

# FDW

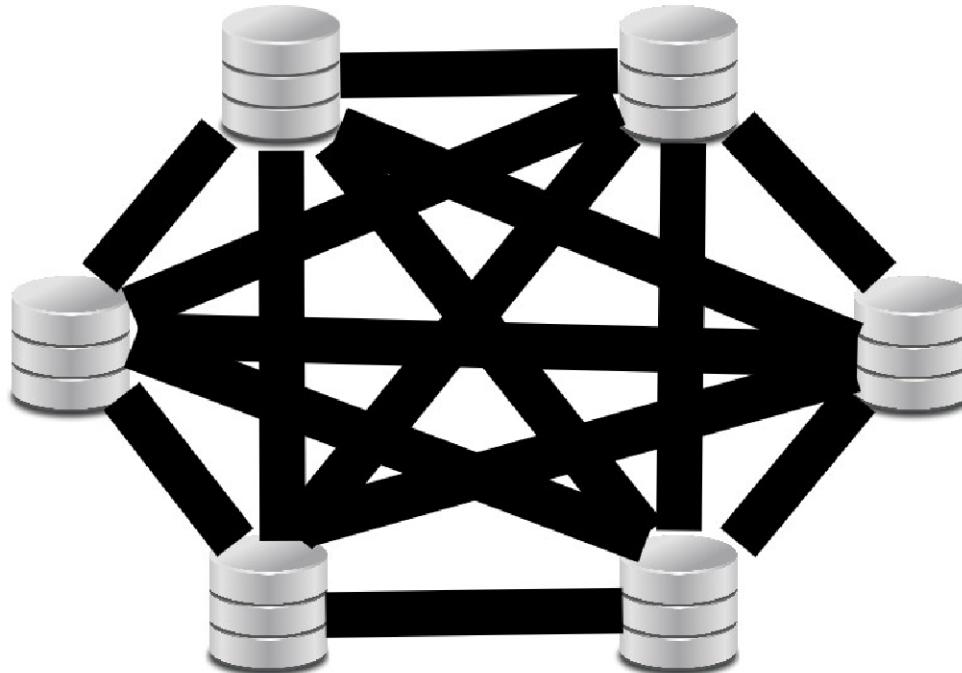
[wim.bertels@ucll.be](mailto:wim.bertels@ucll.be)

Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Unported  
Licentie

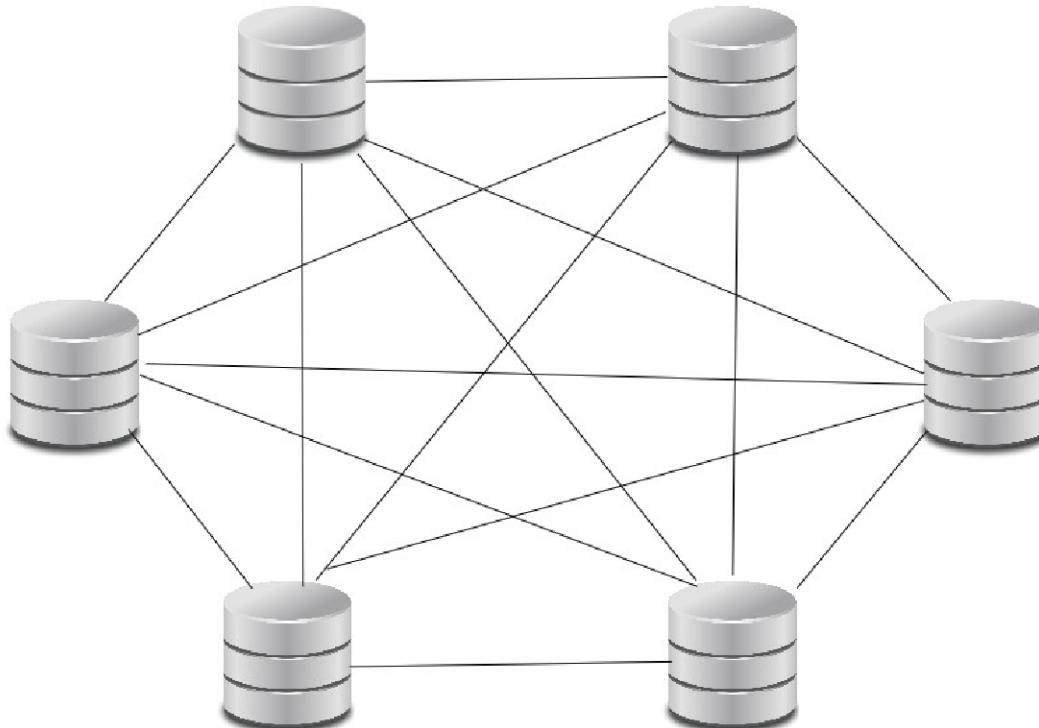
# Wat

- Foreign Data Wrapper (FDW)
- Onderdeel van SQL/MED (ISO)
  - Foreign Table
  - Datalink
    - <https://github.com/lacanoid/datalink>

# Opslag vs Netwerk



# Opslag vs Netwerk



# Bronnen

Local server



Remote servers



<https://www.interdb.jp/pg/pgsql04.html>

# Bronnen

- [https://wiki.postgresql.org/wiki/Foreign\\_data\\_wrappers](https://wiki.postgresql.org/wiki/Foreign_data_wrappers)
  - Databanken
  - Bestanden
  - GIS
  - LDAP
  - Web
  - BS
  - ..

# Basis

- Waar en hoe: server
  - **CREATE SERVER**
- Wie: beveiliging, gebruiker afbeelding
  - **CREATE USER MAPPING**
- Wat: welke tabellen
  - **CREATE FOREIGN TABLE**
- Nodig voor gebruik: bibliotheek
  - **CREATE EXTENSION**

# CREATE EXTENSION

- CREATE EXTENSION postgres\_fdw;
- CREATE EXTENSION file\_fdw;
- <https://gdal.org/>
  - <https://github.com/pramsey/pgsql-ogr-fdw>

# Referenties

- Universal Data Access with SQL/MED, David Fetter
- <https://wiki.postgresql.org/wiki/SQL/MED>

# Inleiding procedurele SQL



[Wim.bertels@ucll.be](mailto:Wim.bertels@ucll.be)

# ISO Standaard

- Sinds 1996, 1999, ..., 2023
- SQL/PSM (Persistent Stored Modules)
- Veel producten richten zich op de standaard, maar geen enkele is volgt deze helemaal.
  - Bekijk de documentatie van het concrete product.

# Postgresql

- Ondersteuning voor verschillende andere programmeertalen.
  - SQL
  - PL/pgSQL (standaard)
  - PL/Tcl
  - PL/Perl
  - PL/Python
  - PL/Java (uitbreiding)

# “Vetrouwde vs. Niet-Vetrouwde”

- Trusted (vertrouwd): deze talen zijn beperkt tot het wat is toegelaten door de databank (DBMS)
- Untrusted (niet-vertrouwd): deze talen zijn daartoe niet beperkt. Superuser toegang nodig om deze te gebruiken

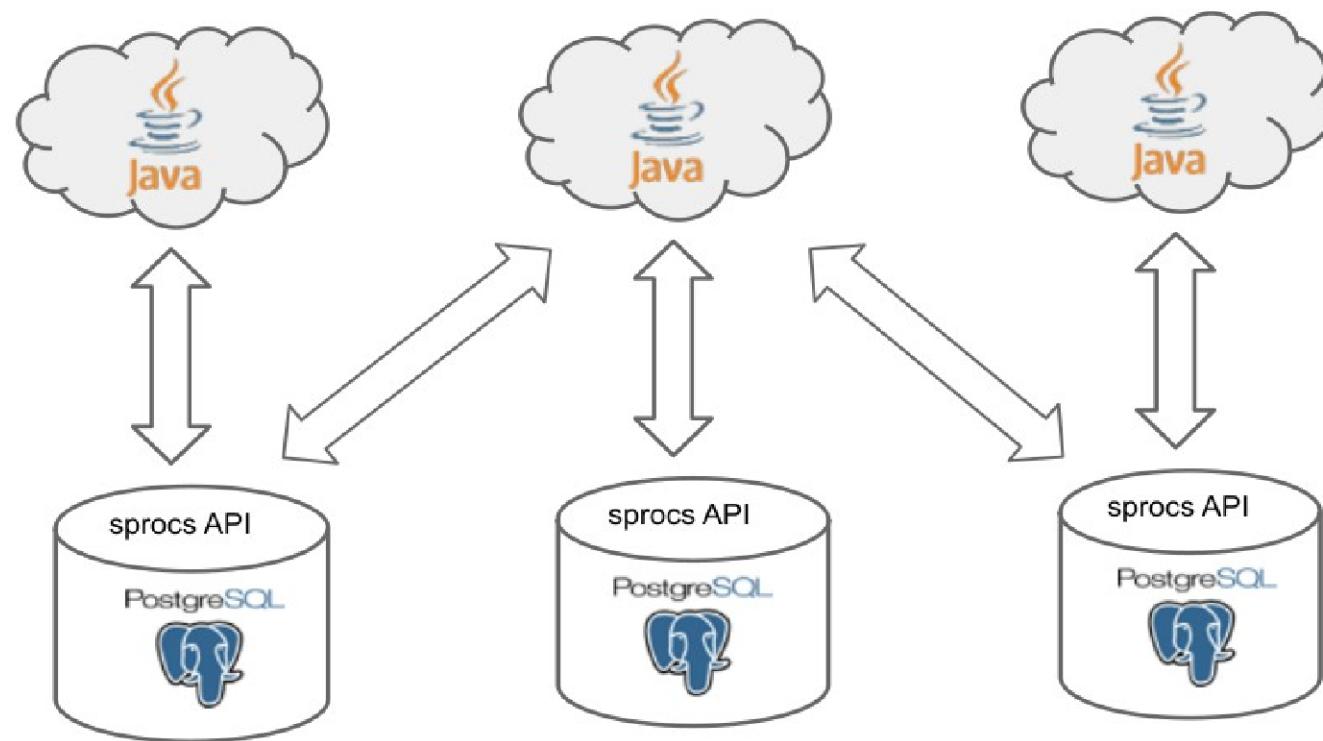
# Extra beschikbare modules

Postgresql komt met sommige extra modules zoals bijvoorbeeld:

- Debugger (bv pldebugger voor pgadmin)
- Profiler (bv plprofiler, of algemener pg\_stat\_statements)

# How Trustly and Zalando are using PostgreSQL

Applications access data in PostgreSQL databases via calls to stored procedures.



(slide copied from the excellent 2014.pgconf.eu presentation by Alexey Klyukin @ Zalando)

# Referenties

- <https://www.postgresql.org/docs/current/static/server-programming.html>
- <https://www.postgresql.org/docs/current/sql-createlanguage.html>
- <https://www.postgresql.org/docs/current/contrib.html>
- <https://en.wikipedia.org/wiki/SQL/PSM>
- "How we use Postgresql at Trustly, 'Joel Jacobson', October 2014, Madrid

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License

# Procedurele SQL met (plpgsql)



Wim.bertels@ucll.be

# Objecten op de server

- Stored procedures
- Stored functions
- Triggers
- Def. : hoeveelheid code die opgeslagen is in de catalogus van een DB en die geactiveerd kan worden
- Vergelijkbaar met wat je in programmeren een (statische) methode zou noemen.

# Voorbeeld: stored function

```
CREATE OR REPLACE FUNCTION increment(i INT)
RETURNS INT
AS
$$
BEGIN
RETURN i + 1;
END;
$$ LANGUAGE 'plpgsql';
```

-- Een voorbeeld van hoe de functie op te roepen:

```
SELECT increment(10);
```

# Verwerking

Verwerking :

- Vanuit programma wordt procedure/functie opgeroepen
- DBMS ontvangt oproep en zoekt procedure
- Procedure wordt uitgevoerd waarbij de instructies op de database verwerkt worden
- M.b.v. een code wordt aangegeven of procedure correct verwerkt is (sqlcode)

# Parameters Algemeen

- Communicatie met buitenwereld
- 3 soorten (signatuur methode) :
  - Invoerparameters
  - Uitvoerparameters
  - Invoer/uitvoerparameters
- Parameter best andere naam dan kolom van tabel !
- Return (kan enkel bij functions)

# Voorbeeld IN OUT

```
CREATE FUNCTION dup(in int, out f1 int, out f2 text)
```

```
AS
```

```
$$
```

```
    SELECT $1, CAST($1 AS text) || ' is text'
```

```
$$
```

```
LANGUAGE SQL;
```

```
SELECT * FROM dup(42);
```

# Signatuur: Functions vs Procedures

- Algemeen:
  - Het grote verschil is dat functies altijd iets teruggeven
  - Terwijl procedures nooit een return hebben
  - In de praktijk afhankelijk van het concrete DBMS dat je gebruikt

# Waarom?

- Historisch verschil tussen berekeningen (functies) en manipulaties (procedures)
- Daarnaast:
  - Transacties kunnen enkel binnen procedures
  - Zo kan de planner hiermee rekening houden

# Instructie mogelijkheden (de essentie)

- Declaratie
- Sequentie
- Selectie
  - IF .. THEN .. (ELSE ..) END IF
  - CASE .. WHEN .. THEN ... END CASE
- Iteratie
  - FOREACH .. LOOP .. END LOOP
  - FOR .. IN .. LOOP .. END LOOP

# Structuur: functies

```
CREATE FUNCTION function_name(arg1,arg2,...)
```

```
    RETURNS type
```

```
    AS
```

```
'
```

```
    BEGIN
```

```
        -- logic
```

```
    END;
```

```
'
```

```
    LANGUAGE language_name;
```

# Structuur: functies

- Specifieer naam van functie
- Lijst van parameters achter naam van functie
- Definieer return type
- Gevolgd door code binnen begin- en end block
- Procedurele taal meegeven

# Verwijderen stored functions

- **DROP FUNCTION:** verwijdert functie
- Vb.
  - `DROP FUNCTION function_name;`
  - `DROP FUNCTION function_name(signatuur);`

# SELECT INTO voorbeeld

- Enkel indien een rij als uitvoer! Vb.

```
create function som_boetes_speler(p_spelersnr integer)
    returns decimal(8,2)
AS
$eenderwat$
    declare    som_boetes decimal(8,2);
begin
    select sum(bedrag)
        into som_boetes
     from boetes
    where spelersnr = p_spelersnr;
    return som_boetes ;
end;
$eenderwat$
language plpgsql;

select som_boetes_speler (27);
```

# \$\$ delimiter dialect

```
create function som_boetes_speler(p_spelersnr integer)
    returns decimal(8,2)
    AS
```

**\$eenderwat\$**

```
declare som_boetes decimal(8,2);
begin
    select sum(bedrag)
    into som_boetes
    from boetes
    where spelersnr = p_spelersnr
    return som_boetes;
end;
```

**\$eenderwat\$**

```
language plpgsql;
```

```
select som_boetes_speler (27);
```

# PERFORM

- Resultaten van een sql statement moeten opgevangen worden, bv via INTO variabele.
- PERFORM alternatief voor SELECT waarbij het resultaat niet wordt opgevangen
  - Bv SELECT now(); zal een fout geven in een code blok
  - PERFORM now(); niet
- Vergelijkbaar met het void maken van een functie in andere programmeer talen

# FOUND globale variabele

- Boolean
- Bijvoorbeeld:
  - PERFORM spelersnr FROM spelers ;
  - IF FOUND THEN .. END IF ;

# RETURN(S)

- Signatuur :
  - RETURNS type
  - (RETURNS setof type)
  - RETURN TABLE (column\_name type,...)
- Code :
  - RETURN scalair..
  - RETURN QUERY ..
  - RETURN NEXT .. + RETURN

# Foutberichten

- Foutberichten :
- SQL-error-code : beschrijvende tekst
- SQLSTATE: code (getal)

```
BEGIN
    -- code
    RAISE DEBUG 'A debug message % ', variable_that_will_replace_percent;
EXCEPTION
    -- welke fout, bv
    WHEN uniqueViolation THEN
        -- code
    WHEN division_by_zero THEN
        RAISE .. -- eventueel omzetten naar uniqueViolation;
    WHEN others THEN
        -- ?
        NULL;
END
```

# RAISE

- RAISE;
- RAISE division\_by\_zero;
- RAISE SQLSTATE '22012';
- RAISE DEBUG/INFO/..  
    -- SET client\_min\_messages TO debug;
- RAISE .. USING
  - ERRCODE = 'uniqueViolation',
  - HINT = 'suggestie voor de reden voor de gebruiker',
  - DETAIL = 'meer detail fout',
  - MESSAGE = 'gedrag van de uniqueViolation';

# ASSERT

- ASSERT condition [, message];

```
do
$$
declare
    film_count integer;
begin
    select count(*)
    into film_count
    from film;
    assert film_count > 0, 'No films found, check the film table';
end
$$;
```

-- do is dialect, <https://www.postgresql.org/docs/current/sql-do.html>

# SECURITY

- GRANT EXECUTE ON haha() TO jomeke ;
- INVOKER : standaard, veilig
- DEFINER : via de rechten van de eigenaar

```
CREATE OR REPLACE FUNCTION haha()
```

```
    RETURNS text
```

```
    AS
```

```
$code$
```

```
    BEGIN
```

```
        DROP TABLE ola();
```

```
        RETURN 'pola';
```

```
    END
```

```
$code$
```

```
LANGUAGE plpgsql
```

```
EXTERNAL SECURITY DEFINER;
```

# LEVEL of immutability

CREATE FUNCTION add(integer, integer) RETURNS integer

AS 'select \$1 + \$2;'

LANGUAGE SQL

**IMMUTABLE**

RETURNS NULL ON NULL INPUT;

- **IMMUTABLE** (alleen afhankelijk van de signatuur)
- **STABLE** (geen aanpassingen)
- **VOLATILE** (standaard)

# Structuur: stored procedure

CREATE OR REPLACE PROCEDURE

<procedure\_name>( <arguments> )

AS <block-of-code>

LANGUAGE <implementation-language>;

# Transactie voorbeeld: stored procedure

CREATE OR REPLACE PROCEDURE voorbeeld(invoer text )

AS

\$code\$

BEGIN

...

