VDAB
TRAINING & OPLEIDING

Vlaamse Dienst voor Arbeidsbemiddeling en Beroepsopleiding

SQL

(MET MYSQL)

Inhoudsopgave

1 WAT IS SQL ?

Structured Query Language is taal om een relationele database te beheren en te onderhouden. Databasegegevens worden beheerd door een afzonderlijk systeem: Relational Database Management System. De RDBMS zelf is verantwoordelijk voor de structuur, het bewaren en ophalen van gegevens. Met behulp van SQL geven wij opdrachten aan de RDBMS. De SQL-instructies worden onderverdeeld in vier categorieën: Data Manipulation Language (DML), Data Definition Language (DDL), Data Control Language (DCL) en Transaction Control (TC).

De DML wordt gebruikt om gegevens te selecteren, toe te voegen, te wijzigen en te wissen en omvat de instructies select, insert, update en delete.

De DDL bestaat uit de creatie en onderhoud van databases, de tabellen, views en indexes en omvat de instructies create, drop en alter.

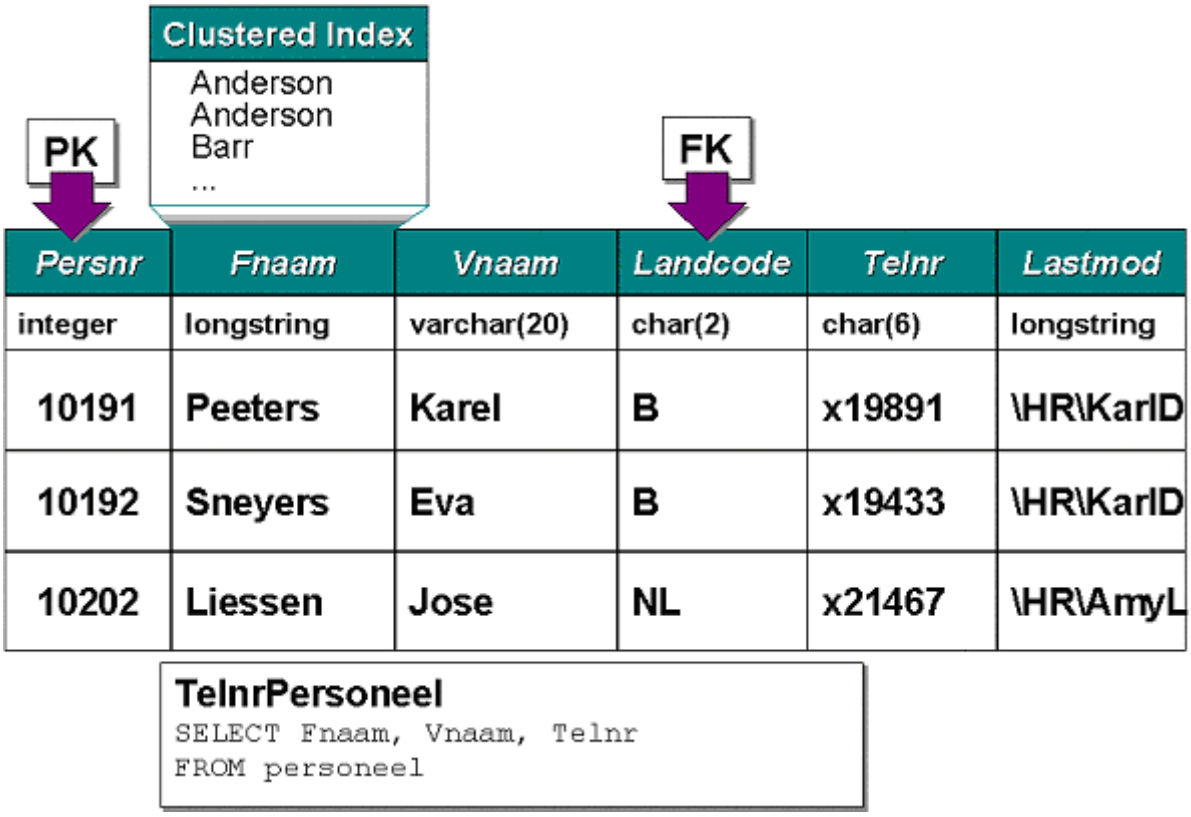
De DCL is verantwoordelijk voor de beveiliging van de gegevens. Voorbeeld: De personeelsdirecteur heeft toegang tot de lonen van de personeelsleden, de secretaresse heeft enkel toegang tot de namen en personeelsnummer. Deze instructies komen niet aanbod.

