- **Uitnormaliseren**: redundantie vermijden & consistentie bewaken
- **Kandidaatsleutels:** minimaal & uniek (constraints, integriteit), primary key verplicht (not null), unique niet verplicht
- **Vreemde sleutels (foreign keys)** : noodzakelijke, minimale vorm van redundantie, constraints, referentiële integriteit, mag null zijn, kan verwijzen naar PK of UN
- **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

```
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)
)
```

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

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

```
DELETE
FROM    hemelobjecten
WHERE objectnaam = 'Pluto'
```

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

```
GRANT USAGE
ON SCHEMA bezoeken
TO public
```

How would you describe the output of a select statement : Tabelexpressie binnen een tabelexpressie, resultaat wordt doorgegeven aan aanroepende tabelexpressie, subqueries mogen genest zijn

Soorten subqueries :

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

Scalaire bv. Geef voor elke planeet hoeveel groter of kleiner die is dan de zon.

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

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

Kolom-subquery bv. Geef alle manen

Rij-subquery bv. Geef alle spelers met hetzelfde geslacht en woonplaats als speler met nummer 7

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

! ALLEMAAL NIET GECORRELEERD !

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

Tabel-subquery : FROM-component bevat subquery, tijdelijk resultaat, geen ORDER BY, subq. moet pseudoniem hebben

```
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;
```

Gecorreleerde subquery : 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.
Hoofdquery: krijgt alles wat in SELECT van subquery staat, weet niets van detail subquery, krijgt alleen output
Subquery: weet alles van hoofdquery
Bv. Geef voor iedere reis het bezoek met de langste verblijfsduur

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

```
SELECT bezoeken.reisnr, bezoeken.objectnaam
FROM bezoeken
WHERE bezoeken.verblijfsduur = (
        SELECT MAX(verblijfsduur)
        FROM bezoeken allebezoeken
        WHERE allebezoeken.reisnr = bezoeken.reisnr);
```

```
SELECT spelersnr
FROM bestuursleden b
GROUP BY spelersnr
HAVING COUNT(*) > (
        SELECT COUNT(*)
        FROM wedstrijden w
        WHERE w.spelersnr = b.spelersnr);
```

Operator EXISTS, bv. Geef alle reizen met een bezoek aan Jupiter

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

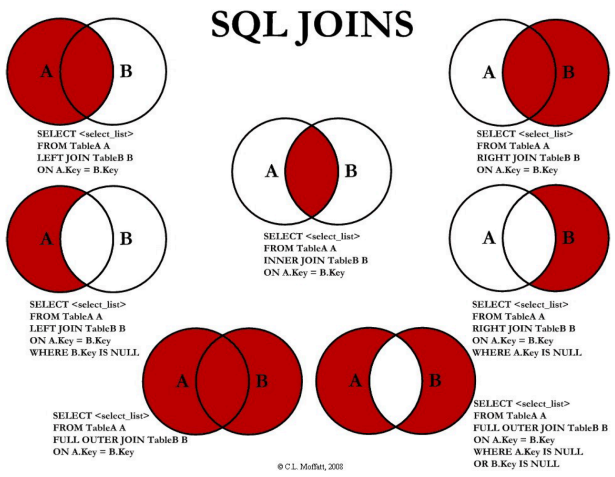
```
SELECT reisnr, vertrekdatum
FROM reizen
WHERE EXISTS (
        SELECT *, 'iserietofnie' OF SELECT reisnr
        FROM bezoeken
        WHERE bezoeken.objectnaam = 'Jupiter'
        AND bezoeken.reisnr = reizen.reisnr);
```

Operator ANY en ALL : extra operator, verwacht scalaire waarde en kolomexpressie
Vergelijken met null = altijd false
> ALL >= ALL
> ANY >= ANY
< ALL < ANY
Bv. Geef de langste reis

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

Operator OVERLAPS
Bv. Geef de spelers en hun functie die in het bestuur zaten van1 januari 1991 tot en met 31 december 1993.

FROM : bevat tabelspecificaties, kunnen subqueries zijn (alias niet vergeten), impliciete joins > expliciete joins
USING: veronderstelt gelijke kolomnamen, alle rijen uit beide tabellen worden getoond, verschil met ON : er blijft maar 1 rij met spelersnr over.
ON : 2 rijen met spelersnr (uit beide tabellen)
Als je al in FROM 'AND …" doet gaat hij daar al checken op de conditie
CROSS JOIN : expliciet cartesisch product
UNION JOIN : elke rij van elke tabel wordt 1 maal opgenomen en aangevuld met null-waardes voor de kolommen uit de andere tabel
NATURAL JOIN : lexicografisch, zelfde als inner join maar geeft de dubbele kolommen niet terug, werkt ook voor andere joins
EQUI-JOIN : vergelijking met "=" **THETA-JOIN** : vergelijking met andere operator



SET OPERATOREN : combineren van resultaten van individuele SELECT-instructies
Mogelijkheden : UNION (ALL), INTERSECT (ALL), EXCEPT (ALL)

UNION : elke rij die in één van de twee select-blokken of in beide voorkomt (dubbele rijen worden verwijderd)
- 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, sorteert het eindresultaat
- SELECT mag geen DISTINCT bevatten (want dubbele rijen worden automatisch al verwijderd)

INTERSECT : rijen die in de resultaten van beide selectblokken voorkomen (de doorsnede), dubbele rijen verwijderd
EXCEPT : rijen die in het resultaat van het 1e maar niet in dat van de 2e voorkomen (dubbele rijen worden verwijderd)

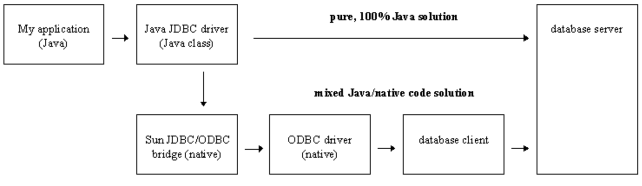
ALL = behoud dubbels, standaard worden dubbele rijen verwijderd, door gebruik ALL worden deze behouden
Rijen met null-waarde worden als gelijk beschouwd voor set-operatoren.
Set-operatoren kunnen gecombineerd worden, door haakjes te gebruiken kunnen we de volgorde aanpassen.
SCHEMA = namespace, binnen een namespace moeten alle objecten een unieke naam hebben
PostgreSQL namespace default public, checken door "SHOW search_path;"
Privileges (standard) : rechten op parentobject nodig om rechten op child te kunnen uitvoeren.
Default : alleen eigenaar heeft toegangsrechten
Rechten toekennen : GRANT <privilege> ON <objecttype> <objectname> TO <role>;
Privileges (passable) : rechten geven om door te geven :
GRANT <privilege> ON <objecttype> <objectname> TO <role> WITH GRANT OPTION;
Localisatie : - character sets: ç or not, Й or not
- collating sequences : abc …
- encoding : UTF-8, LATIN1, …

• LC_COLLATE : string order
• LC_CTYPE : character classification
• LC_MESSAGES : language of the messages
• LC_MONETARY : format currency
• LC_NUMERIC : format numbers
• LC_TIME : time format
SHOW LC_COLLATE;

```
FROM
ON
JOIN
WHERE
GROUP BY
WITH CUBE or WITH
ROLLUP
HAVING
SELECT
DISTINCT
ORDER BY
TOP
```

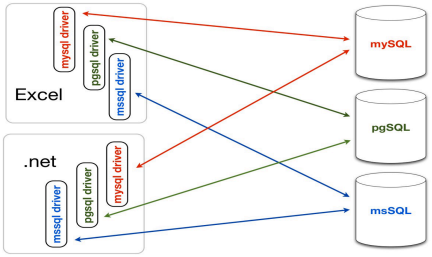
Alter table : Wijzigen van de tabelstructuur
Bv. ALTER TABLE spelers CONVERT TO character set utf8 COLLATE utf8_general_ci
ADD : voegt kolom toe
DROP : verwijdert een kolom
ALTER : verandert kolomeigenschappen
ADD CONSTRAINT : voegt integriteitsregel toe
DROP CONSTRAINT : verwijdert integriteitsregel
Bv. ALTER TABLE wedstrijden ADD CONSTRAINT FK2 FOREIGN KEY (spelersnr) REFERENCES spelers (spelersnr)
ALTER DATABASE : wijzigen van databases
CREATE USER : creëert een user Bv. CREATE USER Frank IDENTIFIED BY Frank_pw
ALTER USER : wijzigt paswoord Bv. ALTER USER Frank IDENTIFIED BY Frank_pasw
DROP USER : verwijdert user Bv. DROP USER Frank

