PG 9.5 – novinky ve vývoji aplikací

P2D2 2016

Antonín Houska

18. února 2016

Část I

GROUPING SETS, ROLLUP, CUBE

Agregace

- Seskupení řádků tabulky (joinu) do podmnožin podle určitého klíče.
- Za každou podmnožinu se ve výsledku objeví přesně jeden řádek.
- Klíč může obsahovat jeden nebo několik výrazů (tzn. nejen jména sloupečků).
- Hodnoty výrazu, který není součástí klíče, se v rámci skupiny mohou lišit. Sloučení do jedné hodnoty zajistí **agregační funkce**.
- Každý výraz uvedený v klauzuli `SELECT` musí figurovat v klíči (`GROUP BY`) nebo v některé agregační funkci.

Agregace

„Každý výraz uvedený v klauzuli SELECT musí figurovat v klíči(GROUP BY) nebo v některé agregační funkci.“

```
SELECT jmeno, cas, count(*)
FROM hovory_20160218
GROUP BY jmeno;
```

```
ERROR:  column "hovory_20160218.cas" must appear in
the GROUP BY clause or be used in an aggregate
function.
```

Příklad

Agregace se často používá k vyhodncení souboru událostí ...

```
CREATE TABLE hovory_20160218 (
    jmeno        varchar     NOT NULL,
    odchozi      bool        NOT NULL,
    soukromy     bool        NOT NULL
    cas          time        NOT NULL,
    trvani       interval    NOT NULL,
);
```

Vstupní data

```
SELECT *
FROM   hovory_20160218;
```

jmeno	odchozi	soukromy	cas	trvani
vaclav	true	true	08:29:06	00:01:26
lubos	true	false	08:53:34	00:03:11
vaclav	false	true	09:01:56	00:02:39
cestmir	true	false	09:42:27	00:05:31
adam	false	true	09:54:08	00:01:05
lubos	true	false	09:44:16	00:02:12
adam	true	false	10:08:27	00:03:43
vaclav	false	false	11:35:09	00:02:50
cestmir	true	false	11:48:15	00:04:13

Jak dlouho telefonovali jednotliví zaměstnanci?

```
SELECT jmeno, sum(trvani) AS provolano
FROM   hovory_20160218
GROUP BY jmeno;
```

jmeno	odchozi	soukromy	cas	trvani
vaclav	true	true	08:29:06	00:01:26
lubos	true	false	08:53:34	00:03:11
vaclav	false	true	09:01:56	00:02:39
cestmir	true	false	09:42:27	00:05:31
adam	false	true	09:54:08	00:01:05
lubos	true	false	09:44:16	00:02:12
adam	true	false	10:08:27	00:03:43
vaclav	false	false	11:35:09	00:02:50
cestmir	true	false	11:48:15	00:04:13

Jak dlouho telefonovali jednotliví zaměstnanci?

```
                    QUERY PLAN
-----------------------------------------------
 HashAggregate
   Group Key: jmeno
   ->  Seq Scan on hovory_20160218
```

jmeno	sum
adam	00:04:48
cestmir	00:09:44
vaclav	00:06:55
lubos	00:05:23

Kolik času se provolalo v jednotlivých hodinách?

```
SELECT date_trunc('hour', cas), sum(trvani)
FROM   hovory_20160218
GROUP BY date_trunc('hour', cas);
```

jmeno	odchozi	soukromy	cas	trvani
vaclav	true	true	08:29:06	00:01:26
lubos	true	false	08:53:34	00:03:11
vaclav	false	true	09:01:56	00:02:39
cestmir	true	false	09:42:27	00:05:31
adam	false	true	09:54:08	00:01:05
lubos	true	false	09:44:16	00:02:12
adam	true	false	10:08:27	00:03:43
vaclav	false	false	11:35:09	00:02:50
cestmir	true	false	11:48:15	00:04:13

Kolik času se provolalo v jednotlivých hodinách?

```
                                  QUERY PLAN
-----------------------------------------------------------------------------
 HashAggregate
   Group Key: date_trunc('hour'::text, (cas)::interval)
   ->  Seq Scan on hovory_20160218
```

date_trunc	sum
08:00:00	00:04:37
09:00:00	00:11:27
10:00:00	00:03:43
11:00:00	00:07:03