**IF** invoer = 'niet doen' **THEN** RAISE WARNING 'Abort, the ship is sinking';

**ROLLBACK;**

**ELSEIF** invoer = 'doen' **THEN** RAISE INFO 'Gaon met die banaan';

**COMMIT;**

**END IF;**

...

**END**

\$code\$

LANGUAGE plpgsql;

CALL voorbeeld('doen');

-- <https://www.postgresql.org/docs/16/plpgsql-transactions.html>

# Praktisch: script

BEGIN;

code en testen

ROLLBACK;

DROP [procedure|function|trigger] naam

CREATE [procedure|function|trigger] naam

ALTER [procedure|function|trigger] naam

CREATE OR REPLACE [procedure|function|trigger] naam

# Triggers

- Def. :  
hoeveelheid code die opgeslagen is in de catalogus en die geactiveerd wordt door het dbms indien een bepaalde operatie wordt uitgevoerd en een conditie waar is.
- Triggers worden door het dbms zelf automatisch opgeroepen (niet door vb. een call)

# PostgreSQL

- Triggers roepen  
trigger functies op
- **CREATE FUNCTION trigger\_functie...**  
**RETURNS TRIGGER**  
..
- **CREATE TRIGGER trigger\_trg ..**  
..  
**EXECUTE PROCEDURE trigger\_functie..**

# Voorbeeld: tabel

```
create table mutaties (
```

```
    gebruiker          varchar(30)  not null,
```

```
    mut_tijdstip       timestamp   not null,
```

```
    mut_spelersnr     smallint    not null,
```

```
    mut_type          char(1)     not null,
```

```
    mut_spelersnr_new smallint    ,
```

```
    primary key
```

```
        (gebruiker, mut_tijdstip, mut_spelersnr, mut_type)) ;
```

```
-- tabel om wijzigingen bij te houden
```

# Voorbeeld: trigger functie

```
create or replace function insert_speler() returns trigger as  
$body$
```

```
begin
```

```
    insert into mutaties values
```

```
        (user, current_date(), new.spelersnr, 'I', null) ;
```

```
end;
```

```
$body$
```

```
language sql;
```

# Voorbeeld: trigger

```
create or replace trigger insert_speler.trg
```

```
after insert
```

```
on spelers
```

```
-- when new.spelersnr < 10
```

```
for each row
```

```
execute procedure insert_speler();
```

-- OLD en NEW verwijzen naar de huidige toestand en de nieuwe toestand

-- welke verschillende onderdelen zie je hier die typisch voor een

-- trigger zijn ?

# Onderdelen Trigger

- Trigger-moment + Trigger-gebeurtenis:
  - Wanneer activeren?
    - AFTER : nadat triggering instructie is verwerkt
    - BEFORE : eerst de trigger-actie
    - INSTEAD OF : alleen de trigger-actie
  - Voor welke rij activeren ?
    - FOR EACH ROW : voor elke rij
    - FOR EACH STATEMENT : voor een statement
  - Voor welke gebeurtenis?
    - INSERT, UPDATE, DELETE, (TRUNCATE)
- Trigger-Conditie: WHEN
- Trigger-actie: wat doet de trigger?

# Syntax

```
CREATE [ OR REPLACE ] [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
ON table_name
[ FROM referenced_table_name ]
[ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]
[ REFERENCING { { OLD | NEW } TABLE [ AS ] transition_relation_name } [ ... ] ]
[ FOR [ EACH ] { ROW | STATEMENT } ]
[ WHEN ( condition ) ]
EXECUTE { FUNCTION | PROCEDURE } function_name ( arguments )
```

where event can be one of:

INSERT

UPDATE [ OF column\_name [, ... ] ]

DELETE

TRUNCATE

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

# Syntax

CREATE/ALTER/DROP TRIGGER

ALTER TABLE .. ENABLE/DISABLE TRIGGER ..

GRANT/REVOKE TRIGGER ON TABLE .. TO ..

( CREATE EVENT TRIGGER – DDL)

# Standaard

- De standaard laat meer acties dan een trigger functie toe (udf : user defined function)
- Bv
  - create trigger delete\_spelers  
after delete on spelers for each row  
begin  
    delete from spelers\_wed  
    where spelersnr = old.spelersnr;  
end ;  
-- geen verwijzing naar udf, maar rechtstreeks de sql code

# Notas

- Er zitten verschillen tussen producten :
  - Meerdere triggers op 1 tabel ? En wat met de volgorde ?
  - Kan een trigger een andere trigger activeren ? (waterval/domino !)
  - Wat mag een trigger allemaal doen ?
  - Hoe worden (complexere) triggers juist verwerkt ?
  - ..

# Gebruik?

- Denk gebeurtenis gestuurd
- Voorbeelden :
  - (Integriteits)regels
  - Audit
  - Consistentie
  - Beveiliging
  - Afgeleide waarden berekenen
  - (Data)validatie
  - ..

# Voordelen

- **Onderhoudbaarheid:**  
Vb. Snellere uitvoering door meer instructies in macro
- **Verwerkingsnelheid**  
Vb. minimaliseert netwerkverkeer
- « Precompilatie » bij stored procedures/functions/triggers
  - Planning en caching
  - Werkt in verschillende host-languages

# Nadelen

- Nadelenken over architectuur en code organisatie
- Algemeen zoals bij andere talen : logische fouten vs syntaxfouten
  - Bv Onverwachte neveneffecten bij gebruik van (veel) (overlappende) triggers

# Referenties

Slides: Stored Procedures Functions Triggers, P. Demazière, 2018

Slides: Procedurel SQL, H.Martens, W.Bertels,2014

Postgresql 11 Server Side Programming Quick Start Guide, L. Ferrari, 2018

<https://www.postgresqltutorial.com/postgresql-plpgsql>

<https://www.postgresql.org/docs/current/plpgsql-trigger.html>

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

<https://www.postgresql.org/docs/current/triggers.html>

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

<https://www.postgresql.org/docs/current/plpgsql.html>

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

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License

# Beveiliging



[Wim.bertels@ucll.be](mailto:Wim.bertels@ucll.be)

# Security

- Hardware
- Het platform waar het op draait
- De databank software
- Binnen sql zelf (grant, revoke, view,...)
- Sql als “vertaler”
- Applicaties
- Gebruikers
- Algemeen terugkerende security problematiek (bv CIA, AAA, maar ook reserve kopies, ..)
- ..

# Ondersteunend platform

- Bijdetijs (Up to date) houden
- Beveiligen (infrastructuur vakken)
- (D)DoS-attack's
- ...

# Databank software

- Up to date houden
- Configuratie
  - Local
  - Remote
    - Eg postgresql: pg\_hba.conf, postmaster.conf en postgresql.conf
- Known exploits -> Wat doe je?

# Binnen SQL zelf

- User management (roles)
- Privileges (grant / revoke)
- Views
- Stored procedures
- Row Security Policies

# SQL

- SQL wordt niet gecompileerd
- SQL wordt “vertaald” in de onderliggende/omsluitende laag
- Dit kan gebruikt worden om sql ongewenste instructies te laten uitvoeren
- Het is probleem dat bij elke taal die “vertaald”, terugkomt (eg php)

# SQL Injection

- Indien we de onderliggende sql code van bv een formulier niet kennen, dan gaan we proberen te raden wat de sql code is die erachter zit.
- Ipv gewone waarden gaan sql code meegeven met het formulier.
- Pas op serial, identity, autoincrement..?

# Incorrectly Filtered Escape Characters

- Fout: input is niet gefilterd op escape characters
- Bv
  - statement := "SELECT \* FROM users WHERE naam = " + userNaam + ";"
  - UserNaam:= a' or 't='t
  - Gevolg: SELECT \* FROM users WHERE naam = 'a' or 't='t';
  - Dus.. , andere voorbeelden?

# Incorrect Type Handling

- Fout: types van de input worden niet gecheckt
- Bv
  - statement := "SELECT \* FROM data WHERE id = "  
+ a + ";" (a wordt enkel verwacht als int)
  - Voor a := 1;DROP TABLE users;
  - Gevolg: SELECT \* FROM data WHERE id = 1;DROP TABLE users;
  - Dus..

# Oplossingen

- Escape character verwijderen uit de invoervelden
- Invoer validatie, controleren of het invoerveld bv wel het juiste type heeft
- Prepared statements
- Stored procedures
  - Type
  - ; en escaping
  - Grants..
  - Dicht bij de bron
  - ..

# Opgepast

- Enkel Escaping is niet voldoende:
- Bv
  - `SELECT * from items where userid=$userid;`
  - `$userid := "33 or userid is not null or userid=44";`
  - `SELECT * from items where userid=33 or userid is not null or userid=44;`
- Dus zeg eerder wat wel mag zijn als invoer, ipv wat niet mag; het eerste is eenvoudiger, er zijn (meestal) maar een eindig aantal mogelijkheden.

# Handige Functies

- `quote_ident ( text ) → text`
- `quote_literal ( anyelement ) → text`
  - `quote_nullable ( anyelement ) → text`

# Dieper

- <https://www.postgresql.org/docs/current/ddl-rowsecurity.html>

```
CREATE TABLE users (
```

```
    manager text,
```

```
    company text );
```

```
ALTER TABLE users ENABLE ROW LEVEL SECURITY;
```

```
CREATE POLICY users_manager
```

```
    ON users TO managers -- groep rol managers
```

```
    USING (manager = current_user);
```

- <https://www.postgresql.org/docs/current/sepgsql.html>
- <https://www.postgresql.org/docs/current/sql-security-label.html>

# Links

- <http://www.w3schools.com/>
- <http://www.postgresql.org/>
- <http://en.wikipedia.org/>

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License

# Replicatie

(CC BY-NC-SA 4.0)

[Wim.bertels@ucll.be](mailto:Wim.bertels@ucll.be)

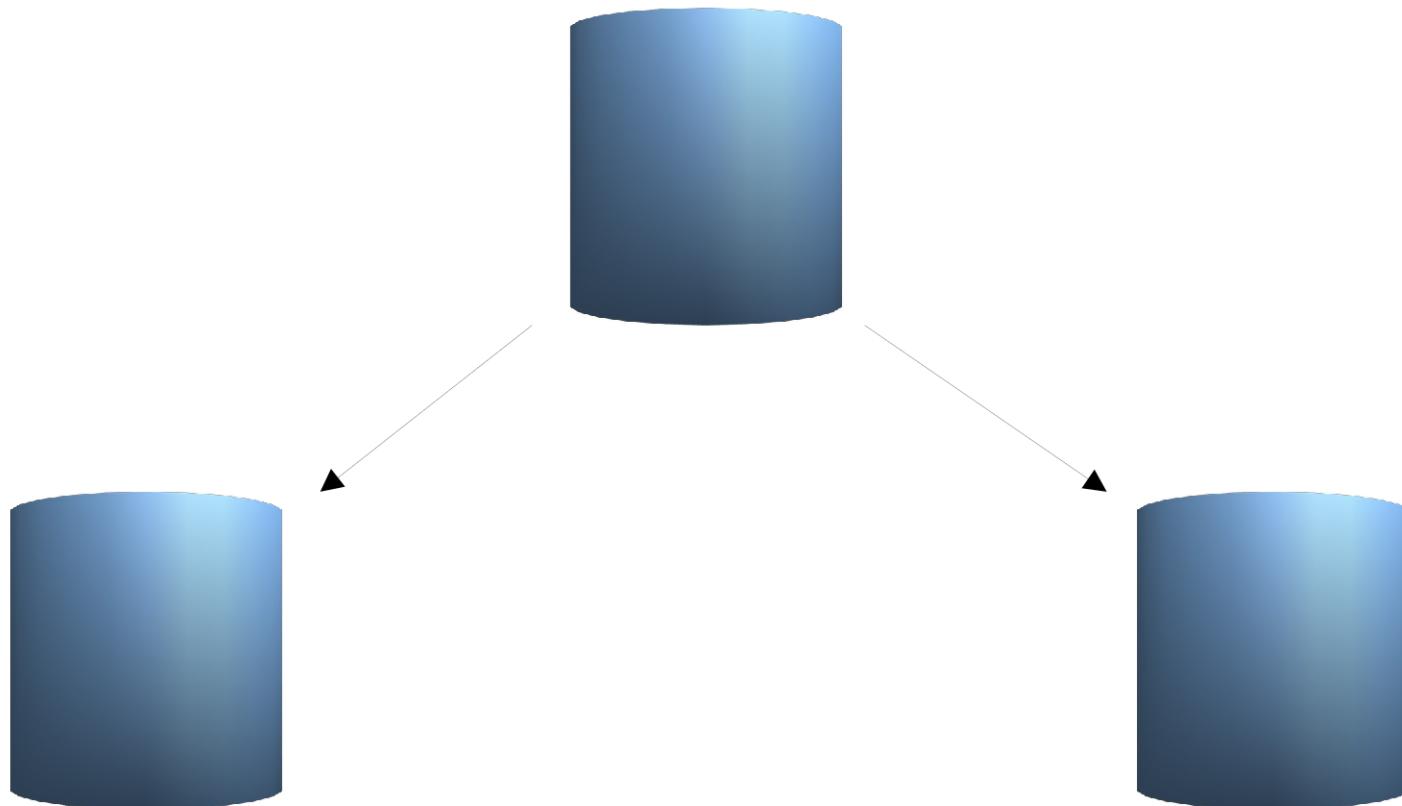
# Replicatie

- Verband CAP theorema
  - Hoge beschikbaarheid (naast data partitionering, parallele query verdeling over meerdere servers, ..)
- Fysische
- Logische
- Andere (bv triggergebaseerd, externe applicatie ea)

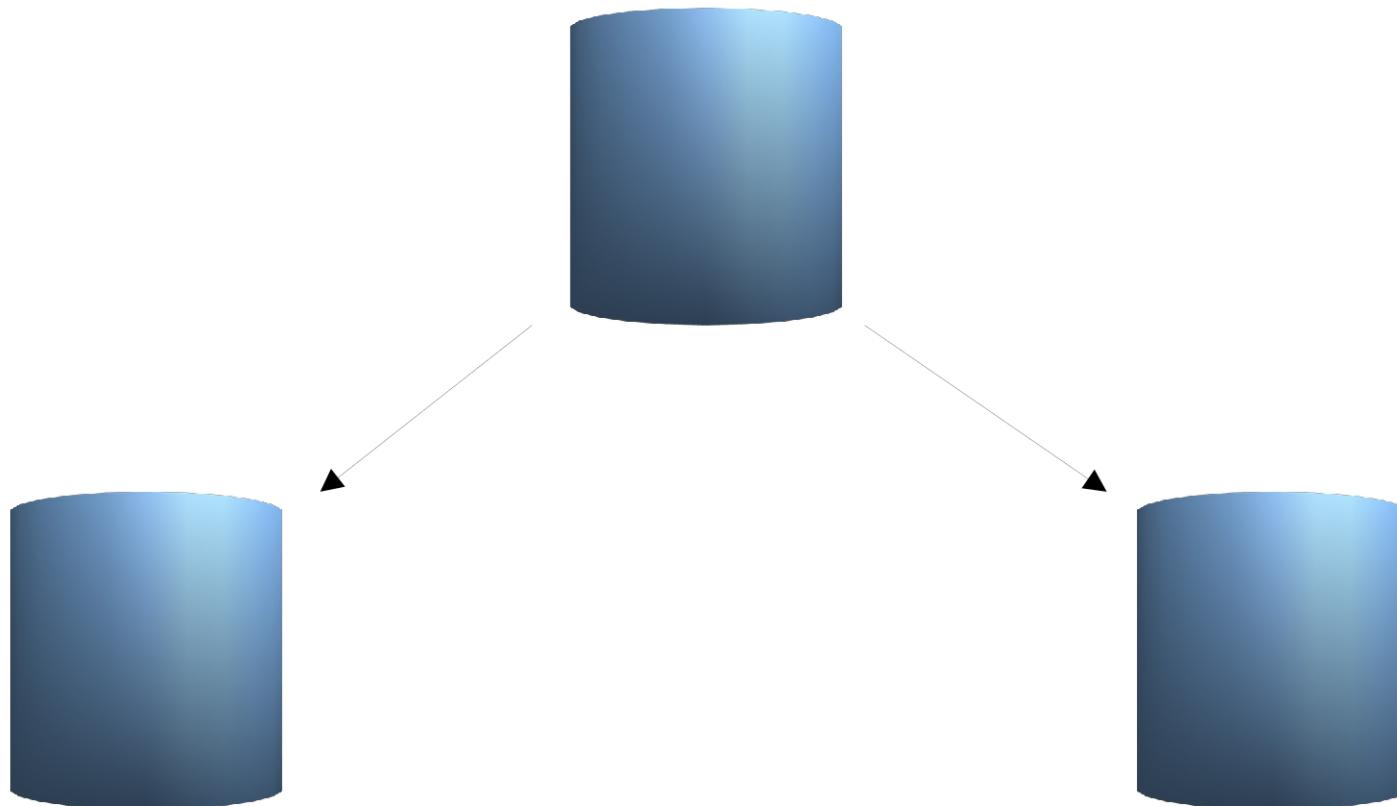
# CAP

- Consistentie:  
alle knooppunten in het systeem zien dezelfde data op hetzelfde moment
- Availability (beschikbaarheid):  
elke aanvraag krijgt een antwoord terug.
- Partitie tolerant:  
als een knooppunt uitvalt, dan blijft het systeem functioneren
  - > 2 van de 3
  - \* uitbreiding: pacelc (latency vs consistency)
  - \* sync/async

