

SUBQUERIES 2

GECORRELEERDE SUBQUERIES

wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0
Unported Licentie

Soorten subqueries

■ Scalaire subquery:

- *1 rij , 1 kolom => 1 waarde*

■ Rij-subquery:

- *1 rij, meerdere kolommen*

■ Kolom-subquery:

- *Meerdere rijën, 1 kolom*

■ Tabel-subquery:

- *Meerdere rijën, meerdere kolommen*

Subquery in WHERE

Scalaire subquery

- =
- >
- <
- ...

Kolom-subquery

- IN()
- >= ALL()
- ...

Subquery in FROM

- Geef voor alle hemelobjecten die minstens 5 keer bezocht zijn alle reizen die dat hemelobject bezocht hebben. Toon reisnr en objectnaam.

```
SELECT b.reisnr, vb.objectnaam  
FROM bezoeken AS b INNER JOIN (  
    SELECT objectnaam  
    FROM bezoeken  
    GROUP BY objectnaam  
    HAVING COUNT(*) >= 5 ) AS v  
ON b.objectnaam = vb.objectnaam;
```

Hoofdqueries en subqueries

- De hoofdquery krijgt enkel de output van de subquery en weet niets over details zoals: gebruikte tabellen, berekeningen, ...
- Alleen de SELECT wordt dus doorgegeven aan de hoofdquery
- De subquery weet alles over de hoofdquery tot in detail en kan alle gegevens van de hoofdquery gebruiken

Gecorreleerde subqueries

Subquery waarin een kolom wordt
gebruikt die tot een tabel behoort uit een
ander select-blok.

Dus een gecorreleerde subquery kan niet
autonomoos uitgevoerd worden.

Oefening 1

■ Geef voor iedere reis
het bezoek met de
langste verblijfsduur

Oplossing 1

- SELECT b.reisnr, b.objectnaam
FROM bezoeken AS **b**
WHERE b.verblijfsduur =
 (SELECT MAX(verblijfsduur)
 FROM bezoeken AS **allebezoeken**
 WHERE **allebezoeken.reisnr = b.reisnr**);

- Geef voor iedere reis het bezoek met de langste verblijfsduur

Oefening 2

- Geef de spelers die meer keer bestuurslid zijn geweest dan dat ze wedstrijden hebben gespeeld. Toon spelersnr.

Oplossing 2

■ SELECT spelersnr

FROM bestuursleden AS b

GROUP BY spelersnr

HAVING COUNT(*) >

(SELECT COUNT(*)

FROM wedstijden AS w

WHERE w.spelersnr = b.spelersnr):

■ Geef de spelers die meer keer bestuurslid zijn geweest dan dat ze wedstrijden hebben gespeeld.
Toon spelersnr.

EXISTS operator (is er iet, of nie?)

- Kijk of er output BESTAAT voor een subquery
- TRUE of FALSE
- Wat er in de SELECT staat maakt niet uit:
 - *Iets* = TRUE
 - *Niets / empty* = FALSE

EXISTS operator oefening

- Geef alle reizen met een bezoek aan Jupiter. Of:
- = Geef alle reizen waarbij er een bezoek aan Jupiter BESTAAT.
- Toon reisnr en vertrekdatum.

EXISTS operator oplossing

■ SELECT reisnr, vertrekdatum

FROM **reizen**

WHERE EXISTS

(SELECT * , 'is erietofnie'

FROM **bezoeken AS b**

WHERE b.objectnaam = 'Jupiter'

AND **b.reisnr = reizen.reisnr**):

EXISTS operator oplossing

■ SELECT reisnr, vertrekdatum

FROM **reizen**

WHERE EXISTS

```
(SELECT    reisnr  
  FROM      bezoeken AS b  
 WHERE     b.objectnaam = 'Jupiter'  
 AND      b.reisnr = reizen.reisnr);
```

NOT EXISTS operator (als er niks is .. dan..)

- Tegenovergestelde van EXISTS
- Geef alle hemelobjecten doe nog nooit bezocht zijn:
- SELECT h.objectnaam

FROM hemelobjecten AS h

WHERE NOT EXISTS

(SELECT reisnr

FROM bezoeken AS b

WHERE b.objectnaam = h.objectnaam);

En Deze?:

- SELECT objectnaam
FROM hemelobjecten
WHERE NOT EXISTS (
 SELECT reisnr
 FROM bezoeken b
);
- --..

ANY en ALL operatoren

■ Deze operatoren verwachten een rij uitdrukking, om te vergelijken met 1 of meerdere waarden (ALL is de 'voor alle' en ANY is de 'er bestaat' uit de wiskunde)

- $> ALL$ $> ANY$
- $\geq ALL$ $\geq ANY$
- $< ...$
- $\leq ...$

ANY en ALL operator oefening

- Geef de langste reis. Of:
- = Geef de reis waarbij de reisduur groter of gelijk is aan alle reizen.
- Toon reisnr

ANY en ALL operator oplossing

■ Geef de langste reis:

■ SELECT reisnr

FROM reizen

WHERE reisduur >= ALL

(SELECT reisduur

FROM reizen);

ANY en ALL operator, andere oplossing?

■ Geef de langste reis

■ SELECT reisnr

FROM **reizen**

WHERE reisduur > ALL

(SELECT reisduur

FROM **reizen AS anderen**

WHERE **anderen.reisnr <> reizen.reisnr**);

■ -- WAT ALS 2 REIZEN DEZELFDE REISDUUR HEBBEN ???

ANY en ALL operator oefening 2

- Geef alle reizen, behalve de langste reis. Of:
- = Er is minstens 1 reis met een langere reisduur
- Toon reisnr

ANY en ALL operator oplossing 2

■ Geef alle reizen, behalve de langste reis:

■ SELECT reisnr

FROM reizen

WHERE reisduur < ANY

(SELECT reisduur

FROM reizen);

ANY en ALL operator: OPGELET!

- Geef een lijst van alle planeten die groter zijn dan al hun satellieten

- SELECT objectnaam

```
FROM hemelobjecten AS h
WHERE satellietvan = 'Zon'
AND diameter > ALL
  (SELECT diameter
   FROM hemelobjecten AS maan
   WHERE maan.satellietvan = h.objectnaam);
```

- -- Klopt dit?

objectnaam

Mercurius
Venus
Aarde
Mars
Jupiter
Saturnus
Uranus
Neptunus
Pluto
(9 rows)

ANY en ALL operator: OPGELET!

- Geef een lijst van alle planeten die kleiner zijn dan al hun satellieten

```
■ SELECT objectnaam  
      FROM hemelobjecten AS h  
     WHERE satellietvan = 'Zon'  
    AND   diameter < ALL  
          (SELECT diameter  
              FROM hemelobjecten AS maan  
             WHERE maan.satellietvan = h.objectnaam);
```

- -- Daarnet groter, nu kleiner, .. ?

objectnaam

Mercurius
Venus
(2 rows)

ANY en ALL operator: OPGELET!

```
■ SELECT diameter  
      FROM hemelobjecten AS manen  
     WHERE manen.satellietvan IN ('Mercurius', 'Venus');
```

diameter

(0 rows)

(ANY en ALL operator) vs NULL

- Vergelijken met NULL: vaak opnieuw NULL (onbekend) tenzij het niet uitmaakt wat deze null waarde ook zou zijn, of het duidelijk is, of ..
- NULL is een geval apart!
- Wanneer de subquery geen output (NULL) heeft dan:
 - *Geeft ALL: waar / TRUE;*
 - *Geeft ANY: onwaar / FALSE;*

UNIQUE operator

■ Geef de spelers voor wie precies één boete betaald werd.

■ SELECT spelersnr

FROM boetes AS BT

WHERE UNIQUE

(SELECT B.spelersnr

FROM boetes AS B

WHERE **B.spelersnr = BT.spelersnr**);

■ -- Niet geïmplementeerd in PostgreSQL, kan vervangen worden door
HAVING

OVERLAPS operator

■ Geef de spelers en hun functie die in het bestuur zaten van 1 januari 1991 tot en met 31 december 1993

■ SELECT spelersnr, functie
FROM bestuursleden
WHERE (begin_datum, eind_datum)
OVERLAPS ('1991-01-01', '1993-12-31');

Combinatie oefening 1

- Geef de klanten die op een reis zijn meegegaan waar ook klant met klantnr 126 op meegegaan is.
- Toon klantnr

Combinatie oefening 1: oplossing

■ Geef de klanten die op een reis zijn meegegaan waar ook klant met klantnr 126 op meegegaan is.

■ SELECT d.klantnr

FROM klanten AS k INNER JOIN deelnames AS **d** USING(klantnr)

WHERE EXISTS

(SELECT *

FROM deelnames AS **andereDeelnames**

WHERE klantnr = 126

AND **andereDeelnames.reisnr = d.reisnr**)

GROUP BY d.klantnr;

Combinatie oefening 2

- Geef de planeten die bezocht zijn op een reis waar klantrn 126 niet op meeging.
- Toon alle gegevens van de hemelobjecten

Extra oefening 2: oplossing

■ Geef de planeten uit ons zonnestelsel die bezocht zijn op een reis waar klantnr 126 niet op meeging.

■ SELECT *

```
FROM hemelobjecten AS h
```

```
WHERE EXISTS
```

```
(SELECT *
```

```
FROM bezoeken AS b
```

```
WHERE NOT EXISTS
```

```
(SELECT *
```

```
FROM deelnames AS d
```

```
WHERE klantnr = 126 AND d.reisnr = b.reisnr)
```

```
AND h.objectnaam = b.objectnaam)
```

```
AND satellietvan = 'Zon';
```

Uitdaging: Wat doet deze query ?

```
SELECT spelersnr
FROM spelers AS s
WHERE NOT EXISTS
  (SELECT *
   FROM wedstrijden AS w1
   WHERE spelersnr = 57
   AND NOT EXISTS
     (SELECT *
      FROM wedstrijden AS w2
      WHERE w1.teamnr = w2.teamnr
      AND s.spelersnr = w2.spelersnr))
AND spelersnr NOT IN
  (SELECT spelersnr
   FROM wedstrijden
   WHERE teamnr IN
     (SELECT teamnr
      FROM teams
      WHERE teamnr NOT IN
        (SELECT teamnr
         FROM wedstrijden
         WHERE spelersnr = 57))));
```

Wim Bertels (CC)BY-SA-NC

Referenties:

- Slides subqueries deel 1 sql 2012-13, K. Beheydt
- Slides Databanken, H. Martens
- SQL Leerboek, R. Van der Lans

Relationeel model

wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0
Unported Licentie

Model

- Een conceptueel en logisch datamodel
- Heeft een basis in verzamelingleer en predikatenlogica
 - databank = verzameling van relaties
- Normalisatie

Terminologie

- (predikaat variabele > tabelstructuur)
- relatie > tabel(inhoud)
- tuple > rij
- (predikaat > constraints en queries)
- selectie
- projectie
- ..

SQL

- Taal van de meeste (relationele) databanken
- Een relationele databank is gebaseerd op het relationele model
- De implementie van SQL komt niet exact overeen met het relationeel model, bv
 - rijen zijn hoeven niet uniek te zijn (cf verzamelingleer)
 - true/false + null (+ unknown?) (cf logica)

RDBMS Sleutel eigenschappen

- Tabellen (rijen en kolommen)
- Constraints (pk, fk, ..)
- Normalisatie
- ACID
- ..

ACID

- Atomicity: cf boolean
- Consistency: cf state
- Isolation: cf concurrency
- Durability: cf commit, levensduur en select

Tot nu doeleteinden RDBMS

- OLTP: databank die beschikbaar is en transakties verwerkt
- OLAP
- ..

Produkt keuze

wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0
Unported Licentie

Belangrijke punten en keuzes

- Volg je de SQL-standaard of niet ?
- Software ?
- Fouten ?
- Hoe zit het met frameworks ?

SQL Standaard

- Ideale wereld vs. Realiteit
- Ken je opties
- Wat is belangrijk?
- Maak je keuzes

Softwarevereisten

- Zijn er één of meerdere gebruikers?
- Dicht bij de SQL-standaard of niet?
- Snel and gemakkelijk of ..
- Duurzaamheid?
- Is er officiële ondersteuning nodig?
- Systeemvereisten, Besturingssysteem, ..
- Maak je een standalone applicatie of een client/server?
- Softwarelicentie van database belangrijk?
- ..

Eenvoudige bron/kost-classificatie

	Gesloten	Open*
Gratis		
Betalend		

*open en vrij is niet hetzelfde

Eenvoudige bron/kost-classificatie

	Gesloten	Open*
Gratis	Bv DB2 Express-C,..	Bv SQLite,..
Betalend	Bv Oracle,..	Bv Timescale cloud,..

*open en vrij is niet hetzelfde, en “open” als marketing

Eenvoudige vereistenclassificatie

	Lokaal	Netwerk
Kantoor		
KMO		
Groot bedrijf		

Een aantal gratis open-bron databanken

- libh2-java
- SQLite
- HSQLDB
- Firebird
- Derby
- MariaDB
- PostgreSQL
- Virtuoso
- Greenplum
- DuckDB
- ..

Fouten

- Te veel verschillende DBMSn .. vs. tijd
- Ken je gegevens! (structuur) [vereisten]
- DB <> rekenblad of tabel ; eigen vaardigheid
- Datatabasenormalisatie (3NF+) vs. gezond verstand
- (alle) applicatielogica in de databank
- Reserve kopie, replicatie
- Versie beheer
- Gebruik de beschikbare hulpmiddelen
- Met een hamer de loodgieterij doen? (kies het juiste gereedschap)

Hoe zit het met frameworks ea?

- Vraag je af:
 - hoe lang zal de software applicatie meegaan?
 - hoe lang hebben we die data dan nog nodig?
 - ..
 - Bv administratie studentendiploma's of ..
- Gegevenslevensverwachting vergeleken met de softwareomgeving > Waar heb je nu aandacht voor..

*[https://www.postgresql.eu/events/pgconfeu2023/sessions/session/5143/slides/448/Why%20using%20Open%20Source%20PostgreSQL%20matters%20\(1\).pdf](https://www.postgresql.eu/events/pgconfeu2023/sessions/session/5143/slides/448/Why%20using%20Open%20Source%20PostgreSQL%20matters%20(1).pdf)

Joins

wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Unported
Licentie

FROM

- Bevat tabelspecificaties
- Kunnen subqueries zijn (alias niet vergeten)
- Impliciete joins > expliciete joins

Impliciete JOIN =

```
SELECT S.spelersnr  
FROM Spelers as S, anderschema.Woonplaatsen as Stad  
WHERE S.plaats = Stad.plaatsnaam
```

Tabellen uit andere schema's moeten gekwalificeerd worden

Of we doen dit via search_path

search_path

```
SHOW search_path;
```

```
SET      search_path = public, myschema;
```

- Opgepast voor tabellen met dezelfde naam!
- Oplossing?

Expliciete INNER JOIN

```
SELECT    s.spelersnr  
FROM      spelers AS s INNER JOIN anderschema_woonplaatsen AS stad  
WHERE     s.plaats = stad.plaatsnaam;
```

```
SELECT    s.spelersnr  
FROM      spelers as s INNER JOIN anderschema_woonplaatsen AS stad  
          ON (s.plaats = stad.plaatsnaam)
```

- INNER JOIN ... ON (.. operator ..)
- Doorsnede, wat we gewoon zijn

Expliciete FULL OUTER JOIN

```
SELECT spelers.spelersnr, naam, bedrag  
FROM spelers FULL OUTER JOIN boetes  
      USING (spelersnr);
```

	spelersnr integer	naam character (15)	bedrag numeric (7,2)
1	6	Permentier	100.00
2	44	Bakker, de	75.00
3	27	Cools	100.00
4	104	Moerman	50.00
5	44	Bakker, de	25.00
6	8	Niewenburg	25.00
7	44	Bakker, de	30.00
8	27	Cools	75.00
9	39	Bischoff	[null]
10	57	Bohemen, van	[null]
11	2	Elfring	[null]
12	83	Hofland	[null]
13	112	Baalen, van	[null]
14	100	Permentier	[null]
15	28	Cools	[null]
16	95	Meuleman	[null]
17	7	Wijers	[null]

Expliecate FULL OUTER JOIN

```
SELECT spelers.spelersnr, naam, bedrag  
FROM spelers FULL OUTER JOIN boetes  
      USING (spelersnr);
```

- USING veronderstelt gelijke kolomnamen
- Alle rijen uit beide tabellen worden weerhouden (getoond)
- Verschil met ON: er blijft maar 1 rij met spelersnr over
- ON: 2 rijen met spelersnr (uit beide tabellen)

- Alle spelers met hun eventuele boetes?
- Met de naam van de spelers in HOOFDLETTERS

Expliecate LEFT OUTER JOIN

```
SELECT spelers.spelersnr, naam, bedrag  
FROM spelers LEFT OUTER JOIN boetes  
USING (spelersnr)
```

- Alle rijen uit de LINKSE tabel spelers worden weerhouden (getoond)
- HOOFDLETTERS?

	spelersnr integer	naam character (15)	bedrag numeric (7,2)
1	6	Permentier	100.00
2	44	Bakker, de	75.00
3	27	Cools	100.00
4	104	Moerman	50.00
5	44	Bakker, de	25.00
6	8	Nieuwenburg	25.00
7	44	Bakker, de	30.00
8	27	Cools	75.00
9	39	Bischoff	[null]
10	57	Bohemens, van	[null]
11	2	Elfring	[null]
12	83	Hofland	[null]
13	112	Baalen, van	[null]
14	100	Permentier	[null]
15	28	Cools	[null]
16	95	Meuleman	[null]
17	7	Wijers	[null]

Documentatie

- www.postgresql.org
- Daarna pas via een alternatieve bronnen

```
1 | SELECT version();
```

Explicitie LEFT OUTER JOIN

```
SELECT spelers.spelersnr, UPPER(naam), bedrag  
FROM spelers LEFT OUTER JOIN boetes  
USING (spelersnr)
```

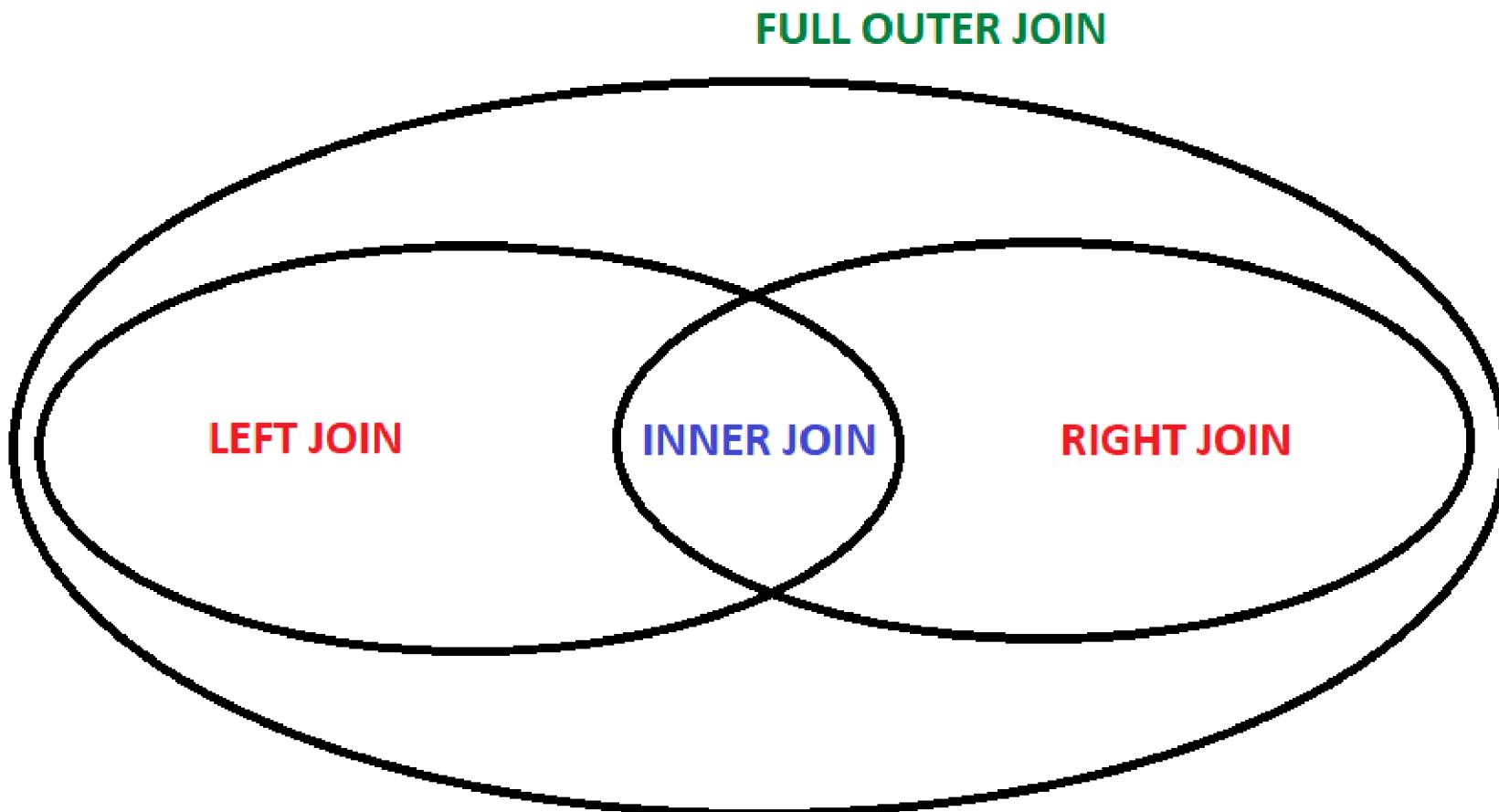
	spelers integer	upper text	bedrag numeric (7,2)
1	6	PERMENTIER	100.00
2	44	BAKKER, DE	75.00
3	27	COOLS	100.00
4	104	MOERMAN	50.00
5	44	BAKKER, DE	25.00
6	8	NIEWENBURG	25.00
7	44	BAKKER, DE	30.00
8	27	COOLS	75.00
9	39	BISCHOFF	[null]
10	57	BOHEMEN, VAN	[null]
11	2	ELFRING	[null]
12	83	HOFLAND	[null]
13	112	BAALEN, VAN	[null]
14	100	PERMENTIER	[null]
15	28	COOLS	[null]
16	95	MEULEMAN	[null]
17	7	WIJERS	[null]