Kolik bylo hovorů jednotlivých typů?

```
SELECT odchozi, soukromy, count(*)
FROM   hovory_20160218
GROUP BY odchozi, soukromy;
```

jmeno	odchozi	soukromy	cas	trvani
vaclav	true	true	08:29:06	00:01:26
lubos	true	false	08:53:34	00:03:11
vaclav	false	true	09:01:56	00:02:39
cestmir	true	false	09:42:27	00:05:31
adam	false	true	09:54:08	00:01:05
lubos	true	false	09:44:16	00:02:12
adam	true	false	10:08:27	00:03:43
vaclav	false	false	11:35:09	00:02:50
cestmir	true	false	11:48:15	00:04:13

Kolik bylo hovorů jednotlivých typů?

```
                        QUERY PLAN
-----------------------------------------------------------
 HashAggregate
   Group Key: odchozi, soukromy
   ->  Seq Scan on hovory_20160218
```

odchozi	soukromy	count
false	false	1
false	true	2
true	false	5
true	true	1

Všechny agregace v jednom dotazu

```
SELECT jmeno, date_trunc('hour', cas),
       odchozi, soukromy,
       sum(trvani), count(*)
FROM hovory_20160218
GROUP BY GROUPING SETS (
  (jmeno),
  (date_trunc('hour', cas)),
  (odchozi, soukromy)
);
```

1. Všechny dotazy, které jsme takto sloučili, musí mít stejné klauzule `FROM` a `WHERE`.
2. Klauzule `GROUPING SETS` obsahuje všechny kombinace klíčů `GROUP BY` z předchozích dotazů.
3. Každý „neagregovaný“ sloupeček v klazuli `SELECT` musí figurovat alespoň v jedné kombinaci klíčů.

Všechny agregace v jednom dotazu

jmeno	date_trunc	odchozi	soukromy	sum	count
adam				00:04:48	2
cestmir				00:09:44	2
vaclav				00:06:55	3
lubos				00:05:23	2
	08:00:00			00:04:37	2
	09:00:00			00:11:27	4
	10:00:00			00:03:43	1
	11:00:00			00:07:03	2
		false	false	00:02:50	1
		false	true	00:03:44	2
		true	false	00:18:50	5
		true	true	00:01:26	1

To samé bez GROUPING SETS

```
SELECT jmeno, NULL, NULL::bool,
       NULL::bool, sum(trvani), count(*)
FROM   hovory_20160218
GROUP BY jmeno

UNION ALL

SELECT NULL, date_trunc('hour', cas),
       NULL::bool, NULL::bool, sum(trvani), count(*)
FROM   hovory_20160218
GROUP BY date_trunc('hour', cas)

UNION ALL

SELECT NULL, NULL,
       odchozi, soukromy, sum(trvani), count(*)
FROM   hovory_20160218
GROUP BY odchozi, soukromy;
```

To samé bez GROUPING SETS

```
                        QUERY PLAN
-------------------------------------------------------
 Append
   ->  HashAggregate
         Group Key: hovory_20160218.jmeno
         ->  Seq Scan on hovory_20160218
   ->  HashAggregate
         Group Key: date_trunc('hour', .cas)
         ->  Seq Scan on hovory_20160218
   ->  HashAggregate
         Group Key: odchozi, soukromy
         ->  Seq Scan on hovory_20160218
```

Protože pro GROUPING SETS zatím PG neumí používat HashAggregate, může toto řešení být někdy efektivnější.

Je-li ve skupinách hierarchie: ROLLUP

```
SELECT jmeno, odchozi, soukromy, sum(trvani)
FROM hovory_20160218
GROUP BY
ROLLUP (jmeno, odchozi, soukromy);
```

To samé pomocí `GROUPING SETS`:

```
SELECT jmeno, odchozi, soukromy, sum(trvani)
FROM hovory_20160218
GROUP BY
GROUPING SETS (
  (jmeno),
  (jmeno, odchozi),
  (jmeno, odchozi, soukromy),
  ()
);
```

Je-li ve skupinách hierarchie: ROLLUP

EXPLAIN

```
                        QUERY PLAN
-------------------------------------------------
 GroupAggregate
   Group Key: jmeno, odchozi, soukromy
   Group Key: jmeno, odchozi
   Group Key: jmeno
   Group Key: ()
   ->  Sort
         Sort Key: jmeno, odchozi, soukromy
         ->  Seq Scan on hovory_20160218
```

