Samenvatting slides gegevensbanken Niels Vermeiren & Simon Cek

1 Non Correlated Subquery's

- Niet gecorreleerde subquery = Tabelexpressie binnen een tabelexpressie
- Resultaat wordt doorgegeven aan aanroepende tabelexpressie
- Subquery's mogen genest zijn

1.1 Scalaire subquery

= 1 rij, 1 waarde

```
SELECT  objectnaam, diameter -
        (SELECT diameter
        FROM hemelobjecten
        WHERE objectnaam = 'Zon') AS verschil
FROM        hemelobjecten
WHERE     satellietvan = 'Zon';
/*
Geef voor elke planeet hoeveel groter of
kleiner die is dan de zon.
*/
```

1.2 Rij – subquery

= 1 rij

```
Kolom-subquery

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

/*Geef alle manen...*/
```

1.3 Kolom – subquery

= elke rij 1 waarde

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

/*Geef alle manen...*/
```

1.4 Tabel – subquery

= verzameling rijen en kolommen

- From – component bevat subquery
- Tijdelijk resultaat
- Geen order by
- Subquery moet pseudoniem krijgen

```
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;

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

2 Joins

2.1 FROM

- Bevat tabelspecificaties
- Kunnen subquery's zijn
- Impliciete joins > expliciete joins

2.2 Expliciete inner join

```
SELECT   s.spelersnr
FROM     spelers as s INNER JOIN
            anderschema.woonplaatsen as stad
            ON (s.plaats = stad.plaatsnaam) ;
```

2.3 Expliciete Full Outer Join

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

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

2.4 Expliciete Left Outer Join

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

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

2.5 Outer Joins

- Left-outer-join:
  - Alle rijen uit linker tabel met eventueel bijhorende gegevens uit rechter tabel, anders NULL waardes
- Right-outer-join:
  - Alle rijen uit rechter tabel met eventueel bijhorende gegevens uit de linker tabel, anders NULL waardes
- Full-outer-join:
  - Alle rijen uit linker tabel en alle rijen uit rechter tabel met eventueel bijhorende gegevens uit andere tabel, anders NULL waardes

2.6 Condities From vs Where

- Extra voorwaarden mogen
- Let op er is een verschil met where!

```
select teams.spelersnr, teams.teamnr,  betalingsnr
from teams left outer join boetes
      on (teams.spelersnr = boetes.spelersnr)
where divisie = 'tweede'
```

↕

```
select teams.spelersnr, teams.teamnr,  betalingsnr
from teams left outer join boetes
     on (teams.spelersnr = boetes.spelersnr)
     and divisie = 'tweede'
```

2.7 Cross Join

= expliciet cartesisch product

2.8 Union Join

= elke rij van elke tabel wordt 1 maal opgenomen en aangevuld met null-waardes voor de kolommen uit de andere tabel.

NIET ondersteund door postgresql

2.9 Natural Join

= natuurlijke join, lexiografisch

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

2.10 Equi/Theta Join

Equi – join = vergelijking met =

Theta – join = vergelijking met een andere vergelijkingsoperator

2.11 Lateral Join

Subqueries die voorkomen in de FROM kunnen vooraf gegaan worden door het sleutelwoord LATERAL. Dit laat hen toe te verwijzen naar kolommen verzorgd door de voorgaande FROM items.

Zonder LATERAL, wordt elke subquery onafhankelijk geëvalueerd en kan er zo geen kruisende verwijzing gemaakt worden naar een ander FROM item.

```
SELECT  *
FROM    spelers s
    LEFT JOIN LATERAL age(s.geb_datum) ON true;
```

3 Gecorreleerde subquery's

3.1 Subquery's in WHERE

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

=, >, <, … : scalaire

IN(…) : kolom

3.2 Hoofdquery en subquery

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

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

3.3 Gecorreleerde subquery

= subquery waarin een kolom wordt gebruikt die tot een tabel behoort uit een andere select-blok.

- Subquery kan dus niet autonoom uitgevoerd worden.

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

/*Geef voor iedere reis het bezoek met de langste
verblijfsduur.*/
```

3.4 Operator EXSISTS and NOT EXSISTS

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

/*Geef alle reizen met een bezoek aan Jupiter.*/
```

Voor NOT EXSITS idem.

3.5 Operator ANY and ALL

= extra operator, verwacht scalaire waarde en kolomexpressie

Voorbeeld all

Geef van elk team het teamnr en de nr van de speler met het laagste aantal gewonnen sets:

SELECT distinct teamnr, spelersnr FROM wedstrijden w1

WHERE gewonnen <= ALL (SELECT gewonnen from wedstrijden w2 where w1.teamnr = w2.teamnr)

Voorbeeld any

Geef de spelersnummers van de spelers voor wie minstens één boete betaald is die groter is dan een boete betaald voor speler 27; deze speler mag zelf niet in het resultaat voorkomen.

SELECT DISTINCT spelersnr

FROM spelers inner join boetes b using(spelersnr)

WHERE b.bedrag > ANY(SELECT bedrag FROM boetes WHERE spelersnr = 27) AND spelersnr <> 27

>ALL >= ALL

>ANY >= ANY

<ALL <AN

3.6 Operator Unique

```
SELECT    spelersnr
FROM      boetes
WHERE     UNIQUE (
      SELECT    B.spelersnr
      FROM      boetes B
      WHERE     B.spelersnr = BT.spelersnr);

/*Geef de spelers voor wie precies één boete betaald werd.*/
/*Niet geimplementeerd*/
/*alternatief: HAVING..*/
```

3.7 Operator Overlaps

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

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

4 Patronen vergelijken

4.1 LIKE – operator (Sql standaard)

= like operatoren matchen altijd op de gehele string!

- Where naam LIKE '%/_%' ESCAPE '/'
- Escapen van tekens zorgt ervoor dat deze gebruikt worden zoals ze zijn, zonder interpretatieve.
- LIKE <> NOT LIKE

4.2 Similar to operator (Sql standaard sinds 1999)

- Zoals LIKE met %, _ en ESCAPE
- + extra's uit regex
  - | staat voor of
  - * staat voor mogelijke herhaling
  - + staat voor mogelijke herhaling
  - () om samen te nemen
  - [] analog aan gewone regex

4.3 Andere operatoren

- BETWEEN
- OVERLAPS
- IS NULL
- NOT

5 Venster functies

5.1 Nummeringsfuncties

5.1.1 Row_number()

Row_number() over() wilt zeggen dat we een extra kolom toevoegen met rijnummers, op basis van een nummering via een order by in de OVER() component. Deze kan verschillen van de order by van de query, waardoor deze rijnummers niet dezelfde hoeven te zijn als de natural numbers.

```
SELECT row_number() OVER(ORDER BY

spelersnr), plaats,

spelersnr

FROM spelers

WHERE geslacht = 'M'

ORDER BY plaats NULLS FIRST;
```

Bij deze query wordt een kolom toegevoegd met rijnummers die toegekend worden op basis van het spelersnr asc (dit komt door de ORDER BY spelersnr in de OVER() component) .

5.1.2 Rank()

Rank doet hetzelfde als row_number() maar gaat twee rijen die eenzelfde kolomwaarden hebben dezelfde rijnummer geven (rij nummer = RANK nummer). Bij het toewijzen van eenzelfde rijnummer aan een tweede rij, wordt het rijnummer achterliggend wel verhoogd, waardoor het rijnummer van de volgenden in de reeks een rijnummer overslaan.

```
SELECT rank() OVER(ORDER BY

plaats), plaats

FROM spelers

WHERE geslacht = 'M'
```

```
ORDER BY plaats NULLS LAST;
```

Hierbij krijgen allee spelers die op de zelfde plaats wonen hetzelfde ranknummer.

5.1.3 Dense_rank()

Idem als rank() maar de rijnummers worden intern niet verhoogd wanneer er dubbels optreden. Hierdoor onstaan er geen gaten in de RANK() nummering.

```
--cf distinct aantal plaatsen

SELECT dense_rank() OVER(ORDER BY

plaats), plaats

FROM spelers

WHERE geslacht = 'M'

ORDER BY plaats NULLS LAST;
```

5.2 Partitioneren

Vergelijkbaar concept van GROUP BY. Per groep/partitie wordt de aggregatie of vensterfunctie toegepast. Hierdoor worden de rijnummers opgeteld voor elke dubbele kolomwaarde. Per nieuwe kolomwaarde wordt er opnieuw van 1 begonnen en opgeteld voor elke rij met dezelfde kolomwaarde.

```
SELECT row_number() OVER(partition BY

plaats), plaats, spelersnr

FROM spelers

WHERE geslacht = 'M'

ORDER BY 2;
```

5.2.1 Partities sorteren

Wanneer we een order by toevoegen in de OVER component na een partitie dan worden de elementen gesorteerd alvorens ze gepartitioneerd worden.

```
SELECT row_number() OVER(partition BY

plaats ORDER BY spelersnr asc),

plaats, spelersnr

FROM spelers

WHERE geslacht = 'M'

ORDER BY 2;
```

5.2.2 Aggregatie functies met OVER()

```
SELECT sum(jaartoe) OVER(ORDER BY

jaartoe), jaartoe

FROM spelers
```

5.3 OVER() VS GROUP BY

- Andere positie, volgorde van verwerking
- Venster != groepering: per groep heb je één waarde, per venster meerdere waarden.

OVER()

- Gebruik venster functies
- Gebruik aggregatiefuncties

GROUP BY

- Gebruik aggregatiefuncties

```
--groepering

SELECT spelersnr, sum(bedrag)

FROM boetes

GROUP BY spelersnr;

--venster

SELECT spelersnr, sum(bedrag)

OVER(PARTITION BY spelersnr)

FROM boetes;
```

Verschil queries: bij een vensterfunctie zullen rijen met dezelfde kolommen niet samen worden genomen. (distinct)

5.4 Bereik

```
--cumulatief op spelersnr de boetesommen geven

SELECT spelersnr, sum(bedrag)

OVER(order by spelersnr rows

between unbounded preceding and

current row)

FROM boetes
```

between unbounded preceding and current row: behandel dubbele kolomwaarden als een nieuwe rij in de vensterfunctie bij het nemen van de cumulatieve som: anders zou de cumulatieve som niet correct genomen worden omdat bij rijen met eenzelfde kolomwaarde binnen het venster , bij de eerste rij onmiddellijk de totale cumulatieve som wordt opgeteld, en bij de andere dubbele waarden niks.

6 GROUP BY

6.1 ROLLUP

SELECT spelersnr, sum(bedrag)

FROM boetes

GROUP BY spelersnr WITH ROLLUP

Wanneer men deze query uitvoert krijgt men eerst het totaal bedrag van boetes per speler met vervolgen nog 1 rij met het totaal bedrag aan boetes van alle spelers.

Volgorde van verwerking : spelersnr => []

Spelersnr, plaats with rollup = > [geslacht, plaats], [geslacht], []

6.2 CUBE

SELECT row_number() over () as volgnr,

geslacht, plaats, count(*)

FROM spelers

GROUP BY geslacht, plaats WITH CUBE

ORDER BY geslacht, plaats

Bij deze query krijgt men eerst per geslacht en plaats het aantal spelers, dan per geslacht het aantal spelers(plaats is hier leeg), daarna per plaats het aantal spelers(geslacht is hier leeg) en als laatste het totaal aantal spelers.