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

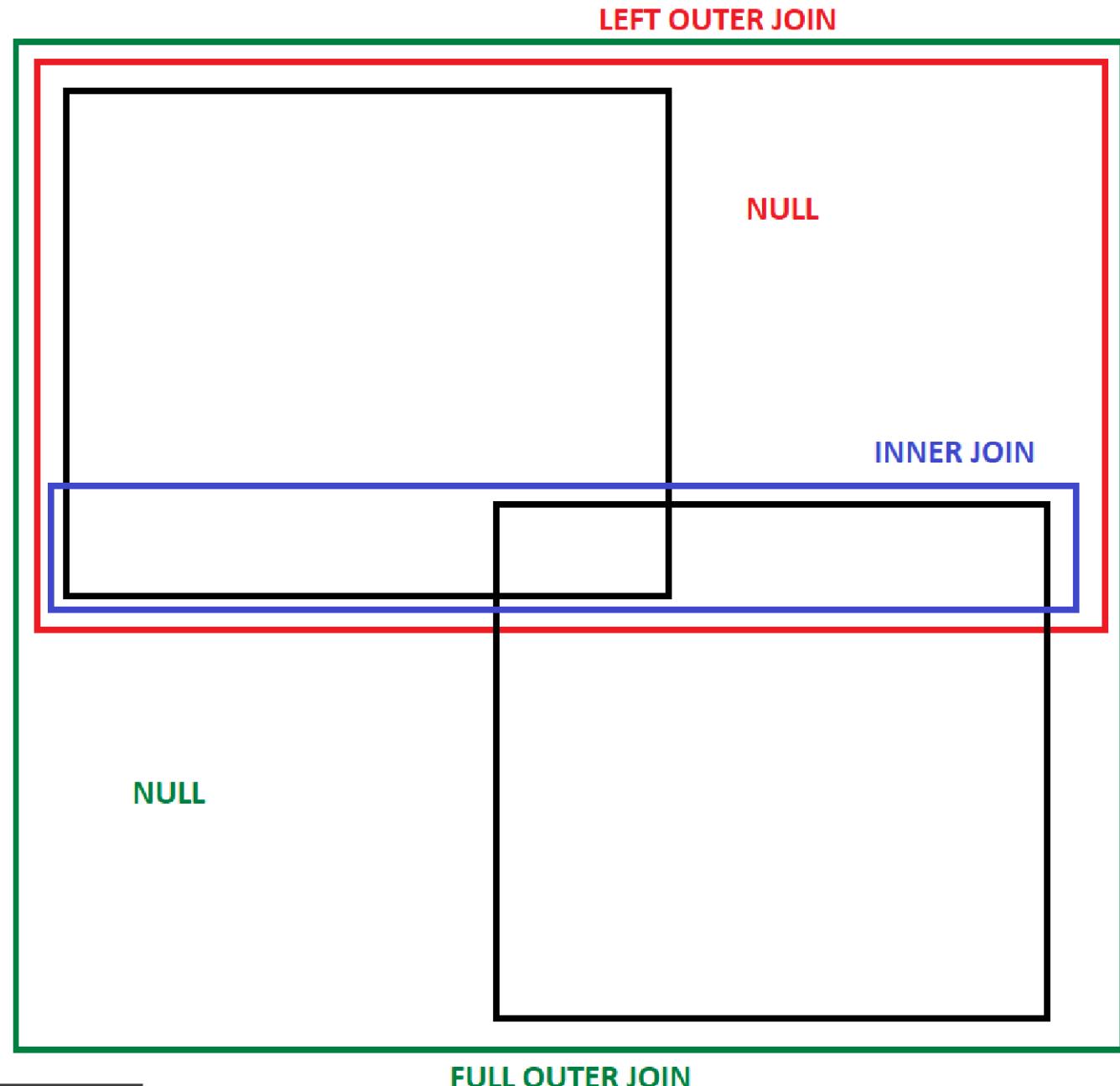
OUTER JOINS

- LEFT OUTER JOIN:
 - Alle rijen uit linker tabel met eventuele bijhorende gegevens uit rechter tabel, anders NULL waardes
- RIGHT OUTER JOIN:
 - Alle rijen uit rechter tabel met eventueel bijhorende gegevens uit linker tabel, anders NULL waardes
- FULL OUTER JOIN:
 - Alle rijen uit linker tabel en alle rijen uit rechter tabel met eventueel bijhorende gegevens uit andere tabel, anders NULL waardes

Verzamelingen



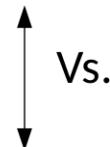
Tabellen



Condities FROM vs. WHERE

- Extra voorwaarden mogen
- Let op: verschil met de WHERE !

```
SELECT teams.spelersnr, teams.teamnr, betalingsnr  
FROM teams LEFT OUTER JOIN boetes  
    ON (teams.spelersnr = boetes.spelersnr)  
WHERE divisie = 'tweede'
```



Vs.

```
SELECT teams.spelersnr, teams.teamnr, betalingsnr  
FROM teams LEFT OUTER JOIN boetes  
    ON (teams.spelersnr = boetes.spelersnr) AND divisie = 'tweede'
```

Query vraag

Geef alle spelers die nog nooit een boete hebben gehad van 50 euro.

Oplossing 1

```
SELECT  *
FROM    spelers
WHERE   spelersnr NOT IN
        (SELECT spelersnr
         FROM   boetes
         WHERE  bedrag = 50);
```

- Niet gecorreleerde oplossing

Oplossing 1 (bis)

	spelersnr integer	naam character (15)	voorletters character (3)	geb_datum date	geslacht character (1)	jaartoe smallint	straat character varying (30)	huisnr character (4)	postcode character (6)	plaats character varying (30)	telefoon character (13)	bondsnr character (4)
1	2	Elfring	R	1948-09-01	M	1975	Steden	43	3575NH	Den Haag	070-237893	2411
2	6	Permentier	R	1964-06-25	M	1977	Hazensteinln	80	1234KK	Den Haag	070-476537	8467
3	7	Wijers	GWS	1963-05-11	M	1981	Erasmusweg	39	9758VB	Den Haag	070-347689	[null]
4	8	Niewenburg	B	1962-07-08	V	1980	Spoorlaan	4	6584WO	Rijswijk	070-458458	2983
5	27	Cools	DD	1964-12-28	V	1983	Liespad	804	8457DK	Zoetermeer	079-234857	2513
6	28	Cools	C	1963-06-22	V	1983	Oudegracht	10	1294QK	Leiden	010-659599	[null]
7	39	Bischoff	D	1956-10-29	M	1980	Ericaplein	78	9629CD	Den Haag	070-393435	[null]
8	44	Bakker, de	E	1963-01-09	M	1980	Lawaalstraat	23	4444LJ	Rijswijk	070-368753	1124
9	57	Bohemen, van	M	1971-08-17	M	1985	Erasmusweg	16	4377CB	Den Haag	070-473458	6409
10	83	Hofland	PK	1956-11-11	M	1982	Mariakade	16a	1812UP	Den Haag	070-353548	1608
11	95	Meuleman	P	1963-05-14	M	1972	Hoofdweg	33a	5746OP	Voorburg	070-867564	[null]
12	100	Permentier	P	1963-02-28	M	1979	Hazensteinln	80	6494SG	Den Haag	070-494593	6524
13	112	Baalen, van	IP	1963-10-01	V	1984	Vosseweg	8	6392LK	Rotterdam	010-548745	1319

Oplossing 2

```
SELECT  *
FROM    spelers s LEFT OUTER JOIN boetes b
        ON (s.spelersnr = b.spelersnr AND b.bedrag = 50)
WHERE   b.spelersnr IS NULL
```

- Oplossing met JOIN

Oplossing 2 (bis)

	spelersnr integer	naam character (15)	voorletters character (3)	geb_datum date	geslacht character (1)	jaartoe smallint	straat character varying (30)	huisnr character (4)	postcode character (6)	plaats character varying (30)	telefoon character (13)	bondsnr character (4)	betelingsnr integer	spelersnr integer	datum date	bedrag numeric (7,2)
1	2	Elfring	R	1948-09-01	M	1975	Steden	43	3575NH	Den Haag	070-237893	2411	[null]	[null]	[null]	[null]
2	6	Permentier	R	1964-06-25	M	1977	Hazensteinln	80	1234KK	Den Haag	070-476537	8467	[null]	[null]	[null]	[null]
3	7	Wijers	GWS	1963-05-11	M	1981	Erasmusweg	39	9758VB	Den Haag	070-347689	[null]	[null]	[null]	[null]	[null]
4	8	Nieuwenburg	B	1962-07-08	V	1980	Spoorlaan	4	6584WO	Rijswijk	070-458458	2983	[null]	[null]	[null]	[null]
5	27	Cools	DD	1964-12-28	V	1983	Liespad	804	8457DK	Zoetermeer	079-234857	2513	[null]	[null]	[null]	[null]
6	28	Cools	C	1963-06-22	V	1983	Oudegracht	10	1294QK	Leiden	010-659599	[null]	[null]	[null]	[null]	[null]
7	39	Bischoff	D	1956-10-29	M	1980	Ericaplein	78	9629CD	Den Haag	070-393435	[null]	[null]	[null]	[null]	[null]
8	44	Bakker, de	E	1963-01-09	M	1980	Lawaaistraat	23	4444LJ	Rijswijk	070-368753	1124	[null]	[null]	[null]	[null]
9	57	Bohemens, van	M	1971-08-17	M	1985	Erasmusweg	16	4377CB	Den Haag	070-473458	6409	[null]	[null]	[null]	[null]
10	83	Hofland	PK	1956-11-11	M	1982	Mariakade	16a	1812UP	Den Haag	070-353548	1608	[null]	[null]	[null]	[null]
11	95	Meuleman	P	1963-05-14	M	1972	Hoofdweg	33a	5746OP	Voorburg	070-867564	[null]	[null]	[null]	[null]	[null]
12	100	Permentier	P	1963-02-28	M	1979	Hazensteinln	80	6494SG	Den Haag	070-494593	6524	[null]	[null]	[null]	[null]
13	112	Baalen, van	IP	1963-10-01	V	1984	Vosseweg	8	6392LK	Rotterdam	010-548745	1319	[null]	[null]	[null]	[null]

CROSS JOIN

- = expliciet cartesisch product

```
SELECT *
FROM teams CROSS JOIN boetes;
```

- Leesbaarheid

UNION JOIN*

- Uit de standaard verwijderd sinds SQL2003
- UNION JOIN: elke rij van elke tabel wordt 1 maal opgenomen en aangevuld met null-waardes voor de kolommen uit de andere tabel

```
SELECT  *
FROM    teams UNION JOIN boetes;
-- (niet ondersteund door postgresql)
```

NATURAL JOIN

- Natuurlijke join, lexicografisch

```
SELECT  *
FROM    teams NATURAL INNER JOIN boetes
WHERE   divisie = 'ere';
```

- hetzelfde als een INNER JOIN USING?
- werkt ook voor andere JOINS (vb. NATURAL RIGHT OUTER JOIN)

EQUI/THETA JOIN

- EQUI JOIN: vergelijking met =
 - THETA JOIN: vergelijking met een andere vergelijkingsoperator
-
- tel hoeveel spelers er zijn met een ander nummer?
 - tel en toon hoeveel spelers er zijn met een even lange naam?
 - we willen telkens de spelersnr, naam en aantal

THETA JOIN

- Tel hoeveel spelers er zijn met een ander nummer
- We willen telkens de spelersnr, naam en aantal

```
SELECT s.spelersnr, s.naam, count(sp.spelersnr)
FROM spelers s INNER JOIN spelers sp
    ON (s.spelersnr <> sp.spelersnr)
GROUP BY s.spelersnr, s.naam
```

	spelersnr integer	naam character (15)	count bigint
1	112	Baalen, van	13
2	83	Hofland	13
3	28	Cools	13
4	7	Wijers	13
5	104	Moerman	13
6	100	Permentier	13
7	2	Elfring	13
8	6	Permentier	13
9	27	Cools	13
10	95	Meuleman	13
11	39	Bischoff	13
12	57	Bohemen, van	13
13	44	Bakker, de	13
14	8	Niewenburg	13

THETA JOIN

- Tel hoeveel spelers er zijn met een even lange naam
- We willen telkens de spelersnr, naam en aantal

THETA JOIN

```
SELECT s.spelersnr, s.naam, count(sp.spelersnr)
FROM spelers s INNER JOIN spelers sp
ON (s.spelersnr <> sp.spelersnr)
WHERE length(s.naam) = length(sp.naam)
GROUP BY s.spelersnr, s.naam
```

```
SELECT s.spelersnr, s.naam, count(sp.spelersnr)
FROM spelers s INNER JOIN spelers sp
ON (s.spelersnr <> sp.spelersnr)
AND length(s.naam) = length(sp.naam)
GROUP BY s.spelersnr, s.naam
```

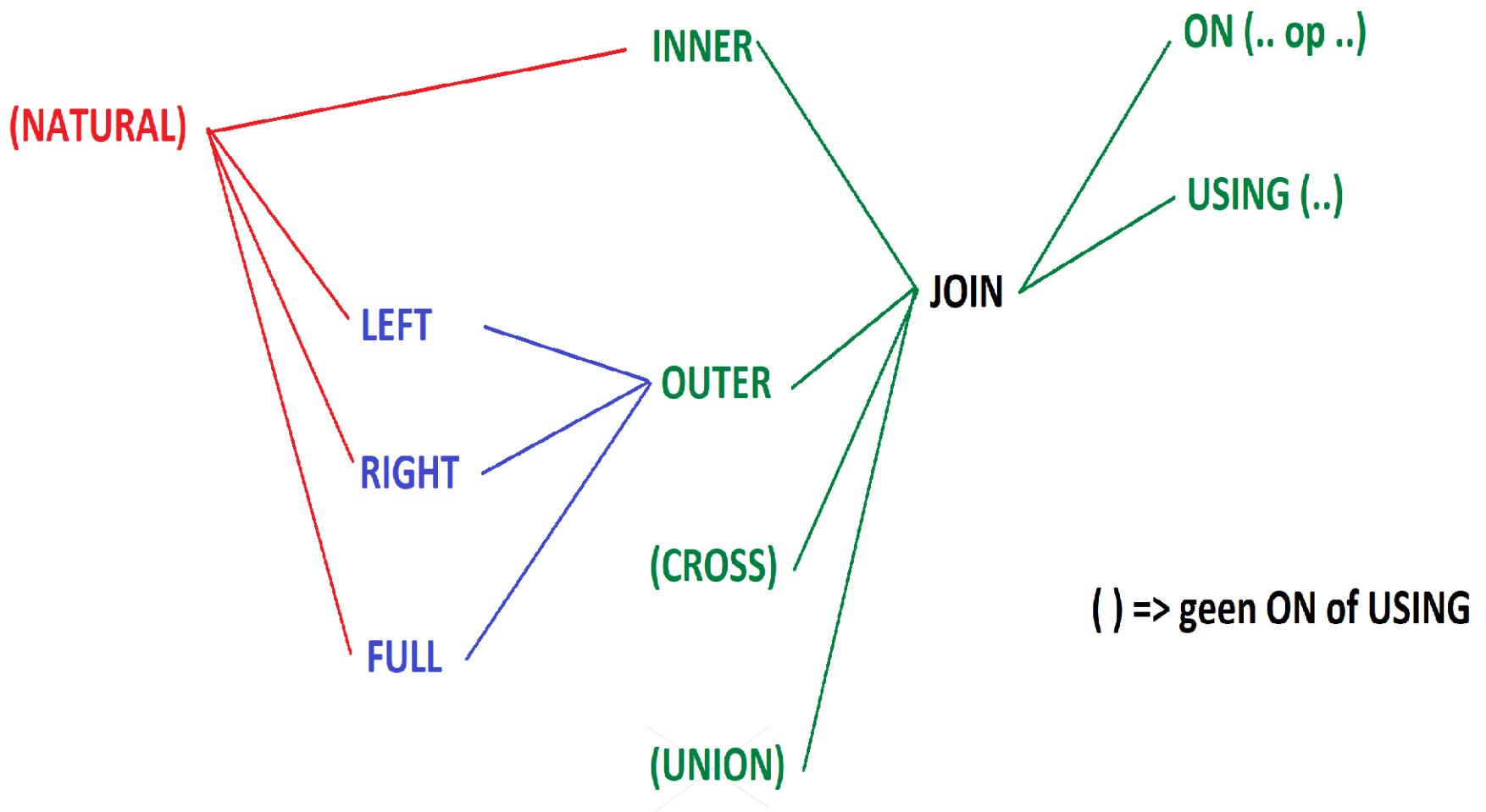
	spelersnr integer	naam character (15)	count bigint
1	104	Moerman	2
2	100	Permentier	3
3	2	Elfring	2
4	6	Permentier	3
5	27	Cools	1
6	95	Meuleman	1
7	83	Hofland	2
8	28	Cools	1
9	39	Bischoff	1
10	44	Bakker, de	3
11	8	Nieuwenburg	3

THETA JOIN

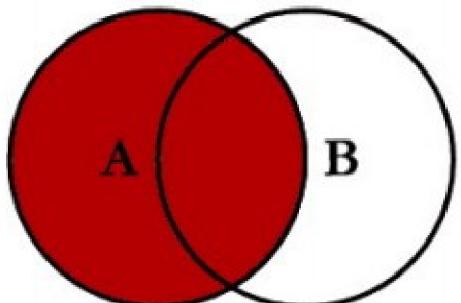
```
1 | SELECT      s.spelersnr, s.naam, count(sp.spelersnr), array_agg(sp.naam)
2 | FROM        spelers s INNER JOIN spelers sp
3 |          ON (s.spelersnr <> sp.spelersnr)
4 | WHERE       length(s.naam) = length(sp.naam)
5 | GROUP BY    s.spelersnr, s.naam
6 | ORDER BY    3
```

	Data Output	Explain	Messages	Notifications	Query History
	spelersnr integer	naam character (15)	count bigint	array_agg character[]	
1	27	Cools	1	{"Cools "}	
2	95	Meuleman	1	{"Bischoff "}	
3	39	Bischoff	1	{"Meuleman "}	
4	28	Cools	1	{"Cools "}	
5	104	Moerman	2	{"Hofland ","Elfring "}	
6	2	Elfring	2	{"Moerman ","Hofland "}	
7	83	Hofland	2	{"Moerman ","Elfring "}	
8	6	Permentier	3	{"Permentier ","Bakker, de ","Nieuwenburg "}	
9	100	Permentier	3	{"Bakker, de ","Nieuwenburg ","Permentier "}	
10	44	Bakker, de	3	{"Permentier ","Nieuwenburg ","Permentier "}	
11	8	Nieuwenburg	3	{"Permentier ","Bakker, de ","Permentier "}	

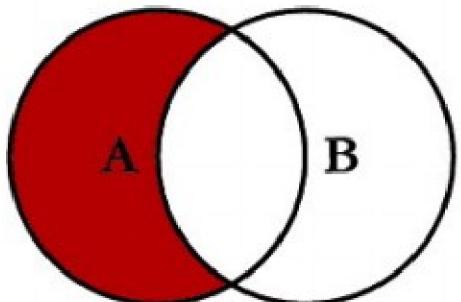
Notedop



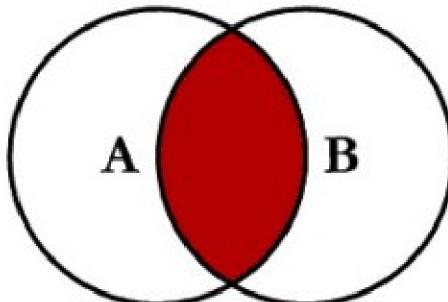
SQL JOINS



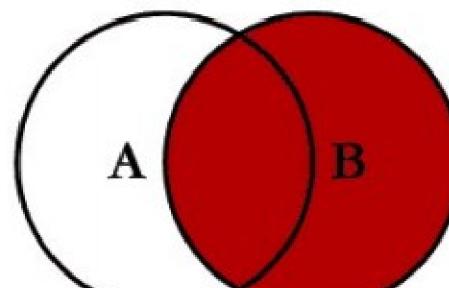
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



