



SAMENVATTING DATABANKEN

1

Martijn Meeldijk



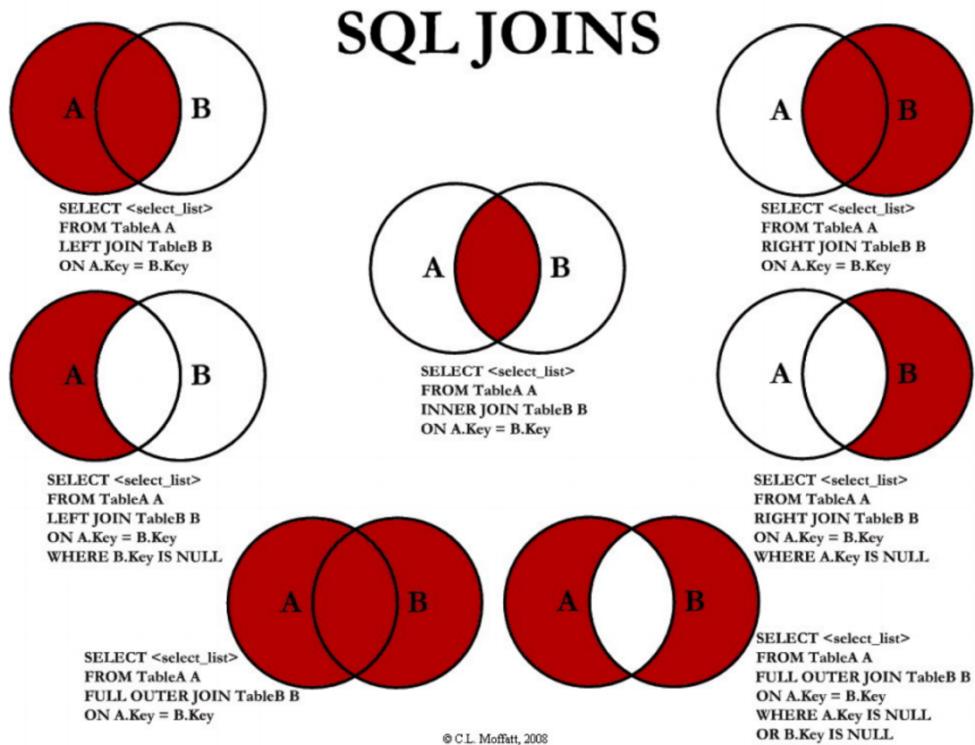
9 JANUARI 2020
UCLL

INHOUD

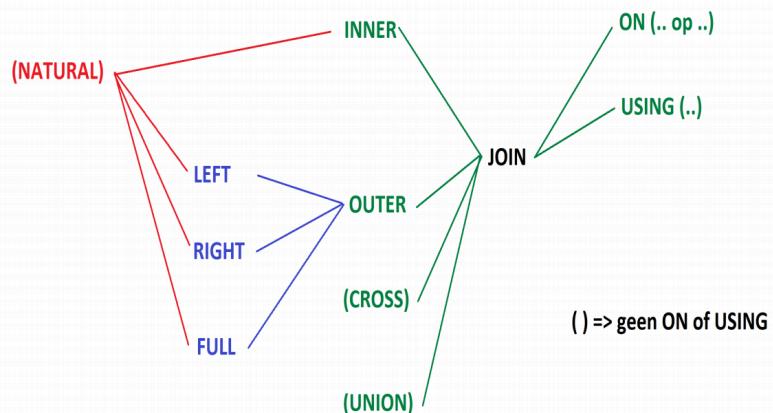
Joins	2
<i>Overzicht</i>	2
<i>Equi/Theta joins</i>	3
<i>Cross join</i>	3
<i>Uitvoer:Natural Join</i>	3
Set-Operators	5
<i>Overzicht</i>	5
<i>Union</i>	6
Views	7
Gebruikers	8
Group by	12
<i>Rollup</i>	12
<i>Cube</i>	13
<i>Grouping sets</i>	14
Filter	15
Subqueries	16
<i>Soorten subqueries</i>	16
Alter	18
<i>Alter Table</i>	18
<i>Alter Database</i>	18
<i>Insert</i>	18
Soorten Databanken	19
<i>Hiërarchische DB</i>	19
<i>Netwerk DB</i>	21
Cast	23
Pattern matching	25
<i>Like</i>	25
<i>Similar to</i>	25