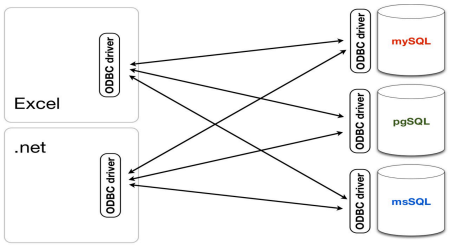
Pattern matching :
LIKE-operator (SQL Standaard)
SIMILAR TO-operator (SQL Standaard sinds 1999)
Regexen, POSIX-vorm, geen SQL Standaard, net als ILIKE
ESCAPE ' ' : om letterlijk _ en % aan te geven in een regex
SIMILAR TO : LIKE maar dan met echte regexen
JDBC : Java Database Connectivity



Connection pooling : cache van database connections
onderhouden door database zodat connections hergebruikt kunnen worden bij toekomstige requests (**client side pooling** ook mogelijk)

+ : boost in startup performance (iedere verbinding heeft een startup cost)
- : verbindingen gebruiken (een beetje) geheugen wanneer niet gebruikt
ODBC : Open Database Connectivity = manier om op een gestandaardiseerde manier toegang te geven aan een database, onafhankelijk van het type applicatie en van het type database.
Ieder programma en iedere DBMS heeft een ODBC-driver die met elkaar communiceren i.p.v. allemaal aparte drivers.





Verschil JDBC :

1.ODBC is for Microsoft and JDBC is for java applications.
2.ODBC can't be directly used with Java because it uses a C interface.
3.ODBC makes use of pointers which have been removed totally from java.
4.ODBC mixes simple and advanced features together and has complex options for simple queries,

But JDBC is designed to keep things simple while allowing advanced capabilities when required.
5.ODBC requires manual installation of the ODBC driver manager and driver on all client machines.JDBC drivers are written in java and JDBC code is automatically installable,secure,and portable on all platforms.
6. JDBC API is a natural Java Interface and is built on ODBC. JDBC retains some of the basic feature of ODBC

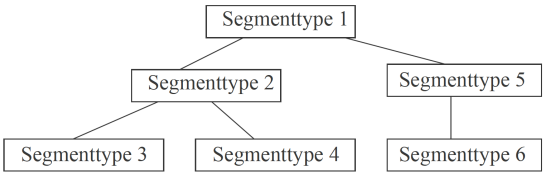
ID's : Serial of Auto-increment
Serial : geen echt datatype, willekeur!
SERIAL is an alias for BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE.
SERIAL DEFAULT VALUE in the definition of an integer column is an alias for NOT NULL AUTO_INCREMENT UNIQUE.
Serial creëert extra object , sequence, gebruiker moet rechten hebben tot tabel EN sequence.

```
GRANT { { USAGE | SELECT | UPDATE }
        [, ...] | ALL [ PRIVILEGES ] }
        ON { SEQUENCE sequence_name [, ...]
                | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
        TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

Auto-increment : geeft automatisch waarde van id mee en verhoogt die per nieuw object. (int datatype)
DBMS : Database Management System

<u>Hiërarchisch</u> :
Ontstaan '60, enorm populair geweest. IMS (van IBM) = bekendste vb. Verliest momenteel aan belang
Bouwstenen :

- Segmenttypes
- Parent-child relationship types
- Wortelsegment – bladeren
- N-m verbanden niet toegelaten
- Veel-op-veel relaties omzetten naar één-op-veel.
- 1 segmenttype als parent en 1 als child kiezen.
- Child MOET parent hebben en maximum 1.
- Als parent wegvalt, vallen alle children weg



Beperkingen :

- 15 niveaus diep
- 255 verschillende segmenttypes
- slechts 1 wortel

Parent-segment : The segment on which it is dependent
Child-segment : A segment that is hierarchically dependent on a segment one level up in the hierarchy
Twin-segment : Multiple segment occurrences of one segment type with the same parent occurrence
Segmentcode :
Volgorde-veld (sequence field) : is a field that identifies and provides access to segments in a database
3 schema-architectuur :

- Conceptueel niveau : DBD (Database Description)
- Intern niveau : bestand
- Extern niveau : logische databank PCB (meerder PCB's : PSB)

Gegevensdefinitietaal : DDL (Data Definition Language)
Gegevensmanipulatietaal : DML (Data Manipulation Language)

DBD :
Bv.

```
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
```

Macrobevelen :

- DBD-macro
- SEGM-macro
- FIELD-macro

PCB :

Bv. PCB TYPE=DB, DBDNAME=ZIEKADM, KEYLEN=19
SENSEG NAME = DOKTER, PROCOPT=K
SENSEG NAME = PATIENT, PROCOPT=G
SENSEG NAME = CONSULT, PROCOPT=G

Macrobevelen :

- PCB-macro
- SENSEG-macro
- PROCOPT : G-I-R-D-A-K
- SENFLD-macro
-

DML :

Koppelen met de **gasttaal** :

- **JCL**
- **PCB-masker**
- 4 soorten parameters (CBLTDLI) :
  - Functiecode → WSS
    - GU : overloopt boom en stopt bij eerst gevonden segment
      Bv. GU DOKTER(DFNAAM = DEPRET)
      PATIENT
    - GN : zoekt verder en stopt bij eerst gevonden segment
      Bv. GN PATIENT
    - GNP : zoekt verder maar alleen in de afhankelijke segmenten van een parent
      Bv. GU DOKTER (DNAAM = DEPRET)
      IF STATUS_CODE = SPACE
      DISPLAY I_O_DOKTER
      GNP PATIENT
      PERFORM UNTIL STATUS_CODE NOT = SPACE
      DISPLAY I_O_PATIENT
      GNP PATIENT
      END-PERFORM
      END-IF
    - GHU – GHN – GHNP : analoog maar H moet voor een delete of replace
      Bv. GHU DOKTER (DNR = 12)
      DLET

  DML Functiecode : ISRT (toevoegen segment), DLET (verwijderen segment + alle afhankelijke), REPL (aanpassen segment)

  ISRT Bv. MOVE 112 TO CNR
  MOVE 25 TO CPRIJS
  MOVE 16092005 TO CDATUM
  MOVE 'STD' TO CTYPE
  MOVE 'GRIEP MET ZWARE BRONCHITIS' TO CBESCHR
  ISRT DOKTER(DFNAAM = DEPRET)
  PATIENT (PFNAAM = MAES)
  CONSULT

  DLET Bv. GHU DOKTER (DNR = 12)
  DLET

  REPL Bv. GHU DOKTER (DNR = 12)
  MOVE 'GEMEENTESTRAAT' TO DSTRAAT
  MOVE '15A' TO DHUISNR
  REPL

  - PCB-masker → Linkage section

```
Bv. 01 ZIEK_PCB.
    03 DBD_NAAM          PIC X(8).
    03 SEG_LEVEL         PIC XX.
    03 STATUS_CODE       PIC XX.
    03 PROC_OPT          PIC X(4).
    03                   PIC XXXX.
    03 SEG_NAAM_FB       PIC X(8).
    03 NSENSEG           PIC S9(5) BINARY.
    03 LENGTE_FB_KEY     PIC S9(5) BINARY.
```

  - Input-output-area → WSS
  - Zoekargumenten (optioneel) → WSS
    Segmentnaam [(conditie)]
    Veldnaam operatiecode waarde

```
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

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

<u>Netwerk :</u>

Bouwstenen :

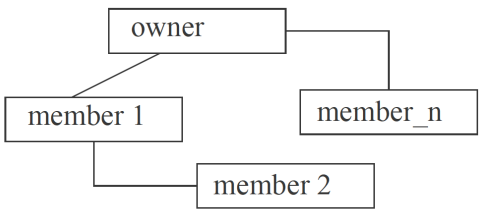
- Recordtypes → recordinstantiatie
- Settypes → setinstantiatie
- 1:n verband (owner recordtype, member recordtype)

ERM naar netwerkmodel :

Veel-op-veel-relatie omzetten naar één-op-veel-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

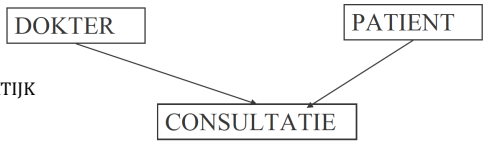


Verschil netwerk – hiërarchisch model :
Bij netwerkgegevensbanksystemen :

- Kan een recordtype member zijn in meerder settypes
- Kunnen meerdere settypes bestaan tussen dezelfde recordtypes
- Kunnen members bestaan zonder owners