Volgorde van verwerking: [geslacht,plaats] , [geslacht], [plaats], []

6.3 GROUPING SETS

=>Uitgebreide vorm van GROUP BY

- Meer mogelijkheden

bv.

SELECT geslacht, plaats, count(*)

FROM spelers

GROUP BY GROUPING SETS ((plaats),(geslacht))

ORDER BY 2, 1

- GROUP BY() : alle rijen in één groep

bv.

SELECT geslacht, plaats, count(*)

FROM spelers

GROUP BY GROUPING SETS ((geslacht, plaats),(geslacht),

())

ORDER BY 1, 2

6.4 ROLL & CUBE

- Vereenvoudiging door ROLLUP

bv.

SELECT geslacht, plaats, count(*)

FROM spelers

GROUP BY ROLLUP (geslacht, plaats)

ORDER BY 1, 2

- Vereenvoudiging door CUBE

bv.

SELECT geslacht, plaats, count(*)

FROM spelers

GROUP BY CUBE (geslacht, plaats)

ORDER BY 2, 3

6.5 Combinaties

Meerdere groeperingen zijn samen mogelijk

Mogelijkheden :

- 1 grouping set + 1 simpele : toevoeging: *GROUP BY E1, GROUPING SETS((E2)) = GROUPING SETS((E2, E1))*
- 2 grouping sets : « vermenigvuldiging » van specificaties (binnen één GROUP BY)
- Meerdere grouping sets samen : omzetting
- Bv GROUP BY GROUPING SETS (E1,E2),E3= GROUP BY GROUPING SETS ((E1,E3),(E2,E3))

● Union !

● CHECK BOEK PG 303 en verder voor UITLEG

7 Limiting result sets

- EXPLAIN : plaats bovenaan de query om uitvoertijd etc. te bekijken.
- FETCH FIRST # ROWS ONLY: haal alleen de eerste # rijen op (na ORDER BY)
- OFFSET: plaats vanwaar te starten (na ORDER BY, voor FETCH)
- LIMIT wordt niet gebruikt!

Vb.

SELECT spelersnr, naam, geb_datum

FROM spelers

ORDER by geb_datum

OFFSET 2 ROWS

FETCH FIRST 3 ROWS ONLY;

= haalt rij [3,5] op

8 Set-operatoren

SET-operatoren: combineren van resultaten van individuele SELECT-instructies.

8.1 UNION

= elke rij die in één van de twee selectblokken of in beide voorkomt. (dubbele rijen worden verwijderd)

Vb. Geef het spelersnummer van elke speler voor wie minstens één boete is betaald, of die aanvoerder is of voor wie beide geldt

SELECT spelersnr

FROM boetes

UNION

SELECT spelersnr

FROM teams

Regels:

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

8.2 INTERSECT

= alleen die rijen die in de resultaten van de beide selectblokken voorkomen (dubbele rijen worden verwijderd).

Vb.

Geef het spelersnummer van de spelers die aanvoerder zijn en voor wie minstens één boete is betaald

SELECT spelersnr

FROM teams

INTERSECT

SELECT spelersnr

FROM boetes

8.3 EXCEPT

= alleen die rijen die wel in het resultaat van het eerste select-blok voorkomen maar niet in het resultaat van de tweede select blok(dubbele rijen worden verwijderd).

Vb. Geef het spelersnummer van de spelers voor wie minstens

één boete is betaald, maar die geen aanvoerder zijn

SELECT spelersnr

FROM boetes

EXCEPT

SELECT spelersnr

FROM teams

8.4 ALL = behoud van dubbels

- Standaard worden dubbele rijen verwijderd
- ALL-variant: gebruiken om dubbele rijen te behouden
  - UNION ALL
  - INTERSECT ALL
  - EXCEPT ALL

Vb. Geef het spelersnummer van de spelers voor wie minstens één boete is betaald, maar die geen aanvoerder zijn. Behoud dubbele rijen.

SELECT spelersnr

FROM boetes

EXCEPT ALL

SELECT spelersnr

FROM teams

8.5 NULL

Rijen met een NULL-waarde worden als gelijk beschouwd door de set-operatoren!

8.6 Combinaties

- Meerdere set-operatoren
- Haakjes kunnen de volgorde wijzigen

Vb.Geef het spelersnummer van de spelers voor wie minstens één boete is betaald, maar die geen aanvoerder zijn, en daarenboven de spelers uit Hove.

(SELECT spelersnr

FROM boetes

EXCEPT

SELECT spelersnr

FROM teams)

UNION

SELECT spelersnr

FROM spelers

WHERE plaats = 'Hove'

9 ALTER object

9.1 Wijzigen van tabelstructuur

- ALTER TABLE: wijzigt structuur

  Vb. Karakterset veranderen

  convert to character set utf8 collate utf8_general_ci

  Vb. Rename table

  RENAME TABLE spelers TO tennissers

  Of ALTER TABLE spelers RENAME TO tennissers

9.2 Wijzigen van kolommen

- ALTER TABLE: wijzigt de structuur
  - ADD: kolom toevoegen
  - DROP verwijdert een kolom
  - ALTER: verandert de kolomeigenschappen

Vb. Alter table spelers

alter plaats varchar(5) not null

9.3 Wijzigen van integriteitsregels

- ALTER TABLE: wijzigt de structuur
  - ADD CONSTRAINT: voegt integriteitsregel toe
  - DROP CONSTRAINT: verwijderd integriteitsregel

Vb. Foreign key

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

Vb. Primary key

```
ALTER TABLE table_name
ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);
```

Vb. Check constraint

```
ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);
```

Vb. Unique constraint

```
ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);
```

9.4 Wijzigen van database

- ALTER DATABASE

Vb. create database tennis2

default character set sjis

default collate sjis_japanese_ci

9.5 Invoeren/verwijderen van gebruikers

- CREATE USER: creëert een user
  Vb. Create user Frank identified by Frank_pw
- ALTER USER: verandert het paswoord
  Vb. Alter user Frank identified by Frank_pasw
- DROP USER: verwijdert een user
  Vb. Drop user Frank

9.6 Insert voorbeeld

INSERT INTO table_x

SELECT *

FROM table_y;

INSERT INTO teams VALUES (null, 6, ‘Derde’);

10 Common Table Expressions

= vergelijkbaar met subquery’s in the FROM, maar met extra’s.

```
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)
);
```

10.1 Recursie

```
/*Alle getallen van 1 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;

 sum
------
 5050
```

```
WITH RECURSIVE kind_van(bijnaam, vader, moeder) AS (
        SELECT  bijnaam, vader, moeder
        FROM    familieboom
        WHERE   vader = 'vader'
  UNION ALL
        SELECT  f.bijnaam, f.vader, f.moeder
        FROM    familieboom f, kind_van k
        WHERE   f.vader = k.bijnaam
)
SELECT  *
FROM    kind_van;
```

Opgepast:

- Oneindige lussen
- Geen controle door CTE’s
- Zelf controles inbouwen!

10.2 Strategieën voor oneindige lus detectie

- Teller
- Lus detectie

Deze 2 zijn combineerbaar!

Optie1

```
WITH RECURSIVE kind_van(bijnaam, vader, moeder, diepte) AS (
        SELECT  bijnaam, vader, moeder, 1
        FROM    familieboom
        WHERE   vader = 'vader'
  UNION ALL
        SELECT  f.bijnaam, f.vader, f.moeder, diepte + 1
        FROM    familieboom f, kind_van k
        WHERE   f.vader = k.bijnaam
        AND     k.diepte<7
)
SELECT  *
FROM    kind_van;
```

Optie2

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

10.3 Inserts met CTE's

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

          avg
______________________
 0.55633943341672433333
```

10.4 Delete met CTE's

```
WITH RECURSIVE kind_van(bijnaam) AS (
        SELECT 'vader'::varchar
        UNION ALL
        SELECT f.bijnaam
        FROM   kind_van k JOIN familieboom f
                ON (k.bijnaam=f.vader)
)
DELETE FROM familieboom
USING kind_van
WHERE kind_van.bijnaam = familieboom.bijnaam;

DELETE 7
```

10.5 Voorbeelden

```
WITH lengte_namen AS (
        SELECT naam, length(naam) AS n
        FROM   spelers),
     evenlange_namen AS (
        SELECT l1.naam AS naam1, l2.naam AS naam2
        FROM   lengte_namen l1 JOIN lengte_namen l2
               ON l1.n = l2.n
        WHERE  l1.naam <> l2.naam),
     overzichts_lijst AS (
        SELECT naam1, array_agg(naam2) AS lijst
        FROM   evenlange_namen
        GROUP BY naam1)
SELECT l.naam, o.lijst
FROM   lengte_namen l LEFT JOIN overzichts_lijst o
+        ON l.naam = o.naam1;
```

Deze geeft voor elke naam in tabel spelers, een lijst van alle niet-zelfde namen die even lang zijn terug.

```
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;
```

Tussen resultaten wegschrijven naar debug_table om eventuele fouten te vinden.

ROW(l.*) = pak alles van de geselecteerde rij

Resultaat:

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

11 XML

- eXtensible Markup Language
- DTD (of xml schema)
- Eerste versie van xml standaard verscheen in 1998
- Hierachische structuren in een nieuw kleedje

- Platte tekstbestanden
- ISO standaard

11.1 Eenvoudige XML syntax

- Gebruik een tag om aan te geven waar men begint en eindigt
- Elk document moet in 1 root tag omvat zijn
- Beginnen met een cijfer mag NIET!
- Case sensitive!
- Tags mogen genest worden

Vb. <example>tekst</example>

11.2 Doel

- Xml(data) vs html(vorm) => (xsl vs css)
- Om zowel hiërarchische datastructuren te omschrijven als ze te bevatten
- Om data uit te wisselen tussen verschillende gegevensbronnen

11.3 META data

- Je kan attributen aan je tags meegeven (deze staan tussen “ “ of ‘ ‘)
- Gebruik dit enkel om META data weer te geven (bv eigenschappen)
- Data zelf hoort daar niet thuis!

Vb <example type =’music’> lalalala.mp3</example>

11.4 Uitbreidingen

- Xml word bv ook gebruikt om configuraties van software bij the houden vb menubalk,…
- Er zijn verschillende formaten die van deze standaard gebruik maken: Xhtml, xml dom, xsl, xslt, xpath,xsl-fo,xlink, xpointer, dtd, xsd, xforms, etc.

11.5 Pro vs Contra

ISO	Redundantie tov relationeel model
Uitwisselbaarheid	Sequentieel
Eenvoud	Traag
Gn hiërarchische engine nodig	…
…	

11.6 RDBMS

- Verschillende RDBMS voorzien manieren om xml te gebruiken!

11.7 Voorbeelden

SELECT xmlcomment('hallo');

xmlcomment

<!--hallo-->

----------vb 2 ----------

select xmlforest(divisie, teamnr) from teams;

xmlforest

<teams>

<divisie>ere </divisie><teamnr>1</teamnr>

<divisie>tweede</divisie><teamnr>2</teamnr>

</teams>

12 Datatypes

- Numeric
- currency
- Character
- Binair
- time
- Boolean
- …

12.1 Algemen verschillen voor teken-datatypes

Char(n) : vaste lengte => plaats wordt gereserveerd, ook indien deze plaats niet gebruikt wordt

Varchar(n): variabele lengte, flexibele lengte met maximum => trager

Tekst: onbeperkte variabele lengte, meest flexibel => traagst

12.2 Strings vs Identifiers

Een String wordt aangeduid d.m.v. ‘ ‘ quotes.

Een identifier wordt aangeduid d.m.v “ “ quotes.

Sql = standaard uppercase.

13 Creatie en ontwerp

13.1 Invoeren van nieuwe rijen

INSERT INTO

- Één rij invoeren in een tabel
- Tabel vullen met rijen van een andere tabel

Command: INSERT
Description: create new rows in a table
Syntax:
[WITH [RECURSIVE] with_query [, ...]]
INSERT INTO table_name [(column_name [, ...])]
{ DEFAULT VALUES | VALUES ({ expression | DEFAULT } [, ...]) [, ...] | query }
[RETURNING * | output_expression [[AS] output_name] [, ...]]

13.2 Tabel vullen vanuit andere tabel

Tabel wordt gevuld met rijen uit een andere tabel

Regels:

- Dit mag dezelfde tabel zijn
- In de tabelexpressie mag alles staan zoals in een “gewone” tabelexpressie

- Aantal kolommen in **INSERT INTRO** moet gelijk zijn aan aantal expressies in **SELECT**
- De datatypes van de kolommen moeten overeenkomen met de expressies in de **SELECT**

Vb.