Transaction control bestuurt de transacties binnen de server. Deze instructies worden niet behandeld in deze cursus.

SQL kan gebruikt worden in programmeertalen (Cobol, Visual Basic, Perl, C …), in tools (report writers, form generators, application programs).

SQL is een gestandaardiseerde taal en dus bruikbaar voor elke soort relationele database. Toch kunnen aan de RDBMS nog extra instructies toegevoegd zijn.

2 BEGRIPPEN



- Tabellen (Tables)

  Na de normalisatie bekom je verschillende entiteiten. Deze entiteiten vormen de tabellen van de database.

- Attributen (Columns)

  De attributen van elke entiteit vormen de kolommen (velden) van de tabel.

- Rijen (Tuples)

  De gegevens worden in rijen opgeslagen. In de literatuur over relationele databases spreekt men eerder over tuples.

- Primary Key (PK)

  De sleutel van de tabel, deze bepaalt de rij van de tabel. Elke rij heeft een unieke sleutelwaarde. Vb. personeelsnr, artikelnr, …

- Foreign key (FK)

  Dit is een attribuut in de tabel die verwijst naar een andere tabel waar deze attribuut het sleutelveld is. Vb. klantnr in tabel fakturen verwijst naar het klantnr in de tabel klanten. Het klantnr van de tabel klanten is de PK van de tabel.

- Views

  Is een select instructie die bewaard wordt. Wanneer we de view gebruiken wordt de select instructie opnieuw uitgevoerd. Op een view kan je dezelfde bewerkingen uitvoeren als bij een tabel, deze bewerkingen worden dan doorgevoerd naar de onderliggende tabellen waaruit de view samengesteld is.
  Voorbeeld:
  TelnrPersoneel is een view die een aantal velden selecteert van de tabel Personeel. Wanneer je een rij toevoegt aan de view wordt er automatisch een rij toegevoegd aan de tabel Personeel.

- Index

  Op de primary key wordt automatisch een index geplaatst. Deze index kan je het best vergelijken met een inhoudstafel waarbij elke lijn een verwijzing bevat naar de rest van de uitleg. Deze index wordt gebruikt om snel gegevens op te zoeken. Je kan nog bijkomende indexen plaatsen om snel informatie te zoeken op die kolom(men).

3 HET PROGRAMMA

Deze versie van de cursus (*'SQL met MYSQL'*) gebruikt de database server **MySQL** 5.x met de GUI Tools om de taal SQL te leren.

In de standaardversie van de cursus (gewoon *'SQL'* genoemd) wordt het programma *SQLTryOut* gebruikt.

Ga na bij je coach of je deze versie, zoniet de standaardversie moet gebruiken.

Voor de MySQL server zijn er twee mogelijkheden:

- ofwel gebruik je een centraal geïnstalleerde MySQL server (vraag je coach of er een MySQL server in het centrum gebruikt wordt)
- ofwel moet je zelf op je eigen PC MySQL installeren.
    - mensen in het traject web / web-ontwikkelaars installeren de XAMPP-omgeving. ...
    - mensen in het traject Java / Java-ontwikkelaars installeren enkel de gratis 'MySQL dat kan door de omgeving XAMPP te installeren. Voor de installatie van XAMPP verwijzen we naar de vdab cursus MYSQL

Gebruikt MySQL standaard SQL?

Ja, MySQL past de SQL-99 standaard toe. Er zijn enkele verschillen, waar wij echter geen rekening mee te hoeven houden.

Verschilt de MySQL syntax in deze cursus met de standaard SQL cursus?

Ja, er zijn enkele verschillen. Een overzicht vind je achteraan in Appendix A.