3-schema-architectuur :

- Conceptueel niveau : schema (schema DDL)
  - Naamgeving
    - Schema MOET een naam krijgen
      Bv. SCHEMA DOKTERSPRAKTIJK
  - Beschrijving recordtypes
    - Recordtype MOET een naam krijgen
      Bv. RECORD Dokter
    - Per recordtype wordt record key en velden gespecifieerd
      Bv. Key dnr duplicates are not allowed
      02 dnr PIC X(3).
      02 vnaam PIC X(30).
      02 naam PIC X(30).
      02 straat PIC X(30).
      02 huisnr PIC X(8).
      02 postkd PIC X(8).
      02 wplaats PIC X(30).
  - Beschrijving settypes :
    - Naamgeving
      Settype MOET een naam krijgen
      Bv. SET dokter-consultatie
    - Specificatie owner – member
      Bv. OWNER IS dokter
    - Set order
      Bv. ORDER FOR INSERTION IS LAST (FIRST, NEXT, PRIOR)
    - Set insertion optie
      Bv. INSERTION IS AUTOMATIC (MANUAL)
    - Set retention optie (retention geeft aan of de memberrecords binnen set losgekoppeld mogen worden)
      Bv. RETENTION IS FIXED (MANDATORY, OPTIONAL)
    - Set selectie clausule
      Bv. SELECTION IS THRU dokter-consultatie KEY Dnr



- Intern niveau : intern schema (DSDL)
- Extern niveau : subschema (subschema DDL)
  Velden en types van één bepaalde applicatie
  Syntax : SS subschema-naam WITHIN schema-naam
  Hernoemen van velden : AD (ALIAS DEFINITIONS)
  Relevante record- en settypes :
  RECORD SECTION
  RECORD NAME IS recordnaam
  SET SECTION
  SET NAME IS setnaam

```
SS Patienten WITHIN
Dokterspraktijk
RECORD SECTION
RECORD Patient
Elements are pnr vnaam naam

RECORD Consultatie
Elements are cnr datum

SET SECTION
SET Patient-Consultatie.
```

Gegevensdefinitietaal : DDL
Gegevensmanipulatietaal : DML
User work area (UWA) : hier worden de gegevens afkomstig van database geplaatst
Run-unit : iedere uitvoering van het programma, per run unit 1 UWA
DB-status : geeft aan of de vorige DML-instructie succesvol is afgelopen
Currency indicatoren :

- CRU (Current of run unit) bevat verwijzing naar laatst gerefereerde record, 1 CRU per run unit
- CRP (Current of record pointer), verwijzing naar laatst gerefereerde record van het recordtype, 1 CRP per recordtype
- CSP (Current of set pointer), verwijz. naar l.g. record van owner/member recordtype van betreffende settype. 1 per settype

Subschema moet geassocieerd worden met applicatie : DB SUBSCHEMA subschema-naam WITHIN schema-naam
DML-instructies :

- FIND- en GET, Find positioneert currency indicatoren.
  Kopiëren inhoud naar CRU : FIND CURRENT record-naam WITHIN set-naam; Get haalt informatie op.
- Wijzigen van velden : MODIFY
- Weglaten van recordinstantiatie : ERASE / ERASE ALL
- Toevoegen van records : STORE record-naam
- RECONNECT-instructie : RECONNECT record-naam WITHIN set-naam
- Associëren van zwevende records (=record van het memberrecordtype dat op het moment niet opgenomen is in een setinstantiatie van het beschouwd settype) : CONNECT recordnaam TO setnaam
- Loskoppelen van members : DISCONNECT recordnaam FROM setnaam
- RETAINING-optie (sommige currency indicatoren niet wijzigen, uitzondering CRU) :
  RETAINING RECORD CURRENCY … setnaam

Zoek de patiënt nummer 112 op, controleer of het handelt over Hedwich Martens. Zo ja, geef de gegevens van de eerste consultatie (consultatie + behandelende dokter). Geef de juiste foutboodschappen als er iets verkeerd is.

```
move 112 to pnr
find any patient using pnr
If DB-status = 0
then get patient
if naam = « Martens » and vnaam = « Hedwich »
then find first consultatie within patient-consultatie
if DB-status = 0
then get consultatie
display consultatie
find owner within dokter-consultatie
if DB-status = 0
then get dokter
display dokter
else display « geen dokter gevonden »
else display « geen consultaties gevonden »
else display « patient 112 is niet Hedwich Martens »
else display « patient 112 bestaat niet »
```

```
move 4 to rnr
find any reis using rnr
If DB-status = 0
then get reis
if naambeg = « Martens Hedwich »
then find first deelname within organiseert
if DB-status = 0
then get deelname
display deelname
find toerist within neemtDeelAan
get toerist
display toerist
else display « geen deelnames gevonden »
else display «Begeleider van reis 4 is niet Hedwich Martens »
else display « reis met nummer 4 bestaat niet »
```

Hierarchische structuren in een relationeel model : kan, conceptueel, vormen :

- Rechtstreekse omzetting (IMS)
- Recursief (CTE)
- Nested sets
- Bomen
- Xml

Netwerkstructuren in rel. model : abstract, deelverzamelingen hierar. C netwerk C relationeel
Relationeel (database) : tabellen, rijen, kolommen
Netwerk (schema) : record- en settypes, records, fields
Hierarchisch (database description) : segmenten, records, fields
CTE's = Common Table Expressions : vergelijkbaar met subqueries in FROM, maar met extra's

```
Kinderen van vader :
SELECT *
FROM familieboom
WHERE vader='vader';
```

```
Kleinkinderen van vader :
SELECT *
FROM familieboom
WHERE vader = 'vader'
OR vader IN
(SELECT bijnaam
FROM familieboom
WHERE vader = 'vader');
```

```
Papa & mama database :
CREATE TABLE familieboom(
bijnaam varchar(16) NOT NULL,
vader varchar(16),
moeder varchar(16),
CONSTRAINT familieboom_pk
PRIMARY KEY (bijnaam),
CONSTRAINT vader_fk
FOREIGN KEY (vader)
REFERENCES familieboom(bijnaam),
CONSTRAINT moeder_fk
FOREIGN KEY (moeder)
REFERENCES familieboom(bijnaam)
);
```

```
Alle getallen van 0 tot 100 optellen :
WITH RECURSIVE t(n) AS (
VALUES (1)
UNION ALL
SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

```
INSERT INTO familieboom VALUES ('bofh','bofh','bofh');
INSERT INTO familieboom VALUES ('moeder','bofh','bofh');
INSERT INTO familieboom VALUES ('vader','bofh','bofh');
INSERT INTO familieboom VALUES ('kindje','vader','moeder');
INSERT INTO familieboom VALUES ('nog_een_kindje','vader','moeder');
INSERT INTO familieboom VALUES ('lelijk_nestje','kindje','nog_een_kindje');
INSERT INTO familieboom VALUES ('lelijker_nestje','kindje','nog_een_kindje');
INSERT INTO familieboom VALUES ('markske','vader','kindje');
INSERT INTO familieboom VALUES ('prutske','lelijk_nestje','lelijker_nestje');
```

```
Tussenresultaten van CTE wegschrijven :
CREATE TEMPORARY TABLE debug_table
(id serial, t text, r text);