```
Create table recreanten
(spelersnr smallint not null,
naam char(15) not null,
plaats char(10) not null,
primary key(spelersnr) )

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

13.3 Wijzigen van waarden in rijen

UPDATE: waarde worden gewijzigd

Vb.

Update wedstrijden

set gewonnen = 0

where spelersnr in

(select spelersnr

from spelers

where plaats = 'Den Haag')

13.4 Verwijderen van rijen uit een tabel

DELETE: rijen worden verwijderd

Vb.

delete

from spelers

where jaartoe >

(select avg(jaartoe)

from spelers)

13.5 Opzetten van nieuwe tabellen

CREATE TABLE: creëert lege tabel

Vb

Create table SPELERS (

spelersnr smallint not null,

naam char(15) not null,

voorletters char(3) not null,

geb_datum date ,

…

primary key (spelersnr))

Tabelschema: tabel(tabelnaam)

Kolomdefinitie: kolomnaam + datatype + null-specificatie+integriteitsregel

13.6 Datatypes van kolommen

Integer: gehele getallen

- tinyint
- smallint
- integer(int)

Decimal: niet-gehele getallen

- decimal(,) (dec) = numeric

Float: heel grote/kleine getallen

- single precision
- double precision

Alfanumerieke: karakter

- charachter(char)
- varchar
- long varchar

Temporal : datum en tijd

- Date
- Time
- Timestamp

Boolean : true / false

Blob (basic large object)

13.7 Creëren van tijdelijke tabellen

Tijdelijke tabellen:

Create temporary table SOMBOETES (

totaal decimal(10,2))

Insert into somboetes

select sum(bedrag)

from boetes

13.8 Kopiëren van tabellen

create table teams_kopie2 as

*(select **

from teams)

13.9 Naamgeving tabellen-kolommen

Beperking in de keuze:

- 2 tabellen uit hetzelfde database_schema : verschillende namen
- 2 kolommen in één tabel : verschillende namen
- lengte van naam is beperkt
- alleen letters, cijfers en enkele symbolen
- beginnen met een letter
- geen gereserveerde woorden
- Zoniet werken met identifiers “”

Naamkeuze is zeer belangrijk!!!

- korte namen (niet cryptisch)
- tabelnaam in meervoud
- geen « informatiedragende » namen
- consistente naamgeving
- geen te lange namen
- kolommen met eenzelfde betekenis, dezelfde naam
- Kolommen met vergelijkbare populaties, dezelfde naam
- geen termen uit het besturingssysteem

13.10 kolom-opties: default – comment

Defaultwaarde als laatste bij kolomdefinitie.

Create table boetes

(betalingsnr integer not null primary key,

spelersnr integer not null,

datum date not null default ‘1990-01-01’,

bedrag decimal(7,2) not null default 50.00);

13.11 tabellen en de catalogus

2 catalogustabellen:

- Tables :
  - table_creator
  - table_name
  - create_timestamp
  - comment
  - pg_catalog ..
- Columns :
  - table_creator
  - table_name
  - columnn_name
  - column_no
  - data_type
  - char_length
  - precision
  - scale
  - nullable
  - comment

13.12 specificeren van de integriteitsregels

- consistentie

- correctheid

13.13 integriteit bij primaire sleutel

- kolom waarvan de waarde uniek moet zijn (nooit null)
- samengestelde primary key

Regels:

- maximaal één primaire sleutel per tabel
- uniciteitsregel : waarde is uniek
- minimaliteitsregel : geen overbodige kolom
- kolomnaam : 1 maal in primaire sleutel
- Integriteitsregel : kolom mag geen NULL zijn

Raad: minimaal één primaire sleutel per tabel

13.14 Integriteit bij alternatieve sleutels

- Kolom waarvan de waarde uniek is, alternatief voor primary key
- Mag niet null zijn
- In SQL: na de kolom, m.b.v. UNIQUE

13.15 Refererende sleutels

- Kolom die verwijst naar primaire sleutel uit een andere tabel
- In SQL: **FOREIGN KEY(kolom) REFERENCES tabel(kolom)**

Regels:

- Gerefereerde tabel : bestaan of gecreëerd worden
- Gerefereerde tabel : een primaire sleutel bevatten
- Gerefereerde tabel :
    - Kolomnaam aanduiden
    - Geen kolomnaam aanduiden (zelfde kolomnaam !)
- Refererende sleutel mag NULL zijn
- Aantal kolommen moet gelijk zijn
- Datatypes van de kolommen moeten gelijk zijn

Merk op:

- Refererende sleutel mag uit meerdere kolommen bestaan
- Een kolom mag deel zijn van verschillende refererende sleutels
- Een deel (of alle) waarden van een primaire sleutel mag een refererende sleutel zijn
- Refererende tabel mag = gerefereerde tabel

13.16 De refererende actie

Bij **UPDATE** & **DELETE:**

- RESTRICT(**default)** : verboden
- CASCADE : doorgetrokken naar de refererende tabel
- SET NULL : NULL waardes in de refererende tabel.

13.17 Check-integriteitsregels

= geven aan welke waardes toegelaten zijn

SQL: **CHECK( )**

geslacht char(1) not null

check (geslacht in ('M', 'V')),

13.18 Namen aan integriteitsregels

Bij foute insturctie: foutboodschap => voor duidelijkheid geeft men een naam aan 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')),

...

13.19 Integriteitsregels voor verwijderen

DROP TABLE:

- Primaire, refererende en alternatieve sleutels verdwijnen
- Refererende sleutels waarvan de tabel de gerefereerde tabel was, verdwijnen

ALTER TABLE:

- Verwijderen van integriteitsregels afzonderlijk

13.20 Integriteitsregels en de catalogus

- Vastleggen van de integriteitsregels gebeurt in de catalogustabellen
- Afhankelijk van het product

13.21 Voorbeelden kolom- & tabel beperkingen

where column_constraint is:

[CONSTRAINT constraint_name]

{ NOT NULL |

NULL |

CHECK (expression) [NO INHERIT] |

DEFAULT default_expr |

UNIQUE index_parameters |

PRIMARY KEY index_parameters |

REFERENCES reftable [(refcolumn)] [MATCH FULL | MATCH

PARTIAL | MATCH SIMPLE]

[ON DELETE action] [ON UPDATE action] }

[DEFERRABLE | NOT DEFERRABLE] [INITIALLY DEFERRED |

INITIALLY IMMEDIATE]

----------tabel----------

and table_constraint is:

[CONSTRAINT constraint_name]

{ CHECK (expression) [NO INHERIT] |

UNIQUE (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 action] [ON UPDATE action] }

[DEFERRABLE | NOT DEFERRABLE] [INITIALLY DEFERRED |

INITIALLY IMMEDIATE]

14 Serial datatype

= GEEN echte sleutel!

Vb.

CREATE TABLE serialdemo(

id serial PRIMARY KEY,

content text

);

Serial creëert extra object een sequence!

Zorg dat gebruiker ook rechten heeft voor die sequence via GRANT!

Sequence

Een sequence is een databaseobject waarmee we oplopende of aflopende, unieke of niet unieke, cyclische of niet-cyclische reeksen met nummers kunnen genereren.

Vb.

GRANT { { USAGE | SELECT | UPDATE }

[, ...] | ALL [PRIVILEGES] }

ON { SEQUENCE sequence_name [, ...]

| ALL SEQUENCES IN SCHEMA schema_name [, ...] }

TO { [GROUP] role_name | PUBLIC } [, ...] [WITH GRANT OPTION]

15 JDBC

15.1 What is JDBC?

- De **J**ava **D**ata**b**ase **C**onnectivity (JDBC) API is de industrie standaard voor database-onafhankelijke verbinding tussen de Java programmeer taal en een groot assortiment van databanken, SQL databanken en andere tabelachtige gegevensbronnen, zoals spreadsheets. De JDBC API verzorgt een call-level API voor SQL gebaseerd databank toegang.
- JDBC technologie laat je toe de Java programmeertaal te gebruiken om de “Write once, run anywhere” capaciteiten te exploiteren voor applicaties die toegang nodig hebben tot entreprise data. Met een JDBC technologie toegelaten driver kan je verbinden met alle bedrijfsdata, zelfs in een heterogene omgeving.

15.2 SSL

- Gebruik maken van een JDBC connectie string met een niet gevalideerd ssl certificaat:
  “jdbc:postgresql://server/database?
  user=me&password=mypassword&ssl=true&sslfactory=org.postgresql.ssl.
  NonValidatingFactory”

16 Schema, rights, locale

16.1 Schema

- Schema = namespace
- Binnen één namespace moeten alle objecten een unieke naam hebben
- Bij postgreSQL is de default namespace “public” (checken met : > SHOW search_path;)
- Bij sommige andere producten default namespace = username

16.2 Rights(standard)

- Rechten op parentobject nodig om rechten op child te kunnen uitvoeren
- Default: alleen eigenaar heeft rechten

16.2.1 Rechten toekennen

GRANT <privilege> ON

<objecttype> <objectname>

TO <role>;

Vb.

```
GRANT SELECT ON mytable TO PUBLIC;

GRANT SELECT, UPDATE, INSERT ON mytable TO admin;

GRANT SELECT (col1), UPDATE (col1) ON mytable TO miriam_rw;
```

16.2.2 Rechten afnemen

REVOKE <privilege> ON

<objecttype> <objectname>

FROM <role>;

16.3 Privileges (passable)

= Rechten geven om rechten door te geven

GRANT <privilege> ON

<objecttype> <objectname>

TO <role>

WITH GRANT OPTION;

16.4 Localisatie

Vb1.

- Character sets: ç or not, Й or not
- Collating sequences: abc…
  A collating sequence (also called a sort sequence) defines how characters in a character set relate to each other when they are compared and ordered.

  Different collating sequences are useful for those who want their data ordered for a specific language. For example, lists can be ordered as they are normally seen for a specific language. A collating sequence can also be used to treat certain characters as equivalent, for instance, a and A.

- Encoding: UTF-8, LATIN1,….

Vb 2.