3.1.1 Installatie van de MySQL Community Server (Java-ontwikkelaars)

- ga naar http://dev.mysql.com
- via de tab 'downloads' bovenaan de pagina kun je de meest recente versie van de MySQL Community Server downloaden:
    - halverwege de pagina kies je het correcte OS (= MS Windows)
    - kies vervolgens voor de versie 'Windows(x86, 32-bit), MSI Installer'
    - op de pagina 'select a mirror to start downloading ...' kies je voor 'no thanks, just take me to the downloads!' (net onder het inlogkader)

- kies voor ‘http’ en bewaar het bestand ‘mysql-x.x.x-win32.msi’ op het bureaublad

- om de server te installeren, dubbelklik je het bestand ‘mysql-x.x.x-win32.msi’ op het bureaublad ; dit start de installatie-wizard op.
  - kies: 'complete install'
  - 'launch configuration' R
  - kies: 'standard'
  - user: root / password: vdab

3.2 Installatie MySQL GUI Tools

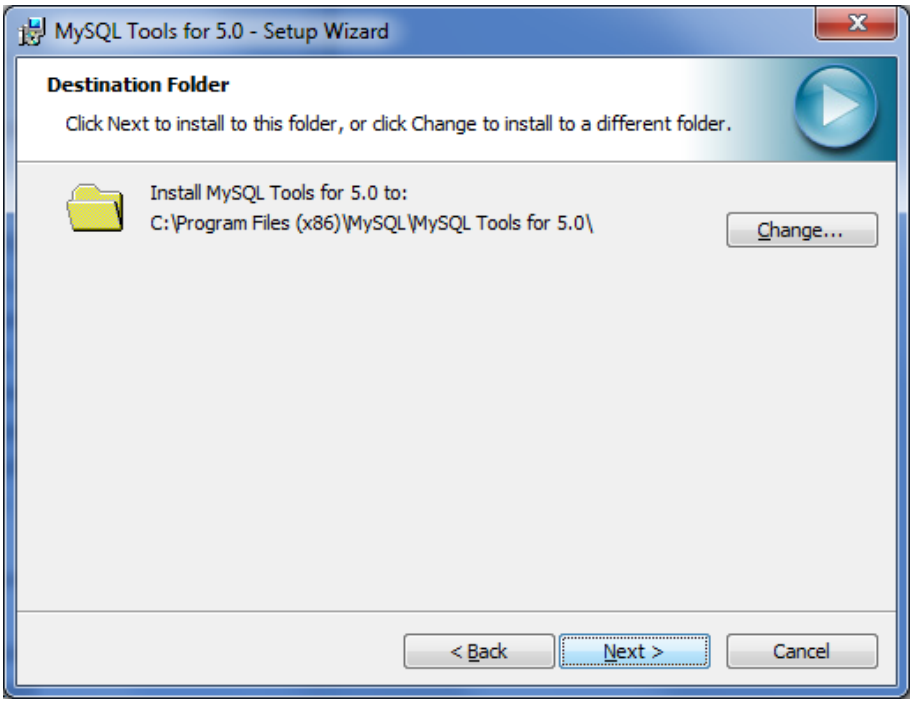
Voor deze cursus gebruiken we de 'oude' collectie GUI Tools (*MySQL Administrator, MySQL Query browser, MySQL Migration Toolkit*) en niet de nieuwere geïntegreerde *MySQL Workbench*. De reden daarvoor is dat deze laatste nog teveel bugs heeft.

