

Uitbreidingen op GROUP BY

wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0
Unported Licentie

Gebruik

Vaak om data te aggregeren (sammennemen)
:: typische aggregatie functies zoals SUM, ...

Uitbreiding

Sleutelwoorden: CUBE, ROLLUP, GROUPING SETS

Meer complex voorbeeld

```
SELECT      avg(totaal)
FROM        (SELECT      spelersnr, sum(bedrag) as totaal
            FROM        boetes
            GROUP BY    spelersnr) as totalen
WHERE       spelersnr IN
            (SELECT      spelersnr
            FROM        spelers
            WHERE       plaats = 'Den Haag'
            OR          plaats = 'Rijswijk')
```

En deze?

```
SELECT B1.betalingsnr, B1.bedrag, sum(B2.bedrag)
FROM boetes as B1, boetes as B2
WHERE B1.betalingsnr >= B2.betalingsnr
GROUP BY B1.betalingsnr, B1.bedrag
ORDER BY B1.betalingsnr
```

ROLLUP

- Verschillende aggregatieneveaus in één instructie
- Als het ware *opgerold*

Vb.

```
SELECT      spelersnr, sum(bedrag)
FROM        boetes
GROUP BY    ROLLUP (spelersnr);
```

ROLLUP uitvoer

```
SELECT      spelersnr, sum(bedrag)
FROM        boetes
GROUP BY   ROLLUP (spelersnr) ;
```

Spelersnr	Sum
6	100
27	175
44	130
8	25
104	50
Null	480

ROLLUP met 2 niveaus

Voorbeeld:

```
SELECT    plaats, spelersnr, sum(bedrag)
FROM      boetes inner join spelers USING (spelersnr)
GROUP BY ROLLUP (plaats, spelersnr);
```

Uitvoer?

ROLLUP met 2 niveaus

- Voorbeeld:

```
SELECT plaats, spelersnr, sum(bedrag)
FROM boetes inner join spelers USING (spelersnr)
GROUP BY ROLLUP (plaats, spelersnr);
```
- Per plaats:
 - Per spelersnr
 - De som
 - De som
 - Gevolgd door de totale som
 - Er wordt als ware van achter naar voor *opgerold*
 - De volgorde in de GROUP BY is dus belangrijk voor ROLLUP
 - Door welke queries kan je ook bovenstaande informatie verkrijgen?

ROLLUP met 2 niveaus uitvoer

Voorbeeld:

```
SELECT    plaats, spelersnr, sum(bedrag)
FROM      boetes inner join spelers USING (spelersnr)
GROUP BY ROLLUP (plaats, spelersnr);
```

=

```
SELECT    plaats, spelersnr, sum(bedrag)
FROM      boetes inner join spelers USING (spelersnr)
GROUP BY plaats, spelersnr
UNION
SELECT    plaats, null, sum(bedrag)
FROM      boetes inner join spelers USING (spelersnr)
GROUP BY plaats
UNION
SELECT    null, null, sum(bedrag)
FROM      boetes inner join spelers USING (spelersnr);
```

ROLLUP met 2 niveaus uitvoer

Voorbeeld:

```
SELECT    plaats, spelersnr, sum(bedrag)
FROM      boetes inner join spelers USING (spelersnr)
GROUP BY ROLLUP (plaats, spelersnr);
```

Uitvoer?

plaats	spelersnr	sum
Den Haag	6	100.00
Den Haag		100.00
Rijswijk	8	25.00
Rijswijk	44	130.00
Rijswijk		155.00
Zoetermeer	27	175.00
Zoetermeer	104	50.00
Zoetermeer		225.00
		480.00

(9 rows)

CUBE

Vergelijkbaar met ROLLUP, maar groepeert voor elke mogelijke combinatie van de meegegeven kolommen.

→ Eigenlijk voor elke mogelijke invalshoek

Voorbeeld:

```
SELECT plaats, spelersnr, sum(bedrag)
  FROM boetes inner join spelers USING (spelersnr)
 GROUP BY CUBE (plaats, spelersnr)
```

Uitvoer:

- Per speler en plaats
- Per plaats
- Per spelers
- Voor alle samen

Speelt de volgorde bij CUBE een rol?

CUBE voorbeeld

Meerdere groeperingen binnen één instructie

Voorbeeld:

```
SELECT      row_number() over () as volgnr,  
            geslacht, plaats, count(*)  
  
FROM        spelers  
  
GROUP BY    CUBE (geslacht, plaats)  
  
ORDER BY    geslacht, plaats
```

CUBE Uitvoer

volgnr	geslacht	plaats	count
1	M	Den Haag	7
2	M	Rijswijk	1
3	M	Voorburg	1
4	M		9
5	V	Leiden	1
6	V	Rijswijk	1
7	V	Rotterdam	1
8	V	Zoetermeer	2
9	V		5
11		Den Haag	7
12		Leiden	1
13		Rijswijk	2
14		Rotterdam	1
15		Voorburg	1
16		Zoetermeer	2
10			14
(16 rows)			

GROUPING SETS

- Uitgebreide vorm van GROUP BY
- Meer mogelijkheden
bv.

```
SELECT geslacht, plaats, count(*)  
FROM spelers  
GROUP BY GROUPING SETS ((plaats),(geslacht))  
ORDER BY 2, 1
```