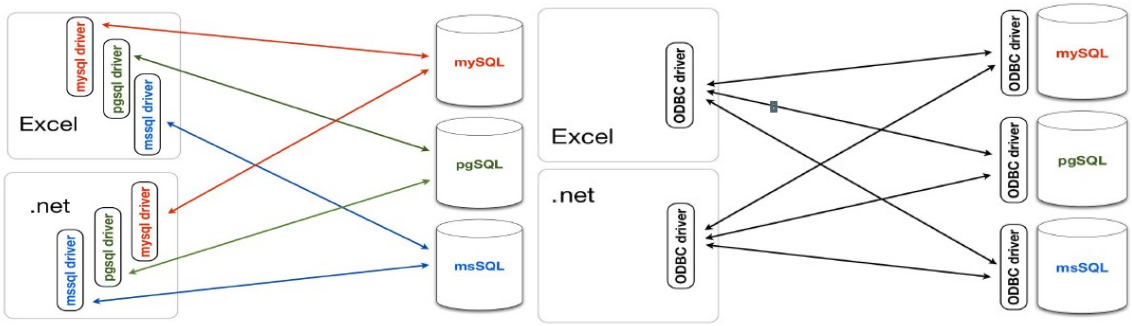
- LC_COLLATE: string order
- LC_CTYPE: character classification
- LC_MESSAGES: language of the message
- LC_MONETARY: format currency
- LC_NUMERIC: format numbers
- LC_TIME: time format
  >SHOW LC_COLLATE;

17 ODBC en native

17.1 Wat is ODBC?

Open **D**ata**B**ase **C**onnectivity (ODBC) is een manier om op een gestandaardiseerde manier toegang te geven aan een database, onafhankelijk van het type applicatie en van het type database.

Zonder ODBC: Met ODBC:



17.2 ODBC of native?

Dit hangt af van:

- Welke drivers zijn er beschikbaar?(Client en Server)
  Afhankelijk van taal.
- Zijn er prestatie of stabiliteit verschillen?
  Native blijft sneller, ODBC beter wanneer databases kunnen worden gewijzigd.
- Wat is het verschil met jdbc?
  ODBC is enkel voor Microsoft-applicaties en JBDC enkel voor Java-applicaties. ODBC kan niet gebruikt worden met Java omdat ODBC onderliggend een C interface voorziet.
- Wat is het verschil met pear?
  Pear (PHP extention and application repository) ’s Pear DB let you access databases in php for which an obdc driver is available.

18 Connection pooling

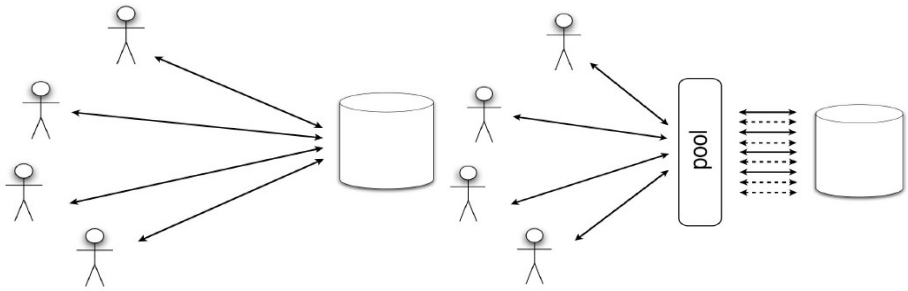
18.1 Wat is connection pooling?

In software ontwikkeling is een **connection pool** een cache van databank connecties onderhouden door de databank zo dat de connecties kunnen worden herbruikt wanneer de databank een toekomstig verzoek voor data ontvangt.

Opmerking: men kan ook aan client-side pooling doen.

zonder pooling:

met pooling:



18.2 Waarom pooling?

Pooling zorgt voor een boost in opstart prestatie (elke connectie heeft een opstart kost)

Nadeel: wanneer connecties niet gebruikt worden, gebruiken ze toch (een deeltje) geheugen.

19 Indexen

- Doel: beïnvloeden van de verwerkingstijd!
- Rijen worden in bestanden opgeslagen op het OS
- Een bestand bestaat uit pagina's
- Als een rij opgehaald wordt:
  - De betreffende pagina wordt opgehaald
  - De betreffende rij wordt opgehaald

19.1 Werking van een index

Er zijn 2 methodes voor het opzoeken:

- Sequentiële zoekmethode: rij per rij
  - Tijdrovend en inefficiënt
- Geïndexeerde zoekmethode: index (B-tree)
  - Een index in SQL is opgezet als een **Boom**
  - Deze boom bestaat dan uit een aantal **Knooppunten** (nodes)
  - Een knooppunt dat naar een rij verwijst = **Leafpage**
  - 2methodes:
    - Zoeken van rijen met een bepaalde, gewenste waarde
    - Doorlopen van de gehele tabel via een gesorteerde kolom(geclusterde index)

Waarom gebruik maken van een geclusterde index?
Wanneer men bv geordend op spelersnr spelers wilt opvragen gaan deze spelers gesorteerd van de schijf gehaald moeten worden. Dit kan ervoor zorgen dat een pagina meerdere malen van de schijf gelezen moet worden.(lees en bekijk tekening index boek pg 495-497). Om dit probleem te verhelpen maakt men gebruik van geclusterde indexen. Deze bepaald de volgorde van de rijen in het bestand waarin ze opgeslagen staan. Hierdoor wordt elke pagina maximaal 1 x opgehaald.

Opmerkingen:

- Bij aanpassing in tabel: index aangepast
- Index: ook op niet-unieke kolom
- Op één tabel: meerder indexen
- Op één tabel: één geclusterde index

- Samengestelde index

Opgelet:

- Index neemt opslagruimte in
- Als index vol: reorganisatie van de index

Meerde indexvormen zijn mogelijk!

19.2 Optimiser

Voor elke ingevoerde query probeert SQL de meest efficiënte strategie voor uitvoering te bepalen. Deze analyse, ook wel query optimalisation genoemd, wordt uitgevoerd door een module binnen SQL, genaamd **optimiser**. Deze ontwikkelt voor elke instructie een aantal alternatieve strategieën voor uitvoering.

Dit doet hij op basis van:

- Verwachte verwerkingstijd
- Aantal rijen
- Indexen

19.3 Create index

- No ANSI of ISO specification

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ name ] ON table [ USING method ]
    ( { column | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] )
    [ WITH ( storage_parameter = value [, ... ] ) ]
    [ TABLESPACE tablespace ]
    [ WHERE predicate ]
```

Postgresql boek pagina 493:

Creëren van een index

CREATE INDEX spelers_postcode_idx

ON spelers (postcode asc);

Creëren van een unieke index

CREATE UNIQUE INDEX spelers_naam_vl_idx

ON spelers (naam, voorletters);

Creëren van een samengestelde index

CREATE INDEX spelers_naam_vl_partial_idx

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

Creëren van een geclusterde index

CREATE UNIQUE CLUSTERED INDEX spelers_clustered

ON SPELERS(spelersnr)

19.4 Reindex

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

19.5 Index management

- CREATE
  CREATE INDEX <indexnaam>
- ALTER
  ALTER INDEX groot_idx
  SET TABLESPACE ergens_anders
- DROP
  DROP INDEX <indexnaam>

Vele SQL-producten:

- Automatische creatie van index op primaire en alternatieve sleutels bij creatie van tabel
- Naam wordt afgeleid uit naam van de tabel en de betreffende kolommen

19.6 Wat indexeren?

- **Voordeel:** index versnelt verwerking
- **Nadeel:** index neemt opslagruimte in en elke mutatie vraagt aanpassing van index (= vertraagde verwerking)

19.7 Welke kolommen?

Richtlijnen voor keuze van kolommen:

- Unieke index op kandidaatsleutels
  ⇨ Hierdoor kan uniciteit van nieuwe waarde snel gecontroleerd worden
- Index op refererende sleutels
  ⇨ Omdat refererende sleutels gebruikt kunnen worden join-kolom moet deze geïndexeerd worden voor een snelle verwerkingstijd bij joins.
- Index op kolommen waarop geselecteerd wordt
  - Grootte/#rijen van de tabel
  - Cardinaliteit (verschillende waarden) van de tabel
  - Distributie (verdeling) van de waarden
- Index op een combinatie van kolommen
  ⇨ Van toepassing als de WHERE-component een AND bevat
- Index op kolommen waarop gesorteerd wordt
  ⇨ Wanneer er gesorteerd moet worden maar geen index op die kolom, is er een extra sorteerslag nodig.

19.8 Speciale vormen

Multi-tabelindex (join-indexen) :

= index op kolommen in meerdere tabellen

Vb.

CREATE INDEX spel_weds

ON SPELERS(spelersnr), WEDSTRIJDEN(spelersnr)

Voordeel: wanneer 2 tabellen met een join op spelersnr gekoppeld worden, wordt deze join zeer snel verwerkt.

Virtuele-kolomindex :

= index op een expressie op voorwaarde dat de expressie geen aggregatiefuncties of subquery's bevat.

Vb.

CREATE INDEX wed_halvesaldo

ON WEDSTRIJDEN(gewonnen-verloren/2)

Voordeel: verbetering van de verwerkingssnelheid van instructies waarbij desbetreffende expressie in de WHERE-component wordt gebruikt. (WHERE gewonne-verloren/2).

Selectieve index:

= index op een gedeelte van de rijen.

Quote pg 514: *"Stel u voor dat de WEDSTRIJDEN-tabel één miljoen rijen bevat. En stel dat de meeste gebruikers voornamelijk gegevens van de laatste 2 jaar geïnteresseerd zijn, wat neerkomt op 200.000 rijen. Al hun vragen bevatten een conditie waar de datum niet ouder is dan twee jaar. Maar er zijn enkele gebruikers voor wie de andere 800.000 rijen wel nodig zijn. Voor indexen geldt hoe meer rijen, hoe groter de indexboom wordt, en hoe langzamer de verwerking wordt. Dit kan men voorkomen met selectieve indexen."*

Vb.

CREATE INDEX boetes_datum

ON BOETES

WHERE DATUM > '1996-12-31'

Voordeel: snellere verwerking bij het werken met gedeelte van tabellen.

Hash-index:

= index op basis van het adres op de pagina maar er wordt geen indexboom gecreëerd. Deze index moet gemaakt worden voor de tabel gevuld wordt. Wanneer een hash-index wordt gecreëerd, wordt er automatisch een bepaalde ruimte op de schijf gereserveerd. Deze ruimte zal gebruikt worden voor het opslagen van rijen.

Vb.

CREATE HASH INDEX spelersnr_hash

ON SPELERS(spelersnr)

WITH PAGES = 100

Voordeel: bij een B-tree (vorige 3 items zijn hierop gebaseerd) moet men eerst de hele indexboom doorlopen voordat men bij de werkelijke rij uitkomt. Bij de hash-index springen we vrijwel meteen naar de rij toe.

Bitmapindex:

= interessant als er veel dubbele waardes zijn.

Vb.

CREATE BITMAP INDEX spelers_geslacht

ON SPELERS(geslacht)

Clustered index: een index waarbij een kolom de fysieke volgorde van de records op het opslagmedium bepaalt.

19.9 Nieuwere datatypes

- GiST (Generalized Search Tree)
- SP-GiST (space-partitioned GiST)
- GIN (Generalized Inverted Index)

Index catalog_table in postgresql = pg_index

http://www.postgresql.org/docs/9.1/static/catalog-pg-index.html

http://www.postgresql.org/docs/current/static/internals.html

20 Optimaliseren van insturcties

20.1 Vermijden van OR-operator

OR: index wordt meestal niet gebruikt

Alternatief: IN of 2x select + UNION

VB:

```
where  spelersnr = 15
or   spelersnr = 29
or   spelersnr = 55

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

20.2 Onnodig gebruik van union

UNION: dezelfde tabel meerdere malen doorlopen

Alternatief: herformuleren waarbij alle voorwaarden in 1 select instructie staan.