Joins

Overzicht



Notedop



Equi/Theta joins

- EQUI JOIN: vergelijking met = (your average join dus)
- THETA JOIN: vergelijking met een andere vergelijkingsoperator

THETA JOIN

- Tel hoeveel spelers er zijn met een ander nummer
- We willen telkens de nr, 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 Nieuwenburg	13

Cross join

- Om expliciet aan te duiden dat het om een cartesisch product gaat.
- Maakt code leesbaarder

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

Uitvoer:

teamnr integer	spelersnr integer	divisie character (6)	betalingsnr integer	spelersnr integer	datum date	bedrag numeric (7,2)
1	1	6 ere	1	6	1980-12...	100.00
2	2	27 tweede	1	6	1980-12...	100.00
3	1	6 ere	2	44	1981-05...	75.00
4	2	27 tweede	2	44	1981-05...	75.00
5	1	6 ere	3	27	1983-09...	100.00
6	2	27 tweede	3	27	1983-09...	100.00
7	1	6 ere	4	104	1984-12...	50.00
8	2	27 tweede	4	104	1984-12...	50.00
9	1	6 ere	5	44	1980-12...	25.00
10	2	27 tweede	5	44	1980-12...	25.00
11	1	6 ere	6	8	1980-12...	25.00
12	2	27 tweede	6	8	1980-12...	25.00
13	1	6 ere	7	44	1982-12...	30.00
14	2	27 tweede	7	44	1982-12...	30.00
15	1	6 ere	8	27	1984-11...	75.00
16	2	27 tweede	8	27	1984-11...	75.00

Natural Join

- Een natural join is een join (je kunt zowel natural left als natural right hebben) die er van uitgaat dat de join criteria zijn waar de gelijknamige kolommen in beide tabellen overeenkomen.

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

	spelersnr integer	teamnr integer	divisie character (6)	betalingsnr integer	datum date	bedrag numeric (7,2)
1	6	1	ere	1	1980-12...	100.00

Verschil met inner join?

Kijk maar

```
SELECT *
FROM teams INNER JOIN boetes ON(teams.spelersnr = boetes.spelersnr)
WHERE divisie = 'ere'
```

	teamnr integer	spelersnr integer	divisie character (6)	betalingsnr integer	spelersnr integer	datum date	bedrag numeric (7,2)
1	1	6	ere	1	1	1980-12...	100.00

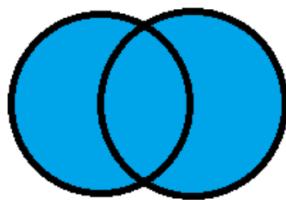
Set-Operators

Overzicht

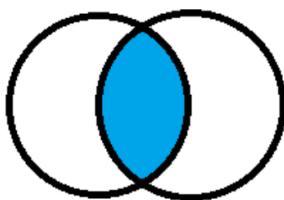
Set-operators

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

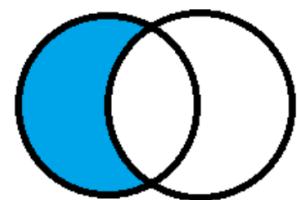
Wiskunde - Verzamelingenleer



Union



Intersect



Except

Union

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)

ALL = behoud dubbels

- Standaard worden dubbele rijen 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.

Views

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 opgebouwe als ie opgevraagd wordt

Gebruikers

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
 - UPDATE: rijen wijzigen m.b.v. UPDATE
 - REFERENCES: refererende sleutels naar deze tabel creëren
 - ALTER: tabel veranderen
 - INDEX: index(en) op de tabel creëren
 - ALL/ALL PRIVILEGES: alle bevoegdheden