- GROUP BY() : alle rijen in één groep
bv.

```
SELECT geslacht, plaats, count(*)  
FROM spelers  
GROUP BY GROUPING SETS ((geslacht, plaats),(geslacht),())  
ORDER BY 1, 2
```

GROUPING SETS vs. ROLLUP

`ROLLUP(c1,c2,c3)` == `GROUPING SETS (`
`(c1, c2, c3),`
`(c1, c2),`
`(c1),`
`()`
`)`

GROUPING SETS vs. CUBE

CUBE(c1,c2,c3) ==

ALLE combinaties

GROUPING SETS (
(c1,c2,c3),
(c1,c2),
(c1,c3),
(c2,c3),
(c1),
(c2),
(c3),
(
))

Combinaties

- Meerdere groeperingen zijn samen mogelijk
- Mogelijkheden :
 - 1 grouping set + 1 simple : toevoeging
 - 2 or meerder grouping sets : « vermenigvuldiging » van specificaties vergelijkbaar met cartesisch produkt
 - Meerdere grouping sets samen : omzetting
 - Bv. GROUP BY GROUPING SETS (E1,E2),E3
= GROUP BY GROUPING SETS ((E1,E3),(E2,E3))
- Union !

Grouping sets

- GROUP BY a, CUBE (b, c), GROUPING SETS ((d), (e))
- GROUP BY GROUPING SETS (
 (a, b, c, d), (a, b, c, e),
 (a, b, d), (a, b, e),
 (a, c, d), (a, c, e),
 (a, d), (a, e)
)

GROUP BY DISTINCT

- GROUP BY ROLLUP (a, b), ROLLUP (a, c)

- GROUP BY GROUPING SETS (

(a, b, c),

(a, b),

(a, b),

(a, c),

(a),

(a),

(a, c),

(a),

()

)

GROUP BY DISTINCT

- GROUP BY DISTINCT ROLLUP (a, b), ROLLUP (a, c)
- GROUP BY GROUPING SETS (
 (a, b, c),
 (**a**, b),
 (**a**, c),
 (**a**),
 ()
)
- <> SELECT DISTINCT

Wim Bertels (CC)BY-SA-NC

Referenties:

- Slides Avanced Group By, 2016, P. De Mazière
- https://www.postgresql.org/docs/current/queries-table-expressions.html#QUERIES-GROUPING-SET_S

FILTER

wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0
Unported Licentie

Bij aggregatie

- De doorgegeven waarden voor de aggregatie beperken, filteren.
- aggregate_name (expression [, ...] [order_by_clause]) [FILTER (WHERE filter_clause)]
- aggregate_name (ALL expression [, ...] [order_by_clause]) [FILTER (WHERE filter_clause)]
- aggregate_name (DISTINCT expression [, ...] [order_by_clause]) [FILTER (WHERE filter_clause)]
- aggregate_name (*) [FILTER (WHERE filter_clause)]

Voorbeeld

```
SELECT spelersnr, sum(bedrag), sum(bedrag)
      FILTER (WHERE bedrag>50)

FROM boetes

GROUP BY spelersnr;
```

Voorbeeld en uitvoer

```
SELECT spelersnr, sum(bedrag), sum(bedrag)
      FILTER (WHERE bedrag>50)

FROM boetes

GROUP BY spelersnr;
```

spelersnr	sum	sum
6	100.00	100.00
27	175.00	175.00
44	130.00	75.00
8	25.00	
104	50.00	

(5 rows)

Oefening

Geef het spelernummer en het aantal boetes voor alle spelers met boetes, evenals het aantal boetes waarvoor het bedrag hoger of gelijk is aan 50 euro.

Tijd voor een micropauze

Oplossing

Geef het spelernummer en het aantal boetes voor alle spelers met boetes, evenals het aantal boetes waarvoor het bedrag hoger of gelijk is aan 50 euro.

```
SELECT spelersnr, count(*) as "aantal_boetes",
       count(*) FILTER (WHERE bedrag >= 50) as "aantal
boetes
>= 50"
FROM boetes
GROUP BY spelersnr;
```

Extra oefening constraint: output

```
SELECT spelersnr, count(*) as "aantal_boetes",
       count(*) FILTER (WHERE bedrag >= 50) as "aantal
boetes >= 50"
FROM boetes
GROUP BY spelersnr;
```

spelersnr integer	aantal_boetes bigint	aantal boetes >= 50 bigint
8	1	0
44	3	1
6	1	1
27	2	2
104	1	1

Wim Bertels (CC)BY-SA-NC

Referenties:

<https://www.postgresql.org/docs/current/sql-expressions.html>

PATRONEN VERGELIJKEN

SIMILAR TO & LIKE

wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Unported
Licentie

Patronen vergelijken

- LIKE – operator (Sql standaard)
- SIMILAR TO – operator (sql standaard sinds 1999)
- Reguliere expressies, POSIX vorm, geen sql standaard, net als ILIKE
- <http://www.postgresql.org/docs/current/static/functions-matching.html>

LIKE operator

- Aanvulling
 - **WHERE naam LIKE '%_%' ESCAPE '\'**
 - Het escappen van tekens zorgt ervoor dat deze gebruikt worden *zoals ze zijn, zonder ze te interpreteren*. Dit is handig om _ en % als expliciet teken aan te geven in een query
 - **WHERE naam NOT LIKE 'br0l'**