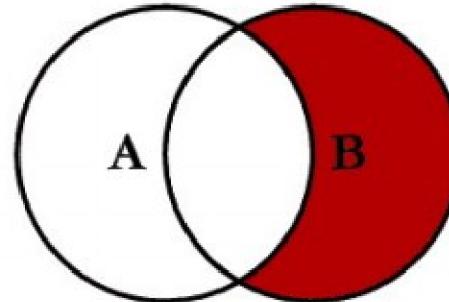
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



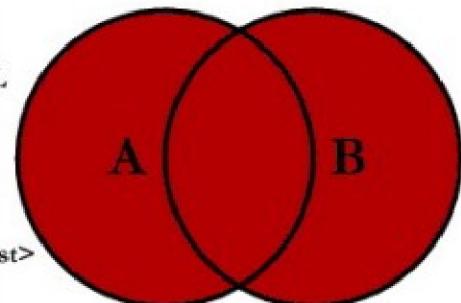
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



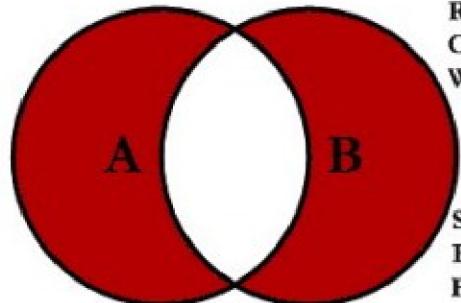
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Wim Bertels (CC)BY-SA-NC

Referenties:

- SQL Leerboek, R. Van der Lans
- Slides Databanken, H. Martens
- C. L. Moffatt, 2008
- <http://users.atw.hu/sqlnut/sqlnut2-chp-1-sect-2.html>

Lateral Joins

wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Unported
Licentie

Inleiding

```
CREATE TEMPORARY TABLE numbers AS  
SELECT generate_series(1,3) AS max_num;
```

```
SELECT *  
FROM numbers;
```

-- Tijdelijke tabellen worden periodiek opgeruimd

max_num

1
2
3
(3 rows)

Probleem?

```
SELECT  *
FROM    numbers,
        (SELECT generate_series(1,max_num)) AS max_lijst;
```

Probleem?

```
SELECT *
FROM   numbers,
       (SELECT generate_series(1,max_num)) AS max_lijst;
```

```
/*
psql:04_2_LATERAL.sql:11: ERROR:  column "max_num" does not exist
LINE 3:   (SELECT generate_series(1,max_num)) AS max_lijst;

HINT:  There is a column named "max_num" in table "numbers",
but it cannot be referenced from this part of the query.
*/
```

LATERAL

Subqueries die verschijnen in FROM kunnen voorafgegaan worden door het sleutelwoord LATERAL.

Hierdoor kunnen ze verwijzen naar kolommen uit voorafgaande FROM-items.

(Zonder LATERAL, wordt elke subquery onafhankelijk geëvalueerd en kan deze dus niet verwijzen naar een ander FROM item).

Oftewel: gecorreleerde subqueries in de FROM-komponent

LATERAL Detail

Via LATERAL kunnen we verwijzen naar een eerdere referentie uit de FROM-komponent:

- naar een eerdere tabelreferentie
- naar een eerdere subquery
- naar een eerdere functie die een verzameling teruggeeft (SRF)
 - (In dit geval wordt een LATERAL gedrag door de standaard bepaalt, je kan dus LATERAL weglaten in dit geval)

Oplossing

```
SELECT *
FROM   nummers,
       (SELECT generate_series(1,max_num) AS max_lijst;
```

```
/*
psql:04_2_LATERAL.sql:11: ERROR:  column "max_num" does not exist
LINE 3:  (SELECT generate_series(1,max_num)) AS max_lijst;
HINT:  There is a column named "max_num" in table "nummers",
but it cannot be referenced from this part of the query.
*/
```

```
SELECT *
FROM   nummers, LATERAL
       (SELECT generate_series(1,max_num) AS max_lijst;
```

Uitvoer

```
SELECT *
FROM   numbers, LATERAL
       (SELECT generate_series(1,max_num) AS max_lijt;
```

max_num
1
2
3
(3 rows)

max_num	generate_series
1	1
2	1
2	2
3	1
3	2
3	3
(6 rows)	

PostgreSQL uitbreiding

```
SELECT      *
FROM        numbers, LATERAL generate_series(1,max_num);
```

```
SELECT      *
FROM        numbers, generate_series(1,max_num);
-- SELECT voor functie is optioneel in dit geval
```

Voorbeeld

```
SELECT      *
FROM        klanten k
LEFT JOIN LATERAL
(SELECT    age(k.geboortedatum) as leeftijd) AS l
ON (leeftijd=age(k.geboortedatum));
```

Voorbeeld

```
SELECT      *
FROM        klanten k
LEFT JOIN LATERAL
(SELECT    age(k.geboortedatum) as leeftijd) AS l
ON        (leeftijd=age(k.geboortedatum));
```

-- toon per klant de leeftijd

klantnr	naam	vnaam	geboortedatum	leeftijd
121	Hassoui	Sjeik	1975-06-12	43 years 8 mons 27 days
122	Martens	Hedwigh	1978-06-30	40 years 8 mons 9 days
123	Ellison	Larry	1975-10-10	43 years 4 mons 30 days
124	Van Rossem	Jean-Pierre	1975-01-12	44 years 1 mon 28 days
125	Frimout	Dirk	1980-11-29	38 years 3 mons 10 days
126	Gates	Bill	1982-12-25	36 years 2 mons 15 days

JOIN Conditie

- Is de join conditie nodig?
 - De subquery is gecorreleerd.
- Wat is het effect van het cartesisch produkt in dit geval?

Voorbeeld

```
SELECT      *
FROM        klanten k
LEFT JOIN LATERAL
(SELECT    age(k.geboortedatum) as leeftijd) AS l
ON (leeftijd=age(k.geboortedatum));
```

```
SELECT      *
FROM        klanten k
LEFT JOIN LATERAL
(SELECT    age(k.geboortedatum) as leeftijd) AS l
ON true;
```

JOIN Conditie

When a FROM item contains LATERAL cross-references, evaluation proceeds as follows:

for each row of the FROM item providing the cross-referenced column(s), or set of rows of multiple FROM items providing the columns, the LATERAL item is evaluated using that row or row set's values of the columns. The resulting row(s) are joined as usual with the rows they were computed from. This is repeated for each row or set of rows from the column source table(s).

> Cartesisch produkt gedraagt zich hier als een foreach lus

JOIN Conditie

Welk soort JOIN?

- CROSS, INNER, LEFT
- Niet: RIGHT of FULL!
- De subquery is gecorreleerd.

Gedrag LEFT blijft tov INNER

Een geval

Geef voor elke reis
de twee kleinste objecten die bezocht worden

- Ter vergelijking:

```
SELECT *
```

```
FROM bezoeken NATURAL INNER JOIN hemelobjecten
```

```
WHERE reisnr = 32
```

```
ORDER BY diameter;
```

objectnaam	reisnr	volgnr	verblijfsduur	satellietvan	afstand	diameter
Deimos	32	3		0	Mars	23.400
Phobos	32	2		1	Mars	9.270
Maan	32	5		2	Aarde	384.400
Maan	32	1		0	Aarde	384.400
Mars	32	4		3	Zon	227900.000

(5 rows)

Een geval

Geef voor elke reis
de twee kleinste objecten die bezocht worden

```
SELECT *
FROM reizen r LEFT JOIN LATERAL
  (SELECT *
   FROM bezoeken b NATURAL INNER JOIN hemelobjecten h
   WHERE b.reisnr=r.reisnr
   ORDER BY h.diameter
   FETCH FIRST 2 ROWS ONLY) AS I
ON (true);
```

Wim Bertels (CC)BY-SA-NC

Referenties:

- <https://www.postgresql.org/docs/current/sql-select.html>
- <https://www.postgresql.org/docs/current/static/queries-table-expressions.html>
- <https://blog.2ndquadrant.com/join-lateral/>
- <https://modern-sql.com/slides>
- Becoming A SQL Guru, Stella Nisenbaum

Set-operatoren

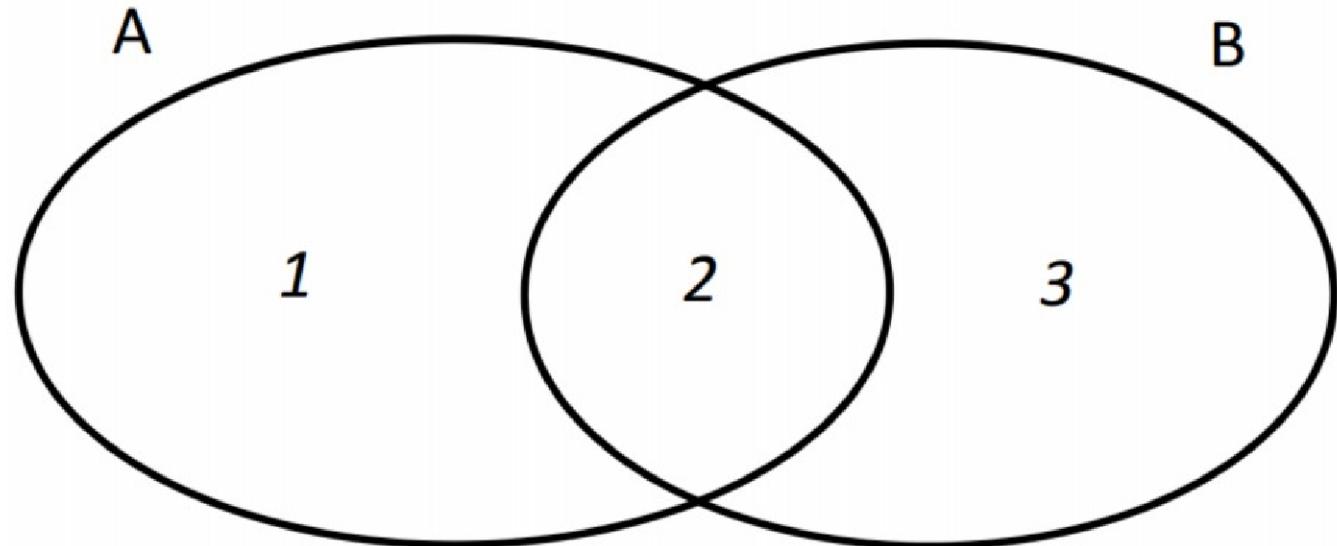
verzamelingenleer

wim.bertels@ucll.be

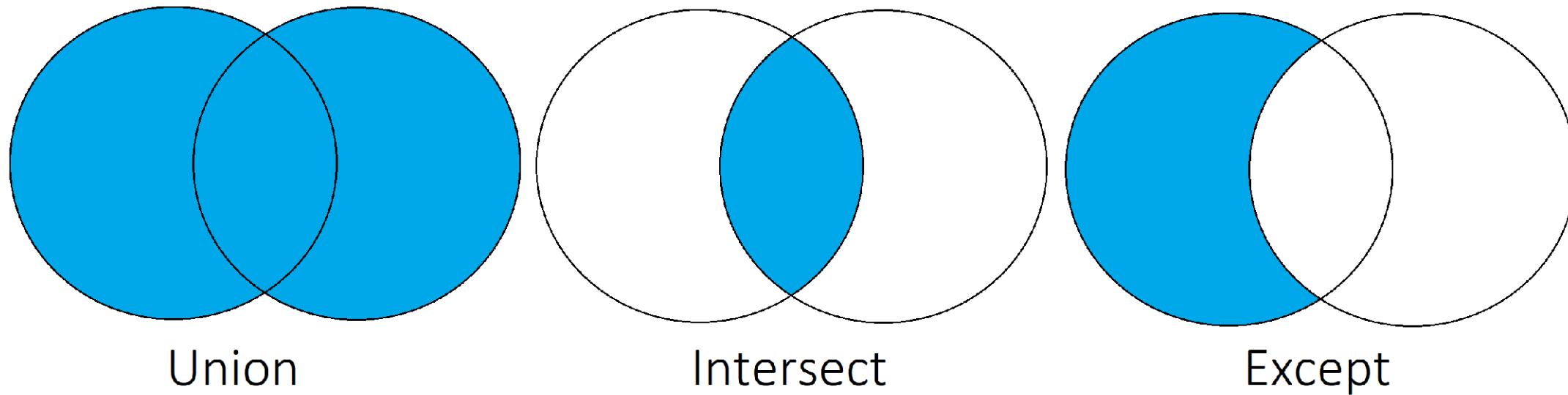
Naamsvermelding-NietCommercieel-GelijkDelen 4.0
Unported Licentie

Set-operatoren

- Combineren van resultaten van individuele **SELECT**-instructies
- Mogelijkheden:
 - **UNION**
 - **INTERSECT**
 - **EXCEPT**
 - **UNION ALL**
 - **INTERSECT ALL**
 - **EXCEPT ALL**



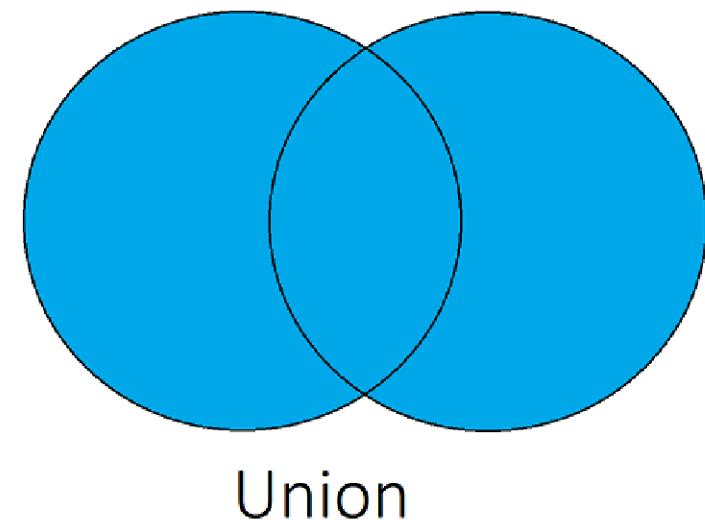
Wiskunde - Verzamelingenleer



UNION

- Elke rij die in één van de twee **SELECT**-blokken of in beide voorkomt. (dubbele rijen worden verwijderd)
 - Vb. Geef het spelersnummer van elke speler voor wie minstens één boete is betaald of die aanvoerder is of voor wie beide geldt

```
SELECT spelersnr  
FROM boetes  
UNION  
SELECT spelersnr  
FROM teams
```



Regels UNION

- De verschillende blokken moeten hetzelfde aantal kolommen hebben en de kolommen die aan elkaar « geplakt » worden, moeten hetzelfde datatype hebben.
- Alleen op het einde mag een **ORDER BY** voorkomen, deze sorteert het eindresultaat.
- **SELECT** moet geen **DISTINCT** bevatten (dubbele rijen worden automatisch verwijderd)

UNION en GROUP BY

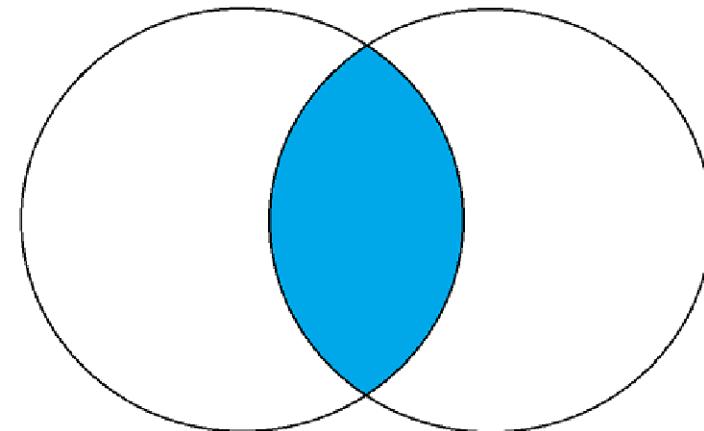
Vb. Berekenen van totalen en subtotalen

```
SELECT      CAST(teamnr AS char(4)) AS teamnr,  
           CAST(spelersnr AS char(4)) AS spelersnr,  
           SUM(gewonnen + verloren) AS totaal  
           wedstrijden  
           teamnr, spelersnr  
  
UNION  
SELECT      CAST(teamnr AS char(4)), 'subtotaal',  
           SUM(gewonnen + verloren)  
           wedstrijden  
           teamnr  
  
UNION  
SELECT      'totaal', 'totaal', SUM(gewonnen + verloren)  
           wedstrijden  
           1, 2
```

INTERSECT

- Alleen die rijen die in de resultaten van beide **SELECT**-blokken voorkomen (dubbele rijen worden verwijderd)
 - Vb. Geef het spelersnummer van de spelers die aanvoerder zijn en voor wie minstens één boete is betaald

SELECT spelersnr
FROM teams
INTERSECT
SELECT spelersnr
FROM boetes



Intersect

Tijd voor een micropauze

EXCEPT

- Alleen die rijen die wel in het resultaat van het eerste **SELECT**-blok voorkomen, maar niet in het resultaat van het tweede **SELECT**-blok (dubbele rijen worden verwijderd)
 - Vb. Geef het spelersnummer van de spelers voor wie minstens één boete is betaald, maar die geen aanvoerder zijn

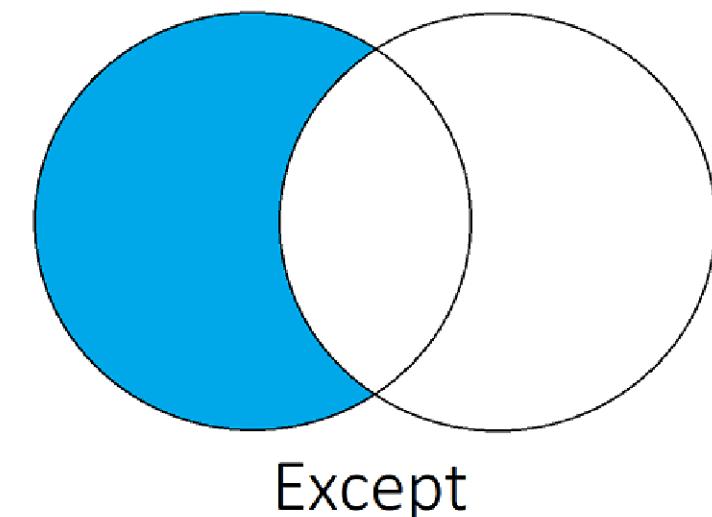
SELECT spelersnr

FROM boetes

EXCEPT

SELECT spelersnr

FROM teams



ALL = behoud dubbels

- Standaard: dubbele rijen worden verwijderd
- ALL-variant : om dubbele rijen te behouden
 - UNION ALL
 - INTERSECT ALL
 - EXCEPT ALL
 - Vb. Geef het spelersnummer van de spelers voor wie minstens één boete is betaald, maar die geen aanvoerder zijn. Behoud dubbele rijen.

```
SELECT      spelersnr  
FROM        boetes  
EXCEPT ALL  
SELECT      spelersnr  
FROM        teams
```

NULL

Rijen met een **NULL**-waarde worden als gelijk beschouwd voor de set-operatoren.