20.3 Vermijd NOT-operator

NOT: index wordt niet gebruikt

Alternatief: vervang NOT door vergelijkingsoperatoren

```
where not (jaartoe > 1980)

    =>    where jaartoe <= 1980
```

20.4 Isoleer kolommen in condities

Kolom in een berekening of in een scalaire functie: index wordt niet gebruikt.

Alternatief: isoleer de kolom

```
where  jaartoe + 10 = 1990

    =>    where jaartoe = 1980
```

20.5 Gebruik de BETWEEN-operator

AND: gebruikt de index meestal niet

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

   =>   where jaartoe between 1985 and 1990
```

20.6 Bepaalde vormen van LIKE-operator

LIKE: index wordt niet gebruikt als patroon begint met % of _

LIKE '#_%' ESCAPE '#'

LIKE '\#_%'

Alternatief: GEEN

20.7 Redundante condities bij joins

Redundante condities: om SQL te verplichten om een bepaald pad te keizen

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

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

20.8 Vermijd de Having-component

HAVING: index niet gebruikt

Alternatief: zoveel mogenlijk in WHERE

```
group by spelersnr
having spelersnr >= 40

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

20.9 SELECT-component: smal

SELECT-component zo klein mogelijk

- Onnodige kolommen weglaten uit **SELECT**
- Bij gecorreleerde subquery met exists: één expressie bestaande uit één constante

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

20.10 Vermijd DISTINCT

DISTINCT: verwerkingstijd verlengt

Alternatief: vermijden als het overbodig is

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


=>  select wedstrijdnr, naam
```

20.11 ALL-optie bij set operatoren

Zonder ALL: verwerkingstijd verlengt, data moet gesorteerd worden om dubbels eruit te halen

VB: ipv UNION => UNION ALL

20.12 Kies outer-joins boven UNION

UNION: verwerkingstijd verlengt

Alternatief: outer-join is beter

20.13 Vermijd datatype-conversies

Converteren van datatypes: verwerkingstijd verlengt

Alterntief: conversie datatypes vermijden

20.14 Grootste tabel als laatste

Volgorde van tabellen kan belangrijk zijn!

Alteratief: grootste tabel als laatste plaatsen

20.15 Vermijd ANY- en ALL- operatoren

ANY en **ALL:** index wordt niet gebruikt

Alternatief: geen

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

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

Vervang door min of max

21 EXPLAIN

- Explain = geeft informatie over kostprijs, etc. van query
- Explain Analyze = naast kostprijs ook echte uitvoeringstijd
- CREATE INDEX ON TABEL(KOLOMNAAM) = index creëren

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

                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Index Scan using een_miljoen_teller_idx on een_miljoen
        (cost=0.00..34612.36 rows=999487 width=37)
        (actual time=0.053..2188.616 rows=999500 loops=1)
   Index Cond: (teller > 500)
 Total runtime: 3303.901 ms
(3 rows)

Niet vergeten:)=
SET enable_seqscan TO on;
```

SET enable_seqscan TO off = sequentieel zoeken uitzetten => zoeken op index

```
Conclusie:
• EXPLAIN (afhandelijk van de statistieken)
   Oplossing: ANALYZE voordien
• EXPLAIN ANALYZE
   Actuele Uitvoertijd, maar voert dus ook effectief uit!
• EXPLAIN
   Werkt ook voor ander DML (eg INSERT..)
   EXPLAIN ANALZE (later transacties..)
• EXPLAIN
   Kijk naar de totale cost
   Kijk eventueel naar de Scan methoden
```

22 Views

22.1 CREATE VIEW

- View: tabel die zichtbaar is voor de gebruiker maar geen opslagruimte inneemt.
- De 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 zitten

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

22.2 Nut?

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

View:

- Bevat geen rijen!

- Voorschrift of formule om gegevens uit basistabellen in een ‘virtuele’ tabel te steken

22.3 Creëren van views

- CREATE VIEW: creëert een view
- Views: raadplegen en muteren + synoniemen en commentaar!

22.4 Kolomnamen van views

SELECT definieert de kolomnamen, maar expliciete definitie is ook mogelijk!

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

Expliciete definitie is verplicht als kolom bestaat uit een functie of berekening.

22.5 With check option

Muteren van views => muteren van tabellen

WITH CHECK OPTION controleert:

- Update: aangepaste rijen behoren nog tot view
- Insert: nieuwe rij behoort tot view
- Delete: verwijderde rij behoort tot view

```
. Create view oud as
select *
from spelers
where geb_datum < ‘1950-01-01’
with check option
```

22.6 Verwijderen van views

DROP VIEW: verwijdert view en alle hierop gedefinieerde views

22.7 Beperkingen bij muteren

SELECT, INSERT, UPDATE, DELETE van views

Maar mutatie mag alleen als:

- View moet direct/indirect 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
- (INSERT) In select moeten alle not null-kolommen staan

22.8 Verwerken van instructies op views

- Extra stap waarin viewformule wordt opgenomen

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

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

(check-option)

22.9 Toepassingen van views

- Vereenvoudigen van routinematige instructies
  - Instructies die vaak gebruikt worden
- Reorganisatie van tabellen
  - Bij aanpassingen "oude" programma's laten bestaan
- Stapgewijs opzetten van select-instructies
  - Bij complexe queries stukken "voorprogrammeren"
- Specificeren van integriteitsregels
  - With check option: toegestane waarden controleren
- Gegevensbeveiliging
  - Beveiligen van delen van tabellen

22.10 Beveiliging

Beveiligen!

SQL gebruiker:

- Moet gekend zijn
- Wachtwoord
- Expliciete toekenning van bevoegdheden
  - Kolombevoegheden
  - Tabelbevoegheden
  - Databasebevoegdheden
  - Gebruikersbevoegdheden

22.11 Invoeren/verwijden van gebruikers

CREATE USER: creëert een user

Vb. Create user Frank identified by Frank_pw

ALTER USER: verandert het paswoord

Vb. Alter user Frank indentified by Frank_pw

DROP USER: verwijdert een user

Vb. Drop user Frank

22.12 Tabel- en kolombevoegdheden

GRANT: kent bevoegdheden toe

Soorten tabelbevoegheden:

- **SELECT:** bevoegheid tot select en view
- **INSERT**: rijen toevoegen
- **DELETE**: rijen verwijderen
- **UPDATE:** rijen wijzigen
- **REFERENCES:** refererende sleutels naar deze tabel creëren
- **ALTER:** tabel veranderen
- **INDEX:** indexen creëren (tabel)
- **ALL-ALL PRIVILEGES:** alle bevoegdheden

22.13 Databasebevoegdheden

Idem vorige +

- **DROP:** tabellen verwijderen
- **CREATE TEMPORARY TABLES:**
- **CREATE VIEW**
- **CREATE ROUTINE:** nieuwe stored procedure/functie
- **ALTER ROUTINE**:
- **EXECUTE ROUTINE**
- **LOCK TABLES:** bestaande tabellen blokkeren

22.14 Gebruikersbevoegdheden

= databasebevoegheden toekennen aan gebruiker

CREATE USER: gebruiker aanmaken

```
Vb.   Grant create, alter, drop
on *.*
to Frank
```

22.15 With grant option

= de gebruikers die toegang krijgen, kunnen deze machtiging doorgeven aan andere gebruikers

22.16 Werken met rollen

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

```
Create role admin
grant select, insert
on spelers
to admin
grant admin to Frank, Marc, Ann, Greet
```

DROP ROLE: verwijdert de rol

```
Drop role admin
```

22.17 Intrekken van bevoegheden

REVOKE: verijdert bevoegdheid (en de afhankelijke bevoegdheden)

```
Revoke all
on spelers
from Frank

Revoke admin from Marc

Revoke select
on spelers
from admin
```

22.18 Beveiliging van en met views

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

Vb. Create user …

```
Create view zichtbaar_deel as ..

Grant select
on zichtbaar_deel
to …
```

23 Beveiliging

- Hardware
- Het platform waar het op draait
- De databank software
- Binnen sql zelf (grant, revoke, view,..)
- Sql als “vertaler”
- Algemeen terugkerende security problematiek
- (zie 3ti, bv authenticatie, maar ook backups, ..)
- ..

23.1 Ondersteunend platform

- Up to date houden
- Beveiligen
- DDOS attacks

23.2 Databank software

- Up to date houden
- Configuratie
  - Local
  - Remote
- Known exploits -> Wat doe je?

23.3 Within SQL

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

23.4 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)

23.5 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.
- Watch out with serial, identity, auto increment

23.6 Incorrectly filtered escape characters

- Fout: input is niet gefilterd op escape characters

– 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?

23.7 Incorrect type handling

- Fout: types van de input worden niet gecheckt

– 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..

23.8 Oplossingen

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

23.9 Opgepast

- Enkel Escaping is niet voldoende:

- 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.

24 Embedded SQL

= SQL binnen een host language

24.1 Hoe?

- Host-language herkent SQL niet en geeft fouten => precompileren
- Precompiler (afhankelijk van product) :
  - identificeert de SQL-instucties in het programma
  - vertaalt de SQL-instructies naar de host-language
  - controleert de syntax van de SQL-instructies
  - controleert of de gebruikte tabellen/kolommen bestaan
  - controleert de bevoegdheden
  - bepaalt de verwerkingsstrategie
  - voert de SQL-instructies uit

25 Transacties en multi-user gebruik

Single – user <-> multiple-user

- Wat als meerderen tegelijk dezelfde gegevens willen gebruiken

25.1 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 een transactie
- Elke transactie is permanent
  commit: permanent maken van een transactie

Rollback: ongedaan maken van een transactie
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: instructies die de catalogus wijzigen

25.2 Hoe?

Inpliecete start bv na ROLLBACK of COMMIT

Cf autocommit

Explicite start

- begin; sql code; commit ;
- begin; sql code; rollback;
- start transaction; sql code; commit ;
- start transaction; sql code; rollback;

25.3 Savepoint

= maken een deel van een actuele transactie ongedaan.

Vb

Update ...

insert ...

savepoint S1

insert ...

savepoint S2

delete ...

rollback work to savepoint S2

...

25.4 Problemen multi-user gebruik

Dirty read (uncommitted read) :

een gebruiker leest een gegeven dat nooit gecommit werd

Nonrepeatable – 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

25.5 Oplossing

Oplossing :

Transacties serieel verwerken !

Oplossing indien honderden gebruikers tegelijk willen werken

Transacties parallel verwerken !