SIMILAR TO operator

- Zoals LIKE operator met % en _ en ESCAPE
- Extra's zoals in 'reguliere' expressies:
 - | staat voor of
 - * staat voor een mogelijke herhaling (0,1,...)
 - + staat voor een mogelijke herhaling (1,2,...)
 - ? staat voor nul of 1 herhaling (0,1)
 - {m} exact m herhalingen
 - {m,} m of meer herhalingen
 - {m,n} minstens m, maximaal n herhalingen
 - () om te samen te nemen
 - [] analoog aan gewone reguliere expressies (posix) bv [abc]

SIMILAR TO operator: voorbeelden

- 'abc' SIMILAR TO 'abc'
- 'abc' SIMILAR TO 'a'
- 'abc' SIMILAR TO '%(b|d)%'
- 'abc' SIMILAR TO '(b|c)%'
- 'abbc' SIMILAR TO 'ab+_'
- 'a' SIMILAR TO 'ab+'
- 'a' SIMILAR TO 'ab*''
- 'a' SIMILAR TO 'ab?'
- 'abbz' SIMILAR TO 'ab*[xyz]'
- 'aba' SIMILAR TO '(ab*){2,}'
- 'abaaz' SIMILAR TO '(ab*){1,2}[az]+'

SIMILAR TO operator: voorbeelden

- 'abc' SIMILAR TO 'abc' true
- 'abc' SIMILAR TO 'a' false
- 'abc' SIMILAR TO '%(b|d)%' true
- 'abc' SIMILAR TO '(b|c)%' false
- 'abbc' SIMILAR TO 'ab+_-' true
- 'a' SIMILAR TO 'ab+' false
- 'a' SIMILAR TO 'ab*' true
- 'abbz' SIMILAR TO 'ab*[xyz]' true
- 'aba' SIMILAR TO '(ab*){2,}' true
- 'abaaz' SIMILAR TO '(ab*){1,2}[az]+' true

UPPERCASE and lowercase

- Postgresql specifiek: ILIKE
- Algemeen:
 - gebruik bv. een functie om alles om te zetten naar kleine letters

```
SELECT *  
FROM   table t  
WHERE  lower(t.field) SIMILAR TO  
       'somelowercasestring';
```

- <http://www.postgresql.org/docs/current/interactive/functions-string.html>

Andere operatoren

- BETWEEN
 - WHERE x between 1 and 100**
- OVERLAPS
 - WHERE (datum1,datum2)**
 - OVERLAPS (datumA, datumB)**
- IS NULL
 - WHERE x IS NULL** --gebruik niet x=null
- NOT (ontkenning)

Enkele conditionele functies

- COALESCE
 - SELECT coalesce(null,'(sch)elvis');
- NULLIF
 - SELECT nullif(7,5);
 - SELECT nullif(5,5);
- GREATEST
 - SELECT greatest(gewonnen, verloren), gewonnen
FROM wedstrijden;
- LEAST

Venster functies

OVER ()