Je-li ve skupinách hierarchie: ROLLUP

1. krok – seřazení

jmeno	odchozi	soukromy	cas	trvani
adam	false	true	09:54:08	00:01:05
adam	true	false	10:08:27	00:03:43
cestmir	true	false	09:42:27	00:05:31
cestmir	true	false	11:48:15	00:04:13
lubos	true	false	08:53:34	00:03:11
lubos	true	false	09:44:16	00:02:12
vaclav	false	false	11:35:09	00:02:50
vaclav	false	true	09:01:56	00:02:39
vaclav	true	true	08:29:06	00:01:26

Je-li ve skupinách hierarchie: ROLLUP

2. krok – GROUP BY jmeno, odchozi, soukromy

jmeno	odchozi	soukromy	cas	trvani
adam	false	true	09:54:08	00:01:05
adam	true	false	10:08:27	00:03:43
cestmir	true	false	09:42:27	00:05:31
cestmir	true	false	11:48:15	00:04:13
lubos	true	false	08:53:34	00:03:11
lubos	true	false	09:44:16	00:02:12
vaclav	false	false	11:35:09	00:02:50
vaclav	false	true	09:01:56	00:02:39
vaclav	true	true	08:29:06	00:01:26

Je-li ve skupinách hierarchie: ROLLUP

3. krok – GROUP BY jmeno, odchozi

jmeno	odchozi	soukromy	cas	trvani
adam	false	true	09:54:08	00:01:05
adam	true	false	10:08:27	00:03:43
cestmir	true	false	09:42:27	00:05:31
cestmir	true	false	11:48:15	00:04:13
lubos	true	false	08:53:34	00:03:11
lubos	true	false	09:44:16	00:02:12
vaclav	false	false	11:35:09	00:02:50
vaclav	false	true	09:01:56	00:02:39
vaclav	true	true	08:29:06	00:01:26

Je-li ve skupinách hierarchie: ROLLUP

4. krok – GROUP BY jmeno

jmeno	odchozi	soukromy	cas	trvani
adam	false	true	09:54:08	00:01:05
adam	true	false	10:08:27	00:03:43
cestmir	true	false	09:42:27	00:05:31
cestmir	true	false	11:48:15	00:04:13
lubos	true	false	08:53:34	00:03:11
lubos	true	false	09:44:16	00:02:12
vaclav	false	false	11:35:09	00:02:50
vaclav	false	true	09:01:56	00:02:39
vaclav	true	true	08:29:06	00:01:26

Je-li ve skupinách hierarchie: ROLLUP

Výsledek

jmeno	odchozi	soukromy	sum
vaclav	false	false	00:02:50
vaclav	false	true	00:02:39
vaclav	false		00:05:29
vaclav	true	true	00:01:26
vaclav	true		00:01:26
vaclav			00:06:55

Obdobně pro další jména . . .

. . . a nakonec . . . všechno do jedné skupiny:

			00:26:50

Všechny způsoby rozdělení do skupin: CUBE

```
SELECT date_trunc('hour', cas), odchozi, soukromy,
       count(*),
       GROUPING
       (date_trunc('hour', cas), odchozi, soukromy)
FROM hovory_20160218
GROUP BY
CUBE (date_trunc('hour', cas), odchozi, soukromy);
```

Všechny způsoby rozdělení do skupin: CUBE

```
CUBE (date_trunc('hour', cas), odchozi, soukromy);
```

...je zkrácený zápis výrazu ...

```
GROUPING SETS (
  (date_trunc('hour', cas), odchozi, soukromy),
  (date_trunc('hour', cas), odchozi),
  (date_trunc('hour', cas), soukromy),
  (odchozi, soukromy),
  (date_trunc('hour', cas)),
  (soukromy), (odchozi), ()
);
```

Výraz GROUPING říká, podle které „grouping set“ byl vytvořen konkrétní řádek výsledku.

Všechny způsoby rozdělení do skupin: CUBE

date_trunc	odchozi	soukromy	count	grouping
08:00:00	true	false	1	0
08:00:00	true	true	1	0
09:00:00	false	true	2	0
09:00:00	true	false	2	0
10:00:00	true	false	1	0
11:00:00	false	false	1	0
11:00:00	true	false	1	0
08:00:00	true		2	1
09:00:00	false		2	1
09:00:00	true		2	1
10:00:00	true		1	1
11:00:00	false		1	1
11:00:00	true		1	1