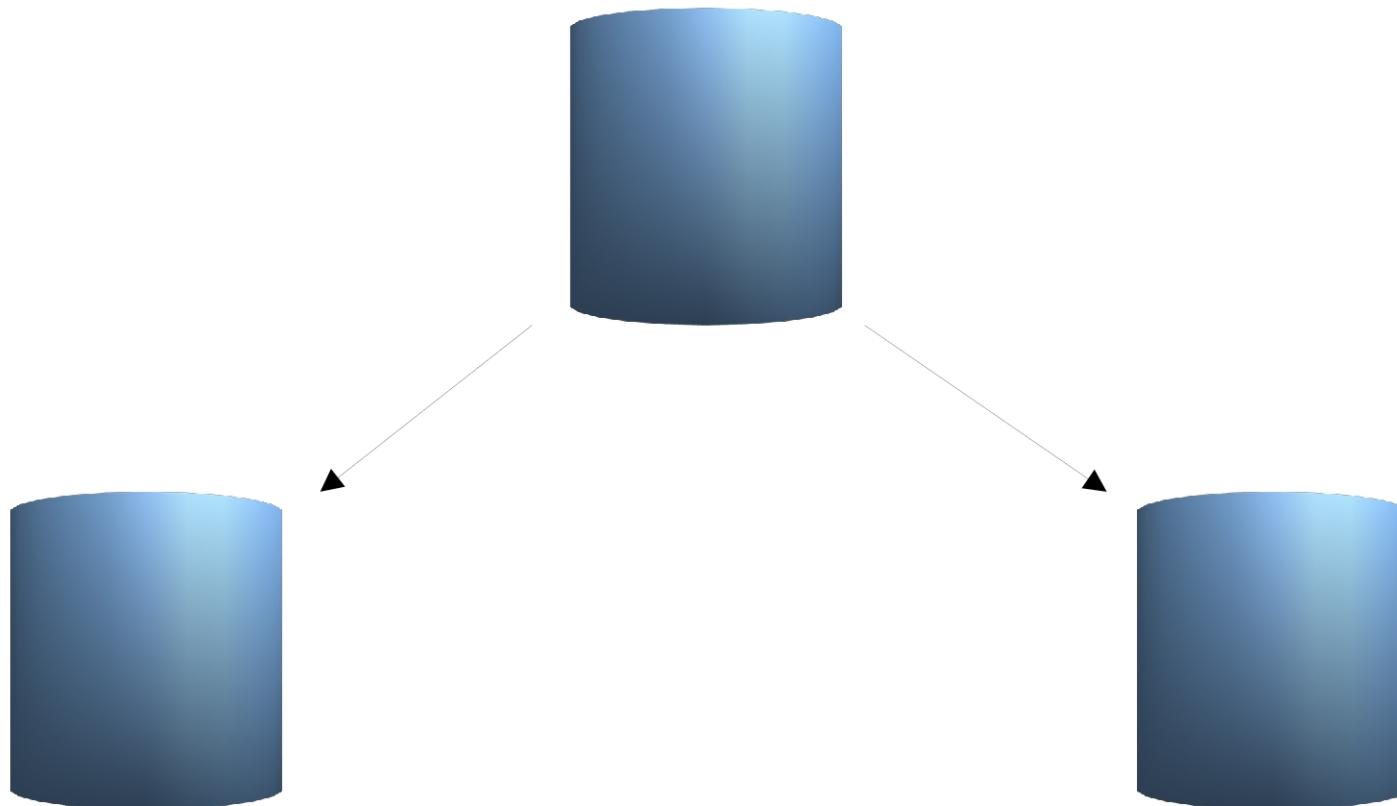
# CAP Voorbeeld 1



# CAP Voorbeeld 3



# CAP Voorbeeld 2



# Andere: Niet ingebouwd

- <https://www.symmetricds.org/>

HOME    ABOUT    DOWNLOAD    DOCUMENTATION    DEVELOPER    GET HELP

## Fast & Flexible Database Replication

*SymmetricDS is open source database replication software that focuses on features and cross platform compatibility.*



### CROSS PLATFORM

Replicate data across different platforms, with compatibility for many databases. Sync from any database to any database in a heterogeneous environment.

[READ MORE +](#)



### SCALE OUT PERFORMANCE

Optimized for performance and scalability, replicate thousands of databases asynchronously in near real time, and span replication across multiple tiers.

[READ MORE +](#)



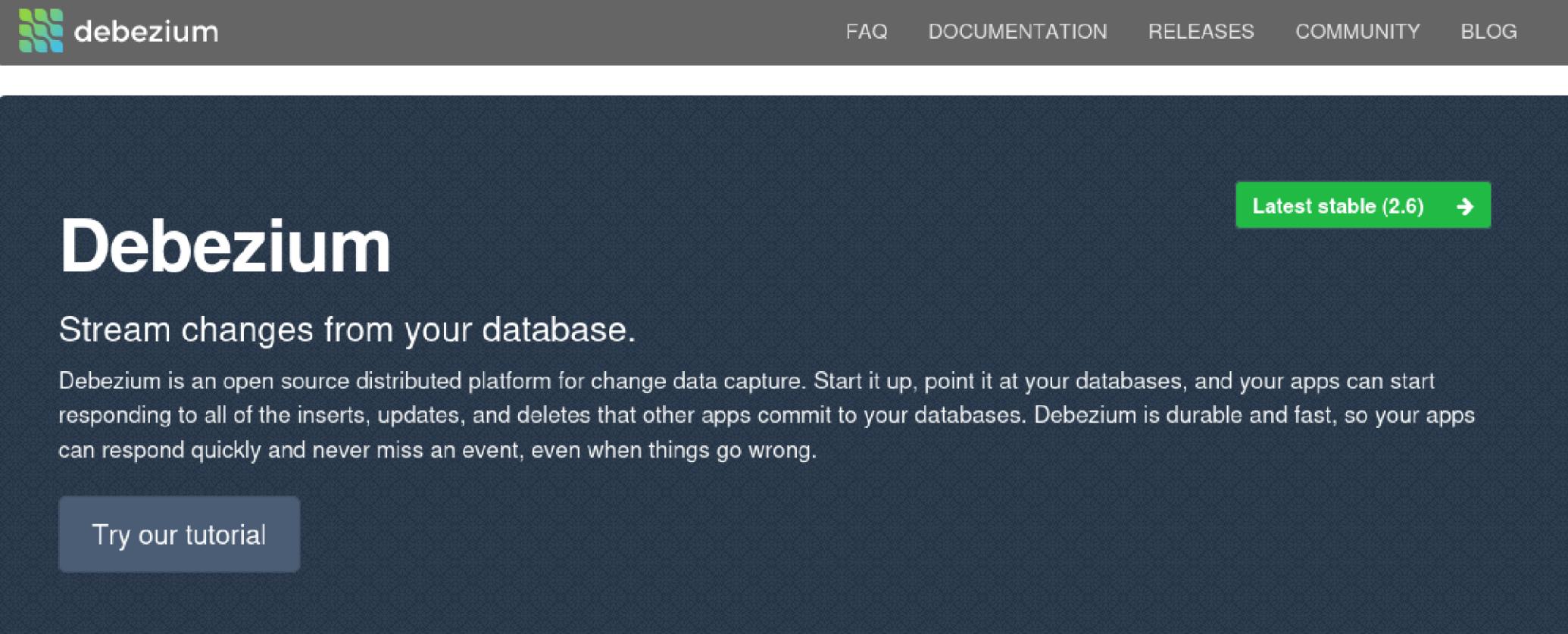
### FLEXIBLE CONFIGURATION

Configure which tables and columns to sync, and in which direction. Subset rows and distribute them across databases. Combine, filter, and transform data.

[READ MORE +](#)

# Andere: Niet ingebouwd

- <https://debezium.io/>



The screenshot shows the official Debezium website. At the top, there's a dark navigation bar with the Debezium logo on the left and links for FAQ, DOCUMENTATION, RELEASES, COMMUNITY, and BLOG on the right. Below the header, the word "Debezium" is prominently displayed in large white letters. To its right is a green button labeled "Latest stable (2.6)" with a white arrow pointing right. Underneath the title, the tagline "Stream changes from your database." is written in white. A detailed description follows: "Debezium is an open source distributed platform for change data capture. Start it up, point it at your databases, and your apps can start responding to all of the inserts, updates, and deletes that other apps commit to your databases. Debezium is durable and fast, so your apps can respond quickly and never miss an event, even when things go wrong." At the bottom left, there's a blue button with the text "Try our tutorial".

# Fysisch

- Exacte kopie
- “Transactielog” (xlog > pg:WAL)

# Logische

- Fijner
- “SQL” : logical decoding
- Typische gebruik:
  - Incrementeel veranderingen doorsturen
  - Verdere afhankelijkheden op de subscriber
  - Verzamelen van data van verschillende databanken
  - Replicatie over verschillende besturingssystemen en/of versies van db software
  - Toegangsbeleid (rechten op subscriber naar rollen)
  - Een deel van een db delen met andere dbn

# Overzicht binnen planeet pg

Feature	Shared Disk	File System Repl.	Write-Ahead Log Shipping	Logical Repl.	Trigger-Based Repl.	SQL Repl. Middle-ware	Async. MM Repl.	Sync. MM Repl.
Popular examples	NAS	DRBD	built-in streaming repl.	built-in logical repl., pglogical	Londiste, Slony	pgpool-II	Bucardo	
Comm. method	shared disk	disk blocks	WAL	logical decoding	table rows	SQL	table rows	table rows and row locks
No special hardware required		•	•	•	•	•	•	•
Allows multiple primary servers				•		•	•	•
No overhead on primary	•		•	•		•		
No waiting for multiple servers	•		with sync off	with sync off	•		•	
Primary failure will never lose data	•	•	with sync on	with sync on		•		•
Replicas accept read-only queries			with hot standby	•	•	•	•	•
Per-table granularity				•	•		•	•
No conflict resolution necessary	•	•	•		•	•		•

# Links

- <https://www.postgresql.org/docs/current/high-availability.html>
- <https://www.postgresql.org/docs/current/logical-replication.html>
- [https://en.wikipedia.org/wiki/CAP\\_theorem](https://en.wikipedia.org/wiki/CAP_theorem)

Wim Bertels

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License

# Data Management

(CC BY-NC-SA 4.0)

Wim Bertels  
Serhat Erdogan

# Lesmateriaal

- Cursus (alphacopy):  
[Inleiding tot data management](#)
- Elektronisch materiaal beschikbaar via Toledo
- Optioneel (gedetailleerd stappenplan voor SQL): Het SQL Leerboek, R. Van der Lans

# Evaluatie

- Examen: 20
  - Schriftelijk (gesloten boek): 10
  - PC (met examenbeperkingen) : 10
- Optioneel Opdracht > gebruik papieren cursus bij het PC examen
- Database Foundations: voorkennis die opnieuw (verdiepend) kan gevraagd worden.
  - Veronderstelde voorkennis: Programming 1 en Computer systems

# Extra Aandachtspunten Examen

- Niet kunnen normaliseren, technisch (pk,un,fk) of conceptueel (SDG)
- Fouten tegen de conceptuele verwerking van een select-instructie
- Vergeten te joinen wanneer dit moet
- Niet kunnen werken met transakties
- Data niet kunnen beveiligen (DCL, sec)
- De query cost niet kunnen vergelijken (perf)

Indien u 1 van deze fouten maakt op een examen =  
Onvoldoende

# Praktisch

- Lessen structuur
  - 2\*2u/week (theorie, practica en oefeningen)  
(onder voorbehoud)
    - 2u grote groep
    - 2u kleinere groep
- Software
  - SQL: RDBMS : (<https://www.postgresql.org>)
  - SQL: PGAdmin (<http://www.pgadmin.org/>)
  - SQL: <https://dbeaver.io/>
  - SQL: psql, ..
  - GIT
  - ERM: speeltijd is voorbij
  - Bestandsformaten: open ISO standaarden

# Kort Onvolledig Overzicht

- Verschillende DBMS paradigma's
- Modellering en implementatie: verbreding
- Basisinstallatie
- Verband programmeren
- SQL verbreding en verdieping
- Beveiliging
- Performantie
- ..
- Zie planning ..

# Studie belasting

- 6 ects credits: 150-180 uren
- Contacturen:  $10 \times 4 = 40$  uren
- Examen: 4u + 48u voorbereiding
- Zelf studeren: 62 – 92 uren
  - Bv wekelijks 6 uur

# Vragen

- Die stel je tijdens de les
- Dit wil niet zeggen dat je geen epost mag sturen bij problemen; maar het antwoord op een inhoudelijke vraag krijg je tijdens de les.
- Als de lessen gedaan zijn, wordt er niet meer op (inhoudelijke) vragen geantwoord.

# Een Inleding tot CLA

[Wim.bertels@ucll.be](mailto:Wim.bertels@ucll.be)  
(CC BY-NC-SA 4.0)

# CA

## Contributor Agreement

- CLA
- CAA

# CAA

- Contributor Assignment Agreement

De CAA vereist een toewijzing en daarbij een overdracht van de rechten tot de project eigenaar.

# CLA

- Contributor License Agreement

waarbij de CLA een onherroepbare licentie toekent aan de project eigenaar om de bijdrage te gebruiken

# CLA in het kort (bijdragen studenten)

- Eigenaarshap (studenten en lectoren)
- Relatie (gezond)
- Juridisch (problemen vermijden)

# Concreet

- Impliciet tussen studenten en lectoren (=project eigenaars)
- Zie eventueel het document  
[A1\\_CLA\\_contributor\\_license\\_agreement.pdf](#)  
impliciet.