WITH lengte_namen AS (
SELECT naam, length(naam) AS n
FROM spelers),
debug_lengte_namen AS (
INSERT INTO debug_table(t,r)
SELECT 'lengte_namen', ROW(l.*)::text
FROM lengte_namen l),
evenlange_namen AS (
SELECT l1.naam AS naam1, l2.naam AS naam2
FROM lengte_namen l1 JOIN lengte_namen l2
ON l1.n = l2.n
WHERE l1.naam <> l2.naam),
overzichts_lijst AS (
SELECT naam1, array_agg(naam2) AS lijst
FROM evenlange_namen
GROUP BY naam1)
SELECT l.naam, o.lijst
FROM lengte_namen l LEFT JOIN overzichts_lijst o
ON l.naam = o.naam1;
```

OFFSET start { ROW | ROWS }
FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY
Voor bv. de 3 jongste spelers, offset voor de 3e tot de 5e jongste spelers,…

Datatypes tekst:

- Char(n) : vaste lengte, plaats gereserveerd, ook indien niet gebruikt
- Varchar(n) : variabele lengte, flexibel met maximum, trager
- Text : onbeperkte variabele lengte, meest flexibel, traagst

'string' ← → "identifier" , verschil aanhalingstekens (sql : standard uppercase)
ROLLUP : verschillende aggregatieniveaus in 1 instructie (dus bv totaalbedrag voor 1 speler en als laatste rij totaalbedrag van ALLE spelers)
CUBE : meerdere groeperingen in 1 instr. , bv. op geslacht+plaats, geslacht, plaats
Verschil : cube bevat ieder mogelijke combinatie van waarden voor elke kolom maar rollup houdt de hiërarchie intact.

Cube: it's an additional switch to GROUP BY clause. It can be applied to all aggregation functions to return cross tabular result sets.
Rollup: It's an extension to GROUP BY clause. It's used to extract statistical and summarized information from result sets. It creates groupings and then applies aggregation functions on them.

Indexen : doel = beïnvloeden van verwerkingstijd. Rijen worden in bestanden opgeslagen, bestand bestaat uit pagina's.
Als een rij opgehaald wordt, wordt de betreffende pagina opgehaald en de betreffende rij.
2 methodes voor opzoeken :

- **Sequentiële zoekmethode :** rij per rij (tijdrovend, inefficient)
- **Geïndexeerde zoekmethode** : index (B-tree), boom, knooppunten, leafpage, 2 methodes:
  - Zoeken van rijen met bepaalde warden
  - Doorlopen van hele tabel via een gesorteerde kolom (geclusterde index)

Bij aanpassing in tabel : index aangepast
Index ook op niet unieke kolom
Op één tabel : meerdere indexen
Op één tabel : één geclusterde index
Samengestelde index
Index neemt opslagruimte, als index vol → reorganisatie van de index
Meerdere indexvormen mogelijk
Optimiser : zoekt de beste strategie (verwachte verwerkingstijd, aantal rijen, indexen)

```
CREATE INDEX spelers_postcode_idx
ON spelers (postcode asc);

CREATE UNIQUE INDEX spelers_naam_vl_idx ON
spelers (naam, voorletters);

CREATE INDEX spelers_naam_vl_partial_idx ON
spelers (naam, voorletters) WHERE spelersnr < 100;

CLUSTER spelers USING speler_naam_vl_idx;
```

Bij vele SQL-producten : automatische creatie index op primaire en alternatieve sleutels bij creatie tabel; naam afgeleid uit naam van tabel en betreffende kolommen

\+ : index versnelt verwerking
\- : opslagruimte, elke mutatie vraagt aanpassing index (vertraagt verwerking)

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

Welke kolommen?

- Unieke index op kandidaatsleutels
- Index op refererende sleutels
- Index op kolommen waarop geselecteerd wordt (grootte tabel, cardinaliteit, distributie
- Index op combinatie van kolommen
- Index op kolommen waarop gesorteerd wordt

```
CREATE
ALTER : ALTER INDEX groot_idx SET TABLESPACE ergens_anders;
DROP
```

Speciale tabellen :
- **Multi-tabelindex** = index op kolommen in meerdere tabellen
- **Virtuele kolomindex** = index op een expressie
- **Selectieve index** = index op gedeelte van de rijen
- **Hash-index** = index op basis van het adres op de pagina
- **Bitmapindex** = interessant bij voorkomen van veel dubbele waarden

Catalog table : pg_index

Optimaliseren van instructies : optimiser, indexen, inzicht → soms niet optimaal → herformulering om tot efficiënte strategie te komen.
- Vermijd OR operator (index meestal niet gebruikt) , alternatief : vervangen door conditie met IN of 2 selects met UNION
- Onnodig gebruik UNION (dezelfde tabel meermaals doorlopen), alternatief : herformuleren (alle voorwaarden in 1 SELECT)
- Vermijd NOT operator, alternatief : vervangen door vergelijkingsoperatoren
- Isoleer kolommen in condities (index wordt niet gebruikt), alternatief : bv. where jaartoe + 10 = 1990 → where jaartoe = 1980
- Gebruik BETWEEN operator (AND gebruikt index meestal niet), alternatief : BETWEEN
- Bepaalde vormen van LIKE (index niet gebruikt als patroon begint met % of _ , alternatief : geen tenzij … ???
- Redundante condities bij joins (om SQL te verplichten bepaald pad te kiezen)
- Vermijd HAVING, alternatief : zoveel mogelijk in de WHERE
- Hou SELECT zo smal mogelijk (onnodige kolommen weg, gecorreleerde subquery met EXISTS : 1 expressie met 1 constante) bv. SELECT spelersnr, naam FROM spelers WHERE EXISTS (SELECT '1' FROM boetes WHERE b.spelersnr = s.spelersnr)
- Vermijd DISTINCT (verlengt verwerkingstijd), alternatief : vermijden wanneer overbodig
- Onnodig gebruik ALL (verlengt verwerkingstijd), alternatief : vermijden wanneer overbodig
- Kies OUTER JOINS boven UNION (verlengt verwerkingstijd), alternatief : OUTER JOIN is beter
- Vermijd datatype conversies (verlengt verwerkingstijd)
- Grootste tabel als laatste (volgorde kan van belang zijn), alternatief : grootste tabel als laatste plaatsen
- Vermijd ANY en ALL (index wordt niet gebruikt), alternatief : vervang door MIN() of MAX()

Checken door **EXPLAIN of EXPLAIN ANALYZE**

• EXPLAIN (afhandelijk van de statistieken)
Oplossing: ANALYZE voordien
• EXPLAIN
Werkt ook voor ander DML (eg INSERT..)
EXPLAIN ANALYZE (later transacties…)

• EXPLAIN ANALYZE
Actuele Uitvoertijd, maar voert dus ook effectief uit!
• EXPLAIN
Kijk naar de totale cost, kijk eventueel naar de Scan methoden

View : tabel die zichtbaar is voor de gebruiker maar geen opslagruimte inneemt. Inhoud van een view wordt afgeleid bij het gebruik van een SELECT view, een view kan dus maar opgebouwd worden op basis van gegevens die in andere tabellen opgeslagen zit.
Nut = vereenvoudigen routinematige instructies, reorganiseren tabellen, stapsgewijs opzetten SELECT instructies, beveiligen gegevens.

Basistabellen ←→ afgeleide tabellen (views)

View bevat geen rijen. Voorschrift/formule om gegevens uit basistabellen in virtuele tabel te steken.
CREATE VIEW maakt een view. Raadplegen en muteren! Synoniemen en commentaar!
SELECT definieert de kolomnamen, maar expliciete definitie is ook mogelijk :
expliciete definitie verplicht als kolom bestaat uit functie/berekening.

```
CREATE VIEW leeftijden (spelersnr,
leeftijd) AS
SELECT spelersnr,
2008 – year(geb_datum)
FROM spelers;

CREATE VIEW leeftijden AS
SELECT spelersnr,
2008 – year(geb_datum) AS leeftijd
FROM spelers;
```

Muteren van views → muteren van tabellen.

WITH CHECK OPTION controleert update (aangepaste rijen behoren nog tot view), insert (nieuwe rij behoort tot view), delete (verwijderde rij behoort tot view)
DROP VIEW verwijdert view en hierop gedefinieerde views.
SELECT, INSERT, UPDATE, DELETE bij views, mutatie mag enkel als :
- View moet indirect/direct gebaseerd zijn op één of meerdere basistabellen
- SELECT mag geen DISTINCT bevatten
- SELECT mag geen aggregatiefunctie bevatten
- FROM mag slechts één tabel bevatten
- SELECT mag geen GROUP BY bevatten
- SELECT mag geen ORDER BY bevatten
- SELECT mag geen SET operatoren bevatten
- (UPDATE) Virtuele kolom mag niet gewijzigd worden
- (UPDATE) In SELECT moeten alle NOT NULL-kolommen staan

