

Venster functies

OVER ()

Window functions

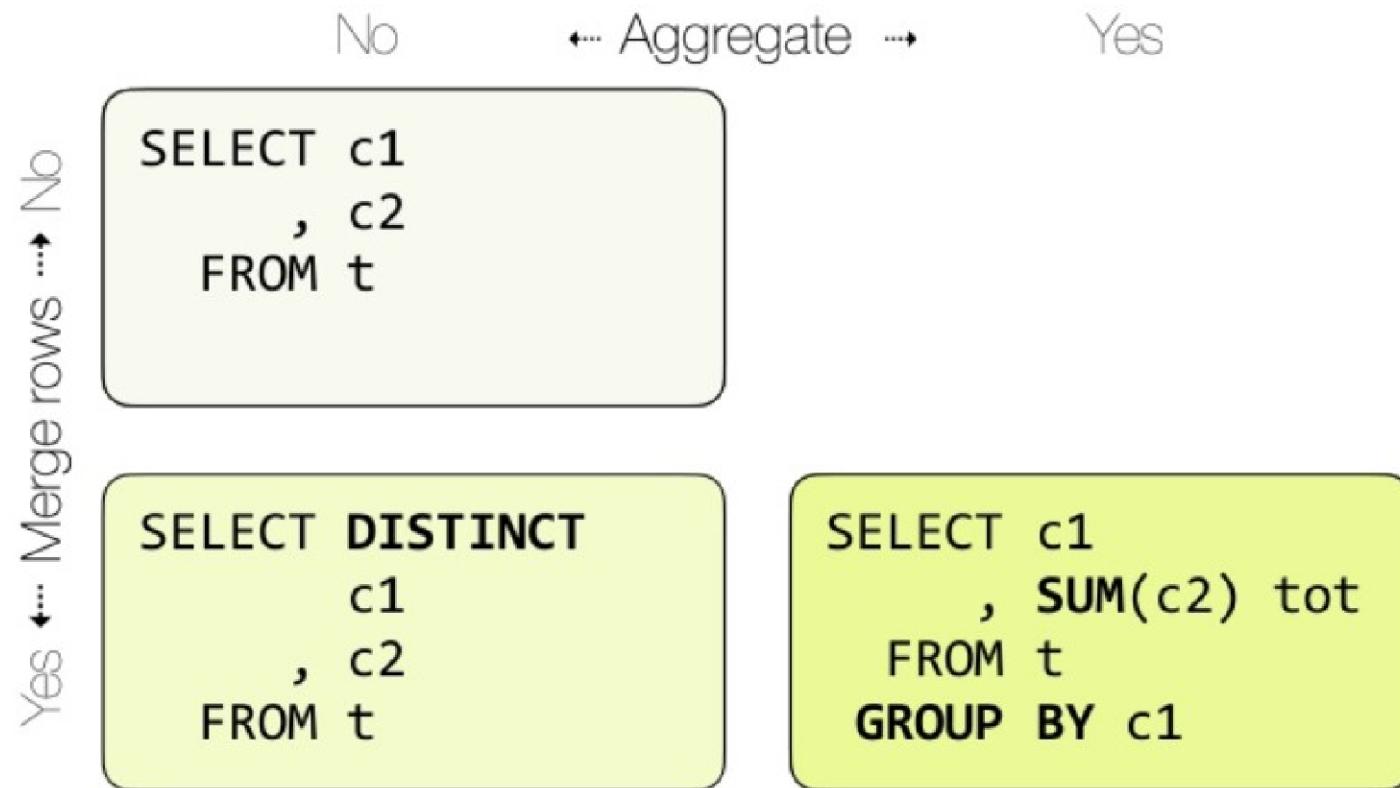
Wim.bertels@ucll.be

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