Combinaties

- Meerdere set-operatoren.
- Haakjes kunnen de volgorde wijzigen.
 - Vb. Geef het spelersnummer van de spelers voor wie minstens één boete is betaald, maar die geen aanvoerder zijn, en daarenboven de spelers uit Hove.

```
(SELECT spelersnr  
      FROM boetes  
    EXCEPT  
  SELECT spelersnr  
      FROM teams)  
  
UNION  
  SELECT spelersnr  
      FROM spelers  
    WHERE plaats = 'Hove'
```

Volgorde

- Van links naar rechts
 - INTERSECT heeft voorrang
 - daarna UNION en EXCEPT

```
SELECT spelersnr  
FROM boetes  
EXCEPT  
SELECT spelersnr  
FROM teams  
INTERSECT  
SELECT spelersnr  
FROM spelers  
WHERE plaats = 'Hove'
```

```
SELECT spelersnr  
FROM boetes  
EXCEPT  
(SELECT spelersnr  
FROM teams  
INTERSECT  
SELECT spelersnr  
FROM spelers  
WHERE plaats = 'Hove')
```

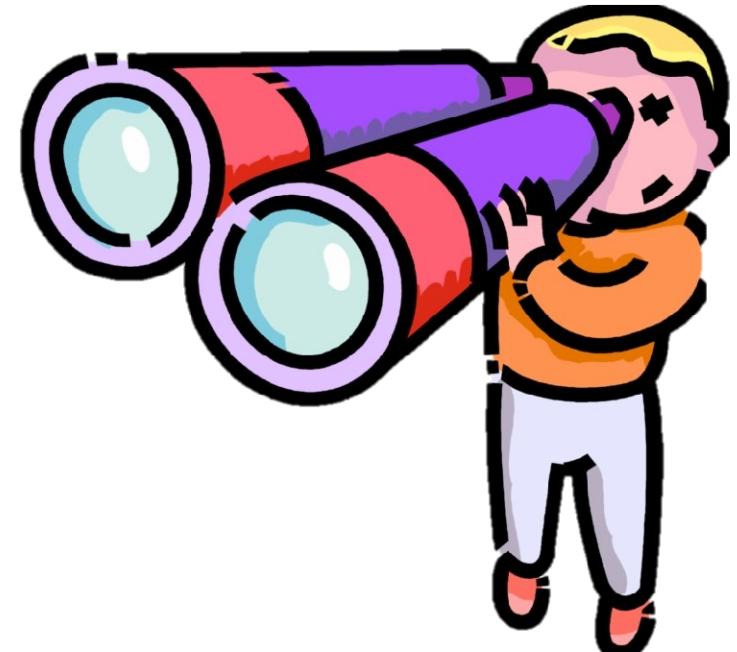
<https://www.postgresql.org/docs/current/queries-union.html>

Wim Bertels (CC)BY-SA-NC

References:

- * Slides Databanken, H. Martens
- * SQL Leerboek, R. Van der Lans

Views



wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0
Unported Licentie

Views

- **CREATE VIEW**

- View: tabel die zichtbaar is voor gebruiker maar geen opslagruimte inneemt.
- Je kan conceptueel een view beschouwen als zijnde een virtuele tabel, dewelke wordt opgebouwd als die opgevraagd wordt.
- Een view wordt gemaakt op basis van een query.

```
CREATE VIEW leeftijden (spelersnr, leeftijd) AS  
SELECT spelersnr, age(geb_datum)  
FROM spelers;
```

```
CREATE VIEW leeftijden AS  
SELECT spelersnr, age(geb_datum) AS leeftijd  
FROM spelers;
```

Nut?

- Vereenvoudigen van routinematige instructies
- Reorganiseren van tabellen
- Stapsgewijs opzetten van SELECT-instructies
- Beveiligen van gegevens

Views

1. Inleiding
2. Creëren van views
3. Kolomnamen van views
4. Muteren van views, WITH CHECK OPTION
5. Verwijderen van views
6. Beperkingen bij muteren
7. Verwerken van instructies op views
8. Toepassingen van views

1. Inleiding

- Basistabellen /= afgeleiden tabellen (views)
- View:
 - Bevat geen fysieke rijen !!!!
 - Voorschrijft of formule om gegevens uit basistabellen in een ‘virtuele’ tabel te steken.

2. Creëren van views

- CREATE VIEW: creëert een view

Vb. CREATE VIEW wspelers AS
 SELECT spelersnr, plaats
 FROM spelers
 WHERE plaats IS NOT NULL;

- Views: raadplegen en muteren

Synoniemen en commentaar



3. Kolomnamen van views

- SELECT definieert de kolomnamen
- Maar expliciete definitie is ook mogelijk:

Vb. CREATE VIEW leuvenaars (snr, naam, geboorte) AS
SELECT spelersnr, naam, geb_datum
FROM spelers
WHERE plaats = 'Leuven';
- Expliciete definitie is verplicht als kolom bestaat uit een functie of berekening.

```
CREATE VIEW leeftijden (spelersnr, leeftijd) AS  
SELECT spelersnr, age(geb_datum)  
FROM spelers;
```

```
CREATE VIEW leeftijden AS  
SELECT spelersnr, age(geb_datum) AS leeftijd  
FROM spelers;
```

Tijd voor een micropauze

4. WITH CHECK OPTION

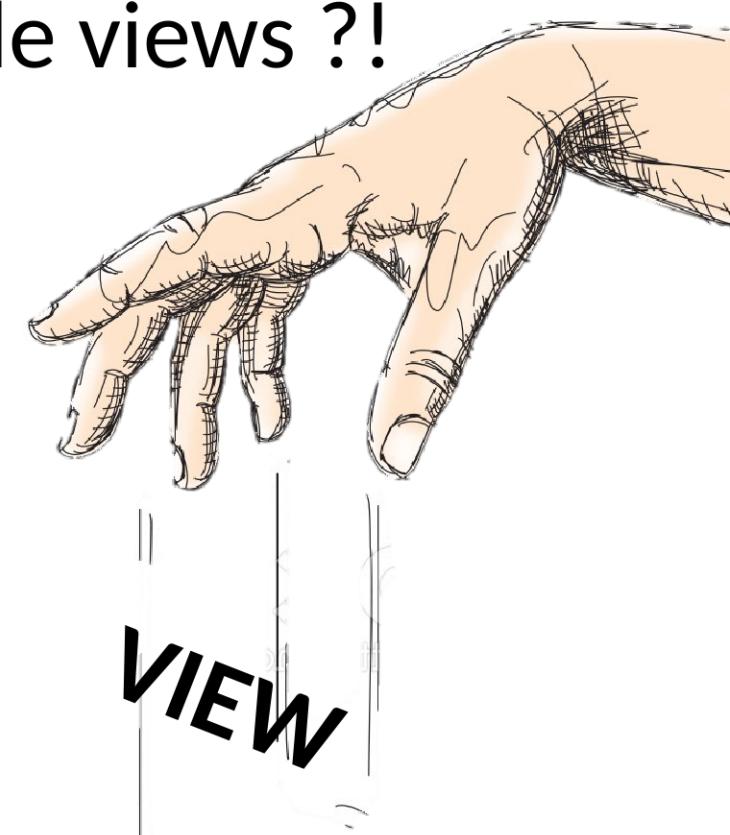
- Muteren van views -> muteren van tabellen
- WITH CHECK OPTION controleert:
 - UPDATE: aangepaste rijen behoren nog tot view
 - INSERT: nieuwe rij behoort tot view
 - DELETE: verwijderde rij behoort tot view

```
CREATE VIEW stok_oud AS
    SELECT *
    FROM spelers
    WHERE geb_datum < '1980-01-01'
    WITH CHECK OPTION;
```

5. Verwijderen van views

- DROP VIEW: verwijdert view en alle hierop gedefinieerde views ?!

RESTRICT vs CASCADE



6. Beperkingen bij muteren

Mogelijke mutaties op views:

- SELECT
- INSERT
- UPDATE
- DELETE

6. Beperkingen bij muteren

- Maar mutatie mag alleen als:
 - View moet (in)direct gebaseerd zijn op één of meerdere basistabellen
 - SELECT mag slechts één tabel bevatten
 - SELECT mag geen WITH, DISTINCT, GROUP BY, HAVING, FETCH .., of OFFSET bevatten
 - SELECT mag geen SET-operatoren bevatten
 - SELECT mag geen aggregatiefunctie (*of vensterfuncties*) bevatten
 - UPDATE: virtuele kolom mag niet gewijzigd worden
 - INSERT: in SELECT moeten alle NOT NULL-kolommen staan

7. Verwerken van instructies op views

- Extra stap waarin de viewformule wordt opgenomen

```
SELECT spelersnr  
FROM spelers  
WHERE plaats = 'Leuven'  
    AND spelersnr IN (  
        SELECT spelersnr  
        FROM boetes  
    );
```

```
CREATE VIEW durespelers AS  
    SELECT spelersnr  
    FROM boetes  
    GROUP BY spelersnr;
```

```
SELECT spelersnr  
FROM spelers  
INNER JOIN durespelers  
    USING (spelersnr)  
WHERE plaats = 'Leuven';
```

8. Toepassingen van views

Vereenvoudigen van routinematige instructies

-> Instructies die vaak gebruikt worden

Reorganisatie van tabellen

-> Bij aanpassingen blijven de <<oude>> programma's bestaan

Stapsgewijs opzetten van SELECT-instructies

-> Bij complexe queries stukken << voorprogrammeren >>

Specificeren van integriteitsregels

-> WITH CHECK OPTION: toegestane waarden controleren

Gegevensbeveiliging

-> Beveiliging van delen van tabellen

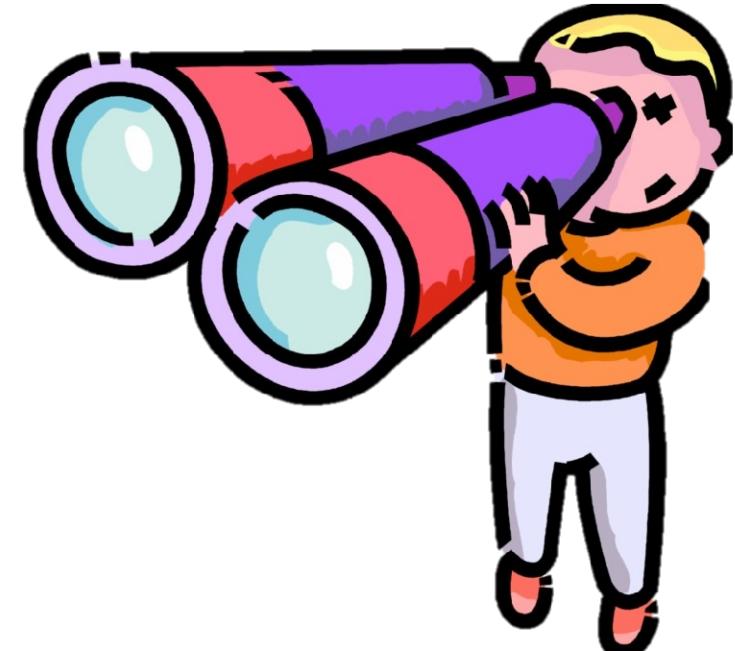
Wim Bertels (CC)BY-SA-NC

References:

- * Slides Databanken, H. Martens
- * SQL Leerboek, R. Van der Lans

Views

Beveiliging



wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0
Unported Licentie

Views

- **CREATE VIEW**

- View: tabel die zichtbaar is voor gebruiker maar geen opslagruimte inneemt.
- Je kan conceptueel een view beschouwen als zijnde een virtuele tabel, dewelke wordt opgebouwd als die opgevraagd wordt.
- Een view wordt gemaakt op basis van een query.

```
CREATE VIEW leeftijden (spelersnr, leeftijd) AS  
SELECT spelersnr, age(geb_datum)  
FROM spelers;
```

```
CREATE VIEW leeftijden AS  
SELECT spelersnr, age(geb_datum) AS leeftijd  
FROM spelers;
```

Nut?

- Vereenvoudigen van routinematige instructies
- Reorganiseren van tabellen
- Stapsgewijs opzetten van SELECT-instructies
- **Beveiligen** van gegevens
- ..

Beveiliging

- Beveiliging
- SQL gebruiker:
 - moet gekend zijn
 - wachtwoord
 - expliciete toekenning van bevoegdheden:
 - Kolombevoegdheden
 - Tabelbevoegdheden
 - Databasebevoegdheden
 - Gebruikersbevoegdheden



Beveiliging

1. Invoeren/verwijderen van gebruikers
2. Tabel- en kolombevoegdheden
3. Databasebevoegdheden
4. Gebruikersbevoegdheden
5. WITH GRANT OPTION
6. Werken met rollen
7. Intrekken van bevoegdheden
8. Beveiliging van en met views

1. Invoeren/verwijderen van gebruikers

- CREATE USER: creëert een user
Vb. CREATE USER Frank IDENTIFIED BY Frank_pw
- ALTER USER: verandert het paswoord
Vb. ALTER USER Frank IDENTIFIED BY Frank_pasw
- DROP USER: verwijdert een user
Vb. DROP USER Frank

2. Tabel- en kolombevoegdheden

- GRANT: kent bevoegdheden toe
- Soorten tabelbevoegdheden:
 - SELECT: bevoegdheid tot SELECT en VIEW
 - INSERT: rijen toevoegen m.b.v. INSERT
 - DELETE: rijen verwijderen m.b.v. DELETE
 - TRUNCATE: tabel leegmaken m.b.v. TRUNCATE
 - UPDATE: rijen wijzigen m.b.v. UPDATE
 - REFERENCES: refererende sleutels naar deze tabel creëren
 - TRIGGER: *trigger op deze tabel aanmaken*
 - (alter en drop zijn gereserveerd voor de eigenaar, via ALTER TABLE gaan)
 - CREATE: *bv INDEX*
 - ALL/ALL PRIVILEGES: alle bevoegdheden

3. Databasebevoegdheden

- Soorten databasebevoegdheden:
 - CREATE: je mag objecten aanmaken
 - CONNECT: je mag de db gebruiken
 - TEMP: je mag tijdelijke objecten aanmaken

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

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

4. Gebruikersbevoegdheden

- Databasebevoegdheden toekennen aan gebruiker(s)
- CREATE USER: gebruiker aanmaken

Vb. GRANT SELECT
ON ALL TABLES IN SCHEMA x
TO franky;

5. WITH GRANT OPTION

- WITH GRANT OPTION:
de gebruikers die toegang krijgen, kunnen deze machtiging doorgeven aan andere gebruikers

Vb. GRANT ALL ON spelers TO Frank
WITH GRANT OPTION;

6. Werken met rollen

- CREATE ROLE: Creëert een rol met bevoegdheden voor een aantal users, als de rol verandert, veranderen de bevoegdheden voor al de betrokken users.

Vb. CREATE ROLE ADMIN

```
GRANT SELECT, INSERT ON spelers
```

```
TO admin;
```

```
GRANT admin TO Frank, Marc, Ann, Greet;
```

- DROP ROLE: verwijdert de rol

Vb. DROP ROLE admin;

7. Intrekken van bevoegdheden

- REVOKE: verwijdert bevoegdheid (en de afhankelijke bevoegdheden)

Vb:- REVOKE ALL
ON spelers
FROM Frank;

- REVOKE admin
FROM Marc;

- REVOKE SELECT
ON spelers
FROM admin;

8. Beveiliging van en met views

- Analoog voor bevoegdheden op views
- Kan gebruikt worden voor de beveiliging van gegevens (gebruiker krijgt enkel machtiging op een view, waarin een deel van de tabel gedefinieerd wordt).

8. Beveiliging van en met views

Vb.

```
CREATE USER Frank;
```

```
CREATE VIEW zichtbaar_deel AS  
    SELECT spelersnr, naam, voorletters  
    FROM spelers;
```

```
GRANT SELECT  
ON zichtbaar_deel  
TO Frank;
```

Vragen

- CREATE MATERIALIZED VIEW
 - Wat is verschil met CREATE VIEW?
 - Waarom zou dit nuttig kunnen zijn?
- REFRESH MATERIALIZED VIEW [CONCURRENTLY]
 - Waarvoor dient refresh en wat is concurrently?
- Is dit ISO sql?
- <https://www.postgresql.org/docs/current/sql-creatematerializedview.html>

Wim Bertels (CC)BY-SA-NC

References:

- * Slides Databanken, H. Martens
- * SQL Leerboek, R. Van der Lans

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

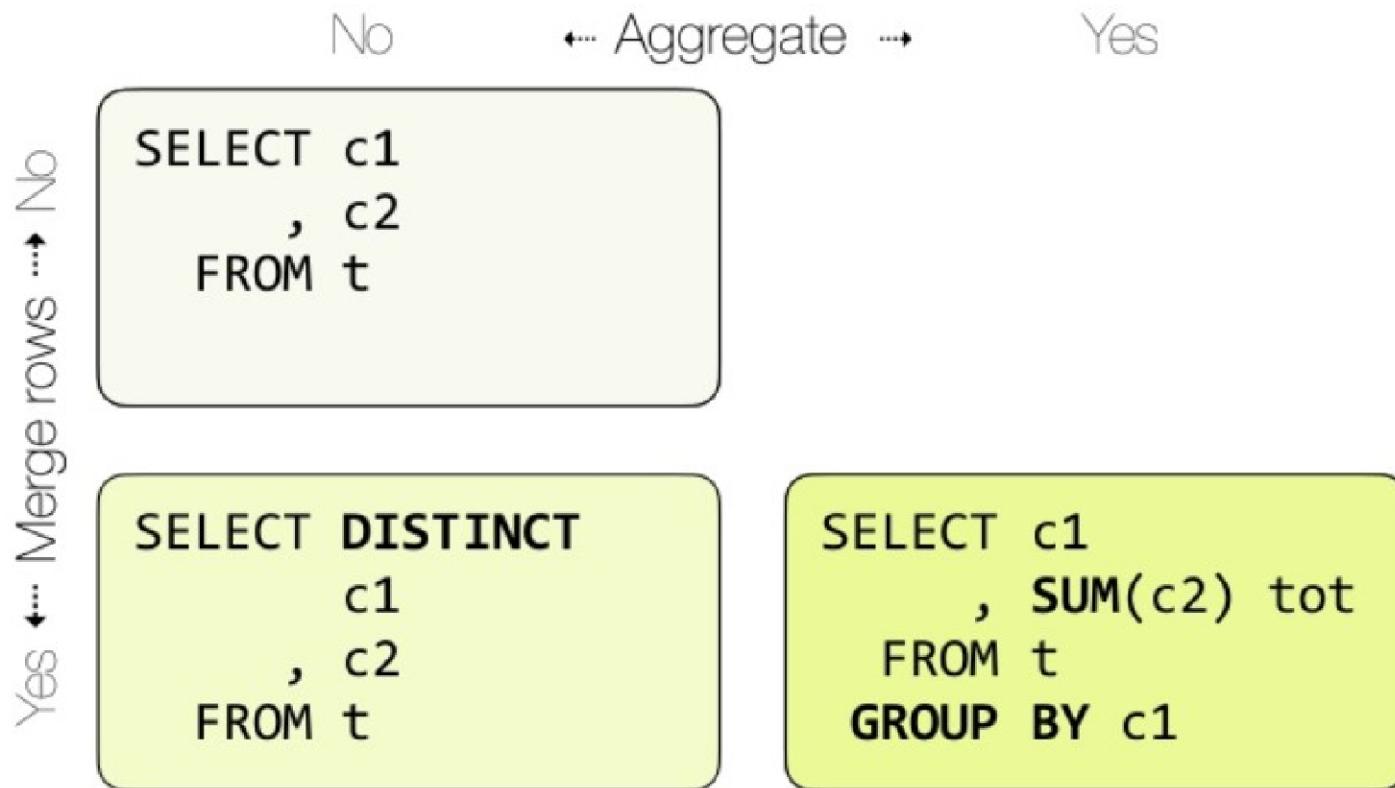
Wim.bertels@ucll.be

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

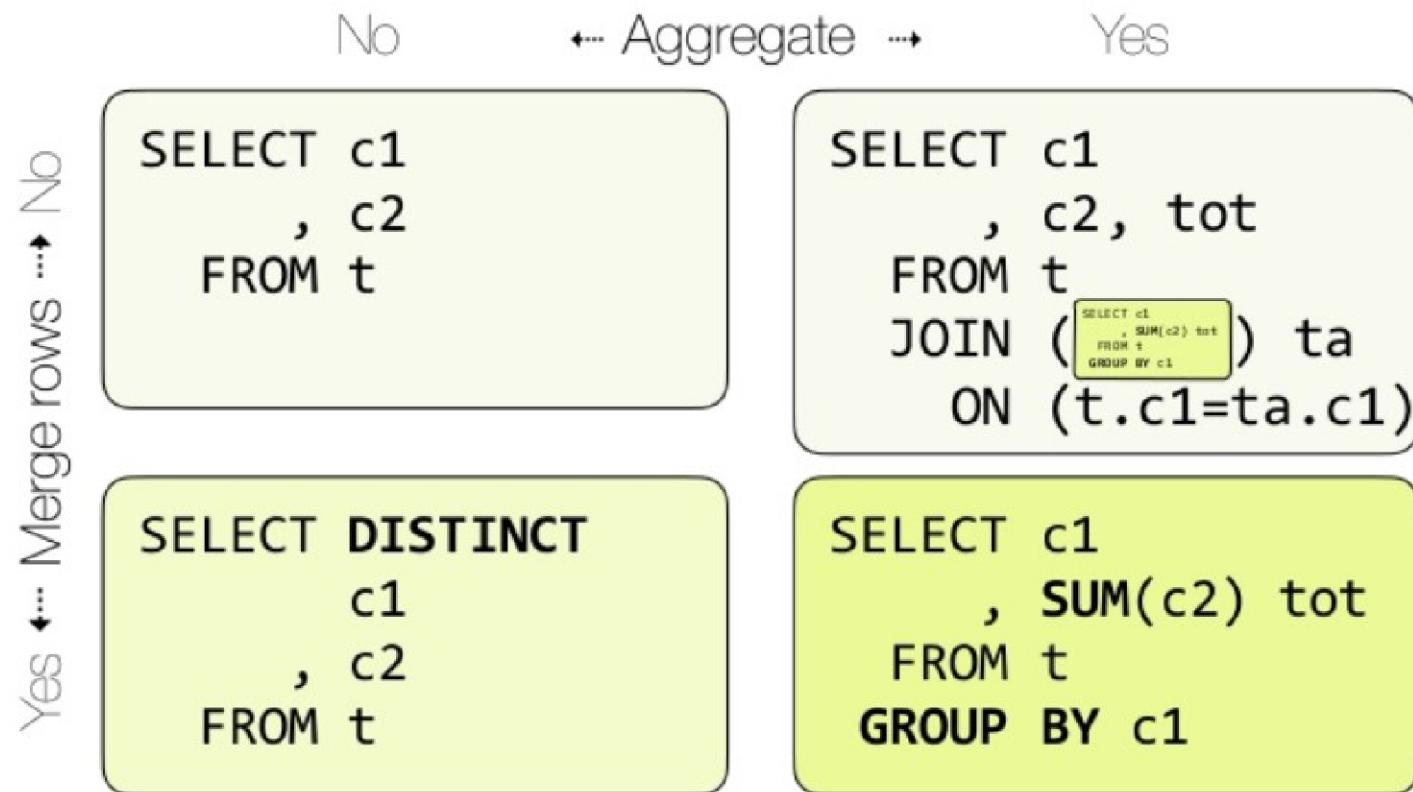
OVER (PARTITION BY)