3. Databasebevoegdheden

- Soorten databasebevoegdheden:
 - SELECT: bevoegdheid tot SELECT en VIEW
 - INSERT: rijen toevoegen m.b.v. INSERT
 - DELETE: rijen verwijderen m.b.v. DELETE
 - UPDATE: rijen wijzigen m.b.v. UPDATE
 - REFERENCES: refererende sleutels naar deze table creëren
 - ALTER: tabellen veranderen
 - DROP : tabellen verwijderen
 - INDEX : indexen op de tabel creëren
 - CREATE TEMPORARY TABLES : tijdelijke tabellen
 - CREATE VIEW : nieuwe views aanmaken
 - CREATE ROUTINE : nieuwe stored procedures/functies aanmaken
 - ALTER ROUTINE : wijzigen stored procedures/functies
 - EXECUTE ROUTINE : aanroepen van stored procedures/functies
 - LOCK TABLES : bestaande tabellen blokkeren
 - ALL – ALL PRIVILEGES : alle bevoegdheden

4. Gebruikersbevoegdheden

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

Vb. GRANT CREATE, ALTER, DROP
ON *.* TO Frank

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

Vb.

```
CREATE USER Frank;
```

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

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

Group by Rollup

ROLLUP

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

Vb.

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

Doet essentieel hetzelfde als group by, maar geeft een subbtotaal (aggregatie) voor elke gedefinieerde groep.

ROLLUP met 2 nivo's

- 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?

Cube

CUBE

Vergelijkbaar met ROLLUP, maar groepeert per elke mogelijke combinatie ; dus 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?

→ Als we plaats en spelersnr omwisselen, worden eerst de subtotalen per spelernummer gegeven, maar de inhoud van de uitkomst van de query is hetzelfde

Dit is de uitvoer van het voorbeeld. Er wordt voor elke plaats, spelersnummer en voor het alles een subtotaal gegeven van het boetebedrag.

	plaats character varying (30)	spelersnr integer	sum numeric
1	[null]	[null]	480.00
2	Den Haag	6	100.00
3	Zoetermeer	104	50.00
4	Rijswijk	8	25.00
5	Rijswijk	44	130.00
6	Zoetermeer	27	175.00
7	Zoetermeer	[null]	225.00
8	Rijswijk	[null]	155.00
9	Den Haag	[null]	100.00
10	[null]	8	25.00
11	[null]	44	130.00
12	[null]	6	100.00
13	[null]	27	175.00
14	[null]	104	50.00

Grouping sets

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
```

→ Je kan groepen maken van meerdere kolommen waarop de group by gaat groeperen.
() = alles

Groupings 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))

Wat is dit?

Filter

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)]

Subqueries

Soorten subqueries

- Scalaire subquery:
1 rij, 1 waarde
- Rij-subquery:
1 rij
- Kolom-subquery:
elke rij 1 waarde
- Tabel-subquery:
verzameling rijen en kolommen

Subqueries in WHERE

Welke soort subquery mogelijk is, hangt af van de gekozen operator.

=, >, <, ...	Scalair
IN(...)	Kolom

Hoofdquery en subquery

- Hoofdquery krijgt alles wat in SELECT staat van subquery
- Hoofdquery weet niets van detail van subquery, krijgt alleen de output.
- Subquery weet alles van hoofdquery.

Gecorreleerde subqueries

- Subquery waarin een kolom wordt gebruikt die tot een tabel behoort uit een ander select-blok. Dus een gecorreleerde subquery kan niet autonoom uitgevoerd worden.

Exists Operator

- Geeft TRUE of FALSE terug. Als er iets in de uitvoer van de subquery zit, dan TRUE.
Of in de wijze woorden van onze heer Bertels:

operator EXISTS

is er iet,
of nie?

:: TRUE/FALSE

Not Exists

- Als je weet wat 'exists' doet,
veronderstel ik dat je ook weet wat
'not exists' doet.

Any en All

- Extra operator, verwacht scalaire waarde en
kolomexpressie
- > ALL, >= ALL, > ANY, >= ANY, < ALL, < ANY, ...