```
Create view Leuvenaars (snr, naam, geboorte) as
select spelersnr, naam, geb_datum
from spelers
where plaats = 'Leuven'
```

Toepassingen van views :
- Vereenvoudigen van routinematige instructies (vaak gebruikte instructies)
- Reorganisatie van tabellen (laten bestaan bij aanpassingen "oude" programma's
- Stapsgewijs opzetten van SELECT instructies (bij complexe queries stukken "voorprogrammeren")
- Specificeren van integriteitsregels (WITH CHECK OPTION : toegestane waarden controleren)
- Gegevensbeveiliging (Beveiliging van delen van tabellen)

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

CREATE USER, ALTER USER, DROP USER

Tabel- en kolombevoegdheden : GRANT kent bevoegdheden toe, soorten bevoegdheden :
- SELECT : bevoegdheid tot SELECT en view
- INSERT : rijen toevoegen
- DELETE : rijen verwijderen
- UPDATE : rijen wijzigen
- REFERENCES : refererend sleutels naar deze tabel creëren
- ALTER : tabel veranderen
- INDEX : indexen op de tabel creëren
- ALL-ALL privileges : alle bevoegdheden

Databasebevoegdheden :
- SELECT : bevoegdheid tot SELECT en view
- INSERT : rijen toevoegen
- DELETE : rijen verwijderen
- UPDATE : rijen wijzigen
- REFERENCES : refererend sleutels naar alle tabellen van de database creëren
- CREATE : nieuwe tabellen in database aanmaken
- ALTER : tabellen veranderen
- DROP : tabellen verwijderen
- INDEX : indexen op deze 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

Gebruikersbevoegdheden : databasebevoegdheden toekennen aan gebruiker (CREATE USER)
WITH GRANT OPTION : de gebruikers die toegang krijgen, kunnen deze machtiging doorgeven aan andere gebruikers
CREATE ROLE : creëert een rol met bevoegdheden voor een aantal users. Als rol verandert, veranderen de bevoegdheden van deze users
DROP ROLE : verwijdert role
REVOKE : verwijdert bevoegdheid (en afhankelijke bevoegdheden)

```
Bv. GRANT ALL
ON spelers
TO Frank
WITH GRANT OPTION
```

```
Bv. CREATE ROLE admin
GRANT SELECT, INSERT
ON spelers
TO admin
GRANT admin TO Frank, Marc, Ann, Greet
```

```
Bv. REVOKE ALL
ON spelers
FROM Frank
```

```
REVOKE admin
FROM Marc
```

```
REVOKE SELECT
ON spelers
FROM admin
```

Beveiliging van en met views : analoog voor bevoegdheden op views, kan gebruikt worden voor beveiliging van gegevens (gebruiker krijgt enkel machtiging op een view, waarin een deel van de tabel gedefinieerd wordt)

```
Bv. CREATE USER …                          GRANT SELECT
CREATE VIEW zichtbaar_deel AS …            ON zichtbaar_deel TO …
```

- Security : hardware, platform waar het op draait, databank software, binnen SQL zelf (grant, revoke, view,…), SQL als 'vertaler', algemeen terugkerende security problematiek (bv. authenticatie maar ook backups,…)
- Ondersteunend platform up to date houden, beveiligen, (D)DoS attacks, …
- Databank software up to date houden, configuratie (Local, Remote), known exploits (wat doe je ertegen?)
- Binnen SQL : user management (roles), privileges (grant/revoke), view, stored procedures
- SQL : wordt niet gecompileerd, wordt "vertaald" in de onderliggende/omsluitende laag. Kan gebruikt worden om SQL ongewenste instructies te laten uitvoeren. = probleem dat bij elke taal die "vertaald" wordt voorkomt.
- **SQL Injection** : proberen raden wat voor SQL code achter formulier zit, ipv gewone waarden SQL code meegeven met formulier.
Uitkijken met serial, identity, autoincrement,…
Incorrectly Filtered Escape Characters : fout = input wordt niet gefilterd op escape characters (bv. username = 'a' OR 't' = 't' …
- Incorrect Type Handling : fout = types van input worden niet gecheckt.
Oplossingen : escape character verwijderen uit invoervelden, controleren of het invoerveld wel het juiste type heeft, stored procedures (type, ; en escaping, grants, dicht bij de bron, …)
Opgepast : enkel escaping is niet voldoende, meestal maar eindig aantal mogelijkheden dus beter zeggen wat WEL mag ipv NIET mag.
Opgeslagen procedure (stored procedure) is een programma dat bewaard wordt binnen een database. Het voordeel van een opgeslagen procedure is dat ze draait binnen de database zelf, op de databaseserver. Daardoor heeft de procedure direct toegang tot de gegevens die ze moet manipuleren, en moet ze maar alleen de resultaten naar de gebruiker terugsturen. Dit vermijdt het over en weer sturen van grote hoeveelheden gegevens. Opgeslagen procedures worden meestal gebruikt wanneer een bepaalde groep wijzigingen op de database logisch bij elkaar horen. Het is dan verstandig deze wijzigingen te groeperen in één opgeslagen procedure.
+ :
  - Veel betere onderhoudbaarheid. Opgeslagen procedures laten toe om "modulair" te programmeren. De procedure kan eenmaal worden aangemaakt en bewaard in de database, en vervolgens meermalen opgeroepen in een programma.
  - Snellere uitvoering. Als de bewerking veel code bevat, of dikwijls herhaald wordt, dan zijn opgeslagen procedures sneller dan herhaalde SQL-opdrachten. Ze kunnen eenmaal ontleed en geoptimaliseerd worden, en daarna kunnen ze in het geheugen blijven. Omdat de procedure in gecompileerde vorm opgeslagen wordt, is het niet nodig de procedure telkens te compileren wanneer ze uitgevoerd wordt.
  - Ze kunnen het netwerkverkeer verminderen. Een operatie die honderden regels SQL-code vereist kan uitgevoerd worden met één enkele opdracht over het netwerk.
  - Ze kunnen gebruikt worden als een beveiligingstechniek. Gebruikers kunnen het recht krijgen een opgeslagen procedure uit te voeren, zelfs als ze niet het recht hebben de afzonderlijke opdrachten in de procedure direct uit te voeren.

Trigger : stukje programmacode, dat bij het bewerken van gegevens nagaat of er in die bewerking aan een vastgestelde voorwaarde wordt voldaan en dat, als dat zo is, een actie onderneemt.
PSQL : - connecten db = psql "service=myservice sslmode=require" $ psql postgresql://dbmaster:5433/mydb?sslmode=require
- help = \h
- help command = \h "command"
- wisselen db = \connect database
- alle tabellen binnen searchpath = pg_*_is_visible()
- uitvoer wegschrijven = \copy tablename to 'filename' csv
- export to html = -H, \pset format html, \H
- script = psql –d database –f script.sql
- historiek tonen/wegschrijven = \s "filename"
- welke poort = \echo :PORT

```
Bv. Partitioneren per plaats beginnen vanaf 1
SELECT row_number() OVER(partition BY
plaats), plaats, spelersnr
FROM spelers
WHERE geslacht = 'M'
ORDER BY 2;
```

Nummeren van rijen : row_number() OVER(), rijnummers zijn niet vast bepaald
Vaste rijnummers : zelfde met ORDER BY 2;
Manipuleren : row_number() OVER(ORDER BY spelersnr), plaats, spelersnr … ORDER BY plaats NULLS FIRST;
Volgens sorteervolgorde : F_i in frequentietabel : rank() OVER(ORDER BY plaats) … ORDER BY plaats NULLS LAST;
Dense rank : zelfde maar dan distinct aantal plaatsen
Partitioneren : zelfde concept als GROUP BY, maar fijner → per groep/partitie wordt de aggregatie/vensterfunctie toegepast
Partitioneren vaste volgorde : ORDER BY in OVER() : OVER(partition BY plaats ORDER BY spelersnr) (PK spelersnr voor vaste volgorde)
Cumulatieve som : SELECT sum(jaartoe) OVER(ORDER BY jaartoe), jaartoe FROM spelers