The Problem



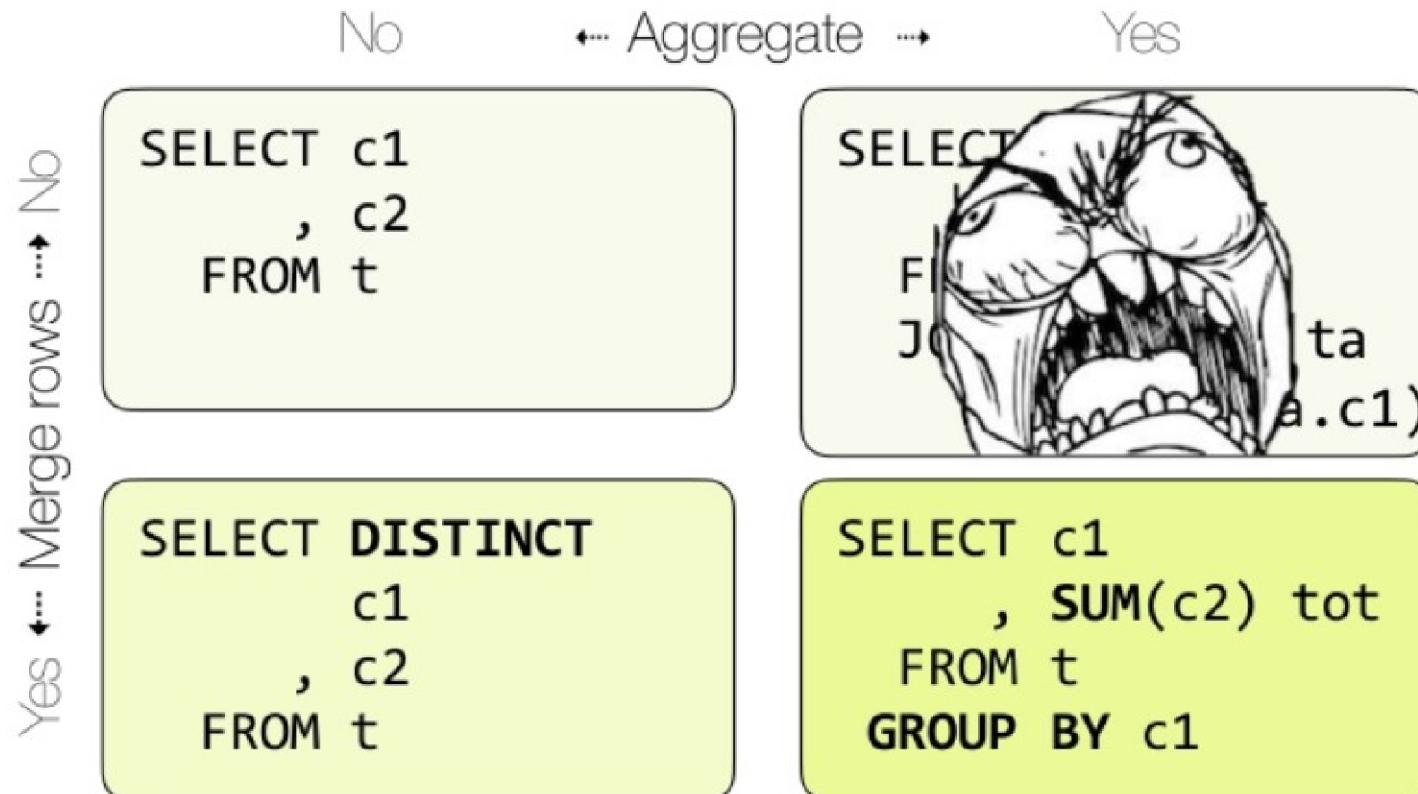
OVER (PARTITION BY)

The Problem



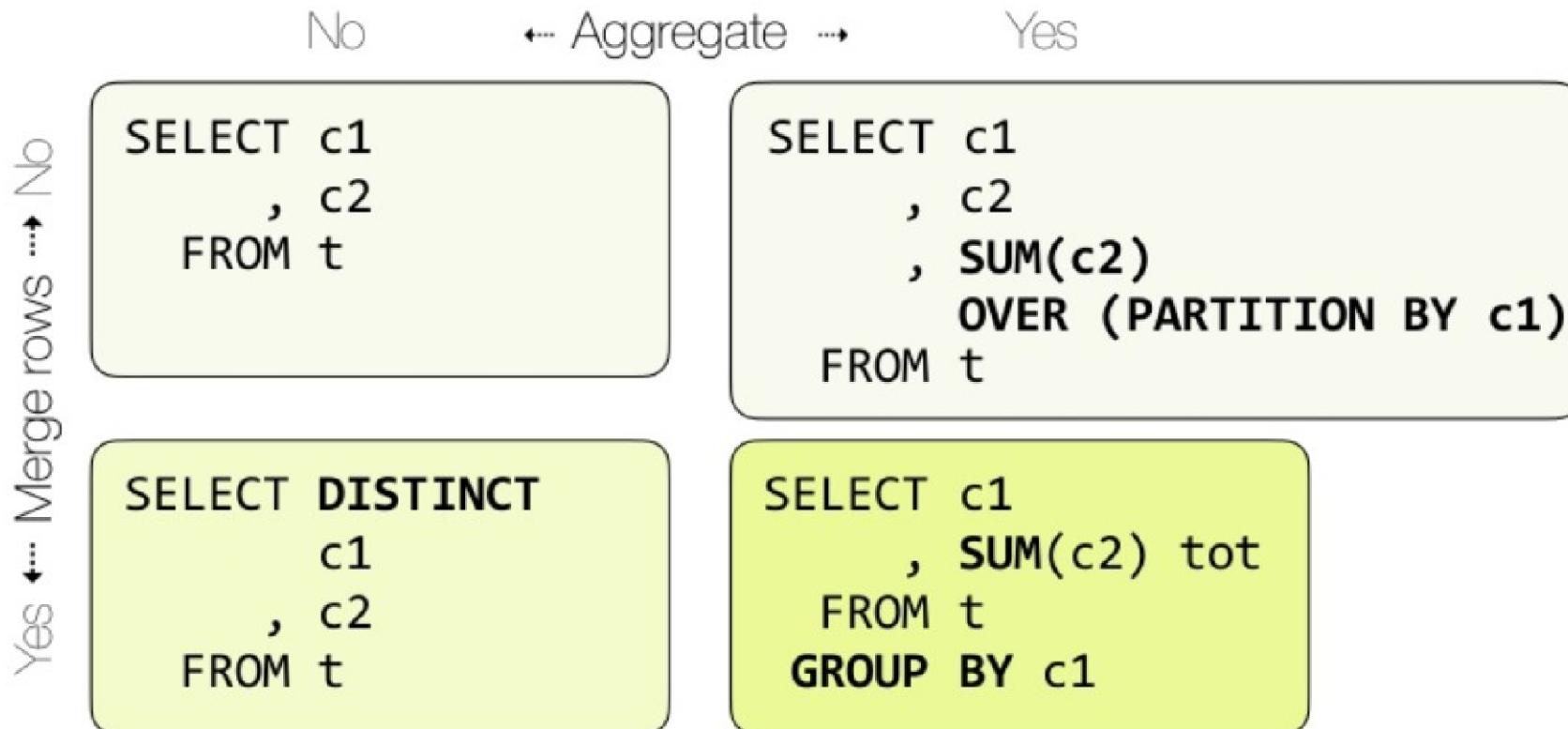
OVER (PARTITION BY)

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(), spelersnr
      spelersnr
  FROM spelers
 WHERE geslacht = 'M';
--zijn deze rijnummers vast bepaald?
          row_number | spelersnr
-----+-----
           1 |      2
           2 |      6
           3 |      7
           4 |     39
           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)
```

-- range/bereik is standaard gedrag

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

Indexen

wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0
Unported Licentie

INDEX

- Doel: beïnvloeden van de verwerkingsstijd!



INDEX

- Doel: beïnvloeden van de verwerkingstijd!

OS:

- Rijen worden in bestanden opgeslagen
- Een bestand bestaat uit pagina's
- Als een rij opgehaald wordt :
 - de betreffende pagina wordt opgehaald
 - de betreffende rij wordt opgehaald

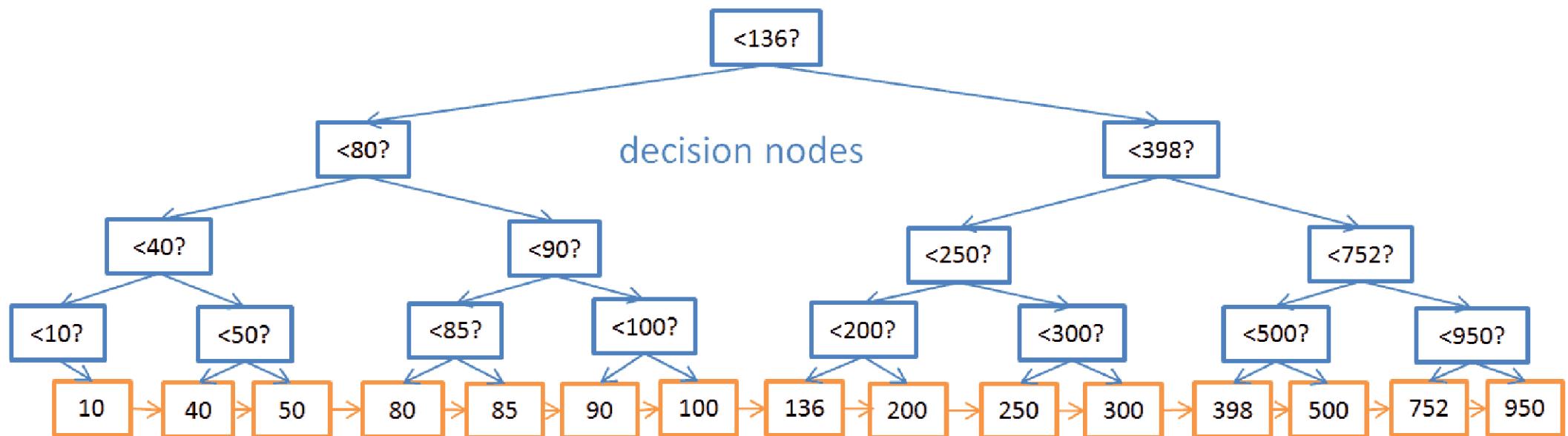
Werking van een INDEX

- 2 methodes voor het opzoeken:
 - Sequentiële zoekmethode: rij voor rij
 - Tijdrovend en inefficiënt
- Geïndexeerde zoekmethode: index (B-tree)
 - Boom
 - Knooppunten
 - Leafpage (bevat referentie naar pagina+rij)
 - 2 methodes:
 - Zoeken van rijen met een bepaalde waarde
 - Doorlopen van de hele tabel via een gesorteerde kolom (geclusterde index)

B-tree (default)

- Binaire zoek boom (node kan 2+leaves hebben)
- Best voor $>$ $<$ $=$ operatoren

B+ Tree / Database index

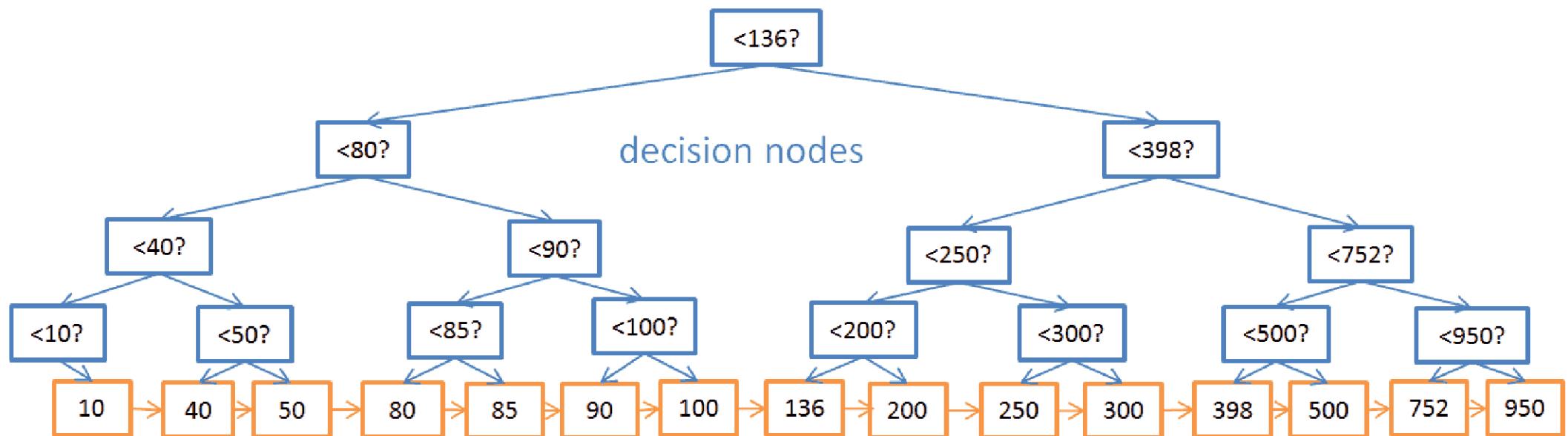


nodes with a pointer to a row in the associated table
they also have a link to their successor in the B+ Tree

B-tree (voorbeeld)

- Sequentiel: max 15 stappen
- Via boom: max 4 stappen, namelijk $\lceil \log_2 15 \rceil$

B+ Tree / Database index



nodes with a pointer to a row in the associated table
they also have a link to their successor in the B+ Tree

Werking van een INDEX

- Opmerkingen:
 - Wanneer tabel word aangepast: word index aangepast
 - Index: ook op niet-unieke kolom
 - Op één tabel:
 - meerdere indexen
 - één geclusterde index
 - Samengestelde index
- Opgelet:
 - Index neemt opslagruimte in beslag
 - Als index vol is: reorganisatie van de index
- Meerdere indexvormen zijn mogelijk

Planner / Optimiser

- Conceptueel: bv denken in lussen of geneste lessen bij joins, de volgorde van verwerking van een SELECT instructie
- Concreet: intern kan dit helemaal anders verwerkt worden
- SQL is een declaratieve (programmeer)taal, geen imperatieve (programmeer)taal.
 - Idee: Je zegt wat er (logisch) moet gebeuren, niet (expliciet) hoe het moet gebeuren.
 - 1 van de sterke punten van SQL
- Redeneer conceptueel en groei

Optimiser

- Zoekt de beste strategie
 - Verwachte verwerkingstijd
 - Aantal rijen
 - Indexen
 - Interne statistieken
 - ..

CREATE INDEX

- Geen ANSI of ISO specificatie
- Vendor specificatie :
<https://www.postgresql.org/docs/current/sql-createindex.html>

CREATE INDEX

- Postgresql:

```
CREATE INDEX spelers_postcode_idx  
    ON spelers (postcode asc);  
CREATE UNIQUE INDEX spelers_naam_vl_idx  
    ON spelers (naam, voorletters);  
-- UNIQUE !  
CREATE INDEX spelers_naam_vl_partial_idx  
    ON spelers (naam, voorletters)  
    WHERE spelersnr < 100;  
CLUSTER spelers USING speler_naam_vl_idx;
```

DELETE/UPDATE

Wat is het effect van een geclusterde index?

REINDEX

- Vendor specificatie:

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

- Postgresql:

- REINDEX INDEX een_index;
- REINDEX TABLE een_tabel;
- REINDEX DATABASE een_database;

INDEX management

- CREATE
- ALTER
 - `ALTER INDEX groot_idx
SET TABLESPACE ergens_anders;`
- DROP
- Meeste SQL-producten :
 - automatische creatie van index op primaire en secudaire sleutels bij het maken van de tabel
 - naam wordt afgeleid uit de naam van de tabel en de betreffende kolommen

Wanneer INDEXeren?

- Index:
 - Voordeel: index versnelt verwerking
 - Nadeel:
 - index neemt opslagruimte
 - elke mutatie vraagt aanpassing van index
=> verwerking vertraagt

Welke kolommen?

- Richtlijnen voor keuze van kolommen :
 - Unieke index op kandidaatsleutels
 - Index op refererende sleutels
 - Index op kolommen waarop (veel) geselecteerd wordt
 - Grootte van de tabel
 - Kardinaliteit (verschillende waarden) van de tabel
 - Distributie (verdeling) van de waarden
 - Index op een combinatie van kolommen
 - Index op kolommen waarop gesorteerd wordt
 - ..

Speciale indexvormen

- Multi-tabelindex :
= index op kolommen in meerdere tabellen
- Virtuele-kolomindex :
= index op een expressie
- Selectieve index
= index op een selectie van de rijen
- Hash-index
= index op basis van het adres van de pagina
- Bitmapindex
= interessant als er veel dubbele waardes zijn

Nieuwere indexvormen: GiST en SP-GIST

- GiST (Generalized Search Tree)
 - gebalanceerd
 - template voor verschillende index schema's
 - Voor "clusters" volgens een afstandsmaat
 - Bv vergelijken van intervallen, GIS, bevat, dichtste buren, tekst
- SP-GiST (space-partitioned GiST)
 - Hoeft niet gebalanceerd te zijn
 - Geen overlap tussen de clusters (GiST)

Nieuwere indexvormen: Gin en Brin

- GIN (Generalized Inverted Index)
 - Interessant bij veel dubbele waarden
 - Er worden meerdere opzoekwaarden tegelijk aangemaakt (handig voor rij, tekst, json,...)
- BRIN : voor grote geclusterde tabellen
 - Klein, minder performant tenzij:
 - min/max waarde per blok

Conclusie

Een index op elke tabel en iedere kolom?

Samengevat

- B-tree:
 - goede standaard, < \leq = \geq >
- Hash:
 - alternatief voor 1 rij, =
- GIN:
 - efficiënt bij dubbels, meerdere opzoekwaarden per veld, <@ @> = &&
- GiST:
 - veel mogelijkheden, minder performant, << &< .. <<| &<| <@ ~= &&
- Sp-GiST:
 - niet overlappende GiST, << >> ~= <@ <<| |>>
- BRIN:
 - grote geclusterde data, < \leq = \geq >

INDEX catalog_table

- Postgresql: pg_index
- <http://www.postgresql.org/docs/current/static/catalog-pg-index.html>
- <http://www.postgresql.org/docs/current/static/internals.html>
- Referentie: Index Internals, Heikki Linnakangas (Pivotal)
- <https://www.pexels.com/photo/blurred-book-book-pages-literature-46274/>

Algemene Richtlijnen Optimalisaties

wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0
Unported Licentie

1. Inleiding

- Optimiser – Beste uitvoeringsplan?
 - Indexen – Query snelheid verhogen?
 - Inzicht – Wat gebeurt er intern?
 - Maar soms niet optimaal
- => Herformulering om tot een efficiënte verwerkingsstrategie te komen !

2. Vermijd de OR-operator

- OR : index wordt meestal niet gebruikt
- Alternatief (indien mogelijk) :
 - Vervangen door een conditie met IN
 - Vervangen door 2 selects met UNION
- Vb.

...
where spelersnr = 15
or spelersnr = 29
or spelersnr = 55

=> where spelersnr in (15, 29, 55)

3. Onnodig gebruik van UNION

- UNION : dezelfde tabel meerdere malen doorlopen
- Alternatief (indien mogelijk) :
 - Herformuleren waarbij alle voorwaarden in één select instructie geplaatst worden

4. Vermijd de NOT-operator

- NOT : index wordt niet gebruikt
- Alternatief (indien mogelijk) :
 - Vervang NOT door vergelijkingsoperatoren
- Vb.

...

where not (jaartoe > 1980)

4. Vermijd de NOT-operator

- Oplossing:

...

where not (jaartoe > 1980)

=> where jaartoe <= 1980

5. Isoleer kolommen in condities

- Kolom in een berekening of in een scalaire functie : index wordt niet gebruikt
- Alternatief (indien mogelijk) :
 - Isoleer de kolom
- Vb.

...

where jaartoe + 10 = 1990

=> where jaartoe = 1980

6. Gebruik de BETWEEN-operator

- AND : gebruikt de index meestal niet
- Vb.

...

where jaartoe >= 1985

and jaartoe <= 1990

6. Gebruik de BETWEEN-operator

- AND : gebruikt de index meestal niet
- Oplossing :
 - Gebruik van BETWEEN
- Vb.

...