25.6 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)

25.7 Deadlocks

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

25.8 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
  - Cursor stability (read committed) :
    - lezen : share blokkades (stopt bij einde select)
    - muteren : exclusive blokkades
  - Dirty read (read uncommitted) :
    - lezen : share blokkades (stopt bij einde select)
    - muteren : exclusive blokkades (stopt bij einde mutatie)
- Tabel boek

25.9 Gevolgen

- Serializable :
  - concurrency is het laagst
  - Snelheid laagst
- Dirty read :
  - concurrency is hoog, moeten weinig op elkaar wachten
  - Kunnen gegevens lezen die enkele momenten later niet meer bestaan

Vb.

Set transaction isolation level serializable

26 Procedurele transacties

26.1 Stored procedures

= hoeveelheid code die opgeslagen is in de catalogus van een DB en die geactiveerd kan worden door deze aan te roepen vanuit een programma, een trigger, of een andere stored procedure.

Stored procedures maken het mogelijk om bepaalde delen van een programma centraal in de catalogus van een database server op te slaan. Hier zijn ze dan vanuit alle programma's aanroepbaar.

Vb.

```
Create procedure delete_wedstrijden

(in p_spelersnr integer)

begin

delete

from wedstrijden

where spelersnr = p_spelersnr ;

end

Call delete_wedstrijden (8) ;
```

verwerking:

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

26.2 Parameters

- Communicatie met buitenwereld
- 3 soorten :
  - Invoerparameters
  - Uitvoerparameters
  - Invoer/uitvoerparameters
- Parameter best andere naam dan kolom !!!

26.3 Body

- Uit te voeren instructies
- BEGIN … END; : samengestelde instructie
- Alle mogelijke SQL-instructies
- Label :

```
Vb.
BLOK1 : BEGIN
BLOK2 : BEGIN
BLOK3 : BEGIN
END BLOK1;
END BLOK2;
END BLOK3;
```

26.4 Lokale veriabelen

- Vasthouden van tijdelijke tussenresultaten
- DECLARE CURSOR noodzakelijk

Vb.

```
declare num1 decimal(7,2);


create procedure test
   (out getal1 integer)
begin
    declare getal2 integer
       default (select count(*) from spelers);
    set getal1 = getal2;
end
```

26.5 SET-INSTRUCTIE

Toekennen van een waarde aan een lokale variabele.

VB: set var1=1;

26.6 Flow-control instructies

Mogenlijkheden:

- 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;

26.7 Aanroepen van stored procedures

CALL-instructie

Mogelijkheden :

- Programma
- Interactief SQL
- Stored procedure
- Recursieve oproep van een stored procedure

26.8 SELECT INTO

Indien gegarandeerd maximaal 1 rij!

Vb.

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

26.9 Foutboodschappen, handlers, conditions

- Foutboodschappen
  - SQL-error-code: beschrijvende tekst
  - SQLSTATE: code
- DECLARE HANDLER: SQL stopt verwerking niet!!
- 3 soorten:
  - Continue
  - Exit
  - Undo

26.10 Stored procedures met een cursor

Indien resultaat uit meerdere rijen bestaat

Vb.
```
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
```

26.11 Stored procedures en transacties

Stored procedure analoog aan SQL-instructies

- Commit
- Rollback
- Start transaction

26.12 Stored procedures en catalogus

ROUTINES: tabel in catalogus

26.13 Verwijderen stored procedures

DROP PROCEDURE: verwijdert procedures

Vb.

Drop procedure delete_spelere

26.14 Compileren en hercompileren

Twee tabellen met zelfde naam gaat als ze verschillende eigenaren hebben.

With recompile: compiler wordt steeds opnieuw opgeroepen. De verwerkingsstrategie voor de procedure wordt steeds opnieuw aan de actuele situatie van de db aangepast. Nadeel: hercompileren kost tijd.

- Wat met tabellen met dezelfde naam maar verschillende eigenaar ?
- Afhankelijk van het moment van compilatie
  - Soms : bij creatie van de procedure
  - Soms : bij uitvoeren van de procedure
  - Soms : kan men kiezen

Vb.

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

26.15 Beveiliging met stored procedures

- GRANT EXECUTE : beveiliging
- Onderliggende machtigingen nodig ?
  - eigenaar van de procedure wel
  - gebruiker van de procedure niet

Vb.

```
grant execute
on delete_wedstrijden
to John
```

26.16 Voordelen van stored procedures

- Onderhoudbaarheid: vb aantal verschillende mutaties samen
- Verwerkingssnelheid: minimaliseert netwerkverkeer
- Precompilatie bij stored procedures
- Oproepen vanuit verschillende host-languages

26.17 Stored function

= Stukken code bestaande uit SQL- en procedurele instructies, opgeslagen in de catalogus

Maar:

- Geen uitvoerparameters mogelijk
- Oproep via allerlei expressies
- Return noodzakelijk

Vb.

```
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
```

26.18 Verwijderen stored functions

DROP FUNCTION: verwijdert function

Vb: drop function delete_speler

26.19 Triggers

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

```
create table mutaties
    (gebruiker              char(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))
```

```
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
```

3 DELEN:

- **Trigger-event + trigger-moment**
  - 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
- **Trigger-conditie:** WHEN
- **Trigger-actie:** wat doet de trigger?
- **NEW:** nieuwe tabel lijkt te bestaan.

26.20 Complexere voorbeelden

Veronderstel extra tabel met per speler het aantal gespeelde wedstrijden. Als speler geschrapt wordt, moet de informatie van die speler ook uit deze tabel verwijderd worden

create trigger delete_spelers

after delete on spelers for each row

begin

delete from spelers_wed

where spelersnr = old.spelersn

end ;

26.21 Triggers als integriteitsregels

Triggers kunnen gebruikt worden voor het implementeren van integriteitsregels.

Vb.

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 ;

26.22 Verwijderen van triggers

DROP TRIGGER: verwijderen van een trigger

Vb. drop trigger gebjaartoe

26.23 Verschillen tussen producten

- Meerdere triggers op één tabel en een bepaalde mutatie ?
- Kan één trigger-actie leiden tot het activeren van een andere (of dezelfde) trigger ?
- Wanneer wordt de trigger-actie precies verwerkt ?
- Welke trigger-events worden ondersteund ?
- zijn triggers op catalogustabellen toegelaten ?

26.24 verschillen

Stored procedure

```
SP: parameters in de signatuur
* IN
* OUT
* INOUT

bv.
CREATE PROCEDURE
 voorbeeld(IN a int, OUT b int, INOUT C int)
..
```

```
SP: geen return

oproepen via
bv. CALL
```

Stored functions

```
SF: parameters in de signatuur
* IN
* Geen OUT
* Geen INOUT
>> RETURN is nodig
```

```
SF:

bv
CREATE FUNCTION
 voorbeeld(a int, b int)
RETURNS int
..

oproepen via
SELECT
```

```
Methoden:
zowel procedures
en functies
kan je zien als een methode,
in 'oudere' programmeertalen
zie je nog dit verschil.
```

```
Trigger?
: 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.
```

27 History

27.1 .Data(banken)_history

0. Bestanden
1. Hierarchisch model
2. Netwerk model
3. Relationeel model
4. Object Relationeel model
5. Object geÖrienteerd model
6. Andere paradigma

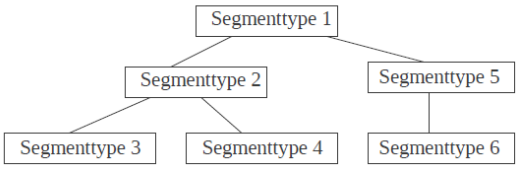
28 Hiërarchische GBD

- Onstaan in jaren ’60 (’66-’68) enrom populaire geweest
- Bekendste voorbeeld: IMS (IBM)
- Momenteel: verliest aan belang?

28.1 Opstellen hiërarchische model

28.2 Bouwstenen

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



28.3 ER-model naar hiërarchisch model

- n-op-m verbanden omzettten naar 1-op-n
- 1 segmenttype als parent en 1 als child kiezen

28.4 Leefregels van het hiërarchisch model

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

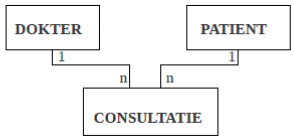
28.5 Terminologie hierarchisch model

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

Twin-segment = zelfde type, zelfde parent **Segmentcode** = nr dat segment identificieert **Volgorde-veld**= veld waarmee twins onderscheiden **samengestelde sleutel**= bepaald segment vinden

28.6 Voorbeeld

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



28.7 3-schema architectuur

- 3 schema-architectuur
  - Conceptueel niveau : DBD
  - Intern niveau : bestand
  - Extern niveau : logische gegevensbank PCB meerdere PCB's : PSB

28.8 GBS talen

Gegevensdefinitietaal: **D**atabase **D**efinition **L**anguage

Gegevensmanipulatietaal: **D**atabase **M**anipulation **L**anguage

28.9 DDL

- Conceptueel niveau: **D**ata **b**ase **d**escription
- Intern niveau: bestand
- Extern niveau: **P**rogram **C**ommunication **B**lock

28.10 Conceptueel niveau: DBD

= in de DBD vind je de beschrijving van de segmenten en hun relaties. (gemaakt m.b.v. macro-taal)

- Macro-bevelen:
  - DBD-macro: geeft een naam aan de databank
  - SEGM-macro: geeft een naam en totale lengte in bytes aan een segment
  - FIELD-macro: geeft naam en startpositie van veld in segment(velden kunnen overlappen)

28.11 DBD (data base description)

CREATE

```
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
```

28.12 Extern niveau: PCB

= beschrijving van het PCB d.w.z. van de sensitieve segmenten en velden(ook met macro-taal).

- PCB-macro: bevat naam van de DBD en de lengte van de sam. Sleutel die het langst is
- SENSEG-macro: voor elk sensitief segment, gebruik van naam segment zoals deze voorkomt in DBD
- PROCOPT: hiermee aangeven voor elk segment welke bewerkingen zijn toegelaten
- SENFLD-macro: voor meerdere sensitieve segmenten dezelfde processing options geven

GIRDAK:

- G:For Get
- I :For Insert
- R:For replace
- D:for delete
- A:For all option (G,I,R,D)
- K: key-sensitief(segment overnemen omdat men een of meerdere van zijn kinderen nodig heeft. Alleen in de samengestelde sleutel merkt men dat dit segment overgenomen is, verdere bewerkingen zijn niet mogelijk).

28.13 Gegevensmanipulatietaal: DML

28.14 Koppeling met de gasttaal

- **JCL**: **J**ob **C**ontrol **l**anguage = geeft je door welke PCB's gebruikt worden in het programma.
- **PCB-masker**: wordt gebruikt om informatie vanuit de gegevensbank door te geven aan toepassingsprogramma. Voor elke gegevensbank die gemanipuleerd wordt het programma een PCB-masker nodig.
- 4 soorten parameters (CBLTDLI = **C**o**b**o**l** **t**ask **DL1)**
  - Functiecode (verplicht) = geeft aan wat men wil doen met gegevensbank(positie = pointer)
  - PCB-masker(verplicht) => linkage section
  - Input-output area(verplicht) => WSS
  - Zoekargumenten(optioneel) = aangeven welk segment men zoekt(bv =,>,<, <>, ..) => WSS

PCB-masker

```
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.
```

28.15 Functiecode

- **GU** (get unique): overloopt de boom hiërarchisch en stopt bij het eerst gevonden segment dat voldoet aan de meegeven voorwaarde. (pointer => volgende segment)
- **GN**(get next): houdt rekening met pointer en begint op huidige plaats in de boom (aangegeven door pointer). Stopt wanneer segment is gevonden dat voldoet aan voorwaarde. (pointer => volgende segment)
- **GNP**(get next within parent): beginnen vanaf pointer maar alleen in afhankelijke segmenten van parent.
- **GHU – GHN – GHNP**: zijn hetzelfde als bovenstaande maar zijn verplicht als de volgende instructie een delete of replace is van het betrokken segment.