```
Cumulatief op spelersnr boetesom:
SELECT spelersnr, sum(bedrag)
OVER(order by spelersnr rows
between unbounded preceding and
current row)
FROM boetes
ORDER BY 1;
```

OVER()
● Gebruik venster functies ● Gebruik aggregatie functies
GROUP BY
● Gebruik aggregatie functies ● Andere 'positie', volgorde van 'verwerking'
● Venster <> identiek aan groepering ● Per groep heb je maar 1 waarde
● Per venster kan je meerder waarden hebben
XML : Extensible Markup Language (DTD of xml schema), 1998, hiërarchische structuren in een nieuw kleedje, platte tekstbestanden, ISO standaard, einde RDBMS?
Eenvoudige syntax : tag om aan te geven waar men begint en eindigt, elk document moet in 1 root tag omvat zijn, beginnen met cijfer mag niet, case sensitive, tags mogen genest worden.
Doel : om zowel hierarchische structuren te omschrijven als deze te bevatten; om data uit te wisselen tussen verschillende geg. Bronnen
METAdata : attributen aan tags meegeven, tussen " " of ' '. Enkel gebruiken om metadata weer te geven (bv. eigenschappen), data zelf niet
Bv. <example type='music'> lalala.mp3 </example>
XML wordt bv ook gebruikt om configuraties van software bij te houden (bv. menubalk)
Er zijn verschillende formaten die van deze standaard gebruik maken (xhtml, xml dom, xsl, xslt, xquery, soap, rss, svg, wap, …)
+ : ISO, uitwisselbaarheid, eenvoud, geen hierarchische engine nodig, …
- : redundantie tov relationeel model (dus niet misbruiken in die zin), sequentieel → traag, …

Tabellen vullen vanuit andere tabel, regels : mag dezelfde tabel zijn, in tabelexpressie mag alles staan zoals in een gewone, aantal kolommen in INSERT INTO moet gelijk zijn aan aantal expressies in SELECT, datatypes kolommen moeten overeenkomen met expressie.
Beperkingen naamgeving tabel/kolom : 2 tabellen uit dezelfde db : verschillende namen, 2 kolommen in 1 tabel : verschillende namen, lengte van naam beperkt, alleen letters, cijfers & enkele symbolen, beginnen met een letter, geen gereserveerde woorden, identifiers " "
Naamkeuze = zeer belangrijk! Korte namen(niet cryptisch), tabelnamen meervoud, geen informatiedragende namen, consistente namen, geen te lange namen, kolommen zelfde betekenis zelfde naam, kolommen met vergelijkbare populaties zelfde naam, geen termen uit BS
Integriteit : consistentie, correctheid!!!
Primaire sleutels : kolommen waarvan de waarde uniek moet zijn nooit NULL! In SQL : na de kolom, mbv PK, samengestelde PK
Regels : max. 1 PK per tabel, uniciteitsregel (unieke waarde), minimaliteitsregel (geen overbodige kolom), kolomnaam (1x in PK), niet null
Alternatieve sleutels : kolom waarvan waarde uniek is = alternatief voor PK MAAR mag NULL zijn
Refererende sleutels : FOREIGN KEY, kolom die verwijst naar PK in andere tabel, waarde MOET voorkomen in andere tabel
Regels : gerefereerde tabel moet bestaan/gecreëerd worden, moet PK bevatten, kolomnaam aanduiden (indien niet zelfde kolomnaam), referring key mag null zijn, aantal kolommen moet gelijk zijn, datatypes van kolommen moeten gelijk zijn.
+ refererende sleutel mag uit meerder kolommen bestaan, kolom mag deel zijn van verschillende referring keys, deel van (of alle) waarden van PK mag RK zijn, refererende tabel mag gelijk zijn aan gerefereerde tabel.
Bij update en delete : wijzigen of verwijderen van waardes in de gerefereerde tabel waarvoor rijen aanwezig zijn in refererende tabel.
3 mogelijkheden : **RESTRICT** [default](verboden), **CASCADE** (doorgetrokken naar refer. tabel), **SET NULL** (waardes refer. tabel = null)
Check-integriteitsregels : geven aan welke waardes toegelaten zijn, bv. geslacht char(1) not null check (geslacht in ('M', 'V'))
Bij foute instructie → foutboodschap, voor duidelijkheid naam geven aan integriteitsregel
Integriteitsregels verwijderen : DROP TABLE (PK, RK en AK verdwijnen, RK waarvan tabel gerefereerde tabel was verdwijnen), ALTER TABLE (verwijderen integ.regels afzonderlijk)
Vastleggen van integriteitsregels gebeurt in de catalogustabellen, afhankelijk van product

```
Naamgeven integriteitsregel
Create table spelers
(spelersnr integer not null
constraint primaire_sleutel_spelers
primary key,
...
geslacht char(1) not null
constraint toegestane_waarden_geslacht
check (geslacht in ('M', 'V')),
```

Transacties : 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 transactie, elke transactie = permanent ←→ commit : permanent maken van een transactie, rollback : ongedaan maken van transactie
Transactie : vanaf begin tot een commit/rollback, zinvol als een bepaald geg. uit meerdere tabellen geschrapt moet of een gebruiker zich vergist heeft bij aanpassingen.
Mogelijke uitzonderingen (product beperkingen) : instructies die catalogus wijzigen.
Impliciete start : autocommit
Expliciete start : begin;sqlcode;commit/rollback starttransaction;sqlcode;commit/rollback
Savepoints : maken een deel van een transactie ongedaan
Problemen multi-usergebruik :

- Dirty read (uncommitted read) : een gebruiker leest een gegeven read dat nooit gecommit werd
- Non-repeatable/non-reproducible read : een gebruiker leest voor en na de commit andere gegevens (geg. worden gewijzigd)
- Phantom read : een gebruiker leest voor en na de commit andere gegevens (er komen nieuwe gegevens)
- Lost update : een wijziging van een gebruiker wordt overschreven door een andere gebruiker

Oplossing : transacties serieel verwerken (indien honderden gebruikers : parallel verwerken)
Locking : de rij waar 1 gebruiker mee werkt wordt gelocked voor andere gebruikers, als transactie afgelopen is → unblocked
Locking gebeurt in de buffer (RAM); verschillende opties voor granulariteit en rechten (bv. SHARE en EXCLUSIVE)
Deadlocks : als 2 of meerdere gebruikers op elkaar wachten, oplossing : 1 transactie afbreken wanneer deadlock voorkomt.
Isolation level : mate van isolatie van gebruikers
Isolation level niveaus :

- Serializable = maximaal gescheiden
- Repeatable read = lezen (share blokkades, stopt bij einde transactie), muteren (exclusieve blokkades)
- Cursor stability [read committed] = lezen (share blokkades, stopt bij einde select), muteren (exclusieve blokkades)
- Dirty read [read uncommitted] = lezen (share blokkades, stopt bij einde select), muteren (excl. blokkades, bij einde mutatie)

Gevolgen serializable : concurrency (± gelijktijdige werking) is het laagst, snelheid is het laagst
Gevolgen dirty read : concurrency is hoog (moeten weinig op elkaar wachten), kunnen gegevens lezen die enkele momenten later niet meer bestaan. Bv. "SET transaction isolation level serializable"

Stored procedure : hoeveelheid code die opgeslagen is in de catalogus van een databank en die geactiveerd kan worden.
Verwerking : vanuit programma wordt procedure opgeroepen, DBMS ontvangt oproep en zoekt procedure, procedure wordt uitgevoerd waarbij de instructies op de database verwerkt worden, mbv code wordt aangegeven of procedure correct verwerkt is (sqlcode)
Parameters : communicatie met buitenwereld
3 soorten :

- Invoerparameters
- Uitvoerparameters
- Invoer/uitvoerparameters

Parameter best andere naam dan kolom!
Body : uit te voeren instructie
Lokale variabelen : vasthouden tijdelijke tussenresults, DECLARE CURSOR noodzakelijk
SET-instructie : toekennen van een waarde aan een lokale variabele
Flow-control instructies, mogelijkheden :

- IF … THEN … ELSEIF … THEN … END IF
- IF … THEN … ELSE … END IF
- CASE WHEN … WHEN … ELSE … END CASE;
- WHILE … DO … END WHILE ;
- REPEAT … UNTIL … END REPEAT;
- LOOP… END LOOP;

Aanroepen van stored procedures : CALL-instructie, mogelijkheden : programma, interactief SQL, stored procedure (recursiev mogelijk)
SELECT INTO : indien gegarandeerd maximaal 1 rij
Foutboodschappen : SQL-error code (beschrijvende tekst), SQLSTATE (code)
DECLARE HANDLER : SQL stopt, verwerking niet!
3 soorten :

- CONTINUE
- EXIT
- UNDO

Stored procedures & transacties:
Stored procedure analoog aan SQL :

- Commit
- Rollback
- Start transaction

ROUTINES : tabel in catalogus

```
Stored procedure :
Create procedure delete_wedstrijden
(in p_spelersnr integer)
begin
delete
from wedstrijden
where spelersnr = p_spelersnr ;
end
Call delete_wedstrijden (8) ;
```

```
Select into :
create procedure som_boetes_speler
(in p_spelersnr integer,
out som_boetes decimal(8,2))
begin
select sum(bedrag)
into som_boetes
from boetes
where spelersnr = p_spelersnr;
end
call som_boetes_speler (27, @som)
select @som
```

```
Stored procedures met een cursor :
(indien resultaat uit meerdere rijen bestaat)
Create procedure aantal_spelers
(out aantal integer)
begin
declare een_spelersnr integer;
declare found boolean default true;
declare c_spelers cursor for
select spelersnr from spelers;
declare continue handler for not found
set found = false ;
set aantal = 0;
open c_spelers;
fetch c_spelers into een_spelersnr;
while found do
set aantal = aantal + 1;
fetch c_spelers into een_spelersnr;
end while;
close c_spelers;
end
```

Verwijderen stored procedures : DROP PROCEDURE
Compileren en hercompileren : wat met tabellen met zelfde naam maar verschillende eigenaar? → afhankelijk van moment compilatie
Soms : bij creatie procedure ; bij uitvoeren procedure ; kan men kiezen
Beveiliging stored procedure : GRANT EXECUTE
Onderliggende machtigingen nodig? → eigenaar procedure wel, gebruiker niet
Voordelen stored procedures :

- Onderhoudbaarheid (bv. aantal verschillende mutaties samen)
- Verwerkingssnelheid (bv. minimaliseert netwerkverkeer)
- Precompilatie bij stored procedures
- Opgeroepen vanuit verschillende host-languages