Window functions

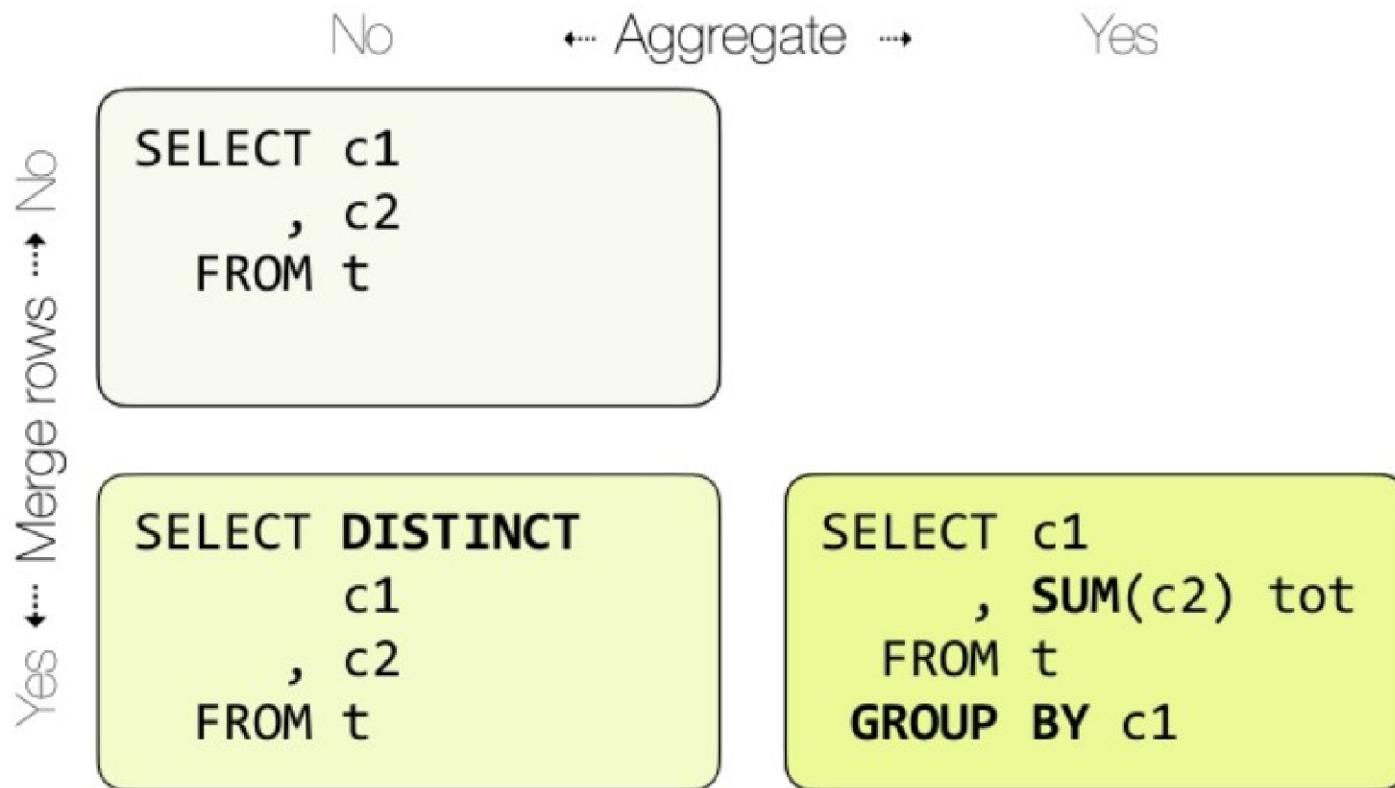
Wim.bertels@ucll.be

Ander perspectief op 2 concepten: samenvoegen en aggregeren

- Samenvoegen van rijen op basis van een eigenschap
 - GROUP BY : meer opties
 - DISTINCT : dubbele rijen verwijderen
- Aggregeren van data van verwante rijen
 - GROUP BY nodig om de groepen te vormen
 - Aggregatie functies, bv count, sum

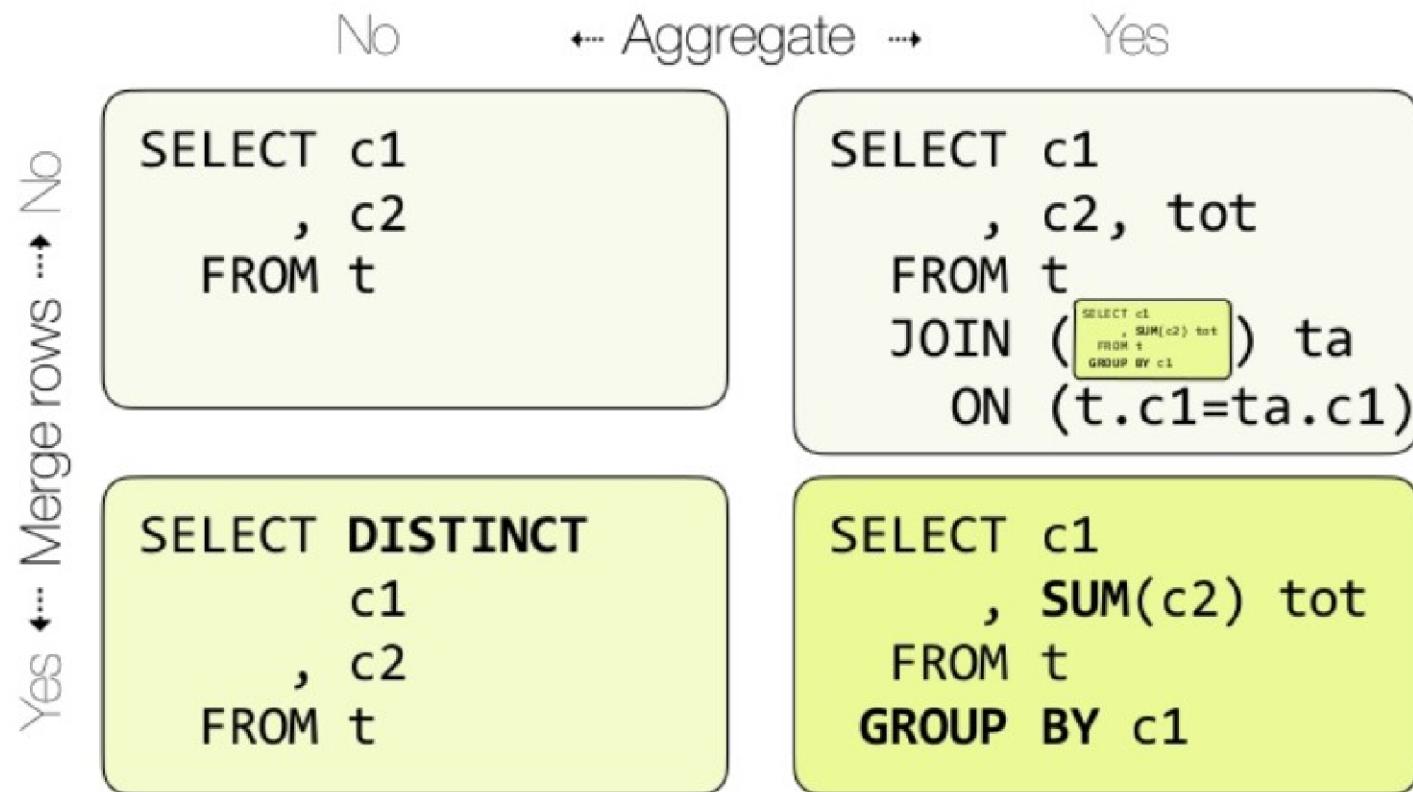
OVER (PARTITION BY)