# Referenties

- <http://www.contributoragreements.org>

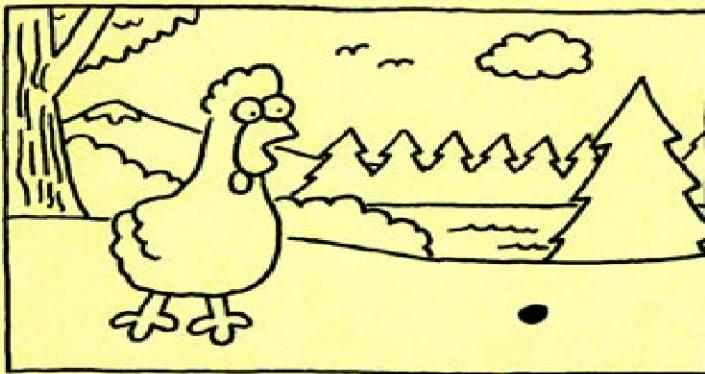
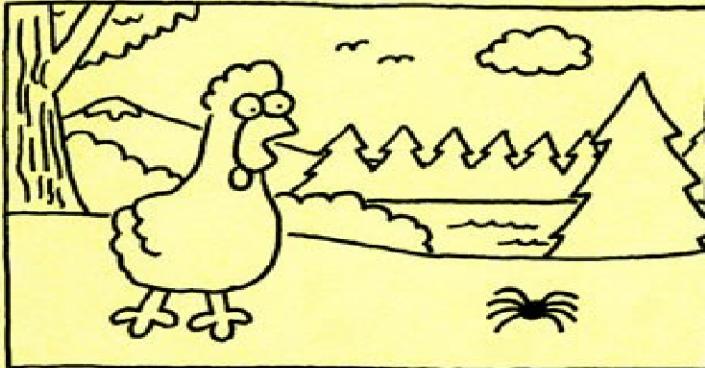
Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License

# Herhaling ERM

[wim.bertels@ucll.be](mailto:wim.bertels@ucll.be)

Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Unported  
Licentie

## SPOT THE 8 DIFFERENCES BETWEEN THESE TWO PICTURES



<http://www.savagechickens.com/2011/06/spot-the-differences.html>

File Edit View Insert Format Tools Data Window Help

Arial 20 A A A | % 0.00 0.00 | Sum=0 Default STD

A13 fΣ = "KBE25AZ21"

	A	B	E	F	I
1	"carrosserienr"	"status"	"merk"	"model"	"beschrijving"
2	"KBE25AZ21	"klaar voor verkoop "	"Renault "	"Trafic "	"nieuwe banden
3	"KBE25AZ21	"klaar voor verkoop "	"Renault "	"Trafic "	"opnieuw gespoten
4	"KBE25AZ21	"klaar voor verkoop "	"Renault "	"Trafic "	"remschijven vervangen
5	"LJB20FK30	"klaar voor verkoop "	"Rover "	"Mini "	"Wielen uitgelijnd
6	"LJB20FK30	"klaar voor verkoop "	"Rover "	"Mini "	"koppelingsplaat vervang
7	"LJB20FK30	"klaar voor verkoop "	"Rover "	"Mini "	"koplampen vervangen
8	"UHDL982HE	"verkocht "	"Mercedes "	"G350 "	"uitlaat veranderd
9	"UHDL982HE	"verkocht "	"Mercedes "	"G350 "	"nieuwe banden
10	"ZZFD73870	"in bewerking "	"Austin "	"Moke "	""
11	"KBE25AZ21	"niet voor verkoop "	"Renault "	"Trafic "	"nieuwe banden
12	"KBE25AZ21	"klaar voor verkoop "	"Renault "	"Trafic "	"opnieuw gespoten
13	"KBE25AZ21	"klaar voor verkoop "	"Renault "	"Trafic "	"remschijven vervangen
14	"KBE25AZ21	"niet voor verkoop "	"Renault "	"Trafic "	"nieuwe banden
15	"KBE25AZ21	"klaar voor verkoop "	"Renault "	"Trafic "	"opnieuw gespoten
16	"KBE25AZ21	"klaar voor verkoop "	"Renault "	"Trafic "	"remschijven vervangen
17	"KBE25AZ21	"niet voor verkoop "	"Renault "	"Trafic "	"nieuwe banden
18	"KBE25AZ21	"klaar voor verkoop "	"Renault "	"Trafic "	"opnieuw gespoten
19	"KBE25AZ21	"klaar voor verkoop "	"Renault "	"Trafic "	"remschijven vervangen

Sheet1 Sheet2 Sheet3

100%

# Uitnormaliseren

Redundantie Vermijden  
Consistentie Bewaken

# Kandidaatsleutels

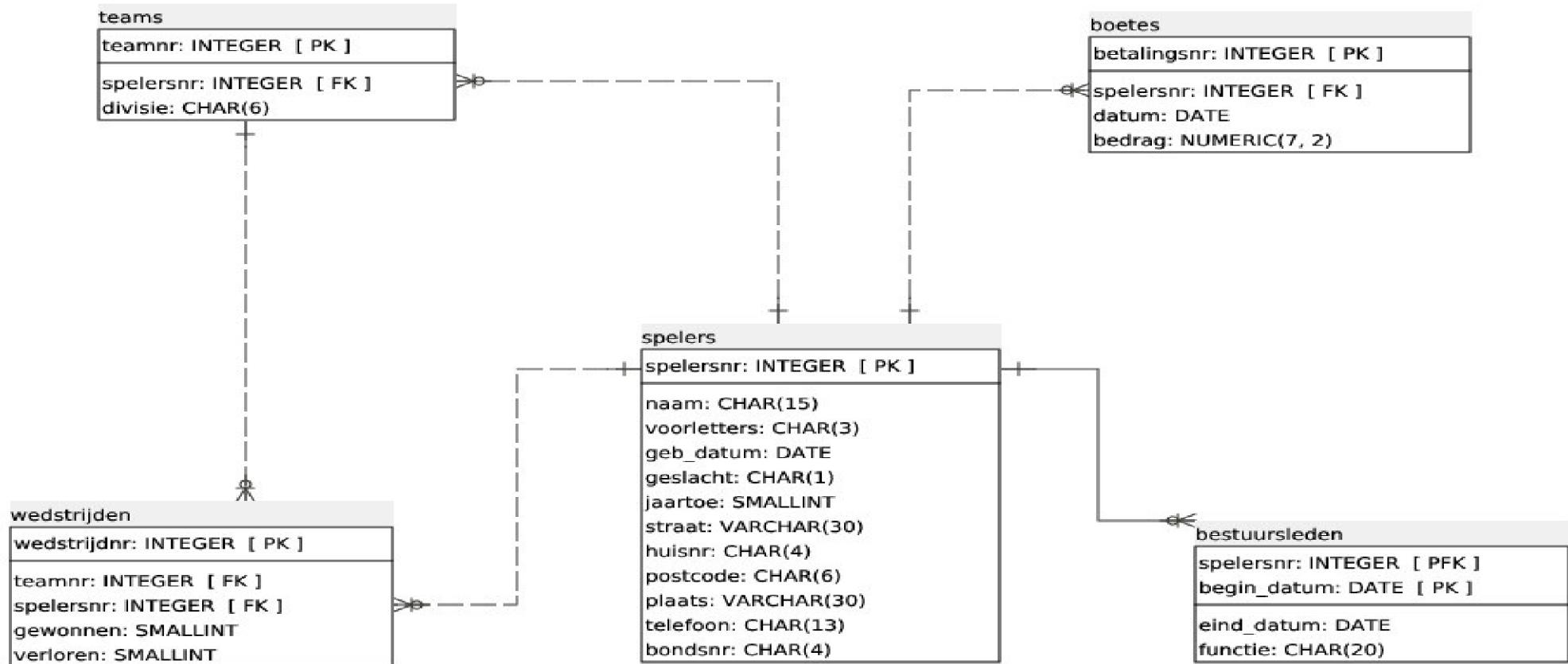
- Minimaal! Anders..?
- Uniek! Anders..?
- Constraints
- Integriteit
- Primary Key is verplicht (NOT NULL)
- Unique is (niet) verplicht

# ToID or not to told

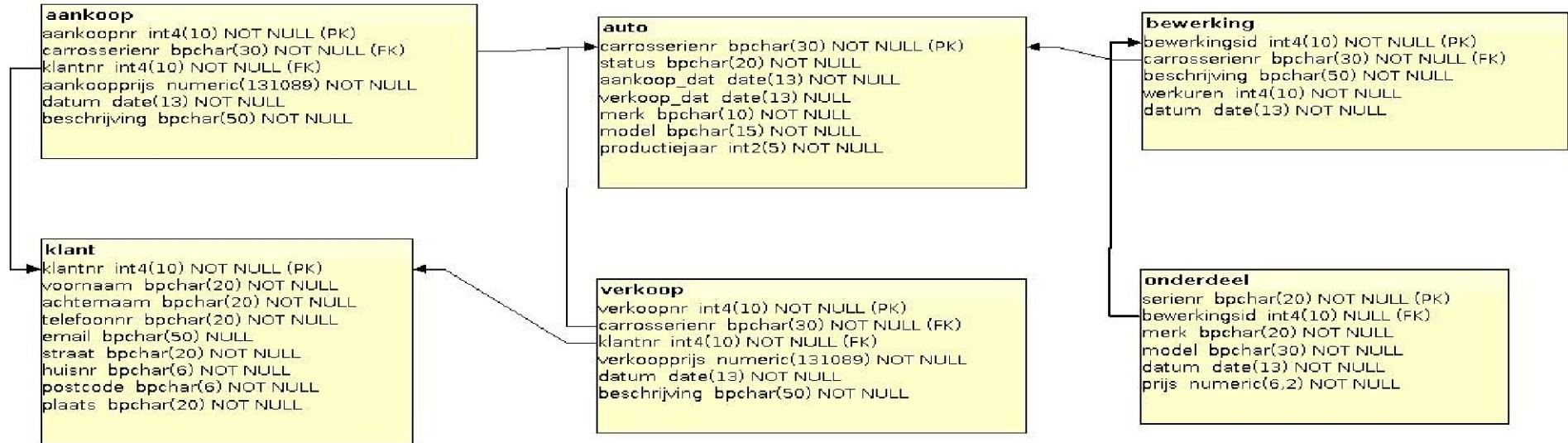
- Reële wereld, virtuele wereld
- Heeft het zin?
- Krijgt deze ID betekenis?
  - Verwijzingen in reële wereld?
  - Als gerefereerde sleutel?
- Of is het een automatisme?
  - Later ORDBMS en frameworks

# Vreemde sleutels

- Noodzakelijke, (minimale) vorm van redundantie.
  - Constraints
  - Referentiële integriteit
- 
- Foreign Key mag NULL zijn
  - Foreign Key kan verwijzen naar een PK of UN



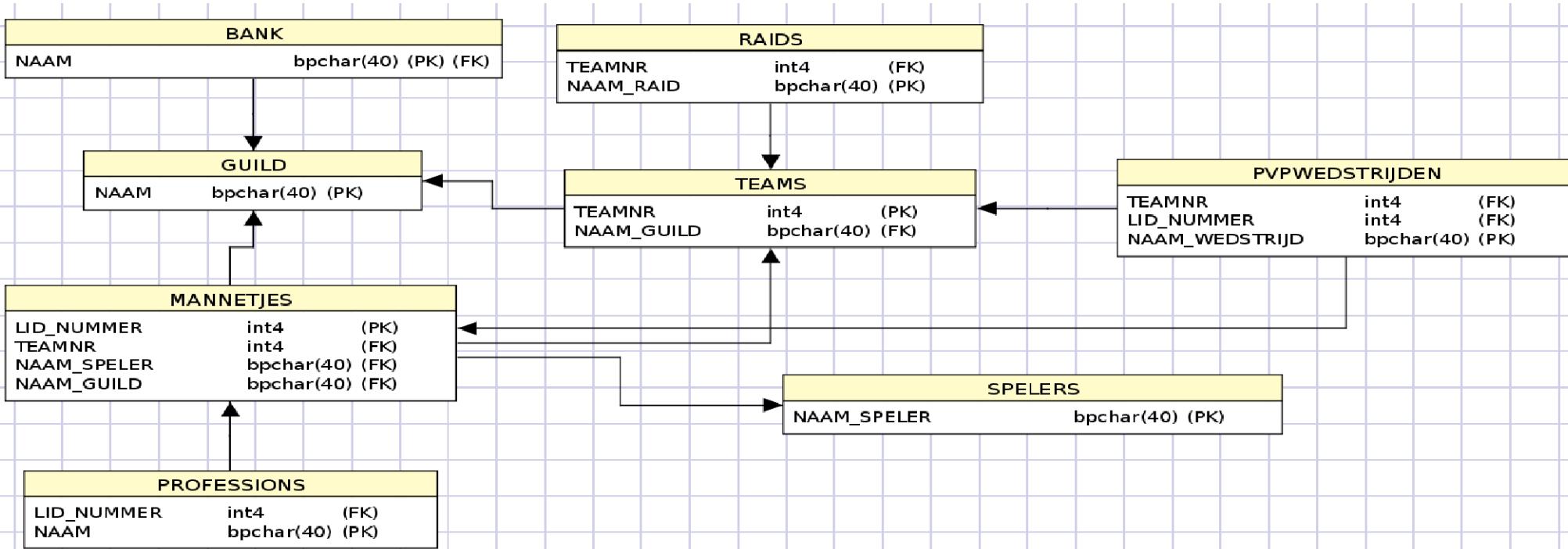
<http://code.google.com/p/power-architect/>



<http://www.squirrelsql.org/>

jdbc:postgresql://databanken.ucll.be:portnr/databasename?ssl=true&sslfactory=org.postgresql.ssl.NonValidatingFactory

<http://jdbc.postgresql.org>



<http://executequery.org/>

# ERM

- Conceptueel?
- Logisch?
- Fysisch?
- Praktisch twee stappen om mee te beginnen:
  - analyse
  - Implementatie
- Aantal oplossingen?

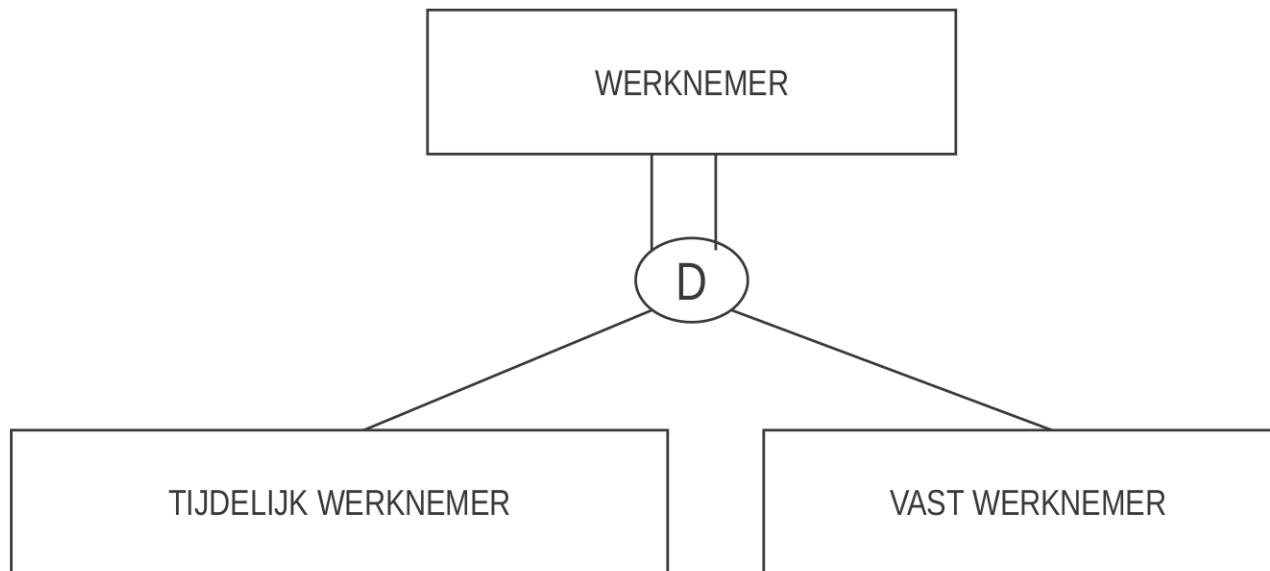
# Opgave

Een eenvoudige studentenadministratie. Een student volgt een vak in een reeks.

Probleem: een student mag voor 1 vak maar in 1 reeks ingeschreven zijn.

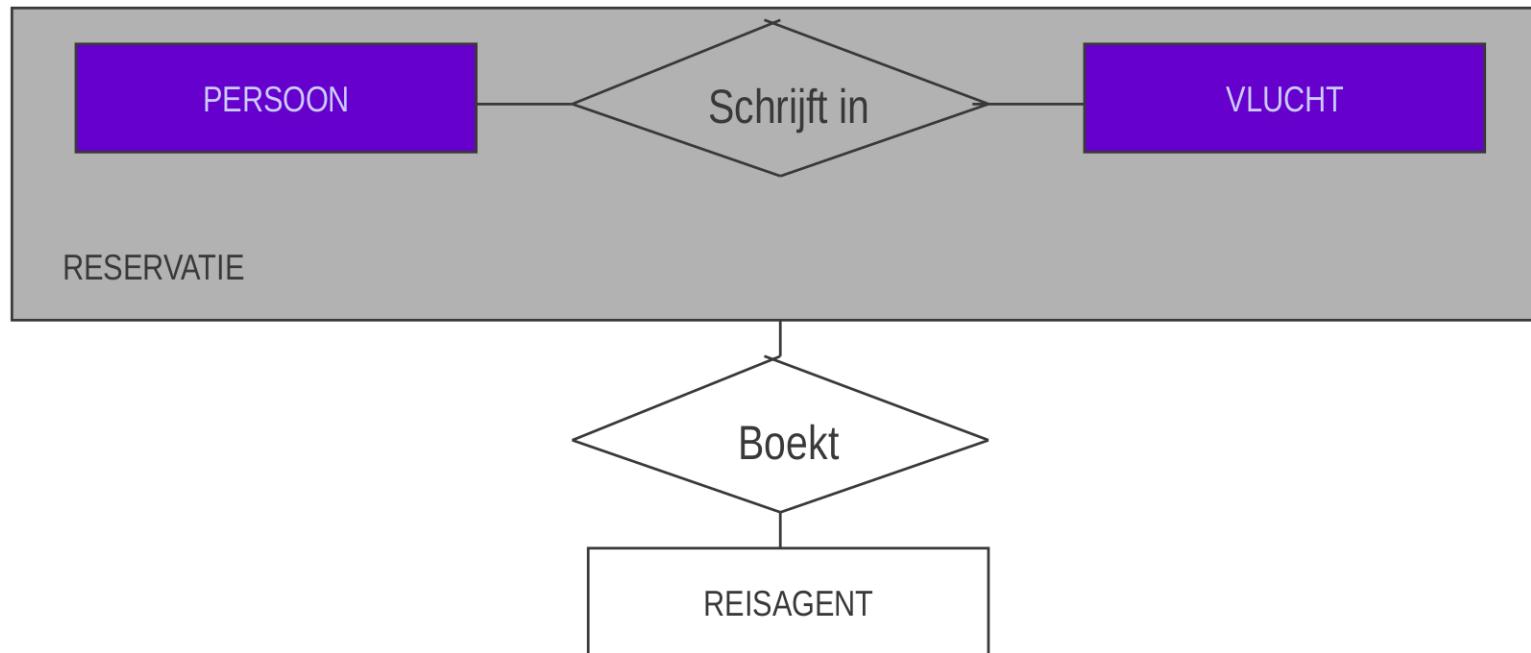
# (E)ERM

- 3 mogelijke omzettingen?