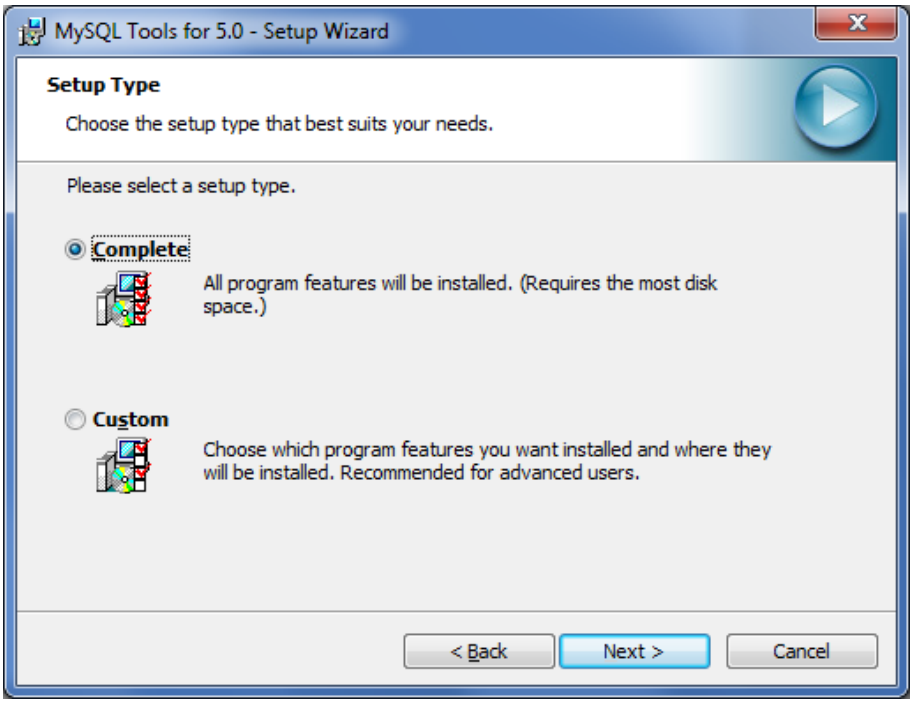
Je kan de GUI Tools downloaden van

http://dev.mysql.com/downloads/gui-tools/5.0.html

of ze vragen aan je coach, want die heeft ze ergens opgeslagen.

Dubbelklik het bestand *mysql-gui-tools-5.0-r17-win32.msi* (voor een Windows omgeving) en volg de installatieprocedure:





Na de installatie vind je in het start menu het nieuwe menu-item "MySql" met daarin de verschillende tools.

3.2.1 MySQL Administrator

In tegenstelling tot een lokaal gebruikersprogramma zoals MS Access, is MySQL een serverprogramma. Enkele duidelijke verschillen met Access zijn:

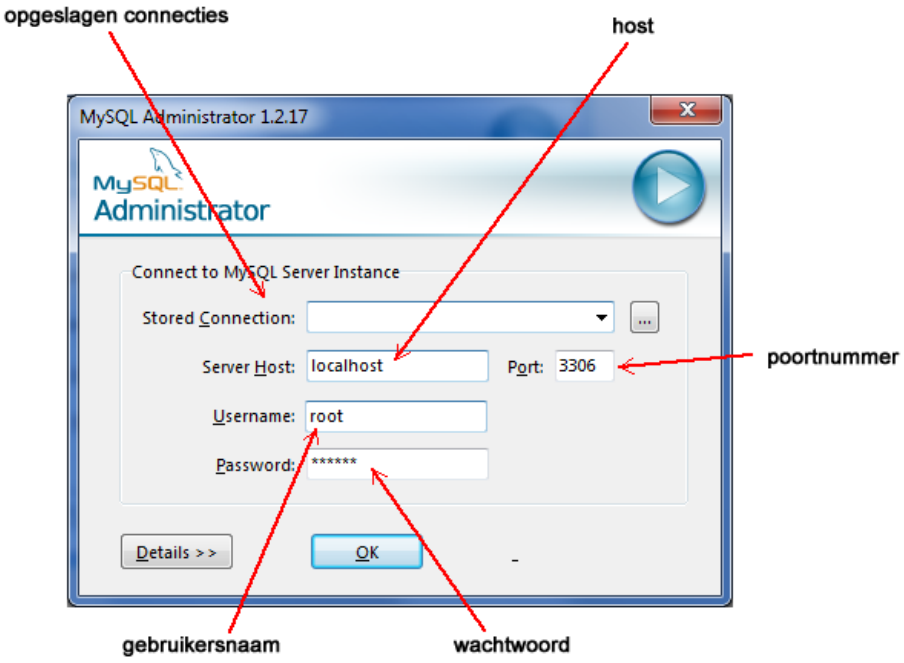
- de server kan op een andere computer staan, zelfs ergens op het net
- je kan er met meerdere gebruikers tegelijk op aanmelden, zelf met duizenden tezelfdertijd

- het heeft geen ingebouwde *User Interface*, dus geen makkelijke menuutjes om formulieren, rapporten en queries te maken, de server begrijpt enkel SQL. Voor de User Interface moet je zelf zorgen…

Om hier een database te gebruiken, volstaat een dubbelklik op een bestand dus niet, we moeten eerst op de server aanmelden.