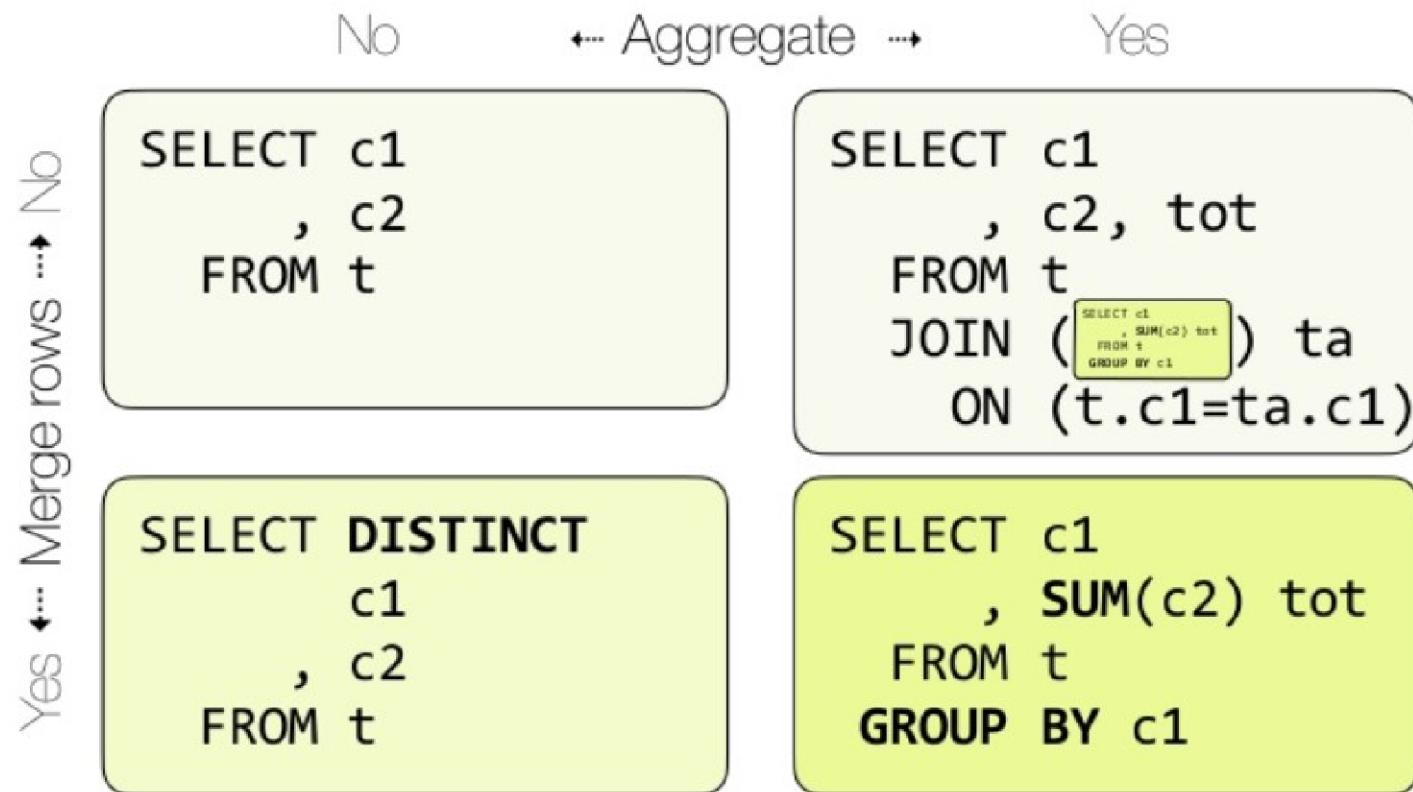
OVER (PARTITION BY)

The Problem



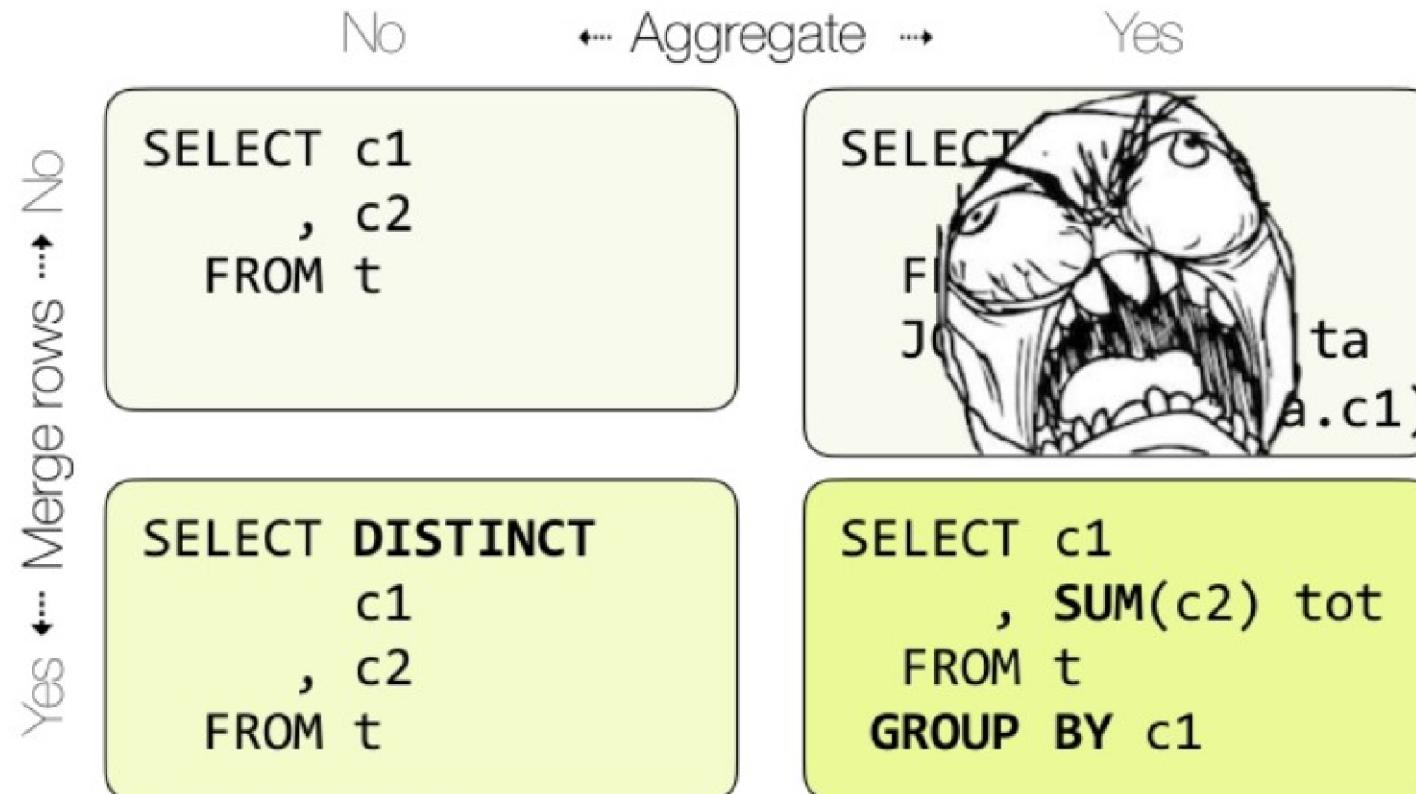
OVER (PARTITION BY)