# (E)ERM

- 1 mogelijk beoogde omzetting?



# Herhaling SQL

wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen 4.0  
Unported Licentie

# SQL

SQL=Structured Query Language

ISO, no dialects

# Lezen van gegevens

SELECT

# Muteren van gegevens

INSERT INTO

UPDATE

DELETE FROM

# Bepalen structuur voor gegevens

CREATE TABLE

GRANT

# Volgorde?

SELECT

FROM

WHERE

GROUP BY

HAVING

ORDER BY

# Voorbeeld

```
SELECT      klantrn, MAX(reisnr) AS  
                      laatste_reis  
FROM        deelnames  
WHERE       reisnr > 10  
GROUP BY   klantrn  
HAVING     COUNT(reisnr) > 1  
ORDER BY   klantrn DESC;
```

# Functies

EXTRACT ( )

CAST ( )

TO\_CHAR ( )

ROUND ( )

...

productspecifieke SQL?!

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

# Alias

```
SELECT voornaam AS naam
```

```
SELECT *
FROM     spelers AS s
```

```
SELECT *
FROM     spelers s
```

## CASE

## CASE

```
WHEN satellietvan IS NULL THEN 'Zon'  
WHEN satellietvan = 'Zon' THEN  
          'Planeet'  
ELSE 'Satelliet'  
END
```

# CASE

```
SELECT
CASE
WHEN satellietvan IS NULL THEN 'Zon'
WHEN satellietvan = 'Zon' THEN 'Planeet'
ELSE 'Satelliet'
END AS naamplaneet
FROM hemellichaam
WHERE
CASE
WHEN satellietvan IS NULL THEN 'Zon'
WHEN satellietvan = 'Zon' THEN 'Planeet'
ELSE 'Satelliet'
END
LIKE '%et'
```

# Joins

```
SELECT *
FROM    reizen INNER JOIN deelnames
ON reizen.reisnr = deelnames.reisnr
```

# LEFT OUTER JOIN

```
SELECT hemelobjecten.objectnaam,  
       bezoeken.reisnr  
  
FROM     hemelobjecten LEFT OUTER JOIN  
        bezoeken  
  
USING (objectnaam)
```

# LEFT OUTER JOIN

```
SELECT      s.spelersnr, w.wedstrijdnr
FROM        spelers s LEFT OUTER JOIN
            wedstrijden w
          ON s.spelersnr = w.spelersnr
          AND w.gewonnen > w.verloren
/* condition in de join */
```

# LEFT OUTER JOIN

```
/* versus: */
```

```
SELECT s.spelersnr, w.wedstrijdnr
FROM spelers s LEFT OUTER JOIN wedstrijden w
    ON s.spelersnr = w.spelersnr
WHERE w.gewonnen > w.verloren
```

```
/* condition in the where */
```

# LEFT OUTER JOIN

```
SELECT      s.spelersnr  
FROM        spelers s LEFT OUTER JOIN  
            wedstrijden w  
ON          s.spelersnr = w.spelersnr  
AND         w.gewonnen > w.verloren  
GROUP BY    s.spelersnr
```

# Selecties op groeperingen

GROUP BY

HAVING

# Aggregatiefuncties

COUNT( )

MAX( )

MIN( )

SUM( )

AVG( )

STDDEV( )

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

# Oefening 1

Toon alle spelers en het aantal wedstrijden dat ze hebben gespeeld.

# Manier 1?

```
SELECT      s.spelersnr, COUNT(*) AS
            aantal

FROM        spelers s INNER JOIN
            wedstrijden w
                    ON s.spelersnr = w.spelersnr

GROUP BY   s.spelersnr;
```

# Manier 2?

```
SELECT      s.spelersnr, COUNT(*) AS
            aantal

FROM        spelers s LEFT OUTER JOIN
            wedstrijden w
            ON s.spelersnr = w.spelersnr

GROUP BY    s.spelersnr;
```

# Manier 3?

```
SELECT    s.spelersnr, COUNT(w.wedstrijd.nr)
          AS aantal
FROM      spelers s LEFT OUTER JOIN
          wedstrijden w
          ON s.spelersnr = w.spelersnr
GROUP BY s.spelersnr;
```

# Toevoegen

```
INSERT INTO bezoeken (
    reisnr,
    volgnr,
    objectnaam,
    verblijfsduur
) VALUES (
    34,
    4,
    'maan',
    2
)
```

# Oefening 2

Hoeveel spelers hebben een wedstrijd gespeeld, geef het totaal.

# Manier 1

```
SELECT COUNT(DISTINCT s.spelersnr)
      AS aantal
FROM   spelers s INNER JOIN
       wedstrijden w
      ON s.spelersnr = w.spelersnr;
```

# Bijwerken

```
UPDATE reizen  
SET vertrekdatum = '2030-12-31',  
    reisduur = 30,  
    prijs = 1.23  
WHERE reisnr = 33;
```

# Verwijderen

```
DELETE  
FROM hemelobjecten  
WHERE objectnaam = 'Pluto';  
  
/* zonder where clause?; */
```

# Aanmaken

```
CREATE TABLE bezoeken (
    reisnr      numeric(4,0)          NOT NULL,
    objectnaam   character varying(10) NOT NULL,
    volgnr      numeric(2,0)          NOT NULL,
    verblijfsduur numeric(4,0)          NOT NULL,
    CONSTRAINT bezoeken_pkey PRIMARY KEY (reisnr, volgnr),
    CONSTRAINT bezoeken_objectnaam_fkey
        FOREIGN KEY (objectnaam) REFERENCES hemelobjecten (objectnaam),
    CONSTRAINT bezoeken_reisnr_fkey
        FOREIGN KEY (reisnr) REFERENCES reizen (reisnr)
);

-- datatypes zijn productspecifieke SQL?
-- http://www.postgresql.org/docs/current/interactive/datatype.html
```

# Rechten geven

```
GRANT      SELECT  
ON         TABLE bezoeken  
TO         student;
```

-- Voldoende?

# Rechten geven

```
GRANT    USAGE  
ON        SCHEMA ruimtereizen  
TO        public;
```

## Referenties:

- \* Slides herhaling sql 2012-13, K. Beheydt
- \* SQL Leerboek, R. Van der Lans

# Transacties en multi-user gebruik

[wim.bertels@ucll.be](mailto:wim.bertels@ucll.be)

Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Unported  
Licentie

# Probleem?

- Single-user vs multi-user
- Wanneer meerdere gebruikers dezelfde data willen lezen/schrijven
  - Conflicten
  - Concurrency control is nodig (gelijktijdigheid)
- Data uit veel tabellen moet geraadpleegd worden



# Transacties

- Def.: verzameling SQL-instructies die door één gebruiker ingevoerd wordt en waarvan de mutaties blijvend moeten zijn of ongedaan moeten worden
- Autocommit:
  - Elke SQL-instructie is een transactie
  - Elke transactie is permanent
- Commit: permanent maken van een transactie
- Rollback: ongedaan maken van een transactie
  - Eenmaal gecommit is er geen rollback mogelijk

# Transacties

- Transactie : vanaf begin tot een commit of rollback
- Laatste : steeds commit of rollback
- Wanneer zinvol ?
  - Als een bepaald gegeven uit meerdere tabellen geschrapt moet worden
  - Als gebruiker zich vergist heeft bij aanpassingen
- Mogelijke uitzonderingen (product beperkingen) : instructies die de catalogus wijzigen

# Hoe

- Impliciete start bv na ROLLBACK of COMMIT
  - Cf autocommit
- Expliciete start
  - begin; sql code; commit ; -- dialect
  - begin; sql code; rollback; -- dialect
  - start transaction; sql code; commit ;
  - start transaction; sql code; rollback;

# Savepoints

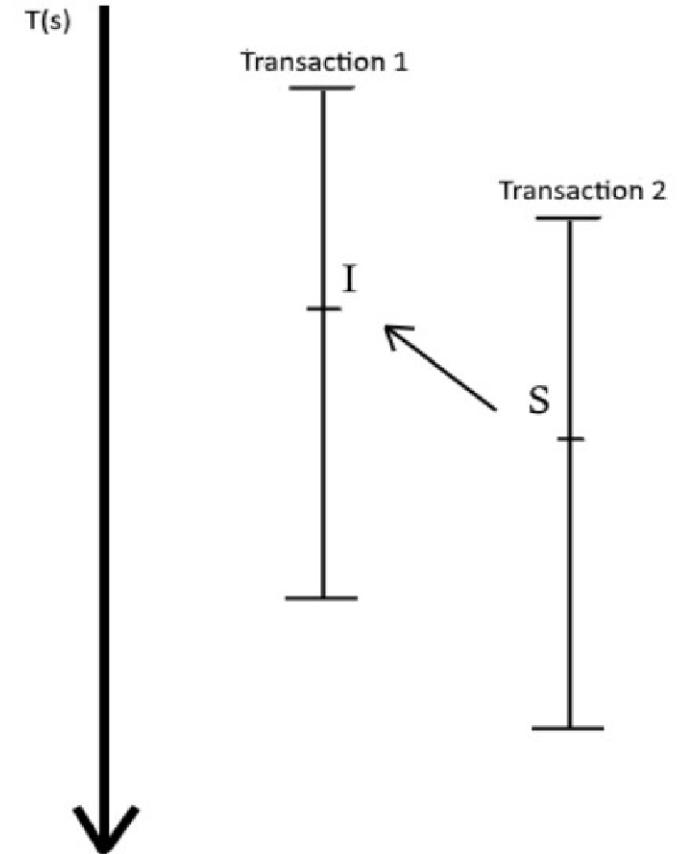
- Savepoints :
  - maken een deel van een actuele transactie ongedaan
  - Tussentijdse momentopnames in een transactie
- Vb.
  - Update ...
  - insert ...
  - savepoint S1
  - insert ...
  - savepoint S2
  - delete ...
  - rollback work to savepoint S2
  - ...

# Problemen multi-user gebruik

- Dirty read (uncommitted read) :
  - een gebruiker leest een gegeven dat nooit gecommit werd
- Nonrepeatable (of nonreproducible) read :
  - Een gebruiker leest voor en na de commit andere gegevens (gegevens worden gewijzigd)
- Phantom read :
  - een gebruiker leest voor en na de commit andere gegevens (er komen nieuwe gegevens)
- Lost update :
  - Een wijziging van één gebruiker wordt overschreven door een andere gebruiker
- Serialization Anomaly:
  - Het resultaat van het succesvol vastleggen van een groep transacties is inconsistent met alle mogelijke volgordes van het één voor één uitvoeren van die transacties.

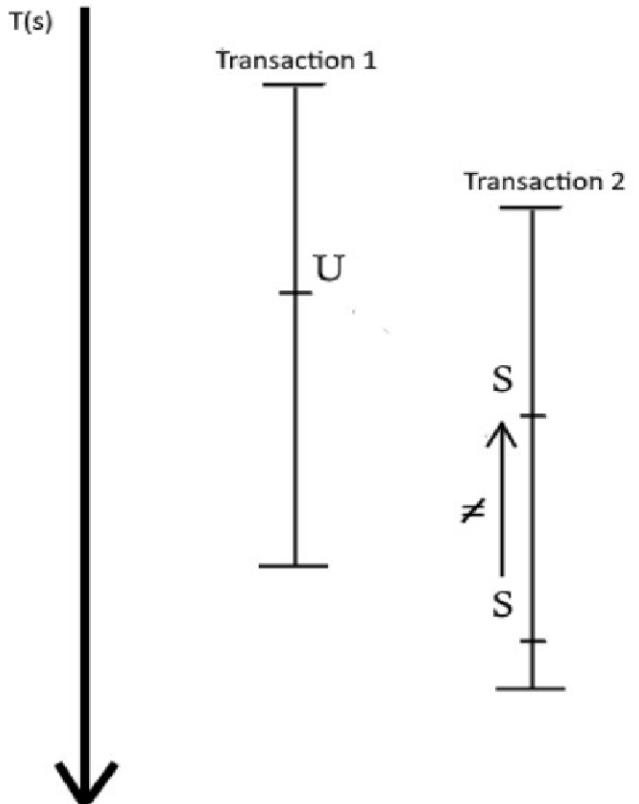
# Dirty read

- Een transactie leest data gecreëerd door een gelijktijdige transactie die nog niet gecommit is



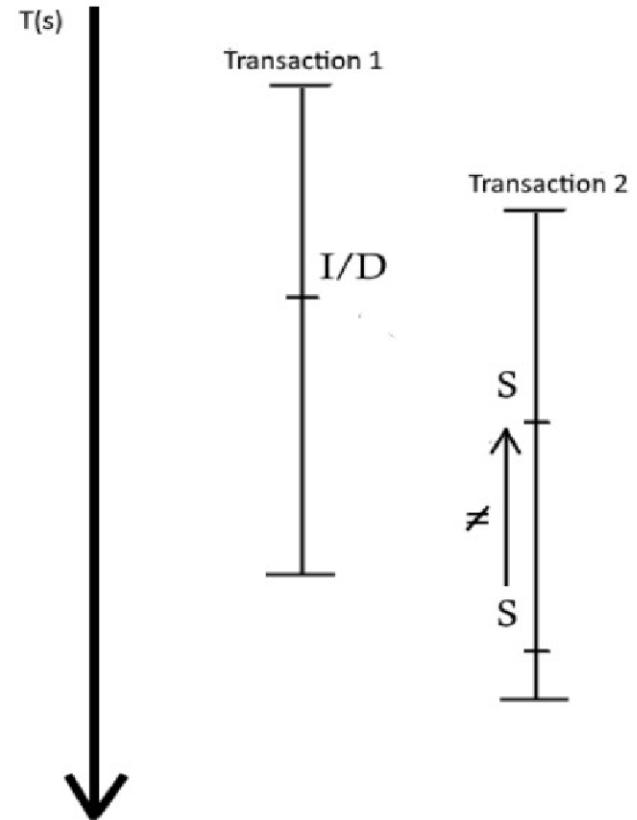
# Nonrepeatable read

- Een transactie leest data die eerder gelezen is en vindt dat de data aangepast is door een transactie die gecommit is sinds de eerste lees operatie.



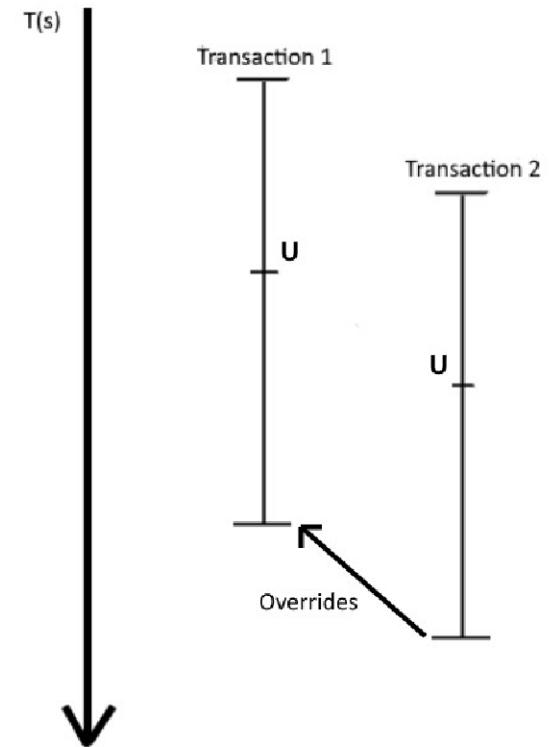
# Phantom read

- Een transactie herleest data die eerder gelezen is en vindt dat er data bijgekomen of verwijderd is door een andere transactie die gecommit is sinds de eerste lees operatie.



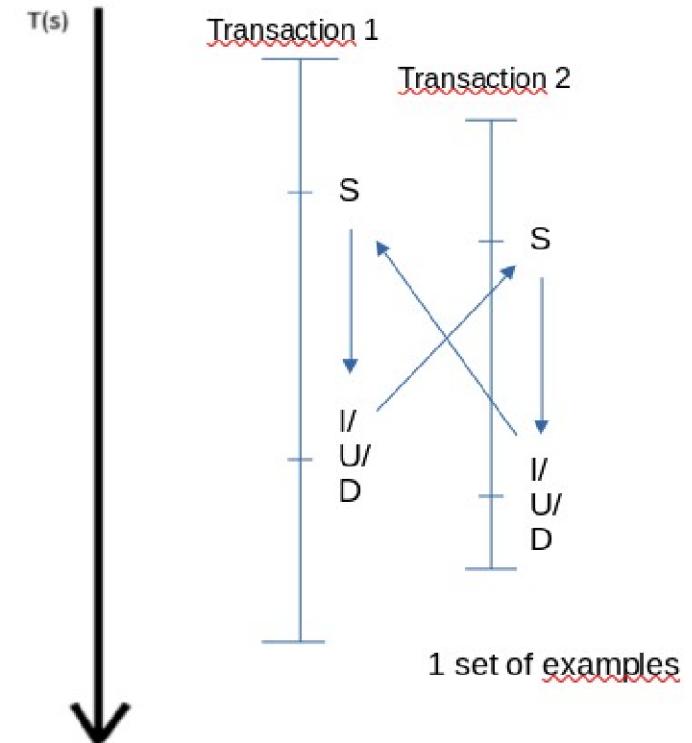
# Lost update

- Enkel de veranderingen van de laatste commit van gelijktijdige transacties die dezelfde rijen updaten zullen behouden worden.