The Problem



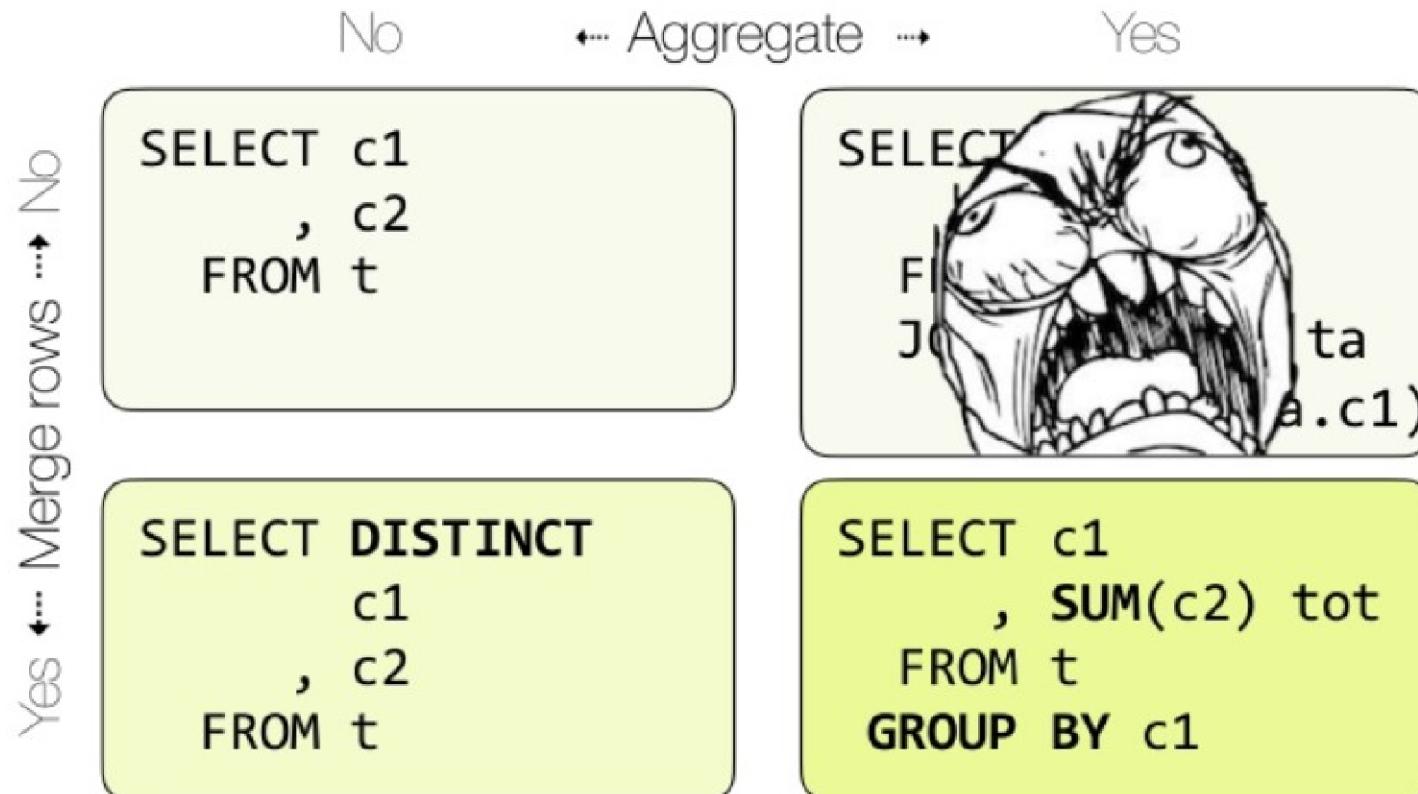
OVER (PARTITION BY)