Všechny způsoby rozdělení do skupin: CUBE

date_trunc	odchozi	soukromy	count	grouping
08:00:00		false	1	2
08:00:00		true	1	2
09:00:00		false	2	2
09:00:00		true	2	2
10:00:00		false	1	2
11:00:00		false	2	2
	false	false	1	4
	false	true	2	4
	true	false	5	4
	true	true	1	4

Všechny způsoby rozdělení do skupin: CUBE

date_trunc	odchozi	soukromy	count	grouping
08:00:00			2	3
09:00:00			4	3
10:00:00			1	3
11:00:00			2	3
		false	6	6
		true	3	6
	false		3	5
	true		6	5
			9	7

GROUPING SETS, ROLLUP a CUBE lze kombinovat

```
GROUPING SETS (jmeno), CUBE(odchozi, soukromy)
```

```
GROUPING SETS (
 (jmeno, odchozi, soukromy),
 (jmeno, odchozi),
 (jmeno, soukromy),
 (jmeno)
);
```

GROUPING SETS, ROLLUP a CUBE lze kombinovat

```
GROUPING SETS ((jmeno), CUBE(odchozi, soukromy))
```

```
GROUPING SETS (
 (jmeno),
 (odchozi, soukromy),
 (odchozi), (soukromy),
 ()
);
```

Další možnosti

- Ordered-set aggregates

```
SELECT jmeno, odchozi, soukromy,
       percentile_disc(0.5)
       WITHIN GROUP (ORDER BY trvani)
FROM hovory_20160218
GROUP BY
CUBE (jmeno, odchozi, soukromy);
```

- FILTER

```
SELECT jmeno, odchozi,
       count(*),
       count(*) FILTER (WHERE soukromy)
FROM hovory_20160218
GROUP BY
CUBE (jmeno, odchozi, soukromy);
```

Část II

UPSERT

Příklad: Sběr dat offline

Zpracovaná data

```
CREATE TABLE cesty (
        id      int NOT NULL PRIMARY KEY,
        uzly    point[] NOT NULL
);
```

Nezpracovaná data od konkrétního autora

```
CREATE TABLE cesty_tmp (
        id      int NOT NULL PRIMARY KEY,
        uzly    point[] NOT NULL
);
```

Příklad: Sběr dat offline

```
SELECT *
FROM   cesty;
```

id	uzly
206	{"(49.908701,14.290768)",...}
94	{"(49.903474,14.283657)",...}
48	{"(49.898872,14.248662)",...}

```
SELECT *
FROM   cesty_tmp;
```

id	uzly
206	{"(49.908358,14.2900923),...}
12	{"(49.908701,14.290768)",...}

Příklad: Sběr dat offline

```
SELECT id, array_length(uzly, 1)
FROM   cesty;
```

id	array_length
206	3
94	3
48	3

```
SELECT id, array_length(uzly, 1)
FROM   cesty_tmp;
```

id	array_length
206	1
12	2

Příklad: Sběr dat offline

Očekávaný výsledek

id	array_length
206	4
94	3
48	3
12	2

Příklad: Sběr dat offline

```
INSERT INTO cesty(id, uzly)
SELECT id, uzly
FROM cesty_tmp;

ERROR:  conflicting key value violates exclusion
constraint "cesty_id_excl"
DETAIL:  Key (id)=(206) conflicts with existing key
(id)=(206).
```

Příklad: Sběr dat offline

```
UPDATE cesty
SET    uzly = cesty.uzly || cesty_tmp.uzly
FROM   cesty_tmp
WHERE  cesty.id = cesty_tmp.id;

UPDATE 1
```

PG <= 9.4

```
WITH inserted(id) AS (
     INSERT INTO cesty(id, uzly)
     SELECT tmp.id, tmp.uzly
     FROM   cesty_tmp tmp
          LEFT JOIN cesty c ON tmp.id = c.id
     WHERE  c.id ISNULL
     RETURNING id)
UPDATE cesty
SET    uzly = cesty.uzly || cesty_tmp.uzly
FROM   cesty_tmp
WHERE  cesty.id = cesty_tmp.id AND NOT EXISTS (
           SELECT *
           FROM inserted
           WHERE id = cesty.id);
```

PG <= 9.4

```
Update on cesty
  CTE inserted
    ->  Insert on cesty cesty_1
          ->  Hash Anti Join
                Hash Cond: (tmp.id = c.id)
                ->  Seq Scan on cesty_tmp tmp
                ->  Hash
                      ->  Seq Scan on cesty c
  ->  Hash Join
        Hash Cond: (cesty.id = cesty_tmp.id)
        ->  Hash Anti Join
              Hash Cond: (cesty.id = inserted.id)
              ->  Seq Scan on cesty
              ->  Hash
                    ->  CTE Scan on inserted
        ->  Hash
              ->  Seq Scan on cesty_tmp
```