# Serialization Anomaly

- Een andere (interne) volgorde van de overlappende transacties zorgt voor een ander resultaat.



# Oplossing

- Oplossing :
  - Transacties serieel verwerken!
- Oplossing indien honderden gebruikers tegelijk willen werken
  - Transacties parallel verwerken!

# LOCK TABLE ..

- Locking :
  - de rij waar één gebruiker mee werkt wordt gelocked voor de andere gebruikers
  - als transactie afgelopen is, wordt de blokkade opgeheven
- Locking gebeurt in de buffer (eg RAM)
- Verschillende opties voor granulariteit en rechten (bv SHARE vs EXCLUSIVE)

# Deadlocks

- Cf operatings systems
- Deadlock :
  - indien twee of meerdere gebruikers op elkaar wachten
- Oplossing :
  - indien deadlock aanwezig, dan wordt één transactie afgebroken

# Transacties: ISOLATION LEVEL

- Isolation level : mate van isolatie van gebruikers
- Niveaus:
  - Serializable : maximaal gescheiden
  - Repeatable read :
    - lezen : share blokkades (stopt bij einde transactie)
    - muteren : exclusive blokkades
  - Read committed (cursor stability):
    - lezen : share blokkades (stopt bij einde select)
    - muteren : exclusive blokkades
  - Read uncommitted (dirty read):
    - lezen : share blokkades (stopt bij einde select)
    - muteren : exclusive blokkades (stopt bij einde mutatie)

# Gevolgen

- Vermijd lang durende transacties
- Serializable :
  - concurrency is het laagst
  - Snelheid laagst
- Read Uncommitted:
  - concurrency is hoog, moeten weinig op elkaar wachten
  - Kunnen gegevens lezen die enkele momenten later niet meer bestaan

Vb.

- Set transaction isolation level serializable

# Usefull links for PostgreSQL

- <http://www.postgresql.org/docs/current/static/mvcc.html>
- <http://www.postgresql.org/docs/current/static/transaction-iso.html>
- <http://www.postgresql.org/docs/current/static/sql-start-transaction.html>

# Prepared statements

[wim.bertels@ucll.be](mailto:wim.bertels@ucll.be)

Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Unported  
Licentie

# Prepared statements

Wat ?

- Server-side object
- Eventueel parameters
- Kan gebruikt worden om performantie te optimaliseren
  - Meerdere keren gelijkaardige query's uitvoeren
  - Complexe query's met veel joins
- Duurt enkel zolang als de huidige DB connectie of wanneer DEALLOCATE gebruikt wordt
  - ⇒ Kan niet door meerdere DB clients/sessions gebruikt worden
  - ⇒ Elke client kan zijn eigen prepared statements hebben, sessie afhankelijk
- Syntax:
  - ⇒ PREPARE name [ ( data\_type [, ...] ) ] AS statement

# Prepared statements

Flow

1. PREPARE

2. EXECUTE

...

3. DEALLOCATE

Voorbeeld:

```
PREPARE add_customer (numeric(4,0), varchar(20), varchar(20), date)
AS INSERT INTO klanten VALUES($1, $2, $3, $4);
EXECUTE add_customer(100, 'Lieve', 'Standaert', now());
```

# Prepared statements

PREPARE *statement* -> statement is...

- Parsed
- Analysed
- Rewritten

EXECUTE *statement* ->

statement is...

- Planned
- Executed

DEALLOCATE *statement* ->  
statement is...

- Dropped

# Prepared statements

Een Prepared Statement kan onderliggend zijn

- SELECT, UPDATE, DELETE, INSERT, ...
- Met parameters gesubstitueerd volgens positie, gebruik \$1, \$2, etc.
- Write **security**: Voordelig omdat geen andere query's uitgevoerd kunnen worden
- Een prepared statement zal uitvoeren wat voorbereid was

Lijst van huidige prepared statements in uw sessie (PostgreSQL):

```
SELECT *  
FROM pg_prepared_statements;
```

# PREPARE

## Syntax

- PREPARE name [ ( parameter [, ...] ) ] AS ..

## Voorbeeld

- PREPARE add\_customer (numeric(4,0), varchar(20),  
varchar(20), date)  
AS INSERT INTO klanten VALUES(\$1, \$2, \$3, \$4);

# EXECUTE

## Syntax

- EXECUTE name [ ( parameter [, ...] ) ]

## Voorbeeld

- EXECUTE add\_customer(100, 'Lieve', 'Standaert', now());

## Nota

- Een lijst van parameter data types kunnen ook gespecificeerd worden, bv.  
EXECUTE add\_customer(100, 'Lieve', 'Standaert', now()::timestamp); -- :: is een dialect voor cast
- Wanneer het datatype van een parameter onbekend is, wordt het datatype afgeleid uit de context waar de parameter gebruikt is (indien mogelijk).

# DEALLOCATE

## Syntax

- DEALLOCATE { name | ALL }

## Voorbeeld

- DEALLOCATE add\_customer;
- DEALLOCATE ALL;

# Embedded Prepared statements

Embedded in een andere taal , vb. in Java:

- PreparedStatement add\_customer =  
c.prepareStatement("INSERT INTO klanten  
VALUES(?, ?, ?, ?);");

# ISO?

Context:

- De SQL standaard bevat een PREPARE statement, maar dit is enkel bedoeld voor embedded SQL.
- De directe PostgreSQL PREPARE statement gebruikt een licht andere syntax dan deze standaard, het is een toevoeging.

Meer info:

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

# Prepared statements

Nota's:

- <https://pgbouncer.github.io/faq.html#how-to-use-prepared-statements-with-transaction-pooling>
- <https://jdbc.postgresql.org/documentation/server-prepare/#server-prepared-statements>
- <https://www.psycopg.org/psycopg3/docs/advanced/prepare.html>

# Creatie en Ontwerp

[wim.bertels@ucll.be](mailto:wim.bertels@ucll.be)

Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Unported  
Licentie

# INSERT

```
insert into recreanten (spelersnr, naam, plaats)
select spelersnr, naam, plaats
from spelers
where bondsnr is null;
```

# UPDATE

```
update wedstrijden  
set     gewonnen = 0  
where   spelersnr in  
        (select spelersnr  
         from   spelers  
         where  plaats = 'Den Haag');
```

# DELETE

```
delete
from    spelers
where   jaartoe >
        (select  avg(jaartoe)
         from    spelers);
```

# Tijdelijke tabellen

```
create temporary table SOMBOETES  
    (totaal decimal(10,2));
```

```
insert into somboetes  
    select sum(bedrag)  
        from boetes;
```

# Kopie tabel

```
create table teams_kopie as  
  (select *  
   from teams);
```

# Default waarden

```
create table boetes (
    bnr      integer not null primary key,
    snr      integer not null,
    datum    date    not null default '1990-01-01',
    bedrag   int     not null default 50);
```

# Tabel beperkingen

```
[ CONSTRAINT constraint_name ]  
{ CHECK ( expression ) [ NO INHERIT ] |  
  UNIQUE [ NULLS [ NOT ] DISTINCT ] ( column_name [, ...] ) index_parameters |  
  PRIMARY KEY ( column_name [, ...] ) index_parameters |  
  EXCLUDE [ USING index_method ] ( exclude_element WITH operator [, ...] ) index_parameters  
[ WHERE ( predicate ) ] |  
  FOREIGN KEY ( column_name [, ...] ) REFERENCES reftable [ ( refcolumn [, ...] ) ]  
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE referential_action ] [ ON  
    UPDATE referential_action ] }  
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

# Kolom beperkingen

```
[ CONSTRAINT constraint_name ]  
{ NOT NULL |  
NULL |  
CHECK ( expression ) [ NO INHERIT ] |  
DEFAULT default_expr |  
GENERATED ALWAYS AS ( generation_expr ) STORED |  
GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY [ ( sequence_options ) ] |  
UNIQUE [ NULLS [ NOT ] DISTINCT ] index_parameters |  
PRIMARY KEY index_parameters |  
REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]  
[ ON DELETE referential_action ] [ ON UPDATE referential_action ] }  
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

# Deferrable

r ..fk1 references t(pk1) deferrable..

```
start transaction;  
set constraints all deferred;  
insert into r(fk1) values (1);  
insert into t(pk1) values (1);  
commit;
```

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

# Referende actie

..fk1 references t(pk1) on delete set null..

Referentiële integriteit?:

- NO ACTION: default
- RESTRICT: =no action, niet uitstelbaar (deferrable)
- CASCADE: Waterval effect!
- SET NULL(fk1): optie om zinvolle selectie van verwijzende kolommen op te geven
- SET DEFAULT: als er een zinvolle default is (optie cf set null)

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

# Catalogus

Meta en organisatie data over de databank in de databank zelf

- information\_schema
- pg\_catalog

Eg. `select * from information_schema.tables;`

# Schema en Locale

[wim.bertels@ucll.be](mailto:wim.bertels@ucll.be)

Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Unported  
Licentie

# Wat

- SCHEMA = namespace
- Binnen een namespace moeten alle objecten (b.v. tabellen, functies, views (later) ...) een unieke naam hebben

# Welk schema

PostgreSQL:

de standaard namespace is "public"

je kan dit nakijken met:

> SHOW search\_path;

# Product specifiek

- Het standaard/default schema is opstelling specifiek
- Sommige andere producten:  
de standaard namespace is de usernaam

# Rechten

## Privileges (standaard)

- Je moet rechten hebben op het parentobject om rechten uit te voeren op de kinderobjecten
- default: enkel eigenaar heeft toegangsrechten
- Dus Server>DB>Schema>Objecten(bv tabel)

# Rechten toekennen

```
GRANT <privilege> ON  
<objecttype> <objectname>  
TO <role>;
```

# Rechten wegnemen

```
REVOKE <privilege> ON  
<objecttype> <objectname>  
FROM <role>;
```

# Privileges doorgeven?

- Recht toekennen om rechten toe te kennen:

```
GRANT <privilege> ON  
<objecttype> <objectname>  
TO <role>  
WITH GRANT OPTION;
```

# Privilege voorbeelden

- SELECT
  - INSERT
  - UPDATE
  - DELETE
  - EXECUTE
  - ALL PRIVILEGES
- 
- GRANT DELETE ON TABLE boetes TO admin WITH GRANT OPTION;

# Localisatie

## Localisatie (1/2)

- Teken sets: speciale tekens ondersteunen (b.v. ç, Й ...)
- collating sequences: abc ...
- encoding: UTF-8, LATIN1, ...
- ..

## Localisatie (2/2)

- LC\_COLLATE : string volgorde
- LC\_CTYPE : character classificatie
- LC\_MESSAGES : taal van het bericht
- LC\_MONETARY : formatteren valuta
- LC\_NUMERIC : formatteren nummers
- LC\_TIME : formatteren tijd

>SHOW LC\_COLLATE;

# Impact Locale?

- Wat zijn de gevolgen hiervan?
- Waarom moeten we hiervoor opletten?

# Datatypes

[wim.bertels@ucll.be](mailto:wim.bertels@ucll.be)

Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Unported  
Licentie

# Kleinere topics

- Datatypes & Functies

# Datatypes & Functies

<https://www.postgresql.org/docs/current/datatype.html>

<https://www.postgresql.org/docs/current/functions.html>

- Numerisch
- Geld
- Character
- Binair
- Tijd
- Boolean
- ..

# Algemene verschillen voor teken-datatypes

- `char(n)`: vaste lengte
  - vaste lengte, plaats wordt gereserveerd ook indien deze plaats niet gebruikt wordt
- `varchar(n)`: variabele lengte
  - flexibele lengte met maximum, trager?
- `text`: onbeperkte variabele lengte
  - meest flexibel, traagst?

# Strings vs Identifiers

'dit een string'

"dit is een identifier"

bv. 'hond' vs "54 mijngekkelabelnaam"

sql : standard uppercase >

in de praktijk: sqlcode is case insensitive tenzij tussen quotes

# IDS

[wim.bertels@ucll.be](mailto:wim.bertels@ucll.be)

Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Unported  
Licentie

# IDENTITY datatype

- Geen echte sleutel!
- Betekenis?
- Willekeur!
- Exploits?

# IDENTITY kolom

```
CREATE TABLE id_demo(  
    id      integer PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,  
    content text  
);
```

## OPMERKING:

- CREATE TABLE zal een impliciete sequentie "id\_demo\_id\_seq" aanmaken voor IDENTITY kolom "id" in tabel id\_demo

## OPMERKING:

- CREATE TABLE + PRIMARY KEY zal een impliciete unieke index "id\_demo\_pkey" aanmaken voor table "id\_demo"

# IDENTITY kolom

Lijst van relaties			
Schema	Naam	Type	Eigenaar
public	id_demo	table	wim
public	id_demo_id_seq	sequence	wim
...			
(103 rijen)			

# IDENTITY kolom

- Creeert extra object
  - Een sequence
- Wat als een gebruiker rechten heeft tot de tabel en niet tot de sequence?

# Rechten op een sequence

```
GRANT <privilege> ON  
SEQUENCE sequence_name  
TO <role>;
```

of

```
GRANT <privilege> ON  
ALL SEQUENCES IN SCHEMA schema_name  
TO <role>;
```

# ISO

- IDENTITY
- SERIAL, AUTOINCREMENT,.. --dialect

# Introductie Verbindingen Bundelen (Connection Pooling)

[wim.bertels@ucll.be](mailto:wim.bertels@ucll.be)

Naamsvermelding-NietCommercieel-GelijkDelen 4.0  
Unported Licentie

# Verbindingen bundelen

Doel

=

De 'overhead' die optreed bij database connecties en lees/schrijf operaties beperken

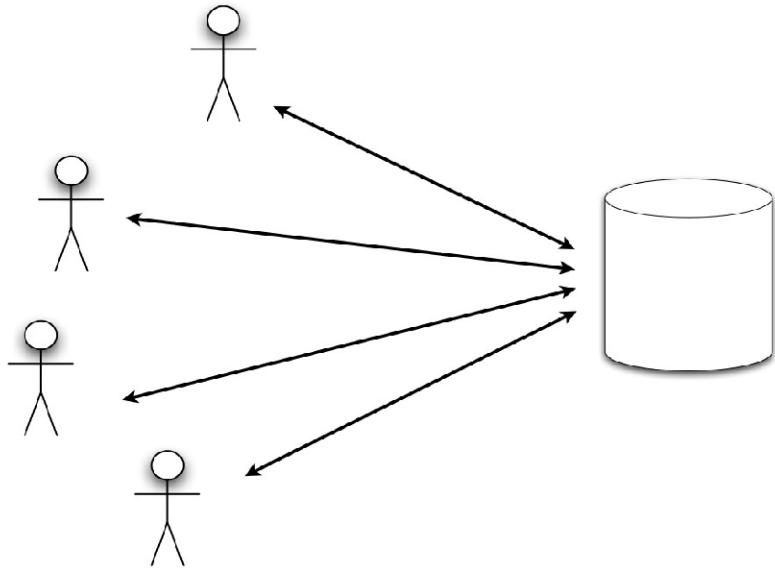
# Wat is connection pooling?

Een **connection pool** is een cache van database connecties die onderhouden worden door de database zodat ze hergebruikt kunnen worden voor request in de toekomst.

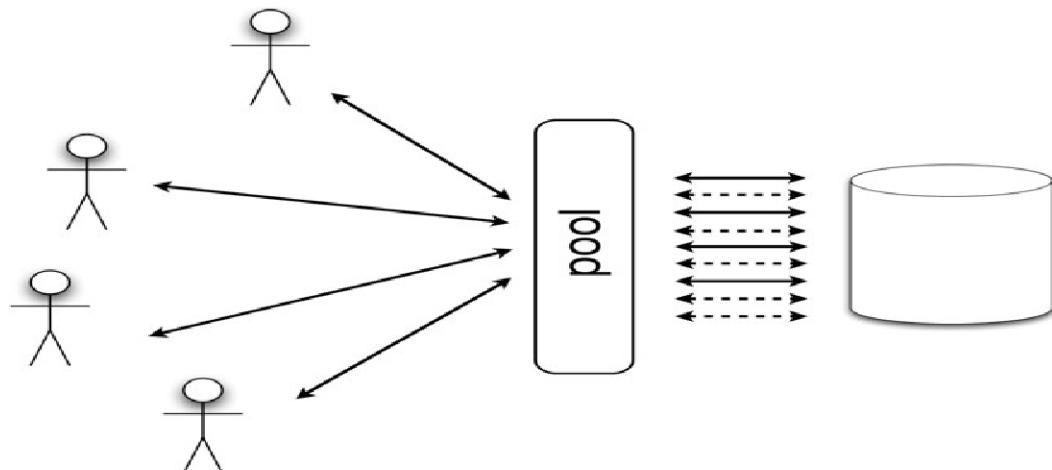
Opmerking:

Je kan verbindingen bundelen aan de client of server kant, of beide.

