EXTENSION moddatetime;
2
3  CREATE TRIGGER add_last_modified_at
4  BEFORE UPDATE ON products
5  FOR EACH ROW EXECUTE FUNCTION
6    moddatetime(last_modified_at);
```

Harkitse triggerin käyttöä huolellisesti!

1. Luodut triggerit unohtuvat helposti.
2. Triggereillä on vaarallisen helppoa räplätä mitä vain mihin vain, käyttöoikeuksien puitteissa.

Rule system

- Query rewrite rule system
- Näkymien sisäinen toteutusyksityiskohta
- Epäintuitiivisia ja monimutkaisia vaikutuksia, piilossa
- Ei samanlainen työkalu kuin Oraclen

Älä käytä.

Kehitystyökaluja

pgFormatter yhtenäistää koodin ulkoasun, kuten rivinvaihdot.

plpgsql_check etsii *PL/pgSQL-koodista* karkeita virheitä ja tarkastaa laatua peukalosäännöillä.

schemalint etsii *skeemasta* karkeita virheitä ja tarkastaa laatua peukalosäännöillä.

Testaustyökaluja

pgTAP	mahdollistaa yksikkötestauksen: Miten pieni osa koodia, yksikkö, käyttäytyy yksittäisillä testitapauksilla?
RegreSQL	mahdollistaa regressiotestauksen: Tuottaako uusi koodi samat tulokset kuin vanha koodi?
IntegreSQL	nopeuttaa integraatiotestausta: Miten PostgreSQL toimii yhteen muiden palvelujen kanssa?
noisia	luo ongelmallista kuormaa testiympäristöön.

Suorituskyvyn optimointi

EXPLAIN

```
                                                                QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=13.12..13.15 rows=10 width=70) (actual time=0.194..0.203 rows=10 loops=1)
   Output: track_id, name, album_id, media_type_id, genre_id, composer, milliseconds, bytes, unit_price
   Buffers: shared hit=4
   ->  Sort  (cost=13.12..13.37 rows=98 width=70) (actual time=0.192..0.196 rows=10 loops=1)
         Output: track_id, name, album_id, media_type_id, genre_id, composer, milliseconds, bytes, unit_price
         Sort Key: track.composer
         Sort Method: top-N heapsort  Memory: 28kB
         Buffers: shared hit=4
         ->  Index Scan using pk_track on chinook.track  (cost=0.28..11.00 rows=98 width=70) (actual time=0.027..0.0
               Output: track_id, name, album_id, media_type_id, genre_id, composer, milliseconds, bytes, unit_price
               Index Cond: (track.track_id < 100)
               Buffers: shared hit=4
 Settings: search_path = 'chinook'
 Planning Time: 0.238 ms
 Execution Time: 0.255 ms
(15 rows)
```

MISTMAP

EXPLAIN ANALYZE

`BEGIN` ja `ROLLBACK` suojaavat vahingoilta.

EXPLAIN VERBOSE

EXPLAIN COSTS

EXPLAIN SETTINGS

EXPLAIN ANALYZE BUFFERS

EXPLAIN ANALYZE TIMING

EXPLAIN SUMMARY

EXPLAIN FORMAT

EXPLAIN (ANALYZE, VERBOSE, COSTS, SETTINGS, BUFFERS, TIMING, SUMMARY)

Visualisointi

https://explain.depesz.com/

Taulujen lukeminen (scan)

Sequential scan

Index scan

Index-only scan

Bitmap scan

Taulujen liittäminen (join)

Nested loop join

Hash join

Merge join

Join-variantit

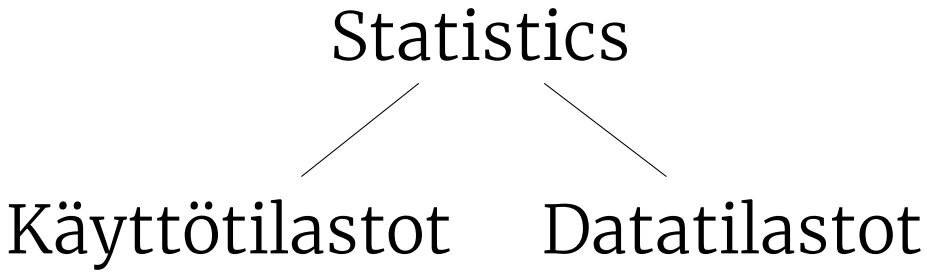
Taulujen lajittelu (sort)

Quicksort

External merge

Top-N heapsort

Statistics


Statistics
Käyttötilastot
Datatilastot

Datatilastot

ANALYZE

VACUUM

autovacuum

Extended statistics

```
CREATE STATISTICS stts2 (ndistinct)
ON city, state, zip FROM zipcodes;
```

Funktionaaliset riippuvuudet

ndistinct

mcv

Käyttötilastot ja monitorointi

Wait events

Explicit locking

JSON

```
2.718

"junttura"