Aanmelden langs de voordeur kan je doen met *MySQL Administrator*, een interface om administratieve taken te doen aan de server. Je kan er databases mee aanmaken, wissen, gebruikers aanmaken en ook doorschakelen naar *MySQL Query Browser*.

In de marge vertellen we hier dat een andere populaire interface voor MySQL, *PHPMyAdmin* is, die ook vervat is in de XAMPP omgeving.



Om aan te melden moet volgende velden invullen:

- ***Server host***: op welke MySQL server meld je aan?

  typ "localhost" om aan te melden op je eigen MySQL server (als je lokaal werkt). Vraag eventueel aan je coach de hostname van een centrumserver (vb. "*dbserver.wevelgem.be*", "*data.intra.local*","*xena.skills.be* ", …)

- ***Port***: het poortnummer van server

  Normaal "3306", laat ongewijzigd

- ***Username***: de gebruikersnaam waarmee je aanmeldt.

  Als je MySQL net geïnstalleerd hebt op je eigen machine zal dat "*root*" zijn. Als je je aanmeldt op een centrumserver, vraag de username aan je coach, dan kan dat bv. "web" of een andere naam zijn.

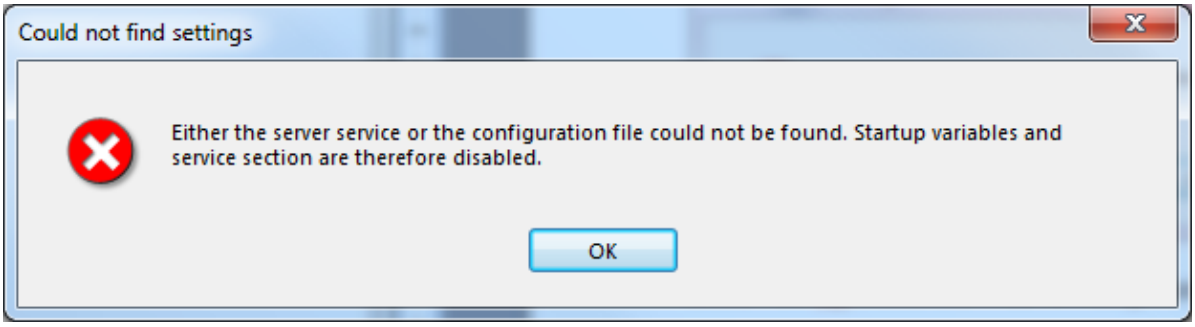
- ***Password***: het wachtwoord dat hoort bij de gebruikersnaam

Als je MySQL net geïnstalleerd hebt op je eigen machine, dan is je een wachtwoord gevraagd tijdens de installatie. Weet je nog wat het was? Misschien heb je het niet ingevuld en kan je dit vak leeglaten.
Als je je aanmeldt op een centrumserver, vraag het wachtwoord voor de *Username* aan je coach, bv. "web" of iets anders.