```
where    jaartoe >= 1985  
and      jaartoe <= 1990
```

=> where jaartoe between 1985 and 1990

7. Bepaalde vormen van LIKE-operator

- LIKE : index wordt niet gebruikt als patroon begint met % of _
- Alternatief :
- Geen, tenzij..
- Vb.

...

wherenaam like ‘%sen’

=> ???

8. Redundante condities bij joins

- Redundante condities : om SQL te verplichten om een bepaald pad te kiezen
- Vb.

...

```
where boetes.spelersnr = spelers.spelersnr  
and   boetes.spelersnr = 44
```

```
=>where boetes.spelersnr = spelers.spelersnr  
      and   boetes.spelersnr = 44  
      and   spelers.spelersnr = 44
```

9. Vermijd de HAVING-component

- Condities in HAVING : index wordt niet gebruikt
- Alternatief (indien mogelijk)
 - Zoveel mogelijk condities in WHERE
- Vb. --hoort dit thuis in de having?

...

```
group by spelersnr  
having    spelersnr >= 40
```

```
=> where    spelersnr >= 40  
        group by spelersnr
```

10. SELECT-component : compact

- SELECT-component zo compact mogelijk
 - Onnodiige kolommen weglaten uit SELECT
 - Bij gecorreleerde subquery met exists : één expressie bestaande uit één constante
- Vb.

```
select spelersnr, naam
from spelers
where exists (select '1'
               from boetes
               where boetes.spelersnr =
                     spelers.spelersnr)
```

11. Vermijd DISTINCT

- DISTINCT : verwerkingstijd verlengd
- Alternatief (indien mogelijk)
 - Vermijden als het overbodig is
- Vb.

```
select distinct wedstrijdnr, naam  
from wedstrijden, spelers  
where wedstrijden.spelersnr =  
      spelers.spelersnr
```

=> select wedstrijdnr, naam

12. ALL-optie bij set operatoren

- Zonder ALL : verwerkingstijd verlengd
- Data moeten gesorteerd worden om dubbels eruit te halen
- Vb.

```
Select    naam, voorletters  
from     spelers  
where    spelersnr = 10  
union all  
select    naam, voorletters  
from     spelers  
where    spelersnr = 18
```

13. Kies outer-joins boven UNION

- UNION : verwerkingstijd verlengd
- Alternatief (indien mogelijk)
 - Outer-join is beter

- Vb.

```
select spelers.spelersnr, naam, bedrag
from spelers, boetes
where spelers.spelersnr = boetes.spelersnr
union
select spelersnr, naam, null
from spelers
where spelersnr not in (select spelersnr
                        from boetes)
order by 1
```

- Vb. Oplossing

```
select spelers.spelersnr, naam, bedrag
from spelers, boetes
where spelers.spelersnr = boetes.spelersnr
union
select spelersnr, naam, null
from spelers
where spelersnr not in (select spelersnr
                           from boetes)
order by 1
```

=> select spelersnr, naam, bedrag
 from spelers left outer join boetes
 using (spelersnr)
 order by 1

14. Vermijd datatype-conversies

- Converteren van datatypes : verwerkingstijd verlengd
- Alternatief (indien mogelijk) :
 - Datatype-conversie vermijden
- Vb.

```
select      *
from        spelers
where       spelersnr = '15'
```

=> where spelersnr = 15

15. Volgorde tabellen

- Volgorde van tabellen kan belangrijk zijn
- Afhankelijk van de juistheid van de interne statistieken
- Vb.

```
select spelers.spelersnr, naam, teamnr  
from spelers, teams  
where spelers.spelersnr = teams.spelersnr  
  
<=> select spelers.spelersnr, naam, teamnr  
from teams, spelers  
where spelers.spelersnr =  
      teams.spelersnr
```

16. Vermijd ANY- en ALL-operatoren

- ANY en ALL : index wordt niet gebruikt
- Alternatief (indien mogelijk)
 - ?
- Vb.

```
Select      spelersnr, naam, geb_datum  
from       spelers  
where      geb_datum <= all (select geb_datum  
                           from spelers)
```

```
=> select spelersnr, naam, geb_datum  
      from spelers  
     where geb_datum = (select min(geb_datum)  
                          from spelers)
```

Vervang door min of max

Hoe nakijken

- Explain
- Explain analyze

EXPLAIN

inleidende voorbeelden query plan

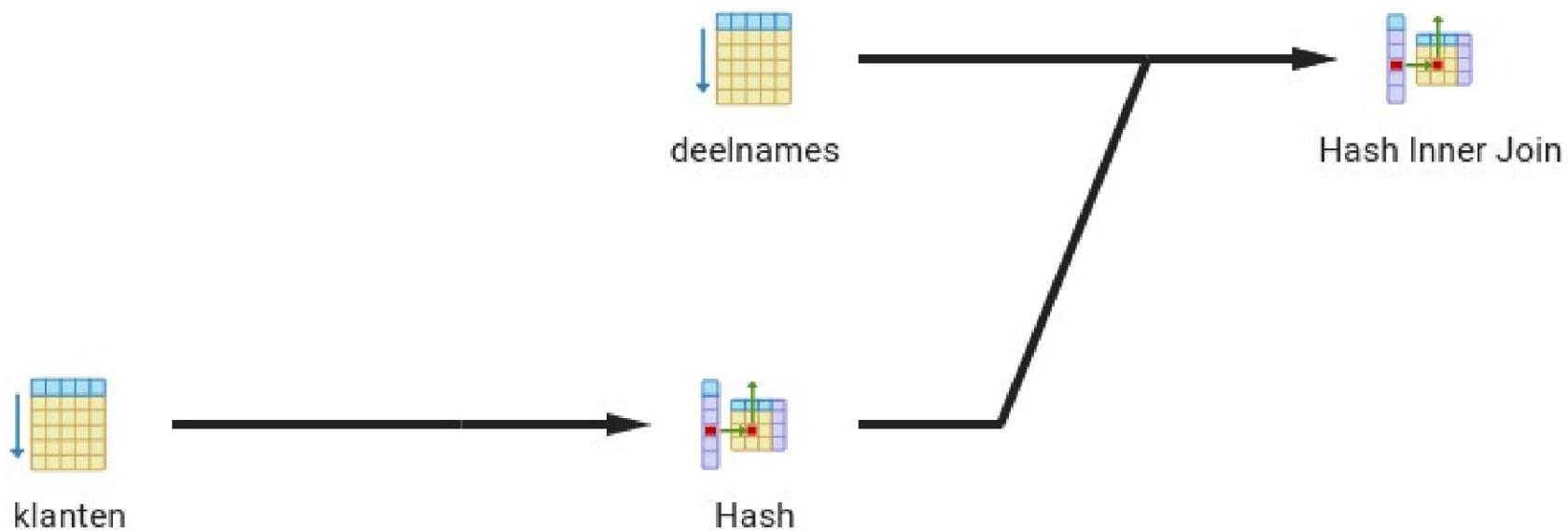
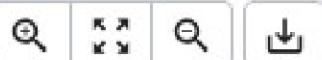
wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0
Unported Licentie

```
3 select *
4 from klanten natural inner join deelnames;
```

Data Output Messages Explain X Notifications

Graphical Analysis Statistics



```

3 select *
4 from klanten natural inner join deelnames;

```

Data Output Messages Explain Notifications

Graphical Analysis Statistics

#	Node
1.	→ Hash Inner Join Hash Cond: (deelnames.klantnr = klanten.klantnr)
2.	→ Seq Scan on deelnames as deelnames
3.	→ Hash
4.	→ Seq Scan on klanten as klanten

Graphical Analysis Statistics

Statistics per Node Type

Node type	Count
Hash	1
Hash Inner Join	1
Seq Scan	2

Statistics per Relation

Relation name	Scan count
Node type	Count
deelnames	1
klanten	1

1 tabel

```
CREATE TABLE een_miljoen  
  (teller      integer,  
   random_tekst  text);
```

```
INSERT INTO een_miljoen  
  SELECT i, md5(random()::text)  
    FROM generate_series(1, 1000000) AS i;
```

```
EXPLAIN
SELECT *
FROM een_miljoen;
```

QUERY PLAN

```
Seq Scan on een_miljoen
(cost=0.00..18918.18
rows=1058418 width=36)
```

```
ANALYZE een_miljoen;
```

```
EXPLAIN
```

```
SELECT *
```

```
FROM een_miljoen;
```

QUERY PLAN

```
Seq Scan on een_miljoen  
(cost=0.00..18334.00  
rows=1000000 width=37)
```

EXPLAIN ANALYZE

```
SELECT *
```

```
FROM een_miljoen;
```

QUERY PLAN

```
Seq Scan on een_miljoen
(cost=0.00..18334.00
rows=1000000 width=37)
(actual time=0.008..78.234
rows=1000000 loops=1)
```

```
Planning Time: 0.012 ms
```

```
Execution Time: 106.864 ms
```

```
EXPLAIN
```

```
SELECT *
```

```
FROM een_miljoen
```

```
WHERE teller > 500;
```

QUERY PLAN

```
Seq Scan on een_miljoen
(cost=0.00..20834.00
rows=999507 width=37)
```

```
Filter: (teller > 500)
```

```
CREATE INDEX ON een_miljoen(teller);
EXPLAIN
SELECT *
FROM   een_miljoen
WHERE  teller > 500;
```

QUERY PLAN

Seq Scan on een_miljoen (cost=0.00..20834.00 rows=999491 width=37)

Filter: (teller > 500)

```
EXPLAIN  
SELECT *  
FROM een_miljoen  
WHERE teller < 500;
```

QUERY PLAN

Index Scan using een_miljoen_teller_idx on een_miljoen
(cost=0.42..25.32 rows=508 width=37)

Index Cond: (teller < 500)

???

filter (> 500) Seq Scan: geen index

filter (< 500) Index Scan

- Misschien..

```
SET enable_seqscan TO off;  
EXPLAIN ANALYZE  
SELECT *  
FROM een_miljoen  
WHERE teller > 500;
```

QUERY PLAN

Index Scan using
een_miljoen_teller_idx on
een_miljoen
(cost=0.42..36800.52
rows=999491 width=37)
(actual time=0.023..255.981
rows=999500 loops=1)

Index Cond: (teller >
500)

Planning Time: 0.051 ms

Execution Time: 296.085 ms

QUERY PLAN

Seq Scan on een_miljoen
(cost=0.00..20834.00
rows=999491 width=37)

Filter: (teller > 500)

SET enable_seqscan TO on;

meerdere tabellen

```
CREATE TABLE half_miljoen
  (teller          integer ,
   random_munt    boolean);

INSERT INTO half_miljoen
  SELECT i, i%2=1
    FROM generate_series(1, 500000) AS i;
ANALYZE half_miljoen;
```

```
CREATE TABLE een_miljoen
  (teller      integer ,
   random_tekst text);

INSERT INTO een_miljoen
  SELECT i, md5(random()::text)
    FROM generate_series(1, 1000000) AS i;
```

```
EXPLAIN ANALYZE
SELECT *
FROM een_miljoen JOIN half_miljoen
ON (een_miljoen.teller=half_miljoen.teller);
```

QUERY PLAN

Hash Join (cost=15417.00..60081.00 rows=500000 width=42)
(actual time=102.561..679.936 rows=500000 loops=1)

Hash Cond: (een_miljoen.teller = half_miljoen.teller)

-> Seq Scan on een_miljoen
(cost=0.00..18334.00 rows=1000000 width=37)
(actual time=0.006..76.141 rows=1000000 loops=1)

-> Hash (cost=7213.00..7213.00 rows=500000 width=5)
(actual time=102.465..102.466 rows=500000 loops=1)

Buckets: 262144 Batches: 4 Memory Usage: 6562kB

-> Seq Scan on half_miljoen
(cost=0.00..7213.00 rows=500000 width=5)
(actual time=0.005..33.529 rows=500000 loops=1)

Planning Time: 0.193 ms

Execution Time: 693.629 ms

```
EXPLAIN ANALYZE
SELECT *
FROM een_miljoen JOIN half_miljoen
ON (een_miljoen.teller=half_miljoen.teller);
```

```
CREATE INDEX ON half_miljoen(teller);
EXPLAIN ANALYZE
SELECT *
FROM een_miljoen JOIN half_miljoen
ON (een_miljoen.teller=half_miljoen.teller);
```

QUERY PLAN

Merge Join (cost=1.31..40111.03 rows=500000 width=42)
(actual time=0.014..298.152 rows=500000 loops=1)

Merge Cond: (een_miljoen.teller = half_miljoen.teller)

-> Index Scan using een_miljoen_teller_idx on een_miljoen
(cost=0.42..34317.43 rows=1000000 width=37)
(actual time=0.006..77.998 rows=500001 loops=1)

-> Index Scan using half_miljoen_teller_idx on half_miljoen
(cost=0.42..15212.42 rows=500000 width=5)
(actual time=0.004..83.697 rows=500000 loops=1)

Planning Time: 0.264 ms

Execution Time: 311.930 ms

Hash Join (cost=15417.00..60081.00

Aggregaties

```
EXPLAIN  
SELECT count(*)  
FROM een_miljoen;
```

QUERY PLAN

```
Finalize Aggregate  (cost=14542.55..14542.56 rows=1 width=8)  
-> Gather  (cost=14542.33..14542.54 rows=2 width=8)  
    Workers Planned: 2  
    -> Partial Aggregate  (cost=13542.33..13542.34 rows=1 width=8)  
        -> Parallel Seq Scan on een_miljoen  (cost=0.00..12500.67 rows=416667 width=0)
```

```
EXPLAIN ANALYZE
```

```
SELECT max(random_tekst)  
FROM een_miljoen;
```

QUERY PLAN

```
Finalize Aggregate  (cost=14542.55..14542.56 rows=1 width=32)  
                      (actual time=75.277..79.133 rows=1 loops=1)
```

```
-> Gather      (cost=14542.33..14542.54 rows=2 width=32)  
                      (actual time=75.198..79.124 rows=3 loops=1)
```

Workers Planned: 2

Workers Launched: 2

```
-> Partial Aggregate  (cost=13542.33..13542.34 rows=1 width=32)  
                      (actual time=72.948..72.948 rows=1 loops=3)
```

```
    -> Parallel Seq Scan on een_miljoen  
                      (cost=0.00..12500.67 rows=416667 width=33)  
                      (actual time=0.008..24.153 rows=333333 loops=3)
```

Planning Time: 0.073 ms

Execution Time: 79.155 ms

```
CREATE INDEX ON een_miljoen(random_tekst);
EXPLAIN ANALYZE
SELECT max(random_tekst)
FROM een_miljoen;
```

QUERY PLAN

Result (cost=0.47..0.48 rows=1 width=32)
(actual time=0.035..0.035 rows=1 loops=1)

InitPlan 1 (returns \$0)

-> Limit (cost=0.42..0.47 rows=1 width=33)
(actual time=0.031..0.032 rows=1 loops=1)

-> Index Only Scan Backward using een_miljoen_random_tekst_idx on een_miljoen
(cost=0.42..46340.43 rows=1000000 width=33)
(actual time=0.030..0.031 rows=1 loops=1)

Index Cond: (random_tekst IS NOT NULL)

Heap Fetches: 0

Planning Time: 0.156 ms

Execution Time: 0.049 ms

Groeperingen

```
DROP INDEX een_miljoen_random_tekst_idx;  
EXPLAIN ANALYZE  
SELECT random_tekst, count(*)  
FROM een_miljoen  
GROUP BY random_tekst;
```

QUERY PLAN

```
HashAggregate  (cost=105834.00..131459.00 rows=1000000 width=41)  
              (actual time=473.085..1013.928 rows=1000000 loops=1)  
  
Group Key: random_tekst  
  
Planned Partitions: 16  Batches: 81  Memory Usage: 8337kB  Disk Usage: 63536kB  
  
-> Seq Scan on een_miljoen  
              (cost=0.00..18334.00 rows=1000000 width=33)  
              (actual time=0.007..63.300 rows=1000000 loops=1)
```

```
CREATE INDEX ON een_miljoen(random_tekst);
EXPLAIN ANALYZE
SELECT random_tekst, count(*)
FROM een_miljoen GROUP BY random_tekst;
```

QUERY PLAN

GroupAggregate (cost=0.42..58840.43 rows=1000000 width=41)
(actual time=0.030..440.101 rows=1000000 loops=1)

Group Key: random_tekst

-> Index Only Scan using een_miljoen_random_tekst_idx on een_miljoen
(cost=0.42..43840.43 rows=1000000 width=33)
(actual time=0.024..133.736 rows=1000000 loops=1)

Conclusie

- EXPLAIN (afhankelijk van de statistieken)
Oplossing: ANALYZE voordien
- EXPLAIN ANALYZE
Actuele Uitvoertijd, maar voert dus ook effectief uit!
- EXPLAIN
Werkt ook voor andere DML (insert, ..)
Kijk in eerste instantie naar de totale kost
Kijk eventueel naar de scan methoden

Referenties

- Understanding Explain, guillaume.lelarge@dalibo.com

Dedectie en referentiepunten



Wim.bertels@ucll.be

Trage queries?

- casus: databank beheerder
- middel: pg_stat_statements
- Bewerk postgresql.conf om deze module beschikbaar te maken
 - produkt specifiek

```
# postgresql.conf  
shared_preload_libraries = 'pg_stat_statements'
```

pg_stat_statements.max = 10000

pg_stat_statements.track = all

Create Extension

- In elke databank waar je dit wil gebruiken:
 - CREATE EXTENSION pg_stat_statements;
- `SELECT *
FROM pg_stat_statements;`

Gebruik?

userid	37919	shared_blk_hit	0
dbid	12413	shared_blk_read	0
queryid	2401821812	shared_blk_dirtied	0
query	SELECT version()	shared_blk_written	0
calls	1	local_blk_hit	0
total_time	0.021	local_blk_read	0
min_time	0.021	local_blk_dirtied	0
max_time	0.021	local_blk_written	0
mean_time	0.021	temp_blk_read	0
stddev_time	0	temp_blk_written	0
rows	1	blk_read_time	0
		blk_write_time	0

Referentiepunten

- Middel: pgbench
- Voorbeeld localhost:
- \$ pgbench -i probeer
- \$ pgbench -T 120 probeer
- ..heeft meerdere opties

Resultaten

starting vacuum...end.

transaction type: TPC-B (sort of)

scaling factor: 1

query mode: simple

number of clients: 1

number of threads: 1

duration: 120 s

number of transactions actually processed: 42594

latency average: 2.817 ms

tps = 354.944850 (including connections establishing)

tps = 354.961873 (excluding connections establishing)

Bruikbaarheid

- Simulaties vs Realiteit
- Opvolging!
- Een redelijk vertrekpunt
- Notas:
 - De *initialization scale factor* (-s) zou minstens zo groot moeten zijn als het hoogste aantal klanten (*clients*) dat je wil testen (-c)
 - Autovacuum en belasting(load)

Referenties

- <https://www.postgresql.org/docs/current/static/pgstatstatements.html>
- <https://www.postgresql.org/docs/current/static/pgbench.html>

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License

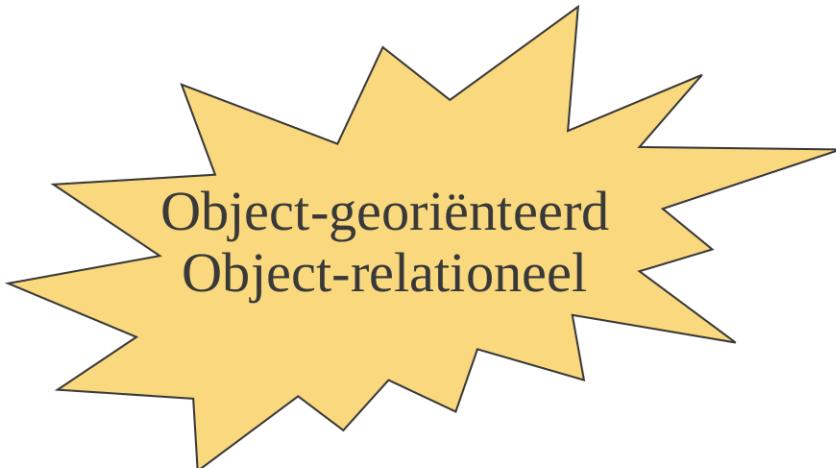
DBMS

OO Soorten

wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Unported
Licentie

Wat



Nummer	Naam		Plaats
1	Peeters		Den Haag
2	Fransen		Arnhem
3	Degent		Den Haag

ORDBMS t.o.v. ODMS

(O)OD(B)MS =

object(-oriented) database management system.

Data as objects (<http://www.odbms.org/introduction-to-odbms/definition/>)

ORDBMS =

object-relational database management system

ORDBMS : belangrijkste speler

ODMS : specialisten – niche markt

ORDBMS -ODMS

- Inleiding
 - ORDBMS en ODMS t.o.v. de rest
 - ORDBMS t.o.v. ODMS
- Kenmerken (ORDMS)
 - Uitbreidung van de basis datatypes
 - Complexe objecten
 - Overerving
 - Regels

ORDBMS/ODMS t.o.v. rest

query	2	4
Geen query	1	3
	Simpele gegevens	complexe gegevens

Zie eventueel: <http://www.odbms.org/wp-content/uploads/2013/11/013.01-Chountas-RDBMS-versus-ORDBMS-versus-OODBMS-August-2005.pdf>

ODMS

wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Unported
Licentie



<http://en.wikipedia.org/wiki/File:Sangreal.jpg>

ODMS – producten – taal - ..

- <https://realm.io/>
- <http://www.mcobject.com/persest>
- <http://www.zodb.org/>
- <http://www.odaba.com/content/tools/odaba/>
- ..
- SQL >> OQL .. (ODMG)

geënt op OOPL concepten

- Nieuw?: smalltalk (1972)
- Kenmerken :
 - objecten
 - State vs Behaviour (Zijn tov Kunnen)
 - object identifier (OID) = unique and immutable
 - Complexe types en structuren
 - atomic, struct(tuple)
 - collection (set, list, bag, array, dictionary(kv))

OOPL kenmerken(2)

- Kenmerken :
 - Inkapseling
 - (Tijdelijk(transient) vs persistent)
 - Overerving
 - Polymorfisme (operator overloading)

ODL (kort)

- Objecten vertaald :
 - In Practice : Value vs Reference
 - Reference : object_id (OID)
- Levensduur : transient vs persistent
- Structuur : atomic of samengesteld
- Create : New
- Overvинг : Extends
- ..

ODL (eenvoudig voorbeeld)