28.16 GU

Voorbeelden

```
GU  DOKTER(DFNAAM = DEPRET)
    PATIENT

GU  DOKTER(DFNAAM = DEPRET)
    PATIENT(PFNAAM = MAES)
    CONSULT(CDATUM = 16092005)
```

28.17 GN

Voorbeelden

```
GN  PATIENT

GN  PATIENT(PFNAAM = MAES)
```

28.18 GNP

```
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
```

28.19 GHU-GHN-GHNP

Voorbeeld

```
GHU DOKTER (DNR = 12)
DLET
```

28.20 DML-functiecode

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

28.21 ISRT

Voorbeeld

```
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
```

28.22 DLET – REPL

Voorbeelden

```
GHU DOKTER (DNR = 12)
DLET


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

28.23 Zoekargumenten

= worden gebruikt om aan te geven welk segment men zoekt.

Segmentnaam [(conditie)]

Veldnaam operatiecode waarde

29 NetwerkDatabank

29.1 Opstellen van het netwerkmodel

29.2 Bouwstenen

Recordtypes ~ eniteitstype uit ERD.

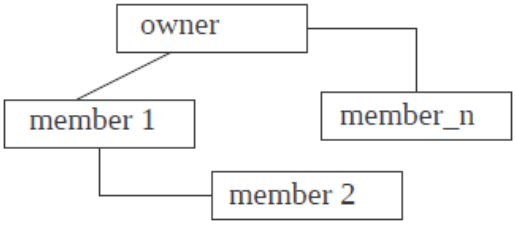
Recordinstantie = één concreet voorkomen van een record van dat type.

Settypes = komen overeen met 1::n verbanden tussen enteitstypes in het ERD

Setinstantie = voorkomen van een ownerrecord met zijn memberrecords

1:n verband

- Owner-recordtype
- Member-recordtype



29.3 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

29.4 Terminologie

- **Recordtype** = beschrijft de gegevens in de gegevensbank
- **Settype** = geven relaties tussen recordtypes aan
- **Record key** = verzameling van de velden die gebruikt worden om rechtstreeks recordinstantiaties op te zoeken

29.5 Verschil hiërarchisch – netwerk

- Bij netwerkgegevensbanksystemen:
  - Kan een recordtype member zijn in meerdere settypes
  - Kunnen meerdere settypes bestaan tussen deze recordtypes
  - Kunnen members bestaan zonder owners

29.6 3-schema architectuur

- Conceptueel niveau : schema (schema DDL) bevat beschrijving van alle record- en settypes
- Intern niveau : intern schema (DSDL = **D**ata **S**torage **D**escription **L**anguage)

- Extern niverau : subschema (subschema DDL) alleen het deel beschreven dat nodig is voor applicatie

29.7 Conceptueel niveau: schema

- Naamgeving
- Beschrijving van de recordtypes
- Beschrijving van de settypes
  - Naamgeving
  - Specificatie owner en member
  - Set order
  - Set insertion optie
  - Set retention optio
  - Set selectie clausule

29.8 Schema DDL – naamgeving

Het schema moet een naam krijgen

Bv. SCHEMA NAME IS schema-naam

SCHEMA DOKTERSPRAKTIJK

29.9 Schema DDL-recordtypes

- Het recordtype moet een naam krijgen

RECORD NAME IS recordnaam

- Per recordtype wordt de record key en de velden gespecificieerd

KEY keynaam DUPLICATES ARE NOT ALLOWED (= unieke identificatie, Primary Key)
02 veldnaam veldbeschrijving

29.10 Schema DDL – settypes

- Het settype moet een naam krijgen

Bv. SET NAME IS setnaam

- Specificatie owner en member

ONWER IS ownernaam
MEMBER IS membernaam

- Set order

ORDER FOR INSERTION IS
SORTED BY DEFINED KEYS
FIRST
LAST
NEXT
PRIOR

= geeft aan welke de logische volgorde van de members is binnen een setinstantie

KEY IS ASCENDING keynaam
DESCENDING

= indien logische volgorde gestorteerd is op sleitel moet men aangeven in welke volgorde de nieuwe memebers opgenomen moeten worden

DUPLICATES ARE NOT ALLOWED
FIRST
LAST

= geeft aan of er duplicaten van members mogelijk zijn op de gespecifieerde velden, zo ja of deze voorop of achteraan opgenomen moeten worden.

- Set insertion optie = aangeven of een member-recordtype bij toevoeging meteen opgenomen moet worden in een setinstantie.
- Manual = niet verplicht
- Automatic =direct opnemen.

INSERTION IS AUTOMATIC
MANUAL

- Set retention optie = geeft aan of memberrecords binnen een setinstantie mogen losgekoppeld worden van hun owner.
- Fixed = altijd bij dezelfde owner
- Mandatory = loskoppelen maar daarna aan andere owner koppelen
- Optional = kan op elk moment losgekoppelt worden

RETENTION IS FIXED
MANDATORY
OPTIONAL

- Set selectie clausules = aangeven op welke manier de owner geïdentificeerd wordt.

SET SELECTION IS THRU setnaam
OWNER IDENTIFIED BY SYSTEM
APPLICATION
KEY veldnaam

Vb.

```
SET dokter-consultatie
OWNER is dokter
  ORDER FOR INSERTION IS LAST
MEMBER IS consultatie
  INSERTION IS automatic
  RETENTION IS fixed
SELECTION IS THRU dokter-consultatie KEY Dnr
```

29.11 Extern niveau: subschema

= velden, record- en settypes worden gedefinieerd die nodig zijn voor een bepaalde applicatie. Men kan hierin ook recordtypes, settypes en velden hernoemen.

- Velden en type van één bepaalde applicatie
- Syntax:
  - Naam: SS subschema-naam WITHIN schema-naam
  - Hernoemen van velden: AD **A**lias **D**efinitions
  - De relevante record- en settypes:
    RECORD SECTION
    RECORD NAME IS recordnaam
    SET SECTION
    SET NAME IS setnaam

29.12 DML

- **User work area (UWA)** = plaats waar gegevens afkomstig van gegevensbank in geplaatst worden
- **Run unit** = voor elk programma één run unit, per run unit één UWA.
- **DB-status** = per run unit één DB-status, geeft aan of vorige DMLinstructie al dan niet succesvol afgelopen is.
- **Currency indicator** = krijgen waarde door het uitvoeren van een DMLinstuctie.

- **Current of run unit(CRU)** = verwijzigen naar de laatst gerefereerde record
- **Current of record pointer(CRP)** = verwijzing naar laatst gerefereerde record van recordtype
- **Current of set pointer (CSP)** = verwijzing naar het laatste gerefereerde record van het owner of van het member recordtype van het betreffende settype.

29.13 Koppeling met gasttaal

- Het subschema moet geassocieerd worden met het applicatieprogramma. Dit is nodig voor opbouw van User Work Area.
  DB SUBSCHEMA subschema-naam
  WITHIN schema-naam

29.14 DML instructies

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

29.15 DML- FIND en GET

- FIND positioneert de currency indicatoren
- Copieren van inhoud naar CRU d.m.v
  FIND CURRENT record-naam
  WITHIN set-naam
- GET haalt de informatie op

2 groepen bij de FIND-instructie:

- Opzoeken van een record van een recordtype
  FIND ANY record-naam USING veldnaam
  FIND DUPLICATE record-naam USING veldnaam
- Opzoeken van een record van een settype:
  - Zoeken naar een owner
    FIND OWNER WITHIN set-naam
  - Zoeken naar members binnen een setinstantie
    FIND FIRST record-naam WITHIN set-naam
    NEXT
    FIND record-naam WITHIN set-naam USING veld

29.16 DML – wijzigen van velden

- Wijzigen van velden binnen een record-instantie: MODIFY

29.17 DML – weglaten van een record

- Weglaten van een recordinstantie uit de gegevensbank: ERASE
- Alle members weglaten: ERASE ALL

29.18 DML – toevoegen van records

- STORE recordnaam

29.19 DML – reconnect

- Een memberrecord van owner veranderen:
  RECONNECT recordnaam WITHIN setnaam

29.20 DML – zwevend record associëren

- Het associëren van een zwevend member-record (record van het memberrecordtype dat op het moment niet opgenomen is in een setinstantiatie van het beschouwde settype):
  CONNECT recordnaam TO setnaam

29.21 DML – loskoppelen van members van hun owner

DISCONNECT recordnaam FROM setnaam

29.22 DML – RETAINING optie

- Sommige currency indicastoren niet wijzigen (met uitzondering van CRU):
  RETAINING RECORD CURRENCY … setnaam

29.23 Oefeningen

VIND DOKTER VAN EEN PATIËNT VIA CONSULATIE

```
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 »
```

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.

2.

```
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 »
```

30 Relationele vergelijking

Hierarchische structuren in een relationeel model => conceptueel

```
Vormen:
* rechtstreekse omzetting (cf IMS)
* recursief (cf CTE)
* nested sets (niet kennen)
* bomen (niet kennen binnen RDBMS)
* (xml ea)
*..
```

Netwerk structuren in een relationeel model => conceptueel

```
Abstract :: deelverzamelingen

hierarchisch
C
Netwerk
C
Relationeel
```

```
relationeel (database):
tabellen, rijen, kolommen
netwerk (schema):
recordtypes (+settypes),records,fields
hierarchisch (database description):
segmenten, records, fields
*sequentieel
```

31 DBMS-soorten

31.1 Object related database management system – ODMS

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

	Simpele gegevens	complexe gegevens
query	2	4
Geen query	1	3

- **Simpele gegevens zonder queries**
  Als men simpele gegevens (geen video, audio, enz.) , gewoon woorden, getallen, enz. gebruikt, waar men geen ondervragen op doet, volstaat het om een tekstverwerker te gebruiken. Het DBMS dat men dan gebruikt is het bestandssysteem van het OS.
- **Simpele gegevens met queries**
  Bv. Gegevensbank met gegevens rond het personeel dat in een firma werkzaam is. Hiervoor gebruiken we tabellen. Vervolgens kunnen we deze tabellen ook ondervragen via queries.
  Hiervoor gebruikt men een relationele gegevensbank.
- **Complexe gegevens zonder queries**
  Kijk voorbeeld cursus pg 58. Aangezien we werken met complexe gegevens maar niet met queries, kan men best gebruik aken van persistent opslaan in programmeertalen.
- **Compelexe gegevens met queries**

Kijk voorbeeld cursus pg 59. Voor deze situatie gaat men gebruik maken van een object-relationeel of object-georiënteerde gegevensbank.

- ORDBMS: belangrijkste speller
- ODMS: specialisten – niche markt

32 ODMS

32.1 Geënt op OOPL concepten

- Nieuw? Smalltalk(1972)
- Kenmerken:
  - Objecten
    - State vs Behaviour (zijn tov kunnen)
    - Object identifier (OID) = unique and immutable
  - Complexe types en structuren
    - Atomic, struct(tuple)
    - Collection(set, list, bag, array, dictionary(kvv))
  - Inkapseling
  - (tijdelijk(transient) vs persistent)
  - Overerving
  - Polymorfisme(operator overloading)

32.2 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
- Overving : Extends
- ..

Vb.