The Problem



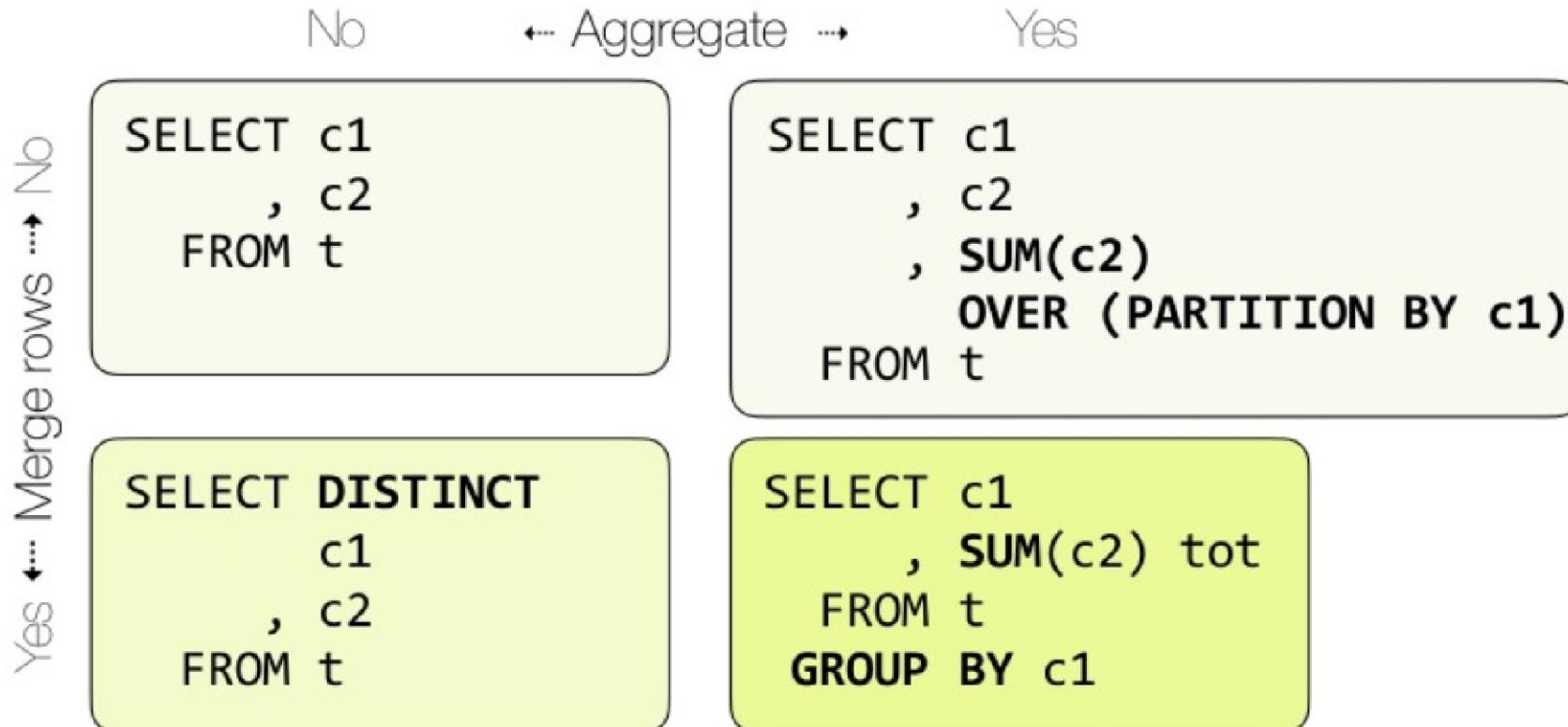
OVER (PARTITION BY)

The Problem



OVER (PARTITION BY)

Since SQL:2003



Nummerings functies

--voeg een rijnummer toe

```
SELECT row_number() OVER(), spelersnr  
FROM spelers  
WHERE geslacht = 'M';
```

--zijn deze rijnummers vast bepaald?

Nummeringsfuncties

```
SELECT row_number() OVER(),      row_number | spelersnr
       spelersnr
-----+-----
FROM   spelers
       1 |     2
WHERE  geslacht = 'M';
       2 |     6
       3 |     7
       4 |    39
--zijn deze rijnummers vast bepaald?      5 |    44
                                           6 |    57
                                           7 |    83
                                           8 |    95
                                           9 |   100
                                         (9 rows)
```

row_number()

--voeg een rijnummer toe

```
SELECT row_number() OVER(), spelersnr  
FROM spelers  
WHERE geslacht = 'M'  
ORDER BY 2;
```

--met 'vaste' rijnummers ?

--ORDER BY komt na SELECT

OVER()

--volgorde van rijnummers manipuleren

```
SELECT row_number() OVER(ORDER BY spelersnr),  
plaats, spelersnr  
FROM spelers  
WHERE geslacht = 'M'  
ORDER BY plaats NULLS FIRST;  
--ORDER BY heeft nog enkele opties
```

row_number() OVER(ORDER BY)

```
SELECT row_number()  
      OVER(ORDER BY spelersnr),  
    plaats, spelersnr  
  FROM spelers  
 WHERE geslacht = 'M'  
 ORDER BY plaats NULLS FIRST;
```

	row_number	plaats	spelersnr
	1	Den Haag	2
	2	Den Haag	6
	3	Den Haag	7
	4	Den Haag	39
	7	Den Haag	83
	9	Den Haag	100
	6	Den Haag	57
	5	Rijswijk	44
	8	Voorburg	95

(9 rows)

rank()

--vergelijkbaar met f_i in een frequentietabel

--volgens de sorteervolgorde

--cf totaal aantal plaatsen

```
SELECT rank() OVER(ORDER BY plaats), plaats  
FROM spelers  
WHERE geslacht = 'M'  
ORDER BY plaats NULLS LAST;
```

rank() OVER(ORDER BY)

```
SELECT rank() OVER(ORDER BY plaats),      rank | plaats
      plaats
-----+-----
FROM   spelers                         1 | Den Haag
WHERE  geslacht = 'M'                   1 | Den Haag
ORDER BY plaats NULLS LAST;          1 | Den Haag
                                         8 | Rijswijk
                                         9 | Voorburg
                                         (9 rows)
```

rank() OVER(ORDER BY)

Pseudo code van voorbeeld in vorige slide

- Rangschikt de rijen gesorteerd op plaats
- Doorloopt de rijen
- Voor elke rij:
 - IF de huidige rij de eerste rij is in de partitie - geef 1 aan
 - ELSE IF plaats is gelijk aan vorige plaats - geef vorige rank aan
 - ELSE geef de telling van de rijen aan tot nu toe

dense_rank()