# Zonder Connection Pooling



# Met Connection Pooling



# Zonder pooling

Databank connecties zijn behoorlijk “dure” operaties. Elke connectie heeft een opstart kost en vraagt geheugen om te kunnen voldoen aan alle mogelijke eisen die van de connectie worden verwacht.

Bv

Er zijn 10 databank verbindingen

# Met Pooling

Verbindingen worden waar het kan gebundeld, gedeeld.

Bv

- Er zijn 10 verbindingen naar een pooling server
- De pooling server heeft maar 1 verbinding met de databank

# Verschillende vormen

Je kan op verschillende manieren gaan bundelen, vergelijk conceptueel een pooling server met een dynamisch wachtrij systeem bv

- Sessie:
  - Elke pooling connectie heeft zijn eigen databank connectie.
  - Deze databank connectie komt terug ter beschikking zo gauw de pooling connectie wordt afgesloten.
  - Alsof je een normale verbinding hebt, alleen is ze sneller ter beschikking
- Transaktie:
  - Elke transactie verloopt door dezelfde databank connectie.
  - Deze databank connectie komt terug ter beschikking zo gauw de transactie beëindigd wordt.
  - Je kan geen veronderstellingen maken over zaken die buiten je transactie gebeuren!
- Instructie (statement):
  - Deze databank connectie komt terug ter beschikking na elke instructie.
  - Het is zinloos expliciet transakties te gebruiken!
  - Je kan hiermee wel het hoogst aantal verbindingen met de laagste kost bereiken voor applicaties

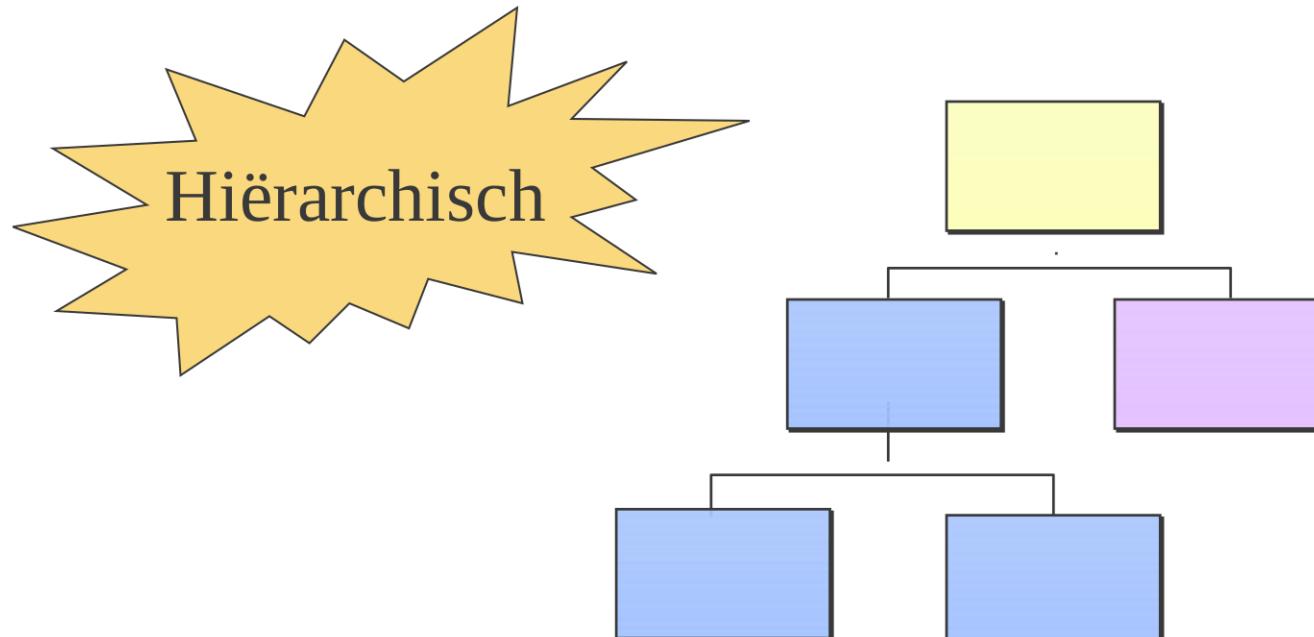


[http://lh5.ggpht.com/-dw2LEB2b75k/SgfogHTvQI/AAAAAAAAGU/OZZ\\_5Gih2rM/s800/sc7.jpg](http://lh5.ggpht.com/-dw2LEB2b75k/SgfogHTvQI/AAAAAAAAGU/OZZ_5Gih2rM/s800/sc7.jpg)

# .Data(banken)\_geschiedenis

1. Bestanden
2. Hiërarchisch model
3. Netwerkmodel
4. Relationeel model
5. Object-geörienteerd model
6. Andere paradigma

# DBMS - Soorten



wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen  
4.0 Unported Licentie

# Hiërarchisch GBS

1. Inleiding
2. Opstellen van het hiërarchisch model
3. 3-schema architectuur
4. Gegevensbanktalen
5. Voorbeeld

# Inleiding

- Ontstaan in jaren '60 ('66 – '68)  
Enorm populair geweest
- Bekendste voorbeeld: IMS (IBM)
- Momenteel: verliest aan belang?



United States [ change ]

Home Solutions Services Products Support & downloads My IBM

## IMS

Features and benefits

System requirements

Library

Success stories

News

Events

Training and certification

Services

Support

Software > Information Management > IMS Family >

# IMS

Information Management System Version 12

### IMS 12: Faster than ever!

IMS 12 delivers double-digit gains  
in performance, throughput, and simplicity



Overview

# IMS 12 (2011)

Firefox

IBM - IMS - Information Management S... +

www-01.ibm.com/software/data/ims/ Google United States Welcome [ IBM Sign in / Register ]

IBM Industries & solutions Services Products Support & downloads My IBM Search

IBM Software > Information Management

# IBM IMS

## 100,000 reasons to move to IMS 13

IMS delivers the lowest cost per transaction in the industry

#IBM\_IMS



Why IMS Downloads Integration Solutions 100K

# IMS 13 (2013)

## Why IMS for OLTP?

IBM Information Management System (IMS™) and the IMS tools portfolio provide industrial strength capabilities for both managing and distributing data. IMS delivers the highest levels of availability, performance, security and scalability for OLTP in the industry.

Expansive integration capabilities enable mobile and cloud applications based on IMS assets, enhanced analytics, new application development, and more.

 See the video (01:48)



IMS is used by many of the top Fortune 1000 companies worldwide. Collectively these companies process more than **50 billion transactions per day** in IMS – securely.

 See more in the infographic

## What's new in IMS?



### IBM Announces IMS 15

Learn how you can build trust into every transaction with IMS 15



### Continuous Delivery

Information about new enhancements to IMS delivered as PTFs is available in the Knowledge Center.



### IMS 14 webcast

In this webcast IMS Chief Architect, Betty Patterson, discusses how IMS 14 offers customers a competitive edge.



### Open access

Making your IMS data accessible to off-platform applications is easier than ever with the Open Access Solution Kit.

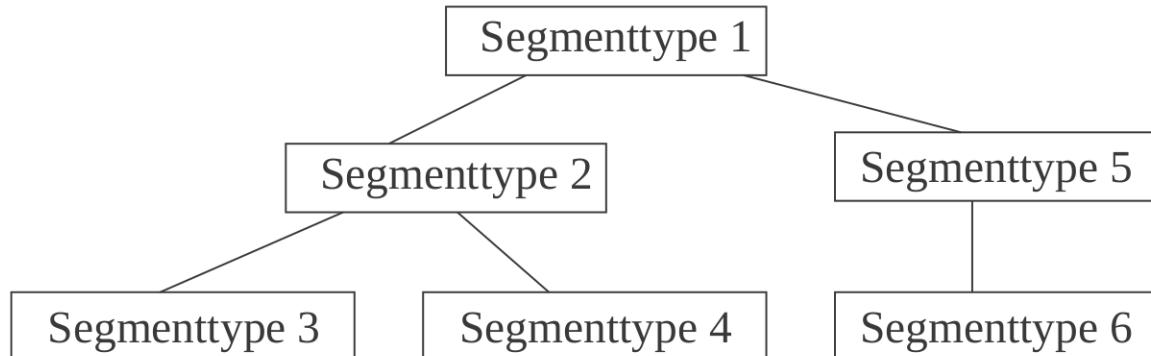
# IMS 15 (2017)

# Opstellen hiërarchisch model

1. Bouwstenen van het hiërarchisch model
2. ER-model naar hiërarchisch model
3. Leefregels van het hiërarchisch model
4. Terminologie van het hiërarchisch model
5. Voorbeeld

# Bouwstenen

- Segmenttypes
- Parent-child relationship-types
- Wortelsegment – bladeren
- n-m verbanden zijn niet toegelaten



# Enkele eigenschappen

- Voordelen:
  - Eenvoudige structuur
  - Snelle hiërarchische toegangsweg (cf pad hiërarchische boom)
- Nadelen :
  - Redundantie (op te vangen door verwijzingen)
  - Minder flexibele structuur, onderhoud
- Gebruik :
  - er is ook een SQL toegang (naast de eigen taal)

# ER-model naar hiërarchisch model

- n-op-m verbanden omzetten naar 1-op-n
- 1 segment type als parent en 1 als child kiezen

# Leefregels van hiërarchisch model

Denk eraan:

- Child moet parent hebben + slechts 1 parent
- Parent weg => alle children weg
- Beperking: Slechts 1 wortel

# Terminologie hiërarchisch model

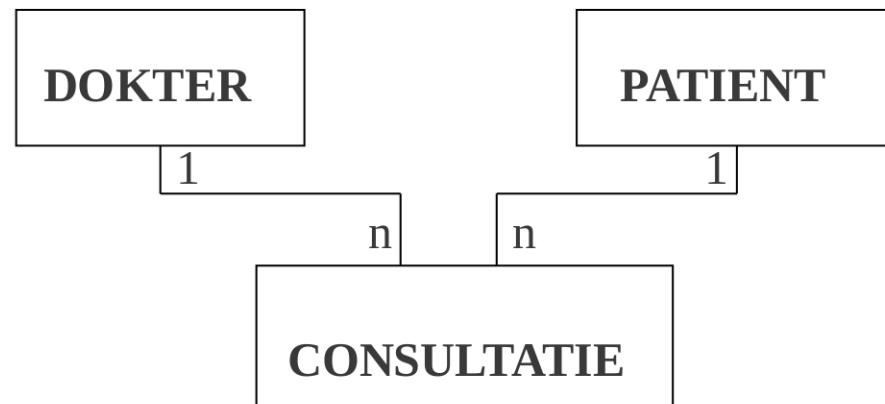
- Parent-segment
- Child-segment
- Twin-segment
- Segmentcode
- Volgorde-veld (sequence field)
- Samengestelde sleutel

# Voorbeeld

- Dokterspraktijk met meerdere dokters.  
Men wil de gegevens van de consultaties  
in een gegevensbank opslaan.

# Voorbeeld

- Dokterspraktijk met meerdere dokters.  
Men wil de gegevens van de consultaties  
in een gegevensbank opslaan.



# Gegevensdefinitietaal: DDL

- ◆ Conceptueel niveau: DBD
- ◆ Intern niveau: bestand
- ◆ Extern niveau: PCB

# DBD (data base description)

DBD NAME = ZIEKADM

SEGM NAME = **DOKTER**, BYTES = 139

FIELD NAME = **(DNR,SEQ)**, BYTES=3, START=1

FIELD NAME = DNAAM, BYTES=60, START = 4

FIELD NAME = DVNAAM, BYTES=30, START = 4

FIELD NAME = DFNAAM, BYTES=30, START = 34

FIELD NAME = DADRES, BYTES=76, START =64

FIELD NAME = DSTRAAT, BYTES=30, START = 64

FIELD NAME = DHUISNR, BYTES=8, START = 94

FIELD NAME = DPOST, BYTES=8, START = 102

FIELD NAME = DPLAATS, BYTES=30, START = 110

# DBD (data base description)

SEGM NAME = **PATIENT**, PARENT= **DOKTER**, BYTES = 144

FIELD NAME = **(PNR,SEQ)**, BYTES=8, START=1

FIELD NAME = PNAAM, BYTES=60, START = 9

FIELD NAME = PVNAAM, BYTES=30, START = 9

FIELD NAME = PFNAAM, BYTES=30, START = 39

FIELD NAME = PADRES, BYTES=76, START =69

FIELD NAME = PSTRAAT, BYTES=30, START = 69

FIELD NAME = PHUISNR, BYTES=8, START = 99

FIELD NAME = PPOST, BYTES=8, START = 107

FIELD NAME = PPLAATS, BYTES=30, START = 115

# DBD (data base description)

SEGM NAME = **CONSULT**, PARENT= **PATIENT**, BYTES = 84

FIELD NAME = **(CNR,SEQ)**, BYTES=8, START=1

FIELD NAME = CPRIJS, BYTES=5, START = 9

FIELD NAME = CDATUM, BYTES=8, START =14

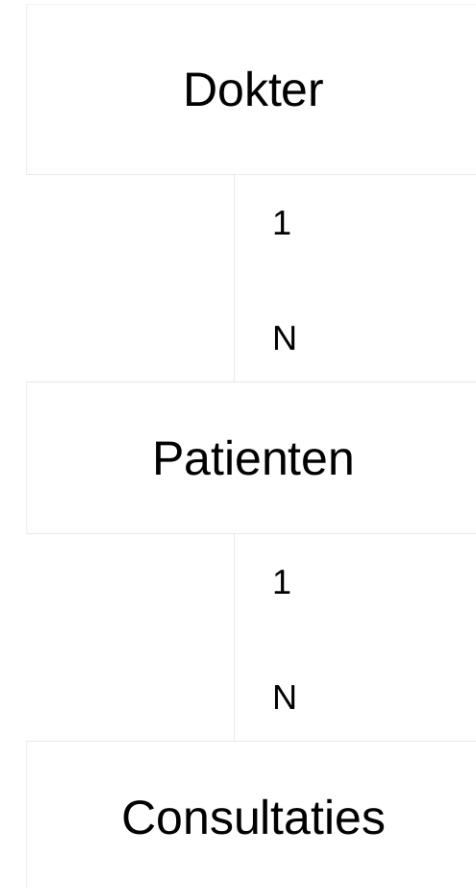
FIELD NAME = CTYPE, BYTES=3, START = 22

FIELD NAME = CBESCHR, BYTES=60, START = 25

# Voorbeeld

Dokterspraktijk met  
meerdere dokters.

Men wil de gegevens  
van de consultaties  
in een gegevensbank opslaan.