```
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)
}
```

32.3 ODMS: kolom objecten

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

32.4 ODMS: geneste objecten

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

32.5 ODMS: overerving

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

32.6 Object query language – OQL

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

REEKSEN;

STUDENT1.Adres;

select      distinct S.Ssid
from        S in REEKS1.heeftStudenten;
```

33 ORDBMS

Kernmerken:

- Uitbreiding van de basis datatypes
- Complexe objecten
- Overerving

        - Overerving van gegevens
        - Overerving van types
- Regels(bv)
    - Update-update regel
    - Query-update regel
    - Update-query regel
    - Query-query regel
    - …

33.1 Zelfgedefinieerde datatypes

= complexere en meer specialistische datatypes

33.2 Zelf definiëren van datatypes

- CREATE TYPE: creëert datatype
- Kenmerken:
    - Gebruikt waar basis-datatypes gebruikt worden
    - Hebben geen populatie
    - Zijn niet manipuleerbaar met een select
    - Heeft een statische inhoud(aantal waarden)
- Voordeel: vergelijking moet zinvol zijn (~strong typing)

- DROP TYPE: verwijdert datatype

Vb.

```
Create type spelersnr as integer;
Create table spelers
   (spelersnr   spelersnr   not null,
    … );

Drop type spelersnr;
```

33.3 Toegang tot datatypes

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

```
grant usage
on type geldbedrag
to Jim
```

- Verwijderen van machtiging:

```
revoke usage
on type geldbedrag
to Frank
```

33.4 Casting van waardes

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

- Vb.

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

Bedrag is van type geldbedrag »

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

insert into boetes (betalingsnr, spelersnr, datum, bedrag)
values (betalingsnr(12), spelersnr(6), '1980-12-08',
                geldbedrag(100.00))
```

33.5 Zelf definiëren van operatoren

- Voorafgaandelijke opmerkingen:
  - Operatoren zijn toegevoegd voor het gemak
  - Bij elk basis-datatype horen operatoren
  - Operatoren hebben een betekenis al naargelang het datatype
  - Definiëren van operatoren op zelf definieerde datatypes ~ operaties op onderliggend type.
  - Definiëren van operatoren in de vorm van scalaire functies

Vb.

```
bedrag1 + bedrag2   :   ongeldig

=>  decimal(bedrag1) + decimal(bedrag2)

    create function « + » (geldbedrag, geldbedrag)
       returns geldbedrag
       source « + » (decimal(), decimal())
```

33.6 Opaque-datatype

= datatype dat niet afhankelijk is van een basisdatatype

⇨ Definiëren van functies om hiermee te werken is noodzakelijk

Vb.

```
create type tweedim
        (internallength = 4) ;


-- base types in postgresql
```

33.7 Named row-datatype

= groeperen van waarden die logisch bij elkaar horen

Vb.

```
create type adres as
   (straat          char(15)  not null,
  huisnr            char(4)             ,
  postcode          char(6)   ,
  plaats            char(10)  not null) ;
-- postgresql : geen not null, types kunnen composite,
  enum of base zijn
```

```
create table spelers
      (spelersnr      integer   primary key,
      …
      woonadres       adres                 ,
      postadres       adres                 ,
      vakantieadres   adres                 ,
      …                                    )

select spelersnr, woonadres
from spelers
where woonadres.plaats = 'Leuven'
```

33.8 Unnamed row-datatype

= groeperen van waarden die logisch bij elkaar horen zonder een naam te geven

```
create table spelers
(spelersnr          smallint    primary key,
 …
 woonadres          row  (straat     char(15) not null,
       huisnr       char(4)               ,
       postcode char(6)                   ,
       plaats       char(10)    not null),
telefoon            char(10) ,
…                 ) ;
- postgresql : unnamed rows worden enkel als input
 toegelaten
```

33.9 Getypeerde tabel

= een datatype toekennen aan een tabel

⇨ Eenvoudig om gelijkende tabellen te creëren

Vb.

```
create type t_spelers as
          (spelersnr  integer      not null,
 naam   char(15)    not null,
 …
 bondsnr                char(4));
create table spelers of t_spelers
(primary key spelersnr) ;
-- postgresql : .. like parent_table ..
```

33.10 Integriteitsregels op datatypes

= beperking leggen op toegestane waardes

Vb.

```
create type aantal_sets as smallint
check (value in (0, 1, 2, 3));

create table wedstrijden
    (wedstrijdnr          integer       primary key,
 teamnr    integer        not null,
 spelersnr integer        not null,
 gewonnen                 aantal_sets   not null,
 verloren  aantal_sets    not null) ;
-- Wat is een DOMAIN in RDBMS?
```

33.11 ALTER TYPE

= wijzigen van datatype

Vb.

```
alter type aantal_sets as smallint
        check (value between 0 and 4)
```

Als conditie strikter is, wijzigingen geweigerd tot al deze waarden in de tabel aangepast zijn.

33.12 Sleutels en indexen

- Is volledig analoog bij zelf gedefinieerde datatypes
- Bij named row-datatypes:
  - Op de volledige waarde
  - Op een deel ervan

33.13 Overerving

= alle eigenschappen van één datatype worden overgeërfd door een ander(supertype en subtype)

Vb.

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

create type buitenlands_adres as
    (land       char(20)  not null) under adres ;
-- postgresql : ..of rechtstreeks of like in base type..
```

```
create table spelers
    (spelersnr          smallint    primary key,
     …
     woonadres          adres                   ,
     vakantieadres      buitenlands_adres       ,
     …                                          );
```

33.14 Koppelen van tabellen

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

Vb.

```
select ref(spelers)
from spelers
where spelersnr = 6 ;
-- postgresql : SELECT oid FROM ..
```

- REF: om tabellen te koppelen

```
create table teams
(teamnrsmallint  primary key,
 speler  ref(spelers)          not null,
 divisie          char(6)      not null)

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

select teamnr, speler.naam
from teams
```

```
create table spelers
    (spelersnr  smallint     primary key,
     naam       char(15)     not null    ,
     vader      ref(spelers)             ,
     moeder     ref(spelers)             ,
     …
     bondsnr    char(4)                  )

select spelersnr, vader.naam
from spelers
where moeder is not null
```

33.15 Voor- en nadelen

- Voordelen:
  - Altijd het juiste datatype bij de refererende sleutel
  - Indien primary keys breed zijn, bespaart het werken met reference-kolommen opslagruimte

- Bij wijzigingen van primary keys wordt geen tijd verloren door het wijzigen van de refererende sleutel
- Bepaalde selects worden eenvoudiger

- Nadelen:
  - Bepaald 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

33.16 Collecties

= verzameling waardes in één cel

Vb.

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

```
select spelersnr
from spelers
where cardinality(telefoons) > 2

select TS.telefoons
from   the (select telefoons
            from spelers)  as TS
order by 1

- kijk ook de functies van het product na
```

33.17 Overerving van tabellen

= alle eigenschappen van één tabel worden overgeërfd door een andere tabel

Beperkingen:

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

```
create type t_spelers as
 (spelersnr        smallint    not null,
  naam             char(15)    not null,
  …
  bondsnr          char(4))

create type t_oude_spelers as
(vertrokken        date        not null) under t_spelers

create table spelers of t_spelers
(primary key spelersnr)

create table oude_spelers of t_oude_spelers under spelers

-- blauwe boek
```

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

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

SELECT *
FROM   (ONLY) spelers;

-- let op met inserts ..
-- postgresql
```

33.18 RULES

- Gaan verder dan triggers: SELECT, INSERT, UPDATE, DELETE
- FK's >> triggered
- Updateble views
  - Rules of triggers
- Maar voorzichtig, systeemlogica

Vb.

```
CREATE RULE "NeKeerIetsAnders" AS
    ON SELECT TO wedstrijden
        WHERE   spelersnr = 7
    DO INSTEAD
       SELECT 'eerst drie toerkes rond tafel
lopen                          en dan nog eens
proberen';
```

34 NOSQL

= **not only SQL (niet relationeel)**

34.1 Mongo db

- Binaire JSON
- CRUD
- Index
- Aggregation
- Replication
- Sharding
- NeedsMapreduce

34.2 KV

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

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

   -- KeyValuePair
   -- index op key maken
```

34.3 ACID(transactions)

- Atomicity: cf Boolean
- Consistency : cf state
- Isolation: cf concurrency
- Durability: cf commit, levensduur en select
- DBMS:
- +RDBMS
- -NOSQL…

35 Database keuze

35.1 Pointers and choices

35.1.1 SqlStandard

- Ideal world vs reality
- Know your options
- What is important?
- Make your choices

35.1.2 Software-requirements

- Standalone vs client/server
- Single user vs multi user
- Close to the SQL standard or ...
- Quick and easy or...
- Durable or ...
- GPL-alike or ...
- Official Support or ...
- Resources, OS, ...
- ...

Some gpl-alike examples: SmallSQL, SQLite, GSQLDB, Firebird, Derby, MySQL, PostgreSQL,...

35.1.3 Simple classification

	Standalone	Network
Office		
Small and Medium		
Enterprise		

35.1.4 Mistakes

- Too many database (abstract levels) .. vs Time
- Know your data (structure) [requirements]
- DB != spreadsheet; skill of it's own
- Third normal form vs common sense
- (All) the application logic in the database
- Backup, replication
- Version control
- Use the tools
- The hammer to fix the plumbing

35.1.5 What about frameworks

- Data live expectancy compared to the programming environment

36 DBMS

- Know your tools/options
- Context/situation
- handCrafted vs ManHours

- Humans vs Mathematics
- Data independence
- Standardization
- ACID vs Performance
- Voor bijna elke klein bedrijf tot KMO is een RDBMS prima

36.1 Toekomst van SQL

- SQL : databasetaal van relationele DBMS
- Integratie met talen als JAVA ?
- Embedded SQL naar achtergrond ⇨ CLI's
- Object-relationele concepten !
- Invloed van Datawarehouses, OLAP … !
- Samenwerking met XML door extensies !
- Verbeteringen van de performance !
- SQL/MED
- Overdraagbaarheid neemt af !

37 Frequently made mistakes

37.1 Joins en tellen

```
Joins
When u have more than one table in your from,
9 times of 10 it is necessary to join the selected tables;
otherwise u take the cartesian product of these tables
without filtering them for nonsense data.
```

```
More than 1 join expression
Use brackets when joining
more than two tables not using the where clause
eg.     SELECT          *
        FROM    (A INNER JOIN B USING(K))
                LEFT OUTER JOIN C USING (K);
```

```
Pitfall:
When joining more than 2 tables
and using aggregate functions
and or grouping sets;
these can kind of queries
can lead to an overestimating
(, in a similar way
as taking the cartesian product)
```

37.2 Logical operators

```
SELECT  *
FROM    spelers s
WHERE   (s.spelersnr <= 10
OR      s.spelersnr > 10)
AND     geslacht = 'M';

 spelersnr |      naam       | voorletters
**---------+-----------------+-------------
(9 rows)
ok
```

Haakjes rond OR Operator!