```
Create procedure delete_wedstrijden_2
(snr_var in smallint) as
with recompile
Begin
…
End
```

Stored functions : stukken code bestaande uit SQL- en procedurele instructies opgeslagen in de catalogus
Maar : geen uitvoerparameters mogelijk, oproep via allerlei expressies, RETURN noodzakelijk
IN, geen OUT, geen INOUT
Verwijderen stored functions : DROP FUNCTION
Triggers : 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.
Worden door het DBMS zelf opgeroepen, niet door bv. een call
3 delen trigger :

- Triggerevent + trigger-moment
  - Wanneer?
    - AFTER (nadat triggering instructie is verwerkt)
    - BEFORE (eerst trigger-actie)
    - INSTEAD OF (alleen trigger-actie)
  - Voor welke rij?
    - FOR EACH ROW (voor elke rij)
    - FOR EACH STATEMENT (voor een statement)
- Triggerconditie : WHEN
- Trigger-actie : wat doet de trigger?

```
Stored function voorbeeld:
create function dollars (bedrag decimal(7,2))
returns decimal(7,2)
begin
return bedrag * (1 / 0,08) ;
end
select betalingsnr, bedrag, dollars(bedrag)
from boetes
```

```
Trigger voorbeeld:
Create trigger insert_speler
after
insert of spelers for each row
begin
insert into mutaties (gebruiker, mut_tijdstip,
mut_spelersnr, mut_type,
mut_spelersnr_new)
values (user, curdate(), new.spelersnr, 'I',
null) ;
end
```

NEW : nieuwe tabel lijkt te bestaan
Triggers als integriteitsregels : kunnen hiervoor gebruikt worden
Verwijderen trigger: DROP TRIGGER
Verschillen tussen producten :

- Meerder triggers op 1 tabel en een bepaalde mutatie?
- Kan 1 trigger-actie leiden tot activeren van andere (of dezelfde) trigger?
- Wanneer wordt trigger-actie precies verwerkt?
- Welke trigger-events worden ondersteund?
- Zijn triggers op catalogustabellen toegelaten?

Zowel procedures als functions kan je zien als een methode, in 'oudere' talen zien we dit verschil nog

ORDBMS = Object Related DataBase Management System = Object-relationeel, belangrijkste speler
ODBMS = Object Database Management System = objectgeoriënteerd, nichemarkt
ODBMS : Smalltalk (nieuwe programmeertaal,1972), geënt op OOPL concepten
Object Oriented Programming Language (OOPL) kenmerken :

- Objecten (state vs. behaviour) + OID (object identifier, unique & immutable)
- Complexe types & structuren (atomic, struct(tuple) + (collection (set, list, bag, array, dictionary(K,V)))
- Inkapseling
- Tijdelijk (transient) vs. persistent
- Overerving
- Polymorfisme (operator overloading)

ODL (Object Definition Language) :

- Objecten vertaald :
  - In Practice (value vs. reference)
  - Reference (object_id OID)
- Levensduur : transient vs. persistent
- Structuur : atomic of samengesteld
- Create : new
- Overerving : extends

```
Trigger als integriteitsregel:
create trigger gebjaartoe
before insert, update(geb_datum, jaartoe) of spelers
for each row
when (year(new.geb_datum) >= new.jaartoe)
begin rollback work ; end ;
```

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

OQL : Object Query Language
ORDBMS kenmerken :

- Uitbreiding van de basis datatypes
- Complexe objecten
- Overerving
  - Van gegevens
  - Van types
- Regels
  - Update-update regel
  - Query-update regel
  - Update-query regel
  - Query-query regel

Complexere en meer specialistische datatypes : **zelf gedefinieerde datatypes** :
CREATE TYPE, kenmerken : gebruikt waar basis-datatypes gebruikt worden, hebben geen populatie, niet manipuleerbaar met SELECT, statische inhoud (aantal waarden). Voordeel : vergelijking moet zinvol zijn (~strong typing)
DROP TYPE : verwijdert
Toegang tot datatypes : eigenaar = diegene die type creëert, user moet machtiging krijgen(GRANT USAGE), verwijderen : REVOKE USAGE
Casting : om verschillende datatypes te kunnen vergelijken
Destructor : transformeert zelfgedefinieeerd datatype naar basistype
Constructor : transformeert basisdatatype naar zelfgedefinieerd datatype
Zelf definiëren van operatoren :
Voorafgaandelijke opmerkingen :

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