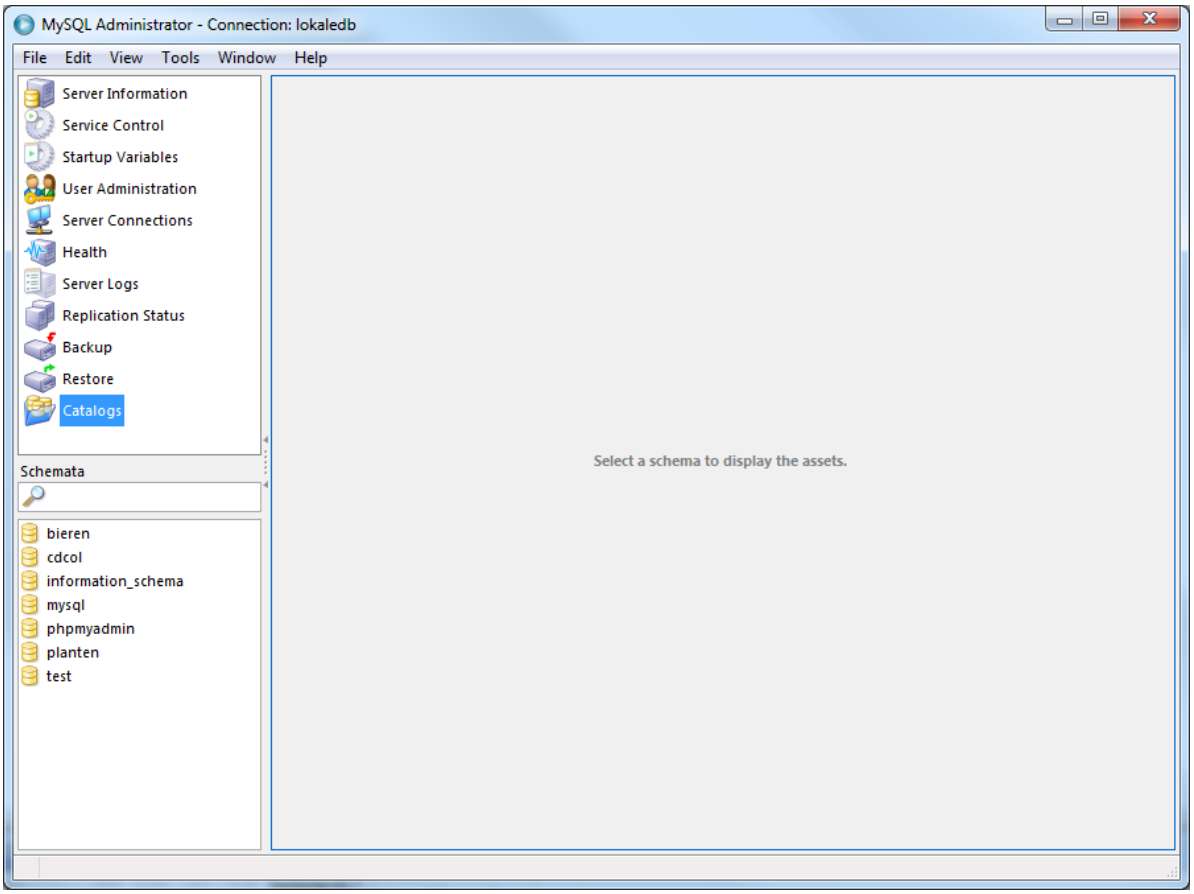
Gebruikers hebben verschillende rechten. De "root" gebruiker heeft alle rechten. De gebruiker "web" heeft waarschijnlijk minder rechten en kan daardoor niet alles **doen** en niet alles **zien**.

- ***Stored connection***: door op het knopje … te klikken kan je verschillende connecties gaan opslaan en later weer gebruiken.
  Je hoeft dit niet in te vullen.

Meld je aan door op OK te klikken. Indien je onderstaande foutboodschap te zien krijgt klik dan op OK.



Je komt uiteindelijk in de “MySQL Administrator” terecht.



Klik op CATALOGS. Je ziet de aanwezige "databases". Hier worden die ook "Schema's" genoemd.

Je ziet (misschien) een aantal schema's *zoals information_schema, mysql, phpmyadmin*, die systeemdatabases zijn. In bovenstaand beeld zijn de db's *bieren* en *planten* reeds aanwezig, bij jouw eigen installatie niet.

3.2.2 De oefendatabases installeren

Als je een eigen installatie gedaan hebt, moet je nog de meegeleverde databases installeren. Je hebt de bestanden *bieren_origineel.sql, plantv_origineel.sql* en *presiden_origineel.sql* nodig.