```
SELECT reisnr
FROM reizen
WHERE reisduur >= ALL (
    SELECT reisduur
    FROM reizen
);

```

Geef de langste reis.

operator ANY en ALL

```
SELECT    reisnr
FROM      reizen
WHERE     reisduur < ANY (
    SELECT    reisduur
    FROM      reizen anderereizen
)
/*Geef alle reizen, behalve de langste reis.*/
```

NULL bij ANY en ALL

- Vergelijken met NULL : altijd false
- ALL Subquery geen resulaat : waar
- ANY Subquery geen resultaat : onwaar

Overlaps

operator OVERLAPS

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

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

Alter

Alter Table

- Wijzigt tabelstructuur

Vb. Alter table spelers

 convert to character set utf8 collate utf8_general_ci

- **ADD** : voegt kolom toe
- **DROP** : verwijdert een kolom
- **ALTER** : verandert de kolomeigenschappen
- **ADD CONSTRAINT** : voegt integriteitsregel toe
- **DROP CONSTRAINT** : verwijdert integriteitsregel

Vb. Alter table wedstrijden

 add constraint FK2 foreign key (spelersnr) references
 spelers (spelersnr)

Alter Database

Vb. create database tennis2
 default character set sjis
 default collate sjis_japanese_ci

- **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

Insert

Vb.

```
INSERT INTO table_x
    SELECT *
    FROM   table_y;
```

Soorten Databanken

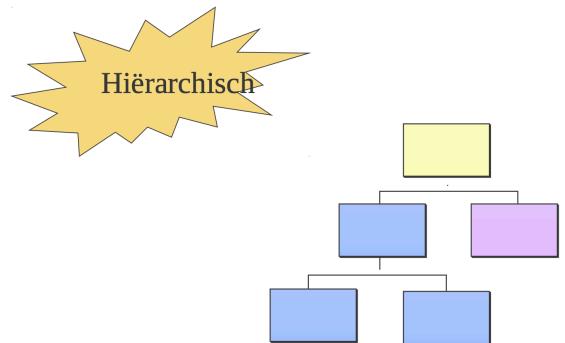
Hiërarchische DB

- ER -> Hierarchisch (1 –n)
- Bouwstenen (segment, parent><child, 1 parent, samengestelde sleutel,...)
- DDL (DBD, Segm , Field)
- DML (PCB , Procopt) Status code
- Functiecodes : GU, GN (Hold, within Parent)

DDL = data definition language

DML = data manipulation language

ER = Entity Relationship



- Child moet parent hebben + slechts 1 parent
- Parent weg => alle children weg
- Beperkingen :
 - 15 niveaus diep
 - 255 verschillende segmenttypes
 - slechts 1 wortel

Functiecode

- GU : overloopt boom en stopt bij eerst gevonden segment
- GN : zoekt verder en stopt bij eerst gevonden segment
- GNP : zoekt verder maar alleen in de afhankelijke segmenten van een parent
- GHU - GHN - GHNP : analoog maar H moet voor een delete of een replace

DML - functiecode

- ISRT : toevoegen van een segment
- DLET : segment en alle afhankelijke worden weggelaten
- REPL : aanpassen van een segment

Voorbeeld

```
GU    REIS(REISOMS = «LUXE CARAIBEN»)
GNP  TOERIST(GEMEENTE = «LEUVEN»)
IF STATUS_CODE = SPACE
    DISPLAY I_O_TOERIST
    GNP TOERIST(GEMEENTE = «LEUVEN»)
    PERFORM UNTIL STATUS_CODE NOT = SPACE
        DISPLAY I_O_TOERIST
        GNP TOERIST(GEMEENTE = «LEUVEN»)
    END-PERFORM
END-IF
```

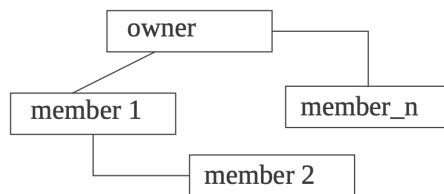
Dit is echt een rotzooi man

Netwerk DB

- ER -> Netwerk model (1-1, 1-n, n-m(toevoeging))
- Bouwstenen (record, owner><member, verschillende parents, record key,...)
- DDL (Schema, intern schema, subschema) Record, SET
- DML (SET) DB status, Currency indidctor (CRU, CRP, CSP)
- Functiecodes : Find, get (DML instructies)

Bouwstenen

- Recordtypes => recordinstantiatie
- Settypes => setinstantiatie
- 1:n verband
 - Owner-recordtype
 - Member-recordtype



ER-model naar netwerkmodel

- n-op-m relatie naar 1-op-n relatie
- Omzettingsregels :
 - ✓ Entiteittype wordt recordtype
 - ✓ Binair 1:1 verband kan settype worden
 - ✓ Binair 1:n verband wordt settype
 - ✓ Binair n:m verband : nieuw recordtype creëren
 - ✓ Unair verband : nieuw recordtype

Verschil hiërarchisch - netwerk

- Bij netwerkgegevensbanksystemen :
 - kan een recordtype member zijn in meerdere settypes
 - kunnen meerdere settypes bestaan tussen dezelfde recordtypes
 - kunnen members bestaan zonder owners

DML instructies

- FIND- en GET-instructie
- Wijzigen van velden
- Weglaten van een recordinstantiatie
- Toevoegen van records
- RECONNECT-instructie
- Associëren van zwevende records
- Loskoppelen van members
- RETAINING-optie

(dit is niet volledig zoek het zelf maar op google ofzo)

Cast

Soms wil je een bepaald type omzetten naar een ander type? Meestal wil je dat zelf niet, maar wilt mr. Bertels dat. Hoe moet dat dan? Waarom? Wanneer? Waarom leven we? Heb ik een existentiële crisis? Het antwoord op al deze vragen (of toch sommige) vind je hier.

Dit is de cast-functie:

```
1 | CAST ( expression AS target_type );
```

Je kan ook zo casten:

```
1 | expression::type
```

Ook heel nuttig als je datums wilt omzetten:

```
1 | SELECT
2 |   CAST ('2015-01-01' AS DATE),
3 |   CAST ('01-OCT-2015' AS DATE);
```

Ik ga hier gewoon wat voorbeelden droppen dan snap je het wel.

```
1 | SELECT
2 |   CAST ('10.2' AS DOUBLE PRECISION);
```

```
SELECT
CAST('1' as BOOLEAN),
CAST('0' as BOOLEAN);
CAST('true' AS BOOLEAN),
CAST('false' as BOOLEAN),
CAST('T' as BOOLEAN),
CAST('F' as BOOLEAN);
```

```
SELECT '2019-07-13 1:20:50'::TIMESTAMP::DATE;
```

Zoek het maar uit, het is niet echt moeilijk.

Nog een leuk voorbeeldje

```
select s.spelersnr, s.naam, s.voorletters,  
(s.jaartoe - (select avg(sub.jaartoe) from spelers sub where sub.plaats = s.plaats))::numeric(5,3)  
from spelers s  
order by 1,2,3,4
```

Syntax “::numeric(precision, scale)

Precision = aantal beduidende cijfers

Scale = aantal na de komma

```
r.vertrekdatum::char(10)
```

Pattern matching

Like

Algemeen:

```
string LIKE pattern [ESCAPE escape-character]  
string NOT LIKE pattern [ESCAPE escape-character]
```

Voorbeeld:

```
'abc' LIKE 'abc'      true  
'abc' LIKE 'a%'      true  
'abc' LIKE '_b_'     true  
'abc' LIKE 'c'        false
```

If **pattern** does not contain percent signs or underscores, then the pattern only represents the string itself; in that case **LIKE** acts like the equals operator. An underscore (**_**) in **pattern** stands for (matches) any single character; a percent sign (**%**) matches any sequence of zero or more characters.

Similar to

Algemeen:

```
string SIMILAR TO pattern [ESCAPE escape-character]  
string NOT SIMILAR TO pattern [ESCAPE escape-character]
```

Voorbeeld:

```
'abc' SIMILAR TO 'abc'      true  
'abc' SIMILAR TO 'a'        false  
'abc' SIMILAR TO '%(b|d)%' true  
'abc' SIMILAR TO '(b|c)%'  false
```

Wat betekenen die tekens nu allemaal?

| Staat voor afwisseling (een van de twee alternatieven).

* staat voor een herhaling van het vorige item, nul of meer keren.

+ staat voor herhaling van het vorige item één of meer keren.

? staat voor de herhaling van het vorige item nul of één keer.

{m} geeft de herhaling van het vorige item precies m keer aan.

{m,} staat voor herhaling van het vorige item m of meer keren.

{m,n} staat voor de herhaling van het vorige item, ten minste m en niet meer dan n keer.

Haakjes () kunnen gebruikt worden om items te groeperen in een enkel logisch item.

Brackets [...] specifieren een karakterklasse, net zoals in POSIX reguliere uitdrukkingen.

“_” als wildcard voor 1 karakter

“%” als wildcard voor 0...* karakters