{
  "name": "Перельман",
  "easy": [
    1,
    2
  ]
}

[1,null,false,"vipstaaki",{"foo":"bar"}]
```

Merkistö ja tulkinta

- UTF-8, mutta UTF-16 escaping
- number: kokonaisluvut *ja* liukuluvut
- ei määriteltyä lukujen tarkkuutta tai tulkintaa

JSON-käyttö relaatiokannassa

JSON-tietotyyppi

Älä käytä.

JSONB-tietotyyppi

JSONB-tyypit ja SQL-tyypit 1

```
1  CREATE TABLE t (j jsonb);
2  INSERT INTO t VALUES
3    ('1'),
4    ('"foo"'),
5    ('false'),
6    ('{"bar": 2}'),
7    ('[3, true]'),
8    ('null'),
9    (null);
10 \pset null (null)
11 SELECT j, jsonb_typeof(j), pg_typeof(j)
12 FROM t;
```

JSONB-tyypit ja SQL-tyypit 2

j	jsonb_typeof	pg_typeof
1	number	jsonb
”foo”	string	jsonb
false	boolean	jsonb
{”bar”: 2}	object	jsonb
[3, true]	array	jsonb
null	null	jsonb
(null)	(null)	jsonb

JSONB:n JSON-epäyhteensopivuudet

- null-tavu (`\0`) ei sallittu JSONB-syötteessä (ei tekemistä JSON-nullin tai SQL-nullin kanssa)
- `NaN` ja `Infinity` eivät sallittuja number-arvoja JSONB-syötteessä

JSONB-operaattoreita: -> ja ->>

Sukellus jsonb-arrayhin luvulla:

```
SELECT '[4,5,6]'::jsonb -> 1; -- => 5 of jsonb
```

Sukellus jsonb-objektiin nimellä:

```
SELECT '{"a":1,"b":2}'::jsonb -> 'b'; -- => 2 of jsonb
```

`->>` tekee saman, mutta palauttaa arvon tyyppiä `text`.

JSONB-operaattoreita: #> ja #>>

```
SELECT '{"a":[1,{"b":2}]}'::jsonb
  -> 'a' -> 1 -> 'b'; -- => 2 of jsonb
```

korvataan lyhyemmin

```
SELECT '{"a":[1,{"b":2}]}'::jsonb
  #> '{a,1,b}'; -- => 2 of jsonb
```

`#>>` tekee saman, mutta palauttaa arvon tyyppiä `text`.

JSONB-operaattoreita: @> ja ?

Sisältääkö ylimmällä tasolla?

```
SELECT '{"a":1,"b":2}'::jsonb
  @> '{"b":2}'::jsonb; -- => true of boolean


SELECT '{"a":1}'::jsonb ? 'a'; -- => true of boolean
SELECT   '["a"]'::jsonb ? 'a'; -- => true of boolean
SELECT     '"a"'::jsonb ? 'a'; -- => true of boolean
```

JSONB-funktiot

SQL/JSON path eli `jsonpath`

JSONB GIN-indeksit

`jsonb_ops` tukee useampaa operaattoria (oletus):

```
CREATE INDEX idx ON my_table
USING GIN (jsonb_column);
```

`jsonb_path_ops` toimii nopeammin:

```
CREATE INDEX idx ON my_table
USING GIN (jsonb_column json_path_ops);
```

Expression indexing

```
CREATE INDEX expr_idx ON my_table
USING GIN ((jsonb_column -> 'often_needed_key'));
```

- hyödynnettävissä vain kyselyihin, jotka käyttävät samaa ilmaisua
- poikkeuksellisesti aja `ANALYZE my_table` heti luomisen jälkeen

Partial index

```
1  CREATE INDEX partial_idx ON my_table
2  USING GIN (jsonb_column)
3  WHERE
4    other_column > 3
5    AND NOT (
6      jsonb_column ? 'too_common_to_index'
7      OR jsonb_column @> '{"not_interesting":1}'::jsonb
8    );
```

Niksi: Generated columns

```
CREATE TABLE my_table (
  ...
  jsonb_column jsonb,
  top_favorite text GENERATED ALWAYS AS
    (jsonb_column #>> '{favorites,0}') STORED,
  ...
);
```

Niksi: CHECK

```
1  CREATE TABLE my_table (
2    ...
3    jsonb_column jsonb DEFAULT {} NOT NULL,
4    CONSTRAINT jsonb_column_must_be_object
5      CHECK (jsonb_typeof(jsonb_column) = 'object'),
6    ...
7  );
```

PostgreSQL 13

B-tree-indeksoinnin tiivistäminen

Aggregoinnin nopeutus

Extended statistics

Rinnakkainen VACUUM B-tree-indekseille

Incremental sorting

LIMIT ... WITH TIES

EXPLAIN WAL

UUIDv4-generointi

PostgreSQL:n jatkokehitys

Suurten yhteysmäärien käsittely

REINDEX partitioiduille tauluille

Lisää instrumentointia utility-käskyille

zheap

Kysymyksiä?

Audit-lokituksen tavoite

Audit-lokituksen menetelmiä

Palaute

Kiitos!

henri.seijo@mistmap.com