PG <= 9.4

Tabulka `cesty_tmp` obsahuje id = 12.

```
                              BEGIN;

                              INSERT INTO cesty(id, ...)
                              VALUES (12, ...)

WITH inserted(id) AS (
  INSERT INTO cesty ...
) UPDATE cesty ...;

                              COMMIT;

ERROR: conflicting key value ...
DETAIL: Key (id)=(12) ...
```

PG <= 9.4

Tabulka `cesty_tmp` obsahuje **jen** id = 12.

```
                                BEGIN;

                                DELETE FROM cesty
                                WHERE id = 12;

WITH inserted(id) AS (
  INSERT INTO cesty ...
) UPDATE cesty ...;

                                COMMIT;
```

Z dotazu `WITH ...` se neprovedl ani INSERT ani UPDATE.

PG <= 9.4

Tabulka `cesty_tmp` obsahuje **jen** id = 12.

```
BEGIN;

DELETE FROM cesty
WHERE id = 12;

COMMIT;
```

```
WITH inserted(id) AS (
  INSERT INTO cesty ...
) UPDATE cesty ...;
```

Z dotazu `WITH ...` se provedl INSERT.

PG <= 9.4

- Složitý plán (vysoká cena)
- INSERT skončí chybou, pokud stejnou hodnotu klíče současně vkládá jiná transakce.
- Provádíme-li UPDATE na řádku, který zrovna jiná transakce maže, naše změny se ztratí. (Kdyby však konkurentní DELETE skončil o něco dříve, náš dotaz by provedl INSERT).

PG >= 9.5

```
INSERT INTO cesty(id, uzly)
SELECT id, uzly
FROM cesty_tmp
ON CONFLICT (id) DO
   UPDATE SET uzly = cesty.uzly || EXCLUDED.uzly;

                  QUERY PLAN
-----------------------------------------
 Insert on cesty
   Conflict Resolution: UPDATE
   Conflict Arbiter Indexes: cesty_pkey
   ->  Seq Scan on cesty_tmp
```

PG >= 9.5

- Jednoduchý plán – logika je „uvnitř“ příkazu INSERT.
- Pokud stejnou hodnotu klíče soušasně vkládá jiná transakce, příkaz INSERT počká, jak tato transakce dopadne, a rozhodování INSERT / UPDATE se vrací na začátek.
- Provádíme-li UPDATE na řádku, který zrovna jiná transakce maže, příkaz INSERT počká, jak tato transakce dopadne ...

Jiný typ konfliktu

```
CREATE TABLE oblasti (
        id      int NOT NULL PRIMARY KEY,
        hranice       polygon NOT NULL
);

ALTER TABLE oblasti
ADD CONSTRAINT no_overlap
EXCLUDE USING gist (hranice poly_ops WITH &&);
```

Jiný typ konfliktu

```
INSERT INTO oblasti(id, hranice)
VALUES
(2, polygon '((0, 1), (1, 1), (1, 2), (0, 2))');

INSERT 0 1

INSERT INTO oblasti(id, hranice)
VALUES
(2, polygon '((0, 1), (1, 1), (1, 2), (0, 2))')

ERROR:  conflicting key value violates exclusion
constraint "no_overlap"
DETAIL:  Key (hranice)=(((0,1),(1,1),(1,2),(0,2)))
conflicts with existing key
(hranice)=(((0,0),(1,0),(1,1),(0,1))).
```

Jiný typ konfliktu

```
INSERT INTO oblasti(id, hranice)
VALUES
(2, polygon '((0, 1), (1, 1), (1, 2), (0, 2))')
ON CONFLICT ON CONSTRAINT no_overlap
   DO NOTHING;

INSERT 0 0

                  QUERY PLAN
-----------------------------------------
 Insert on oblasti
   Conflict Resolution: NOTHING
   Conflict Arbiter Indexes: no_overlap
   ->  Result
```

Logika UPSERTu

```
def upsert(row):
    while True:
        if constraint_conflict(row):
            if heap_update(row):
                return
            else:
                continue
        else:
            heap_insert(row)
            if index_tuple_inserted(row):
                return
            else:
                heap_remove(row)
                continue
```