```
Definiëren operatoren in vorm scalaire functies
bedrag1 + bedrag2 : ongeldig
=> decimal(bedrag1) + decimal(bedrag2)
create function « + » (geldbedrag, geldbedrag)
returns geldbedrag
source « + » (decimal(), decimal())
```

Opaque-datatype : datatype dat niet afhankelijk is van een basisdatatype, definiëren van functies om hiermee te werken is noodzakelijk

Named row-datatype : groeperen van waarden die logisch bij elkaar horen
Getypeerde tabel : een datatype toekennen aan een tabel, eenvoudig om gelijkende tabellen te creëren
Integriteitsregels op datatypes : Beperkingen op toegestane waarden
ALTER TYPE : wijzigen van datatype, als conditie strikter is, wijziging geweigerd tot al deze waarden aangepast zijn in de tabel.
Sleutels en indexen : is volledig analoog bij zelfgedefinieerde datatypes.
Bij named row-datatypes : op de volledige waarde of op een deel ervan
Overerving van datatypes : alle eigenschappen van 1 datatype worden overgeërfd door een ander
Supertype + subtype, bv : create type buitenlands_adres as (land char(20) not null) under adres;
Koppelen van tabellen : in OO-DB hebben alle rijen een unieke identificatie (door het systeem)
REF = om identificatie op te vragen (postgresql : OID)
Voordelen van **koppeling** :

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

Nadelen :

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

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

Voorbeeld getypeerde tabel :
```
create type t_spelers as
(spelersnr integer not null,
naam char(15) not null,
…
bondsnr char(4));
create table spelers of t_spelers
(primary key spelersnr) ;
```

Voorbeeld collecties :
```
create table spelers
(spelersnr smallint primary key,
…
telefoons setof(char(13)),
bondsnr char(4) );
insert into spelers (spelersnr, …, telefoons, …)
values (213, .., {'016-342654', '0475-654387'}, ..);
select spelersnr
from spelers
where '016-342654' in (telefoons);
```

Collecties : verzameling waardes in 1 cel (postgresql : arrays [])

Overerving van tabellen : alle eigenschappen voor 1 tabel worden overgeërfd
Beperkingen :

- Geen cyclische structuur
- Geen meervoudige overerving
- Alleen getypeerde tabellen in de tabelhiërarchie
- Overeenkomst tabelhiërarchie met typehiërarchie

RULES : gaan verder dan triggers : SELECT, INSERT, UPDATE, DELETE
FK's >> triggered, updateable views (rules of triggers), maar voorzichtig : systeemlogica
NOSQL : Not Only SQL (niet relationeel), bv. Wide column store, document store, graph based, key value, multivalue, andere niet sql db
Mongo db : binaire JSON, CRUD, index, aggregation, replication, sharding, Needs MapReduce
KV : look up value using key, structuur?, cf. Dictionary (ODMS), RDBMS oplossingen (postgresql : hstore of gewoon..)
ACID (transactions) : Atomicity (boolean), Consistency (state), Isolation (concurrency), Durability (commit, levensduur & select)
DBMS : + RDBMS, - NOSQL
DB keuze : SQL standard? Software? Mistakes? What about frameworks?
Ideal world vs reality, know your options, what is important, make your choices…
Software requirements :

- Standalone vs. client/server
- Single user vs. multi-user
- Close to the SQL standard or ..
- Quick & easy or ..
- Durable or ..
- GPL-alike or .. (general public license, GNU) → SmallSQL, SQLite, Firebird, Derby, MySQL, PostgresQL, HSQLDB
- Official support or ..
- Resources, OS, ..

Mistakes :

- Too many databases (abstract level) vs. Time
- Know your data (structure) → requirements
- DB != spreadsheet, skill of it's own
- Third normal form vs. common sense
- Application logic in the database
- Backup, replication
- Version control
- Use the tools
- The hammer to fix the plumbing, handcrafted vs. manhours, humans vs. mathematics

What about frameworks? Data live expectancy compared to the programming environment
Remarks :

- ACID vs. performance
- Data independence
- Standardization
- Voor bijna elk klein bedrijf tot KMO is RDBMS prima

Toekomst van SQL :

- SQL : databasetaal van relationele DBMS
- Integratie met talen als Java?
- Embedded SQL naar achtergrond → CLI's (command line interfaces)
- Object-relationele concepten
- Invloed van datawarehouses, OLAP (online analytical processing), … !
- Samenwerking met XML door extensies
- Verbetering performance
- SQL/MED
- Overdraagbaarheid neemt af

Maar SQL bijft de toekomst

Referentiële integriteit in een relationele database is het uitgangspunt dat de interne consistentie tussen de verschillende tabellen binnen die database wordt gewaarborgd. Dat betekent dat er altijd een primaire sleutel in een tabel bestaat als er in een sleutelveld in een andere tabel naar wordt verwezen. Het DBMS waarborgt de consistentie en zorgt er voor dat een transactie die de consistentie doorbreekt niet wordt aangebracht.

De **ACID-regels** zijn een viertal regels waar veel databases aan voldoen. Zonder deze strikte set regels kan een database gemakkelijk verstrikt raken in een situatie waarbij twee transacties tegelijkertijd schrijfrechten verlangen op een enkel tupel.
Dat betekent dat transacties voldoen aan de volgende eisen:

- **A**tomair (Atomic): de mate waarin het DBMS garandeert dat een transactie ofwel geheel wordt uitgevoerd, ofwel geheel nietig is.
- **C**onsistent (Consistent): Een transactie creëert ofwel een nieuwe geldige staat of herstelt de staat die er was (in geval van een fout of een probleem). Dit impliceert dat na de transactie alle integriteitsregels van de database moeten gelden.[1]
- Geïsoleerd (**I**solated): transacties worden geïsoleerd van elkaar uitgevoerd, dat wil zeggen dat transacties die tegelijkertijd worden uitgevoerd geen inzicht hebben in elkaars tussenresultaten.
- **D**uurzaam (Durable): waardoor een voltooide transactie later niet ongeldig gemaakt kan worden.

Database die voldoen aan de ACID-regels worden veelal gebruikt bij financiële instellingen. Sinds ongeveer 2005 gebruiken steeds meer sociale netwerksites zoals bijvoorbeeld Facebook databases die niet aan de ACID-regels voldoen. Databases die niet voldoen aan de ACID-regels zijn uitstekend geschikt voor gedistribueerde gegevensverwerking als consistentie geen vereiste is.
Het tegengestelde van de ACID-term is BASE (Basically Available, Soft state, Eventual consistency).

Hiërarchisch
DDL : gebruikt bij ontwerp & onderhoud van gegevensbank. Conceptueel niveau (definitie records & relaties), intern niveau (fysische aspecten, bestandsorganisatie, indexen **= Storage Definition Language**), extern niv. (benodigde recordtypes, velden & relaties = **View DL**)
DML : verwerking van gegevens. Hoog-niveau DML (interactief mbv querytaal = vragen formuleren om geg. uit gb te halen en aanpassen aan voorwaarden ←→ ingebed in programmeertaal (gasttaal)) ←→ laag niveau DML : instructies reeds ingebed in gasttaal (subroutine)
3-schema architectuur : conceptueel niveau **DBD**, intern niveau bestand (geordend geheel van segmenten), extern niveau **PCB** (Process Communication Block, logische gegevensbank, benodigde segmenten (sensitieve segmenten) en eventueel sensitieve velden.
- DBD : DBD-macro, SEGM-macro (naam + totale lengte in bytes), FIELD-macro (voor elk veld, kunnen overlappen)
- PCB : PCB-macro (naam van DBD en lengte van samengestelde sleutel die het langste is) + SENSEG-macro (voor elk sensitief segment) + SENFLD-macro (voor elk sensitief veld, hiervoor gebruikt men de naam vanuit de DBD en eventueel nieuwe startpositie, enkel noodzakelijk als men de volgorde van de velden wijzigt of bepaalde velden weglaat)
In de PROCOPT (processing options) met je voor elk sensitief segment aangeven welke bewerkingen toegelaten worden op extern zicht :
G : get (segmenttype opzoeken)
I : insert
R : replace
D : delete
A : all (of the above, dus **GIRD**)
K : key-sensitief (segmenttype overnemen omdat men 1/+ van zijn kinderen nodig heeft)
DML : via JCL (Job Control Language) geef je door welke PCB's gebruikt zullen worden, aangestuurd door gasttaal.

Netwerk

Syntax schema DDL:
SCHEMA NAME IS schema-naam
RECORD NAME IS recordnaam
KEY keynaam IS {veldnaam} … DUPLICATES ARE NOT ALLOWED / SYSTEM DEFAULT
SET NAME IS setnaam
OWNER IS recordnaam / SYSTEM
ORDER FOR INSERTION IS SORTED/FIRST/LAST/NEXT/PRIOR BY DEFINED KEYS
MEMBER IS recordnaam
INSERTION IS AUTOMATIC/MANUAL RETENTION IS FIXED/MANDATORY/OPTIONAL
KEY IS ASCENDING/DESCENDING {veldnaam} … DUPLICATES ARE NOT ALLOWED/FIRST/LAST
SET SELECTION IS THRU setnaam OWNER IDENTIFIED BY SYSTEM/APPLICATION/KEYveldnaam

Subschema DDL :
SS subschema-naam WITHIN schema-naam
AD SET set-naam IS nieuwe-naam
AD RECORD record-naam IS nieuwe-naam
AD veldnaam IS nieuwe-naam
RECORD SECTION
RECORD NAME IS recordnaam
SET SECTION
SET NAME IS setnaam

ORDBMS/OODBMS :
Simpele gegevens zonder queries : tekstverwerker, DBMS = file system van computersysteem
Simpele gegevens met queries : bekende situatie, relationele databank
Complexe gegevens zonder queries : persistent opslaan in programmeertalen (zoals Java, C, C++, …)
Complexe gegevens met queries : object-relationele / object-georiënteerde gegevensbanken
Syntax creëren nieuw datatype (ORDBMS) : CREATE TYPE name AS ([attribute_name data_type [COLLATE collation] [, …]])
Bv. CREATE TYPE obj_appartement AS (GebouwNaam varchar(25), Appartementnummer char(4), Aantalslaapkamers smallint();

ORDBMS RULES :
Update-update regels : zorgen ervoor dat bij een bepaalde update een andere update zal plaatsvinden.
Syntax : CREATE RULE name AS ON event
TO table_name [WHERE condition]
DO [ALSO|INSTEAD] {NOTHING | command | (command;command …) }
Query-update regels : bij een bepaalde query zal een update plaatsvinden.
Bv. CREATE RULE controle_salaris_opvraging AS
ON SELECT
TO werknemers.salaris
WHERE current.naam = "Torvalds"
DO INSERT INTO controle
VALUES(current.salaris, user, current_datetime);
Hier zal elke persoon die het salaris van Torvalds raadpleegt, geregistreerd worden in een controletabel. (dergelijke regels meestal niet te realiseren mbv triggers)
Update-query regels : bij een bepaalde update wordt een boodschap aan de gebruiker gegeven (worden vaak alerters genoemd)
CREATE RULE waarschuw_Mike_voor_salaris AS
ON UPDATE
TO werknemers
WHERE current.naam = "Mik"
DO SELECT raise_alert('mike', 'nazicht');
Deze regel zal Mike waarschuwen wanneer er een aanpassing gebeurt aan zijn gegevens.
Query-query regels : bij een bepaalde query zal een andere query uitgevoerd worden.
CREATE RULE Mike_Ann_zelfde_salaris
ON SELECT
TO werknemers.salaris
WHERE current.naam = "Ann"
DO INSTEAD
SELECT salaris
FROM werknemers
WHERE naam = "Mike";
Deze regel wacht tot het salaris van Ann opgevraagd wordt, als dit gebeurt wordt niet het salaris van Ann gegeven maar wel het salaris van Mike.
Als je bij dit soort regels 2 regels tegelijk laat uitvoeren, kan je onvoorspelbare resultaten krijgen. En daarenboven kunnen regels de uitvoering van andere regels veroorzaken.
Let dus op met oneindige loops. Regels zijn zeer krachtig, gebruik ze voorzichtig!