- zoals de index voor x_i in een frequentietabel
- cf distinct aantal plaatsen

```
SELECT dense_rank() OVER(ORDER BY plaats), plaats
FROM spelers
WHERE geslacht = 'M'
ORDER BY plaats NULLS LAST;
```

rank() OVER(ORDER BY)

```
SELECT dense_rank()  
      OVER(ORDER BY plaats),  
            plaats  
FROM   spelers  
WHERE  geslacht = 'M'  
ORDER BY plaats NULLS LAST;
```

dense_rank	plaats
1	Den Haag
2	Rijswijk
3	Voorburg

(9 rows)

Partitioneren

- Vergelijkbaar concept van GROUP BY, maar fijner, anders
- Per groep/partitie wordt de aggregatie of venster functie toegepast
- Venster functies :
 - in SELECT en ORDER BY
- Bv nummeringsfuncties ea venster functies
<http://www.postgresql.org/docs/current/interactive/functions-window.html>
- Aggregatie functies :
<http://www.postgresql.org/docs/current/interactive/functions-aggregate.html>

Merk op

In tegenstelling tot aggregatiefuncties

- vensters groeperen rijen niet in een enkele output rij
- de rijen behouden hun identiteit
- (elke aggregatie functie kan worden gebruikt als venster functie, niet omgekeerd)

PARTITION BY

--voeg een rijnummer toe per plaats

```
SELECT row_number() OVER(partition BY plaats),  
plaats, spelersnr  
FROM spelers  
WHERE geslacht = 'M'  
ORDER BY 2;
```

--betekenis rijnummer?!

row_number() OVER(PARTITION BY)

```
SELECT row_number()           row_number | plaats | spelersnr
      OVER(partition BY plaats),-----+-----+-----+
          plaats, spelersnr
FROM   spelers
WHERE  geslacht = 'M'
ORDER BY 2;
```

--betekenis rijnummer?!

	row_number	plaats	spelersnr
	1	Den Haag	2
	2	Den Haag	6
	3	Den Haag	7
	4	Den Haag	39
	5	Den Haag	83
	6	Den Haag	100
	7	Den Haag	57
	1	Rijswijk	44
	1	Voorburg	95

(9 rows)

PARTITION BY ORDER BY

--voeg een rijnummer toe per plaats

--vaste volgorde

```
SELECT    row_number()
          OVER(partition BY plaats ORDER BY spelersnr),
          plaats, spelersnr
FROM      spelers
WHERE     geslacht = 'M'
ORDER BY  2;  
--pk spelersnr om volgorde vast te leggen
```

OVER(PARTITION BY ORDER BY)

```
SELECT row_number()
      OVER(partition BY plaats
            ORDER BY spelersnr),
      plaats, spelersnr
   FROM spelers
 WHERE geslacht = 'M'
 ORDER BY 2;
```

row_number	plaats	spelersnr
1	Den Haag	2
2	Den Haag	6
3	Den Haag	7
4	Den Haag	39
5	Den Haag	57
6	Den Haag	83
7	Den Haag	100
1	Rijswijk	44
1	Voorburg	95

(9 rows)

ORDER BY

--ORDER BY bepaalt nu de groepjes

--cumulatieve som

```
SELECT sum(jaartoe) OVER(ORDER BY jaartoe),  
       jaartoe  
  FROM spelers  
--ORDER BY 2;
```

OVER(PARTITION BY ORDER BY)

```
SELECT sum(jaartoe)           sum | jaartoe
      ,  
      OVER(ORDER BY jaartoe),-----+-----  
      jaartoe  
FROM   spelers;               1972 | 1972  
                                3947 | 1975  
                                5924 | 1977  
                                7903 | 1979  
                                13843 | 1980  
                                13843 | 1980  
                                13843 | 1980  
                                . .  
                                27725 | 1985  
(14 rows)
```

SELECT-instructie

SELECT
FROM
WHERE
GROUP BY
HAVING
WINDOW
-- window-component

SELECT sum(jaartoe) OVER w1,
sum(jaartoe) OVER w2,
avg(jaartoe) OVER w2,
jaartoe
FROM spelers
WINDOW w1 AS (ORDER BY jaartoe),
w2 AS (PARTITION BY plaats);

-- <https://www.postgresql.org/docs/current/static/queries-table-expressions.html#QUERIES-WINDOW>

ORDER BY

OVER() vs GROUP BY

- OVER()
 - Gebruik venster functies
 - Gebruik aggregatie functies
- GROUP BY
 - Gebruik aggregatie functies
 - Andere 'positie', volgorde van 'verwerking'
 - Venster <> identiek aan groepering
 - Per groep heb je maar 1 waarde
 - Per venster kan je meerdere waarden hebben
 - Toon per speler zijn totaal aantal boetes (2).

OVER() vs GROUP BY

--groepering

```
SELECT spelersnr, sum(bedrag)
FROM boetes
GROUP BY spelersnr;
```

--venster

```
SELECT spelersnr, sum(bedrag)
      OVER(PARTITION BY spelersnr)
FROM boetes;
```

--distinct..?!