```
class STUDENT

(extent      PERSISTENT_STUDENTS /*persistent*/
Key          Ssid)
{attribute string  Ssid;
attribute string  FamilieNaam;
attribute        ..
relationship    REEKS zitIn
                inverse REEKS::heeftStudenten;
void          verplaatsStudent(in string  NewReeks)
                raises(NewReeksBestaatNiet)
}
```

ODL (voorbeeld (2))

```
class REEKS

(extent      REEKSEN
Key        Rnaam)

{attribute  string          Rnaam;
attribute   ..
relationship set<REEKS> heeftStudenten
              inverse  REEKS::zitIn;
void       voegStudentToe(in string  NewReeks)
              raises(NewReeksBestaatNiet)
}
```

ODMS : kolom objecten

```
class AUTO
(..)
)
{attribute      string      Snrplt;
attribute      STUDENT    Eigenaar;
attribute      ..
}
```

ODMS : geneste objecten

```
class STUDENT
(extent      STUDENTEN
..
)
{..
attribute   struct Adres{string straat;
                         string huisnr;
..
}
..
}
```

ODMS : collections

- set<type>
- bag<type>
- list<type>
- array<type>
- dictionary<key,value>

ODMS : overerving

```
class BRAVE_STUDENT extends STUDENT
(..)
{
attribute    string      nieuwJaarBrief;
..
}
```

OQL voorbeelden

```
select S.FamilieNaam  
from   S in PERSISTENT_STUDENTS  
where  S.Ssid = '12345';
```

REEKSEN;

STUDENT1.Adres;

```
select distinct S.Ssid  
from   S in REEKSEN.heeftStudenten;
```

ORDMS

wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Unported
Licentie

ORDBMS : kenmerken

- Uitbreiding van de basis datatypes
 - Complexe objecten
- Overerving
 - Overerving van gegevens
 - Overerving van types
- Regels (bv)
 - Update-update regel
 - Query-update regel
 - Update-query regel
 - Query-query regel
- ..

Inleiding

- Complexere en meer specialistische datatypes
 - => zelf gedefinieerde datatypes

Zelf definiëren van datatypes

- CREATE TYPE : creëert datatype
- DROP TYPE : verwijdert datatype
- ALTER TYPE : haalbare aanpassingen
- Bv

CREATE TYPE coordinaat

AS (x int, y int);

-- composiet

Toegang tot datatypes

- Eigenaar van datatype : degene die type creëert
- Degene die type wil gebruiken, moet machtiging krijgen :

```
grant      usage
on type    geldbedrag
to         Jim
```
- Verwijderen van machtiging :

```
revoke    usage
on type   geldbedrag
from      Jim
```
- Zie ook notas <https://www.postgresql.org/docs/current/sql-createtype.html>

Casting van waardes

- Casting : om verschillende datatypes te kunnen vergelijken
- Destructor : transformeert zelfgedefinieerd datatype naar basisdatatype
- Constructor : transformeert basisdatatype naar zelfgedefinieerd datatype

Voorbeelden

- Vb.

```
select *  
from boetes  
where bedrag > geldbedrag(50);
```

Bedrag is van
type geldbedrag

```
select *  
from boetes  
where decimal(bedrag) > 50 ;
```

```
insert into plaats (waar,naam) values  
(coordinaat(12,57), 'niverance') ;
```

Zelf definiëren van operatoren

- Voorafgaandelijke opmerkingen :
 - Operatoren zijn toegevoegd voor het gemak
 - Bij elk basis-datatype horen operatoren
 - Operatoren hebben een betekenis alnaargelang het datatype
 - Definiëren van operatoren op zelfgedefinieerde datatypes ~ operaties op onderliggend type

Voorbeeld

- Definiëren van operatoren en bijhorende functies :

```
CREATE OR REPLACE FUNCTION plus_vector(in_c1 coordinaat, in_c2 coordinaat)
```

```
    RETURNS coordinaat AS
```

```
$$
```

```
DECLARE
```

```
    result coordinaat;
```

```
BEGIN
```

```
    result.x := in_c1.x + in_c2.x;
```

```
    result.y := in_c1.y + in_c2.y;
```

```
    RETURN result;
```

```
END;
```

```
$$
```

```
LANGUAGE plpgsql;
```

Voorbeeld

- Bijhorende operator :

create operator + (

 leftarg = coordinaat,

 rightarg = coordinaat,

 function = plus_vector,

 commutator = +

);

Voorbeeld

- Gebruik :

```
select (1,10)::coordinaat + (20,200)::coordinaat as vector_som;  
select cast((1,10) as coordinaat) + cast((20,200) as coordinaat) as vector_som;
```

vector_som

(21,210)

(1 row)

Opaque-datatype

- Def. : datatype dat niet afhankelijk is van een basisdatatype
- Definiëren van functies om hiermee te werken is noodzakelijk
- Vb.

```
create type tweedim (internallength = 4) ;  
-- base types in postgresql
```

CREATE TYPE

- ISO :
 - Composiet (bv coordinaat)
 - Opaque (pg : base)
- Niet aanwezig in ISO, pg specifiek :
 - Enum (opsomming, M/V/X)
 - Range (bereik)
 - + Array

Named row-datatype

- Def. : groeperen van waarden die logisch bij elkaar horen (idem composiet)
- Vb.

```
create type adres as
    (straat      char(15) not null,
     huisnr      char(4)      ,
     postcode    char(6)      ,
     plaats      char(10) not null);
```

-- postgresql : zonder not null constraint

Voorbeeld

```
create table spelers
```

```
    (spelersnr      integer primary key,
```

```
    ...
```

```
    woonadres      adres      ,
```

```
    postadres      adres      ,
```

```
    vakantieadres  adres      ,
```

```
    ...
```

```
) ;
```

```
select spelersnr, woonadres
```

```
from spelers
```

```
where (woonadres).plaats = 'Leuven' ;
```

Unnamed row-datatype

Unnamed row-datatype :: groeperen van waarden die logisch bij elkaar horen zonder een naam te geven

- Vb.

```
create table spelers
  (spelersnr      smallint      primary key,
   ...
   woonadres      row (straat    char(15)      not null,
                      huisnr     char(4)       ,
                      postcode   char(6)       ,
                      plaats    char(10)      not null),
   telefoon        char(10) ,
   ...
  );
```

-- postgresql : unnamed rows worden enkel als input toegelaten, niet in DDL

Getypeerde tabel

- Def. : een datatype toekennen aan een tabel
- Eenvoudig om gelijkende tabellen te creëren
- Vb. create type t_spelers as

```
(spelersnr integer      not null,  
naam        char(15)    not null,  
...  
bondsnr     char(4));
```

```
create table spelers of t_spelers  
          (primary key spelersnr) ;
```

Tabel > Tabel

- Basis structuur een bestaande tabel overnemen (fk's verhuizen niet mee!):
 - Vb. `create table nieuwe_klanten
(like ruimtereizen.klanten including indexes);`
 - zie including optie voor verschillende opties

Integriteitsregels op datatypes

- Beperkingen op de toegestane waardes
- Vb.

```
create type aantal_sets as smallint
                  check (value in (0, 1, 2, 3));
create table wedstrijden
  (wedstrijdnr integer      primary key,
   teamnr        integer      not null,
   spelersnr     integer      not null,
   gewonnen     aantal_sets not null,
   verloren      aantal_sets not null);
```

-- Wat is een DOMAIN in RDBMS?

-- PG : via enum

Sleutels en indexen

- Is volledig analoog bij zelfgedefinieerde datatypes
 - Operator nodig voor index moet kunnen gebruikt worden
- Bij named row-datatypes (composiet):
 - op de volledige waarde
 - op een deel ervan
- CREATE INDEX coordinaat_x_idx ON your_table ((coord_column).x);

Overerving, references, collecties

1. Overerving van datatypes
2. Koppelen van tabellen via rij-identificaties
3. Collecties
4. Overerving van tabellen
5. Regels

Overerving van datatypes

- Def. : alle eigenschappen van één datatype worden overgeërfd door een ander (supertype en subtype)
- Vb.

```
create type adres as
    (straat      char(15)      not null,
     huisnr      char(4),
     poscode     char(6),
     plaats      char(10)      not null);
```

```
create type buitenlands_adres as
    (land      char(20)      not null) under adres ;
```

-- postgresql : ..of rechtstreeks of like in base type..

Koppelen van tabellen

- In OO-DB : alle rijen hebben een unieke identificatie (door het systeem)
- REF : om identificatie op te vragen
- Vb.

```
select ref(spelers)
from spelers
where spelersnr = 6 ;
```

-- postgresql : eventueel via oids, weinig gebruikt

Voorbeeld

- REF : om tabellen te koppelen (niet in pg)
- Vb.

```
create table teams
  (teamnr      smallint      primary key,
   spelers      ref(spelers)  not null,
   divisie     char(6)       not null);
```

```
insert into teams (teamnr, spelers, divisie)
values (3, (select ref(spelers)
            from spelers
            where spelersnr = 112), 'ere');
```

```
select teamnr, spelers.naam
from teams;
```

Pro-Contra

- Voordelen :

- Altijd het juiste datatype bij de refererende sleutel
- Indien primary keys breed zijn, bespaart het werken met reference-kolommen opslagruimte
- Bij wijzigen van primary keys wordt geen tijd verloren door het wijzigen van de refererende sleutel
- Bepaalde selects worden eenvoudiger

- Nadelen :

- Bepaalde mutaties zijn moeilijker te definiëren
- References werken in één richting
- Bij DB-ontwerp krijgt men meerdere keuzes, dit wordt dus moeilijker
- References kunnen de integriteit van de gegevens niet bewaken zoals refererende sleutels dat kunnen

Collecties

- Collecties : verzameling waardes in één cel
- Vb.

```
create table spelers
  (spelersnr smallint primary key,
   ...
   telefoons setof(char(13)),
   bondsnr char(4)
  );
```

```
insert into spelers (spelersnr, ..., telefoons, ...) values
  (213, .., {'016-342654', '0475-654387'}, ..);
```

```
select spelersnr
  from spelers
 where '016-342654' in (telefoons);
```

-- pg: meestal via arrays[] ..

Overerving van tabellen

- Def. : alle eigenschappen van één tabel worden overgeërfd door een andere tabel (supertabel – subtabel)
- Beperkingen :
 - Geen cyclische structuur

Voorbeeld

```
create table spelers as
  (spelersnr      smallint      not null,
   naam          char(15)       not null,
   ...
   bondsnr        char(4));
```

```
create table oude_spelers
  (vertrokken     date         not null)
  inherits (spelers, okra);
```

```
SELECT  *
FROM    (ONLY) spelers;
-- let op met inserts, constraints ..
-- postgresql
```

RULES

- Gaan verder dan triggers : SELECT, INSERT, UPDATE, DELETE
- FKs >> triggered
- Updateable views
 - rules of triggers
- Maar voorzichtig, systeemlogica

CREATE RULE (bv)

CREATE RULE "NeKeerletsAnders" AS

ON SELECT TO wedstrijden

WHERE spelersnr = 7

DO INSTEAD

SELECT 'eerst drie toerkes rond tafel lopen
en dan nog eens proberen';

GTE (CTE)

Gemeenschappelijke Tabel Expressies

wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Unported
Licentie

Wat

- Vergelijkbaar met subqueries in the from, maar met extras.
- Soms ook with queries genoemd, meestal cte in het engels
- <https://www.postgresql.org/docs/current/interactive/queries-with.html>

Dummy syntax voorbeeld

- SELECT *

```
FROM  (SELECT *
       FROM spelers) AS spelers_sub;
```

- /*of*/

- WITH spelers_sub as (

```
    SELECT      *
    FROM spelers
)
SELECT *
FROM spelers_sub;
```

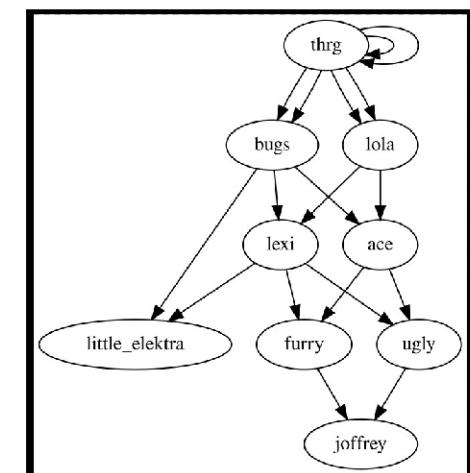
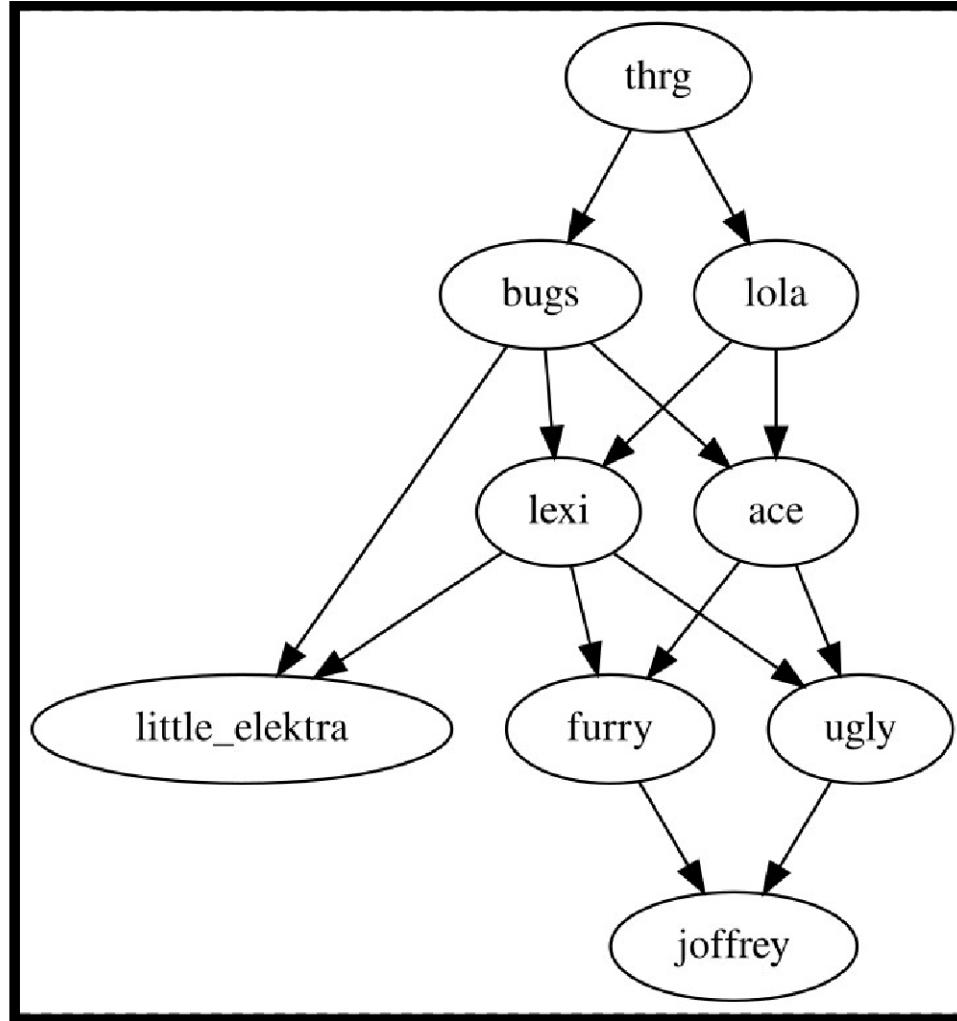
Bugs and Lola

- CREATE TABLE familieboom(
 bijnaam varchar(16) NOT NULL,
 vader varchar(16),
 moeder varchar(16),
 CONSTRAINT familieboom_pk
 PRIMARY KEY (bijnaam),
 CONSTRAINT vader_fk
 FOREIGN KEY (vader) REFERENCES familieboom(bijnaam),
 CONSTRAINT moeder_fk
 FOREIGN KEY (moeder) REFERENCES familieboom(bijnaam)
);

Data

- INSERT INTO familieboom VALUES

```
('thrg','thrg','thrg'),  
('lola','thrg','thrg'),  
('bugs','thrg','thrg'),  
('lexi','bugs','lola'),  
('ace','bugs','lola'),  
('furry','lexi','ace'),  
('ugly','lexi','ace'),  
('little_elektra','bugs','lexi'),  
('joffrey','furry','ugly');
```



De nakomelingen van thrg

```
• SELECT *  
  FROM familieboom  
 WHERE 'thrg' in (vader,moeder);
```

bijnaam	vader	moeder
thrg	thrg	thrg
lola	thrg	thrg
bugs	thrg	thrg
(3 rows)		

De nakomelingen van bugs

- ```
SELECT *
 FROM familieboom
 WHERE vader='bugs';
```

| bijnaam        | vader | moeder |
|----------------|-------|--------|
| lexi           | bugs  | lola   |
| ace            | bugs  | lola   |
| little_elektra | bugs  | lexi   |
| (3 rows)       |       |        |

# De directe nakomelingen van bugs

- ```
SELECT *  
FROM familieboom  
WHERE vader='bugs';
```

bijnaam	vader	moeder
lexi	bugs	lola
ace	bugs	lola
little_elektra	bugs	lexi
(3 rows)		

- -- Dit zijn de lamprelen van bugs
 - We willen ook de kleinlamprelen van bugs zien
 - Hoe doe je dit?

De (klein)lamprelen van bugs

- ```
SELECT *
 FROM familieboom
 WHERE vader = 'bugs'
OR vader IN
 (SELECT bijnaam
 FROM familieboom
 WHERE vader = 'bugs');
```

| bijnaam        | vader | moeder |
|----------------|-------|--------|
| lexi           | bugs  | lola   |
| ace            | bugs  | lola   |
| furry          | lexi  | ace    |
| ugly           | lexi  | ace    |
| little_elektra | bugs  | lexi   |
| (5 rows)       |       |        |

# De(..)lamprenen van bugs