The Problem



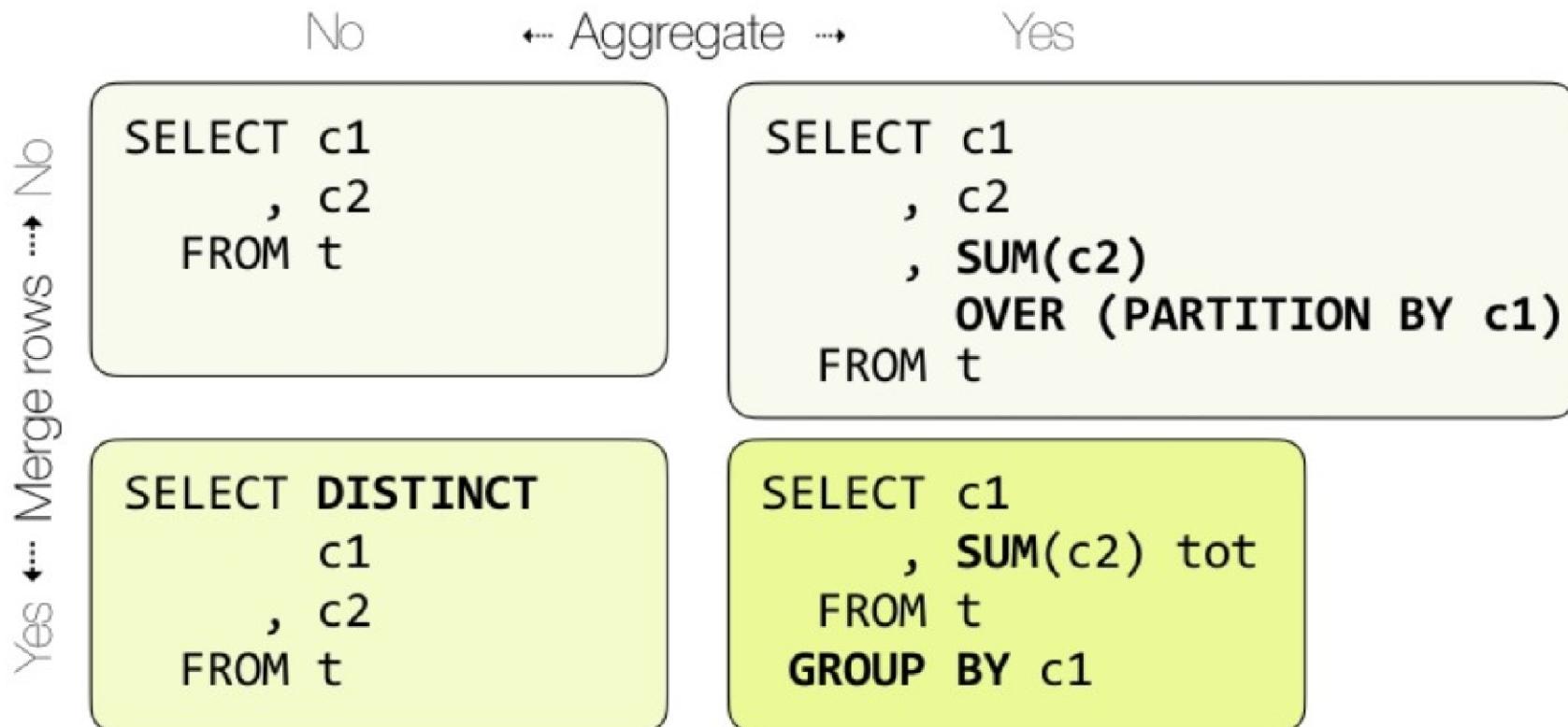
OVER (PARTITION BY)

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
            preceding and current row)
FROM   boetes
ORDER BY 1;
```

	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

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