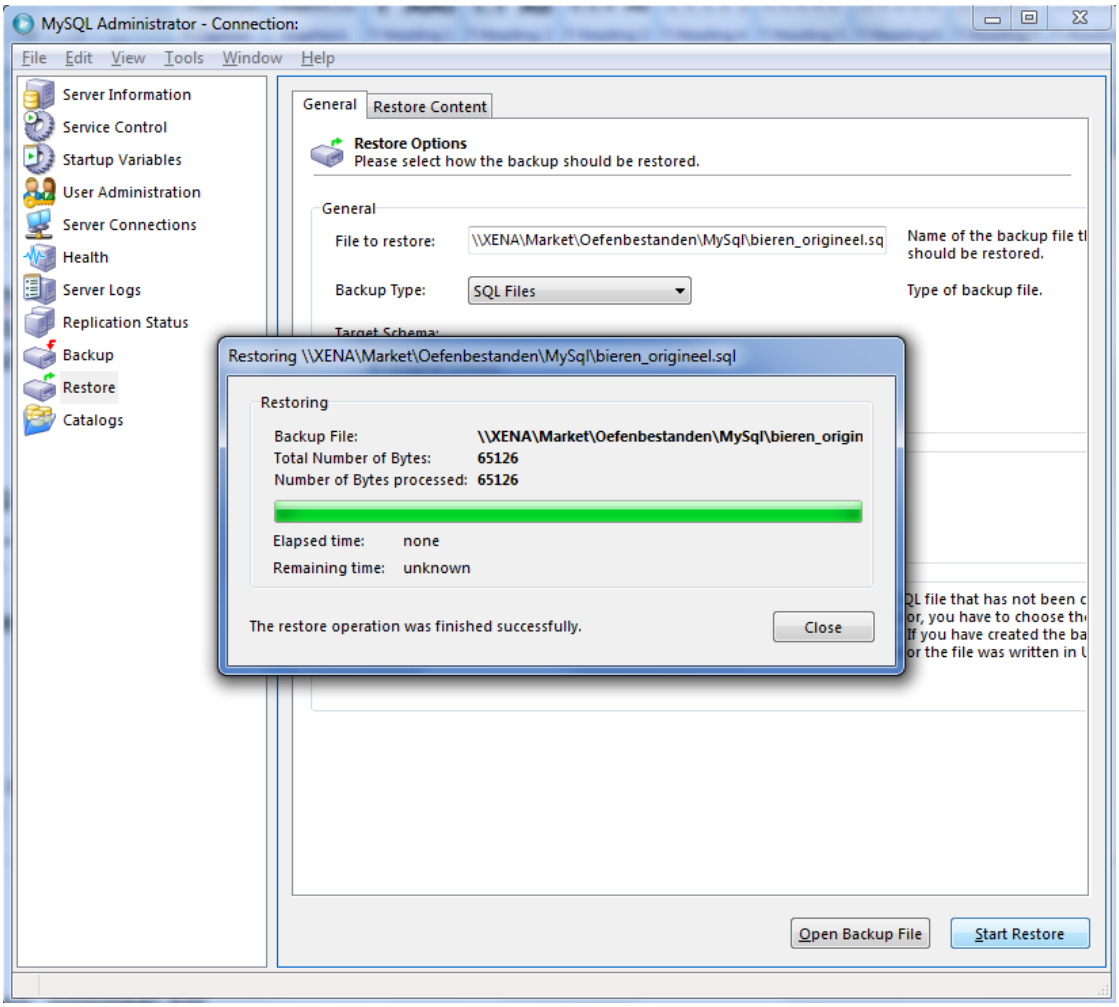
Dit zijn zogenaamde *sql-dumps*, backup bestanden van databanken in MySql. We doen alsof we een backup "*restoren*".

→ Kies RESTORE

→ Op het eerste tabblad GENERAL, klik onderaan op de Knop OPEN BACKUP FILE en kies het bestand *bieren_origineel.sql*

→ Klik dan op de knop *START RESTORE*

De database bieren is toegevoegd aan de *Catalog* lijst.