- ```
SELECT *
  FROM familieboom
 WHERE vader = 'bugs'
OR vader IN
  (SELECT bijnaam
  FROM familieboom
 WHERE vader = 'bugs');
```

bijnaam	vader	moeder
lexi	bugs	lola
ace	bugs	lola
furry	lexi	ace
ugly	lexi	ace
little_elektra	bugs	lexi

(5 rows)

- -- Worden nu ook de achterkleinlamprenen teruggegeven?
-- Waar moeten we stoppen?

Recurzie

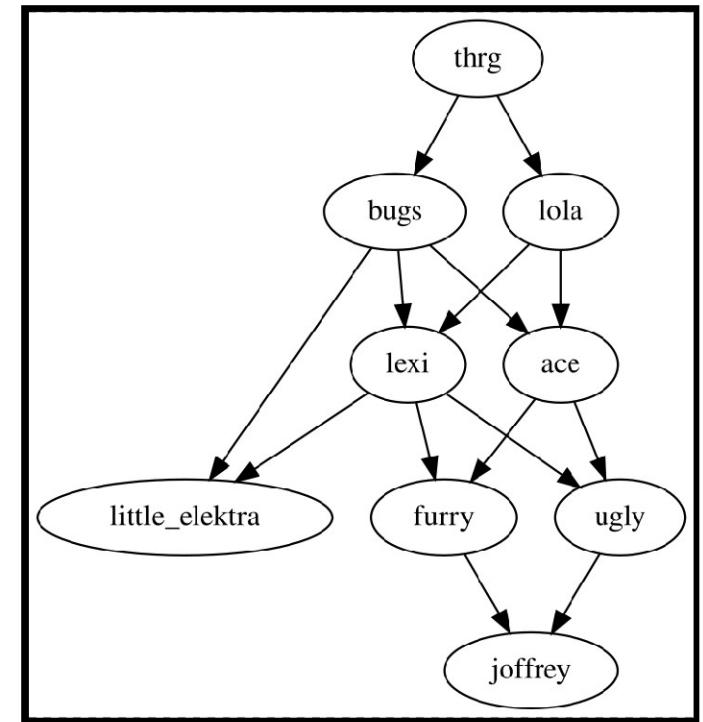
- Cf functionaliteit iteratie, maar men gebruikt het woord recursie
- Alle getallen van 1 tot 100 optellen:

```
cte=# WITH RECURSIVE t(n) AS (
    VALUES (1)
    UNION ALL
    SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
sum
-----
5050
(1 row)
```

Hoe kunnen we dit nu gebruiken?

- WITH RECURSIVE nakomeling(bijnaam, vader, moeder) AS (

```
SELECT bijnaam, vader, lola
FROM familieboom
WHERE vader = 'bugs'
UNION ALL
SELECT f.bijnaam, f.vader, f.moeder
FROM familieboom f, nakomeling k
WHERE f.vader = k.bijnaam
)
SELECT *
FROM nakomeling;
```



Bugs zijn nakomelingen

- WITH RECURSIVE nakomeling(bijnaam, vader, moeder) AS (

```
SELECT bijnaam, vader, lola
FROM familieboom
WHERE vader = 'bugs'
UNION ALL
SELECT f.bijnaam, f.vader, f.moeder
FROM familieboom f, nakomeling k
WHERE f.vader = k.bijnaam
)
SELECT *
FROM nakomeling;
```

bijnaam	vader	moeder
lexi	bugs	lola
ace	bugs	lola
little_elektra	bugs	lexi
furry	lexi	ace
ugly	lexi	ace
joffrey	furry	ugly
(6 rows)		

Opbouw

- Basis geval:

```
SELECT bijnaam, vader, moeder  
FROM familieboom  
WHERE vader = 'bugs'
```

Opbouw

- Referentie kader nakomeling

WITH RECURSIVE **nakomeling**(bijnaam, vader,moeder) AS (

SELECT bijnaam, vader, moeder

FROM familieboom

WHERE vader = 'bugs'

Opbouw

- Recursie

```
WITH RECURSIVE nakomeling(bijnaam, vader,moeder) AS (
```

```
    SELECT bijnaam, vader, moeder
```

```
    FROM familieboom
```

```
    WHERE vader = 'bugs'
```

```
UNION ALL
```

```
    SELECT f.bijnaam, f.vader, f.moeder
```

```
    FROM familieboom f, nakomeling n
```

```
    WHERE f.vader = n.bijnaam
```

Opbouw

- Volledig:

```
WITH RECURSIVE nakomeling(bijnaam, vader,moeder) AS (
    SELECT bijnaam, vader, moeder
    FROM familieboom
    WHERE vader = 'bugs'
    UNION ALL
    SELECT f.bijnaam, f.vader, f.moeder
    FROM familieboom f, nakomeling n
    WHERE f.vader = n.bijnaam
)
SELECT bijnaam
FROM nakomeling;
```

- -- niet elke recursieve query is hetzelfde

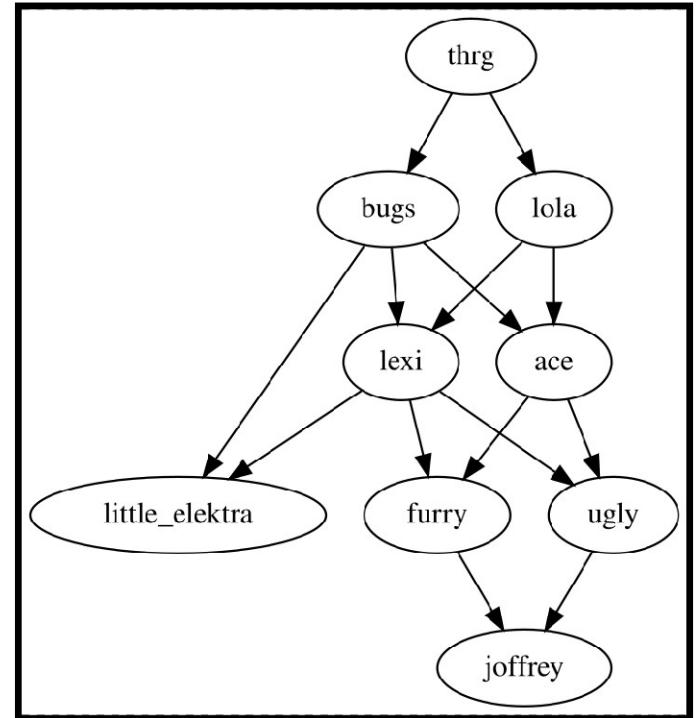
bijnaam

lexi
ace
little_elektra
furry
ugly
joffrey
(6 rows)

Genealogie boom

- Van wie zijn nu al die nakomelingen?

bijnaam	boom
lexi	bugs
ace	bugs
little_elektra	bugs
furry	bugs-lexi
ugly	bugs-lexi
joffrey	bugs-lexi-furry
(6 rows)	



Genealogie boom?

- Probeer de query aan te passen (je mag andere kolommen toevoegen..):

```
WITH RECURSIVE nakomeling(bijnaam, vader,moeder) AS (
```

```
    SELECT bijnaam, vader, moeder
```

```
    FROM familieboom
```

```
    WHERE vader = 'bugs'
```

```
UNION ALL
```

```
    SELECT f.bijnaam, f.vader, f.moeder
```

```
    FROM familieboom f, nakomeling n
```

```
    WHERE f.vader = n.bijnaam
```

```
)
```

```
SELECT bijnaam
```

```
FROM nakomeling;
```

bijnaam	boom
lexi	bugs
ace	bugs
little_elektra	bugs
furry	bugs-lexi
ugly	bugs-lexi
joffrey	bugs-lexi-furry
(6 rows)	

Tijd voor een micropauze

Genealogie boom?

- Probeer de query aan te passen (je mag andere kolommen toevoegen..):

```
WITH RECURSIVE nakomeling(bijnaam, vader,moeder, boom) AS (
```

```
    SELECT bijnaam, vader, moeder, vader::text
```

```
    FROM familieboom
```

```
    WHERE vader = 'bugs'
```

```
UNION ALL
```

```
    SELECT f.bijnaam, f.vader, f.moeder, boom || '-' || f.vader
```

```
    FROM familieboom f, nakomeling n
```

```
    WHERE f.vader = n.bijnaam
```

```
)
```

```
SELECT bijnaam, boom
```

```
FROM nakomeling;
```

bijnaam	boom
lexi	bugs
ace	bugs
little_elektra	bugs
furry	bugs-lexi
ugly	bugs-lexi
joffrey	bugs-lexi-furry
(6 rows)	

Oneindige lussen

- Er is **geen** controle door de CTE
- Zelf controles inbouwen

NO WAY

Donuts

Oneindige lussen

2 strategiën

- teller
- lus dedectie
- combineerbaar



Teller voor de maximum diepte (1)

- WITH RECURSIVE nakomeling(bijnaam, vader, moeder, **diepte**) AS (

```
    SELECT bijnaam, vader, moeder, 1
    FROM   familieboom
   WHERE  vader = 'bugs'
```

```
UNION ALL
```

```
    SELECT f.bijnaam, f.vader, f.moeder, diepte + 1
    FROM   familieboom f, nakomeling k
   WHERE  f.vader = k.bijnaam
   AND    k.diepte<7
```

```
)
```

```
SELECT *
FROM   nakomeling;
```

bijnaam	vader	moeder	diepte
lexi	bugs	lola	1
ace	bugs	lola	1
little_elektra	bugs	lexi	1
furry	lexi	ace	2
ugly	lexi	ace	2
joffrey	furry	ugly	3
(6 rows)			

Iussen dedecteren (2)

- Tabel met lus maken:
- DROP TABLE IF EXISTS familieboom;

```
CREATE TABLE familieboom(  
bijnaam  varchar(16) NOT NULL,  
vader     varchar(16),  
moeder    varchar(16)  
);
```

```
INSERT INTO familieboom VALUES ('lexi','bugs','lola');  
INSERT INTO familieboom VALUES ('bugs','lexi','lola');
```

Parameter met pad bijhouden

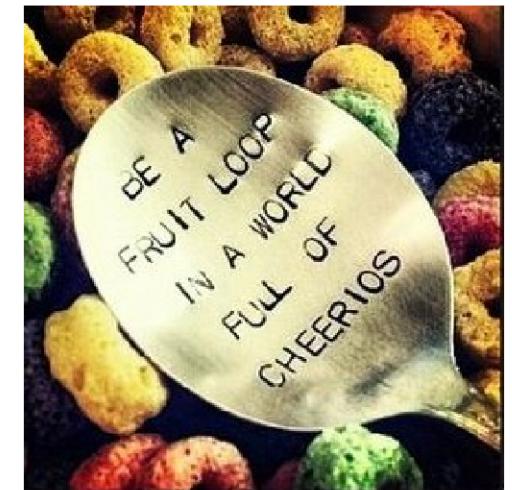
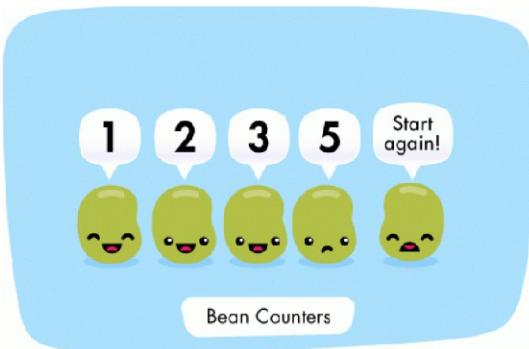
```
WITH RECURSIVE nakomeling(bijnaam, vader, moeder, pad, lus) AS (
    SELECT bijnaam, vader, moeder, ARRAY[vader] as pad, false
    FROM familieboom
    WHERE vader = 'bugs'
UNION ALL
    SELECT f.bijnaam, f.vader, f.moeder, CAST(k.pad || ARRAY[f.vader] as varchar(16)[]) as pad, f.vader = ANY(pad)
    FROM familieboom f, nakomeling k
    WHERE f.vader = k.bijnaam
    AND NOT lus
)
SELECT *
FROM nakomeling;
```

bijnaam	vader	moeder	pad	lus
lexi	bugs	lola	{bugs}	f
bugs	lexi	lola	{bugs,lexi}	f
lexi	bugs	lola	{bugs,lexi,bugs}	t

(3 rows)

Teller of dedectie?

Favoriet?



Uitbreiding

- CTEs werken ook met INSERT, UPDATE, DELETE
- Er is geen ISO gedefinieerd, een uitbreiding van PostgreSQL

Temp tabel

- CREATE TEMPORARY TABLE demo (x NUMERIC);

INSERT INTO demo

VALUES (random()), (random()), (random())

RETURNING x;

x
0.831172323863795
0.867941875721334
0.925603709126386
(3 rows)

Combinatie

- WITH source AS (
 INSERT INTO demo
 VALUES (random()), (random()), (random())
 RETURNING x
)
SELECT AVG(x) FROM source;

avg
0.61717007165672133333
(1 row)

Genetica?!

- Ah, oei, patat: probeer bugs en al zijn kinderen uit de boom te wissen

```
cte=# \h delete
Command:      DELETE
Description:  delete rows of a table
Syntax:
[ WITH [ RECURSIVE ] with_query [, ...] ]
DELETE FROM [ ONLY ] table_name [ * ] [ [ AS ] alias ]
           [ USING from_item [, ...] ]
           [ WHERE condition | WHERE CURRENT OF cursor_name ]
           [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...] ]
```

Genetica?!

- Probeer de query aan te passen (je mag andere kolommen toevoegen..):

```
WITH RECURSIVE nakomeling(bijnaam, vader,moeder) AS (
    SELECT bijnaam, vader, moeder
    FROM familieboom
    WHERE vader = 'bugs'
    UNION ALL
    SELECT f.bijnaam, f.vader, f.moeder
    FROM familieboom f, nakomeling n
    WHERE f.vader = n.bijnaam
)
SELECT bijnaam, boom
FROM nakomeling;
```

Recursief verwijderen

- WITH RECURSIVE **nakomeling**(bijnaam) AS (

```
    SELECT 'bugs'::varchar  
    UNION ALL  
    SELECT f.bijnaam  
    FROM   nakomeling k JOIN familieboom f ON (k.bijnaam=f.vader)  
)
```

```
DELETE FROM familieboom  
USING  nakomeling  
WHERE  nakomeling.bijnaam = familieboom.bijnaam;
```

Ontvlooien met CTEs?

- Zoek alle spelers met een evenlange naam, stapsgewijs:

- WITH lengte_namen AS (

```
    SELECT naam, length(naam) AS n
    FROM spelers),
```

```
evenlange_namen AS (
```

```
    SELECT l1.naam AS naam1, l2.naam AS naam2
    FROM lengte_namen l1 JOIN lengte_namen l2 ON (l1.n = l2.n)
    WHERE l1.naam <> l2.naam),
```

```
overzichts_lijst AS (
```

```
    SELECT naam1, array_agg(naam2) AS lijst
    FROM evenlange_namen
    GROUP BY naam1)
```

```
SELECT l.naam, o.lijst
```

```
FROM lengte_namen l LEFT JOIN overzichts_lijst o ON (l.naam = o.naam1);
```

- -- Klopt dit?



Ontvlooien met CTEs?

- Tussen resultaten wegschrijven om eventuele fouten te vinden:
- CREATE TEMPORARY TABLE debug_table
(id serial,
t text,
r text);

Ontvlooien met CTEs?

- WITH lengte_namen AS (

```
    SELECT naam, length(naam) AS n FROM spelers),
```

```
debug_lengte_namen AS (
```

```
    INSERT INTO debug_table(t,r)
```

```
        SELECT 'lengte_namen', ROW(l.*)::text
              FROM lengte_namen l),
```

```
evenlange_namen AS (
```

```
    SELECT l1.naam AS naam1, l2.naam AS naam2
```

```
    FROM lengte_namen l1 JOIN lengte_namen l2 ON l1.n = l2.n
```

```
    WHERE l1.naam <> l2.naam),
```

```
overzichts_lijst AS (
```

```
    SELECT naam1, array_agg(naam2) AS lijst
```

```
    FROM evenlange_namen
```

```
    GROUP BY naam1)
```

```
SELECT l.naam, o.lijst
```

```
FROM lengte_namen l LEFT JOIN overzichts_lijst o ON l.naam = o.naam1;
```

naam	lijst
Elfring	{"Hofland", "Moerman"}
Permentier	{"Bakker, de", "Nieuwenburg"}
Wijers	
Nieuwenburg	{"Bakker, de", "Permentier"}
Cools	
Cools	
Bischoff	{"Meuleman"}
Bakker, de	"Permentier", "Nieuwenburg"
Bohem, van	
Hofland	{"Elfring", "Moerman"}
Meuleman	{"Bischoff"}
Permentier	{"Bakker, de", "Nieuwenburg"}
Moerman	{"Elfring", "Hofland"}
Baalen, van	

(14 rows)

Ontvlooien met CTEs?

naam	lijst
Elfring	{"Hofland", "Moerman"}
Permentier	{"Bakker, de", "Nieuwenburg", "Bakker, de", "Nieuwenburg"}
Wijers	
Nieuwenburg	{"Bakker, de", "Permentier", "Permentier"}
Cools	
Cools	
Bischoff	{"Meuleman"}
Bakker, de	{"Permentier", "Nieuwenburg", "Permentier"}
Bohemens, van	
Hofland	{"Elfring", "Moerman"}
Meuleman	{"Bischoff"}
Permentier	{"Bakker, de", "Nieuwenburg", "Bakker, de", "Nieuwenburg"}
Moerman	{"Elfring", "Hofland"}
Baalen, van	
(14 rows)	

Debug tabel

```
oefeningen=> SELECT * FROM debug_table ORDER BY regexp_replace(r,'[\D]+','','g');
```

id	t	r
8	lengte_namen	("Bakker, de ",10)
2	lengte_namen	("Permentier ",10)
12	lengte_namen	("Permentier ",10)
4	lengte_namen	("Niewenburg ",10)
14	lengte_namen	("Baalen, van ",11)
9	lengte_namen	("Bohemens, van ",12)
6	lengte_namen	("Cools ",5)
5	lengte_namen	("Cools ",5)
3	lengte_namen	("Wijers ",6)
1	lengte_namen	("Elfring ",7)
10	lengte_namen	("Hofland ",7)
13	lengte_namen	("Moerman ",7)
7	lengte_namen	("Bischoff ",8)
11	lengte_namen	("Meuleman ",8)

```
(14 rows)
```

```
oefeningen=> SELECT * FROM debug_table ORDER BY regexp_replace(r,'[\D]+','','g')::numeric;[]
```

Referenties

- Programming the SQL Way with Common Table Expressions, Bruce Momjian
- Debugging complex SQL queries with writable CTEs, Gianni Ciolfi
- Postgis Latest News, Vincent Picavet

NOSQL

Not Only SQL
-> niet relationeel

Enkele Voorbeelden

- Wide Column Store
 - Bv Cassandra, HBase, ScyllaDB
- Document Store
 - Bv MongoDB, Couchbase, Realm
- Graph based
 - Bv Neo4J, Memgraph, NebulaGraph
- Key Value
 - Bv Redis, Berkeley DB, MemcacheDB
- Multivalue
 - Bv SciDB, Rasdaman, OpenQM
- Search Engines
 - Bv Elasticsearch, Solr, OpenSearch

Enkele Voorbeelden

- Time Series DBMS
 - Bv InfluxDB, Prometheus, TimescaleDB
- RDF Stores
 - Bv Apache Jena – TDB, RDF4J, 4store
- Vector DBMS
 - Bv Chroma, Weaviate, Milvus
- Spatial DBMS
 - Bv PostGIS, spatioLite, GeoMesa
- Native XML DBMS
 - Bv BaseX, Sedna, eXist-db
- Object oriented DBMS
 - Bv ObjectBox, Perst, WakandaDB

Enkele Voorbeelden

- Event stores
 - Bv EventStoreDB, NeventStore
- Content stores
 - Bv Jackrabbit, ModeShape
- Multi-model DBMS
 - Verschillende RDBMS ondersteunen vaak verschillende paradigma
 - Bv PostgreSQL, MariaDB, ClickHouse

Mongo db

- Binaire JSON
- JSON voorbeeld:
- CRUD
- Index
- Aggregation
- Replication
- Sharding

```
{  
    "firstName": "John",  
    "lastName" : "Smith",  
    "age"      : 25,  
    "address"  :  
    {  
        "streetAddress": "21 2nd  
                           Street",  
        "city"          : "New York",  
        "state"         : "NY",  
        "postalCode"   : "10021"  
    },  
    "phoneNumber": [  
        {  
            "type"  : "home",  
            "number": "212 555-1234"  
        },  
        {  
            "type"  : "fax",  
            "number": "646 555-4567"  
        }  
    ]  
}
```

KV

- Look up Value using Key
- Structuur?
- Cf Dictionary (ODMS)
- RDBMS oplossingen..
 - Postgresql : hstore of gewoon..

```
CREATE TABLE kvp
```

```
(id    SERIAL NOT NULL,  
key   text      NOT NULL,  
value text          ,  
CONSTRAINT kvp_pk PRIMARY KEY id );
```

-- KeyValuePair

-- index op key maken

KV

- Look up Value using Key..
- Cf Dictionary (ODMS)
- RDBMS oplossingen..
 - Postgresql : hstore of gewoon..

BASE

- vs ACID
- Basically Available: zo beschikbaar mogelijk
- Soft-state: geen garantie op consistentie, inhoud kan veranderen zonder invoer van de gebruiker
- Eventually consistent: doel is consistentie, gegeven genoeg tijd