# Oefening: Reisbureau

- REIS

- Reisnr
- Reisoms
- Naambeg

- BESTEMMING

- Bestemnr
- Bestoms
- AantDag

- ◆ TOERIST

- Tnr
- Tnaam
- Tadres
- Straat
- Nummer
- Post
- Gemeente

# Oefening

DBD NAME = FREETIME

SEGMENT NAME = **REIS**, BYTES = 64

FIELD NAME = (REISNR,SEQ), BYTES=4, START=1

FIELD NAME = REISOMS, BYTES=30, START = 5

FIELD NAME = NAAMBEG, BYTES=30, START = 35

SEGMENT NAME = **BESTEM**, PARENT= **REIS**, BYTES = 35

FIELD NAME = (BESTEMNR,SEQ), BYTES=3, START=1

FIELD NAME = BESTOMS, BYTES=30, START = 4

FIELD NAME = AANTDAG, BYTES=2, START = 34

# Oefening

**SEGM NAME = TOERIST, PARENT= REIS, BYTES = 93**

**FIELD NAME = (TNR,SEQ), BYTES=3, START=1**

**FIELD NAME = TNAAM, BYTES=30, START = 4**

**FIELD NAME = TADRES, BYTES=60, START = 34**

**FIELD NAME = STRAAT, BYTES=25, START = 34**

**FIELD NAME = NUMMER, BYTES=5, START = 59**

**FIELD NAME = POST, BYTES=4, START = 64**

**FIELD NAME = GEMEENTE, BYTES=26, START = 68**

# Oefening: School

- KLAS

- Klasnr
- Klasoms
- Naam-tit

- VAK

- Vaknr
- Vakoms
- Naam-ler

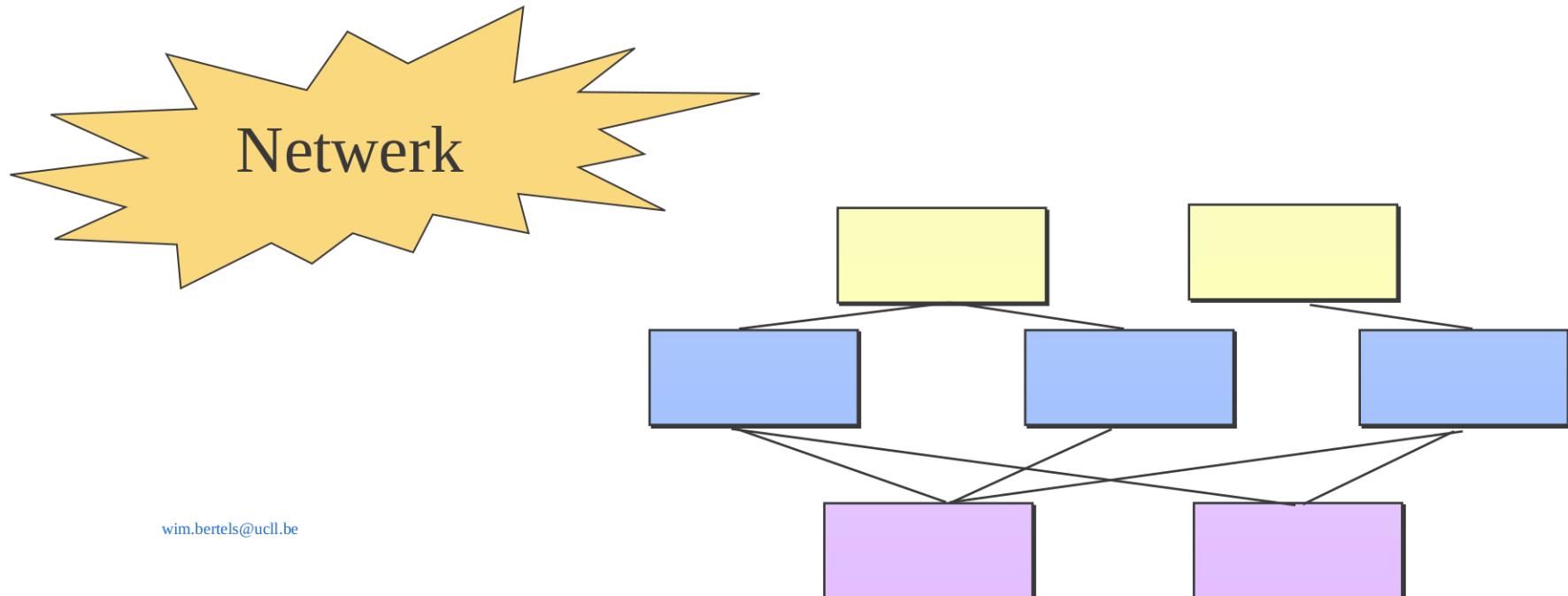
- HANDBOEK

- Boeknr
- Titel
- Auteur
- Uitg

- ◆ LEERLING

- Lnr
- Lnaam
- Ladres
- Straat
- Nummer
- Post
- Gemeente

# DBMS - Soorten



wim.bertels@ucll.be

Naamsvermelding-NietCommercieel-GelijkDelen  
4.0 Unported Licentie

+ products

+ communities & insights

+ services, support & education

+ partners

+ contact

Achieve superior, cost-effective database performance with maximum flexibility using CA's proven, Web-enabled, high-performance mainframe relational database management system. CA IDMS provides unparalleled business value for over a thousand organizations around the world.

## insights & documents



### Industry Analyst Report

Leveraging CA IDMS™ Business Value for Innovation

+ View Industry Analyst Report

## connect

### Communities

[CA Technologies Mainframe Communities](#)

### Blogs

[EXEC I/O Mainframe Blog](#)

### Webcasts

[Mainframe Webcasts](#)

- IDMS(2011)

# CA IDMS™/DB



## At a Glance

CA IDMS™/DB Release 18.5 is a proven, reliable, high-performance, web-enabled DBMS for IBM System z that provides outstanding business value for hundreds of enterprises and government organizations around the world. A powerful database engine and the core of the CA IDMS™ product family, CA IDMS/DB exploits the latest hardware and software technologies, including the IBM zIIP specialty processor.

### Key Benefits / Results

**Reliable, modernized, cost-effective DBMS platform.** CA IDMS/DB is designed to deliver rapid response time and 24/7 availability for critical applications and business services.

**Cost-effective performance and throughput.** CA IDMS/DB takes fewer system and human resources than other leading databases.

**Modernization flexibility.** CA IDMS/DB

### Business Challenges

#### Provide high-performance, continuously available systems

Today, organizations must improve customer service through continuously available high-performance systems while enabling critical data access to customers and business partners 24/7 through a variety of server-based, web-based and mobile applications and services.

# IDMS 18.5 (2013)

# CA IDMS™

Web-enabled, high-performance mainframe relational database management system.

[View eBook](#)

[View Data Sheet](#)

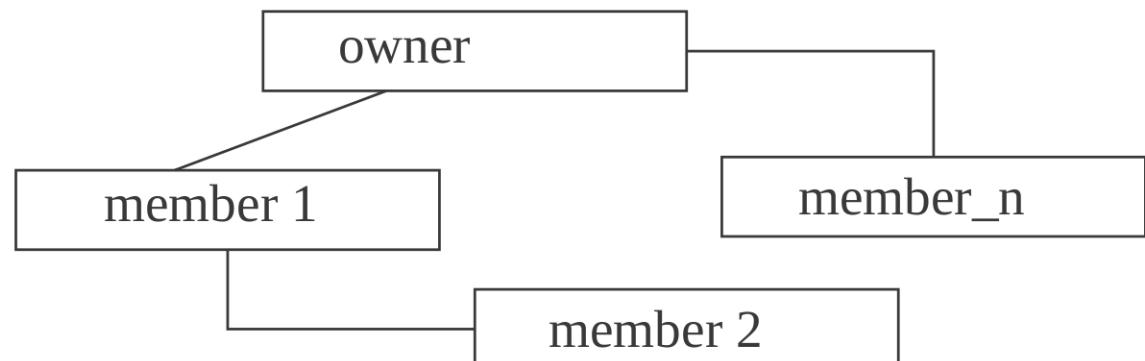
## **Web-enabled, high-performance mainframe relational database management system.**

Achieve superior, cost-effective database performance with maximum flexibility using CA's proven, Web-enabled, high-performance mainframe relational database management system. This legacy DBMS provides unparalleled business value for global organizations, including Fortune 1000 businesses spanning financial, healthcare, manufacturing and more.

## IBMS (2017)

# Bouwstenen

- Recordtypes => recordinstantiatie
- Settypes => setinstantiatie
- Record key
- 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

# Terminologie

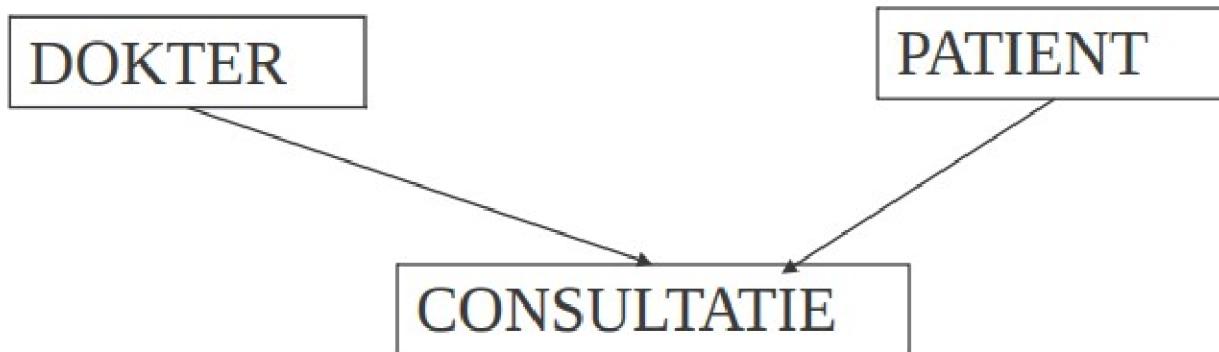
- Recordtype
- Settype
- Record key

# 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

# Gegevensbanktalen

- Gegevensdefinitietaal : DDL
- Gegevensmanipulatietaal : DML



# Vergelijking met relationele structuur

[wim.bertels@ucll.be](mailto:wim.bertels@ucll.be)

Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Unported  
Licentie

# Hiërarchische structuren in een relationeel model

- Kan dit?
- Conceptueel

# Vormen

- Rechtstreeks omzetting (cf IMS)
- Nested sets (niet kennen)
- Bomen
- (xml ea)
- ...

# Netwerk structuren in een relationeel model?

# Abstract : deelverzamelingen

Hierarchisch < Netwerk < Relationeel

# Terminologie

- Relatieel (databank) : tabellen, rijen, kolommen
- Netwerk (schema): recordtypes (+settypes), records, fields
- Hierarchisch (database description): segmenten, records, fields
  - \* Sequentieel



# Wim Bertels (CC) BY-SA-NC

- Links:
- <http://www.postgresql.org/docs/current/static/datatype-xml.html>
- <http://www.postgresql.org/docs/current/static/ltree.html>

# Subqueries 1

Niet-Gecorreleerde Subqueries

[wim.bertels@ucll.be](mailto:wim.bertels@ucll.be)

Naamsvermelding-NietCommercieel-GelijkDelen 4.0  
Unported Licentie

Hoe zou je de output van een SELECT statement beschrijven?

# Wat is een subquery?

- Een tabelexpressie binnen een tabelexpressie
- Resultaat wordt doorgegeven aan aanroepende tabelexpressie
- Subqueries mogen genest zijn

# Waarom gebruiken we subquery's?

- Query opsplitsen in deelproblemen die je kan oplossen en de output ervan verder gebruiken
- Zoals bij programmeren: een complexere methode opsplitsen in eenvoudigere (atomaire) taken

# Soorten subquery's

- Scalaire subquery: output = 1 rij, 1 kolom (dus 1 waarde)
- Rij-subquery: output = 1 rij
- Kolom-subquery: output = meerdere rijen met elk 1 waarde
- Tabel-subquery: output = meerdere rijen en kolommen

# Scalaire subquery (1 rij, 1 kolom)

Voorbeeld:

Geef voor elke planeet hoeveel groter of kleiner deze is dan de zon.

# Scalaire subquery (1 rij, 1 kolom)

Voorbeeld:

Geef voor elke planeet hoeveel groter of kleiner deze is dan de zon.

```
1 SELECT objectnaam, diameter - (SELECT diameter FROM hemelobjecten WHERE objectnaam = 'Zon') AS verschil  
2 FROM hemelobjecten  
3 WHERE satellietvan = 'Zon';  
4
```



The diagram shows a screenshot of a database query editor. At the top, there are tabs for 'Data Output', 'Explain', 'Messages', 'Notifications', and 'Query History'. Below the tabs, there is a table with one row of data. The table has two columns: the first column contains the number '1' and the second column contains the value '1393000'. A red arrow points from the text 'Output van subquery' to the second column of the table, indicating that the value '1393000' is the result of the scalar subquery '(SELECT diameter FROM hemelobjecten WHERE objectnaam = 'Zon')'.

	diameter numeric (7)
1	1393000

Output van  
subquery

# Scalaire subquery (1 rij, 1 kolom)

```
SELECT objectnaam, diameter -  
      (SELECT diameter  
       FROM hemelobjecten  
      WHERE objectnaam = 'Zon')  
           AS verschil  
  
FROM hemelobjecten  
WHERE satellietvan = 'Zon';
```

# Scalaire subquery (1 rij, 1 kolom)

Voorbeeld 2:

Geef de hemellichamen met een diameter groter dan Venus.

# Scalaire subquery (1 rij, 1 kolom)

Voorbeeld 2:

Geef de hemellichamen met een diameter groter dan Venus.

```
1 | SELECT objectnaam, diameter
2 | FROM hemelobjecten
3 | WHERE diameter >
4 | (SELECT diameter
5 | FROM hemelobjecten
6 | WHERE objectnaam = 'Venus');
7 |
```

Data Output		Explain	Messages	Notifications	Q
	diameter numeric(7)				
1	12104				

Output van  
subquery

# Scalaire subquery (1 rij, 1 kolom)

```
SELECT objectnaam, diameter  
FROM hemelobjecten  
WHERE diameter >  
    (SELECT diameter  
     FROM hemelobjecten  
     WHERE objectnaam = 'Venus');
```

# Rij-subquery (1 rij)

Voorbeeld:

Geef alle spelers met hetzelfde geslacht en dezelfde woonplaats als de speler met nummer 7.

# Rij-subquery (1 rij)

Voorbeeld:

Geef alle spelers met hetzelfde geslacht en dezelfde woonplaats als de speler met nummer 7.

```
1  SELECT spelersnr, naam, plaats
2  FROM spelers
3  WHERE (plaats, geslacht) =
4  (SELECT plaats, geslacht
5  FROM spelers
6  WHERE spelersnr = 7);
7
```

Output van  
Explain subquery

	plaats	geslacht
▼	character varying (30)	character (1)
1	Den Haag	M

# Rij-subquery (1 rij)

```
SELECT spelersnr, naam, plaats  
FROM spelers  
WHERE (plaats, geslacht) =  
      (SELECT plaats, geslacht  
       FROM spelers  
       WHERE spelersnr = 7);
```

# Kolom-subquery (meerdere rijen, elk 1 waarde)

Voorbeeld:  
Geef alle manen.

Tijd voor een micropauze

# Kolom-subquery (meerdere rijen, elk 1 waarde)

Voorbeeld:  
Geef alle manen.

```
1 SELECT objectnaam
2 FROM hemelobjecten
3 WHERE satellietvan IN
4 (SELECT objectnaam
5 FROM hemelobjecten
6 WHERE satellietvan = 'Zon');
```

Data Output		Explain	Messages	Notifications
	objectnaam character varying (10)			
1	Mercurius			
2	Venus			
3	Aarde			
4	Mars			
5	Jupiter			
6	Saturnus			
7	Uranus			
8	Neptunus			
9	Pluto			

Output van  
subquery

# Kolom-subquery (meerdere rijen, elk 1 waarde)

```
SELECT objectnaam
FROM hemelobjecten
WHERE satellietvan IN
  (SELECT objectnaam
   FROM hemelobjecten
   WHERE satellietvan = 'Zon');
```

# Tabel subquery (meerdere rijen en kolommen)

- Geeft een tijdelijk resultaat
- Subquery moet een pseudoniem krijgen als de subquery in de from staat van de originele query

# Tabel subquery (meerdere rijen en kolommen)

Voorbeeld:

Geef de reizen die een hemelobject bezoeken dat over alle reizen heen minstens 5 keer bezocht wordt.

# Tabel subquery (meerdere rijen en kolommen)

Voorbeeld:

Geef de reizen die een hemelobject bezoeken dat over alle reizen heen minstens 5 keer bezocht wordt.

Tussenstap:

SELECT objectnaam

FROM

bezoeken

GROUP BY objectnaam

HAVING COUNT(\*) >= 5;

# Tabel subquery (meerdere rijen en kolommen)

Voorbeeld:

Geef de reizen die een hemelobject bezoeken dat over alle reizen heen minstens 5 keer bezocht wordt.

```
1 SELECT reizen.reisnr, reizen.vertrekdatum
2 FROM reizen
3 INNER JOIN bezoeken b using (reisnr)
4 INNER JOIN (
5   SELECT objectnaam
6   FROM bezoeken
7   GROUP BY objectnaam
8   HAVING COUNT(*) >= 5
9 ) AS veelbez ON b.objectnaam = veelbez.objectnaam
10 GROUP BY reizen.reisnr, reizen.vertrekdatum;
```

Data Output		Explain	Messages	Notifications	Query History				
<table border="1"><thead><tr><th></th><th>objectnaam</th></tr></thead><tbody><tr><td>1</td><td>Maan</td></tr></tbody></table>			objectnaam	1	Maan	Output van subquery			
	objectnaam								
1	Maan								

# Tabel subquery (meerdere rijen en kolommen)

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

# Nog iets klein

Geef alle reizen die geen bezoek hebben gebracht aan de maan.

# Nog iets klein

Geef alle reizen die geen bezoek hebben gebracht aan de maan.

```
SELECT    reizen.reisnr  
FROM      reizen INNER JOIN bezoeken USING (reisnr)  
WHERE     bezoeken.objectnaam <> 'Maan'  
GROUP BY  reizen.reisnr;
```

?

# Nog iets klein

Geef alle reizen die geen bezoek hebben gebracht aan de maan.

```
SELECT reizen.reisnr  
FROM reizen INNER JOIN bezoeken USING (reisnr)  
WHERE bezoeken.objectnaam <> 'Maan'  
GROUP BY reizen.reisnr;
```

Waarom fout?

Check:  
Alle reizen

SELECT reisnr  
FROM reizen;

# Oplossing (tussenstap)

Alle reizen die WEL de maan hebben bezocht:

```
SELECT    reizen.reisnr
FROM      reizen INNER JOIN bezoeken USING (reisnr)
WHERE     bezoeken.objectnaam = 'Maan'
GROUP BY  reizen.reisnr;
```

# Oplossing

Alle reizen die GEEN bezoek hebben gebracht aan de maan.

```
SELECT    reisnr
FROM      reizen
WHERE     reisnr NOT IN
          (SELECT  reisnr
           FROM    bezoeken
           WHERE   objectnaam = 'Maan');
```

# Uitdaging

Probeer deze zonder subquery te schrijven (op een regenachtige dag.., als je echt teveel tijd hebt..):

```
SELECT  avg(totaal)
FROM
  (SELECT  spelersnr, sum(bedrag) as total
   FROM    boetes
   GROUP BY spelersnr) as totalen
```

Wat toont deze query?

Wim Bertels (CC)BY-SA-NC

Referenties:

Slides subqueries deel 1 sql 2012-13, K. Beheydt

SQL Leerboek, R. Van der Lans