RANGE

--cumulatief op spelersnr de boetesommen geven

```
SELECT spelersnr, sum(bedrag) OVER(order by spelersnr  
rows between unbounded preceding and current row)
```

```
FROM boetes
```

```
ORDER BY 1;
```

--kan gecombineerd worden met partition by

GELIJKE ORDE

```
SELECT spelersnr, sum(bedrag)          spelersnr | sum
      OVER(order by spelersnr)        -----
FROM   boetes                         6 | 100.00
ORDER BY 1;                          8 | 125.00
                                         27 | 300.00
                                         27 | 300.00
                                         44 | 430.00
                                         44 | 430.00
                                         44 | 430.00
                                         104 | 480.00
                                         (8 rows)
```

GELIJKE ORDE

- Twee rijen zijn equivalent voor ORDER BY als hun volgorde verwisselbaar is wanneer ze geordend zijn door ORDER BY

spelersnr	sum
6	100.00
8	125.00
27	300.00
27	300.00
44	430.00
44	430.00
44	430.00
104	480.00

(8 rows)

OVER(ORDER BY .. ROWS ..)

```
SELECT spelersnr, sum(bedrag)          spelersnr | sum
      OVER(order by spelersnr           -----
rows between unbounded              6 | 100.00
preceding and current row)         8 | 125.00
FROM boetes                         27 | 200.00
ORDER BY 1;                          27 | 300.00
                                         44 | 325.00
                                         44 | 355.00
                                         44 | 430.00
                                         104 | 480.00
                                         (8 rows)
```

RANGE parameters

Voor de optionele kader_clausule kan je kiezen tussen

- ROWS kader_start
- ROWS BETWEEN kader_start AND kader_einde

terwijl kader_start en kader_einde één van de volgende opties is

- UNBOUNDED PRECEDING
- waarde PRECEDING (bv 3 preceding, enkel bij ROWS)
- CURRENT ROW
- waarde FOLLOWING (bv 4 following, enkel bij ROWS)
- UNBOUNDED FOLLOWING

ROWS vs RANGE

- In RANGE modus, CURRENT ROW voor een kader_start betekent dat het kader start met de eerste rij die (volgens ORDER BY) op gelijke orde staat met de huidige rij, terwijl CURRENT ROW voor een kader_einde betekent dat het kader eindigt met de laatste rij die (volgens ORDER BY) op gelijke orde staat.
- In ROWS modus, CURRENT ROW betekent simpelweg de huidige rij.

RANGE vs ROWS

spelersnr	sum
-----------	-----

6	100.00
---	--------

8	125.00
---	--------

27	300.00
----	--------

27	300.00
----	--------

44	430.00
----	--------

44	430.00
----	--------

44	430.00
----	--------

104	480.00
-----	--------

(8 rows)

spelersnr	sum
-----------	-----

6	100.00
---	--------

8	125.00
---	--------

27	200.00
----	--------

27	300.00
----	--------

44	325.00
----	--------

44	355.00
----	--------

44	430.00
----	--------

104	480.00
-----	--------

(8 rows)

RANGE parameters default

De default kader_clausule is

**RANGE BETWEEN UNBOUNDED PRECEDING AND
CURRENT ROW**

Met ORDER BY wordt het kader bepaald als alle rijen vanaf de start van de partitie tot aan de laatste rij met dezelfde orde (volgens ORDER BY) als de huidige rij.

Zonder ORDER BY worden alle rijen in de partitie gebruikt in het venster kader, omdat alle rijen op gelijke orde staan met de huidige rij.

Gebruik

- Aggregaties zonder GROUP BY
- Lopende totalen, glijdende gemiddelden
- Rang, ..
- Interessante toevoegingen sinds SQL:2011
 - lag, lead, nth_value, first_value, last_value, ..
 - bv lag om te vergelijken met de waarde uit de vorige rij

Voorbeeld met lag

```
SELECT spelersnr, bedrag - lag(bedrag)
      OVER(PARTITION BY spelersnr
            ORDER BY datum) AS evolutie
FROM boetes
ORDER BY 1;
```

spelersnr	bedrag	evolutie
6	100.00	¤
8	25.00	¤
27	100.00	¤
27	75.00	-25.00
44	25.00	¤
44	75.00	50.00
44	30.00	-45.00
104	50.00	¤

Opgave en uitvoer?

Geef voor elke teamkapitein het teamnr, spelersnr en de som van zijn boetes. Voeg als laatste kolom ook de ranking toe, met op plaats 1 de kapitein met de grootste som. Sorteer op de rank.

Voorbeeld Uitvoer:

teamnr	spelersnr	sum	rank
1	6	100	1
2	27	175	2

Wat merk je op?

Uitvoer komt niet overeen met vraagstelling, namelijk de kapitein met kleinste som wordt eerst getoond.

Oplossing

```
SELECT    teamnr, spelersnr, sum(bedrag), rank ()  
          OVER (ORDER BY sum(bedrag) DESC)  
FROM      teams LEFT OUTER JOIN boetes USING (spelersnr)  
GROUP BY teamnr, spelersnr  
ORDER BY 4;  
-- boetesom wordt in dit voorbeeld per team en kapitein berekend  
-- dus de boetes van die kapitein voor dat team
```

Wim Bertels (CC)BY-SA-NC

Referenties:

- * <http://www.postgresql.org/docs/current/interactive/sql-expressions.html#SYNTAX-WINDOW-FUNCTIONS>
 - * <https://www.postgresql.org/docs/current/static/tutorial-window.html>
 - * <https://modern-sql.com/>
- * SQL Leerboek, R. Van der Lans