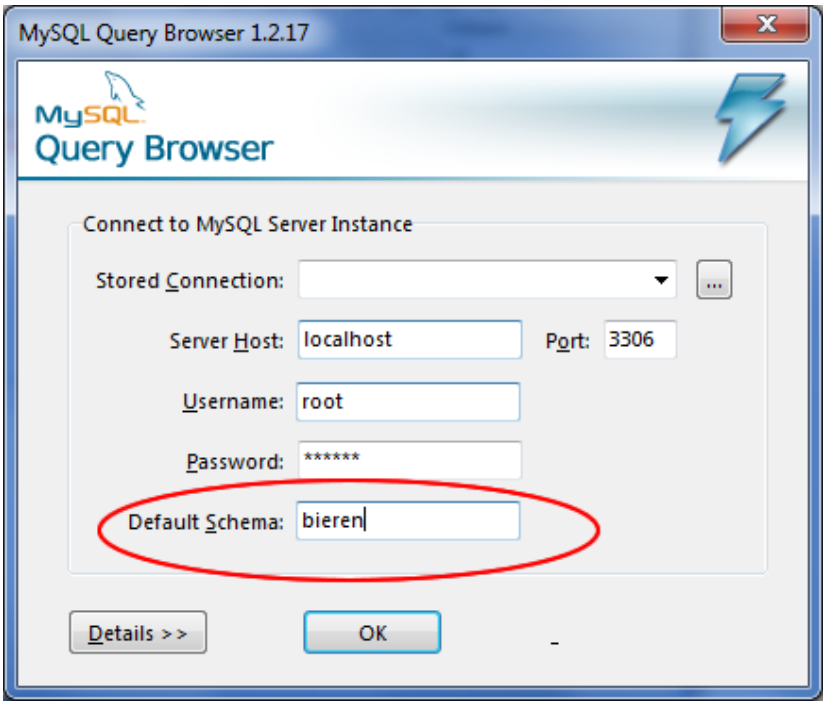


Doe nu hetzelfde met *plantv_origineel.sql* en *presiden_origineel.sql*.

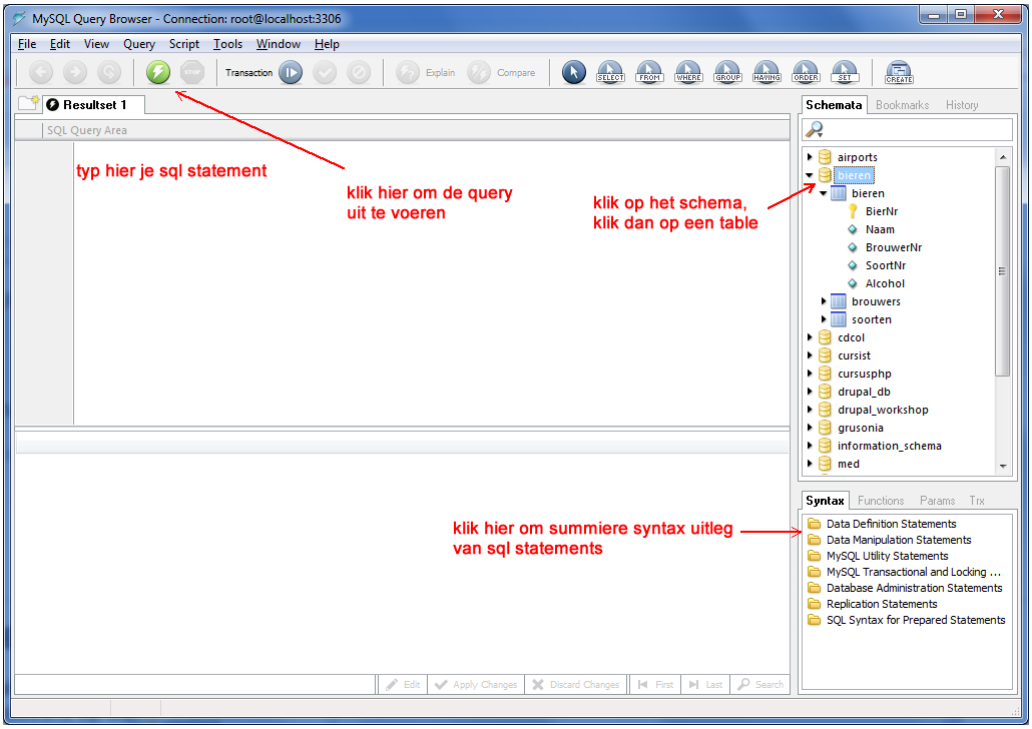
3.2.3 MySQL Query browser

Ook als je *MySQL Query Browser* gebruikt zal je aangemeld moeten zijn. Kom je vanuit de Administrator dan ben je dat reeds, start je het programma direct dan toont het aanmeldscherm een extra veld, *Default Schema*.

Dat hoef je niet noodzakelijk in te vullen, maar als je volgende maal zeker weet dat je met de db *bieren* gaat werken, bespaart het je straks wat werk.



Je kan het programma apart opstarten of het starten vanuit de *MySQL Administrator*, menu TOOLS.

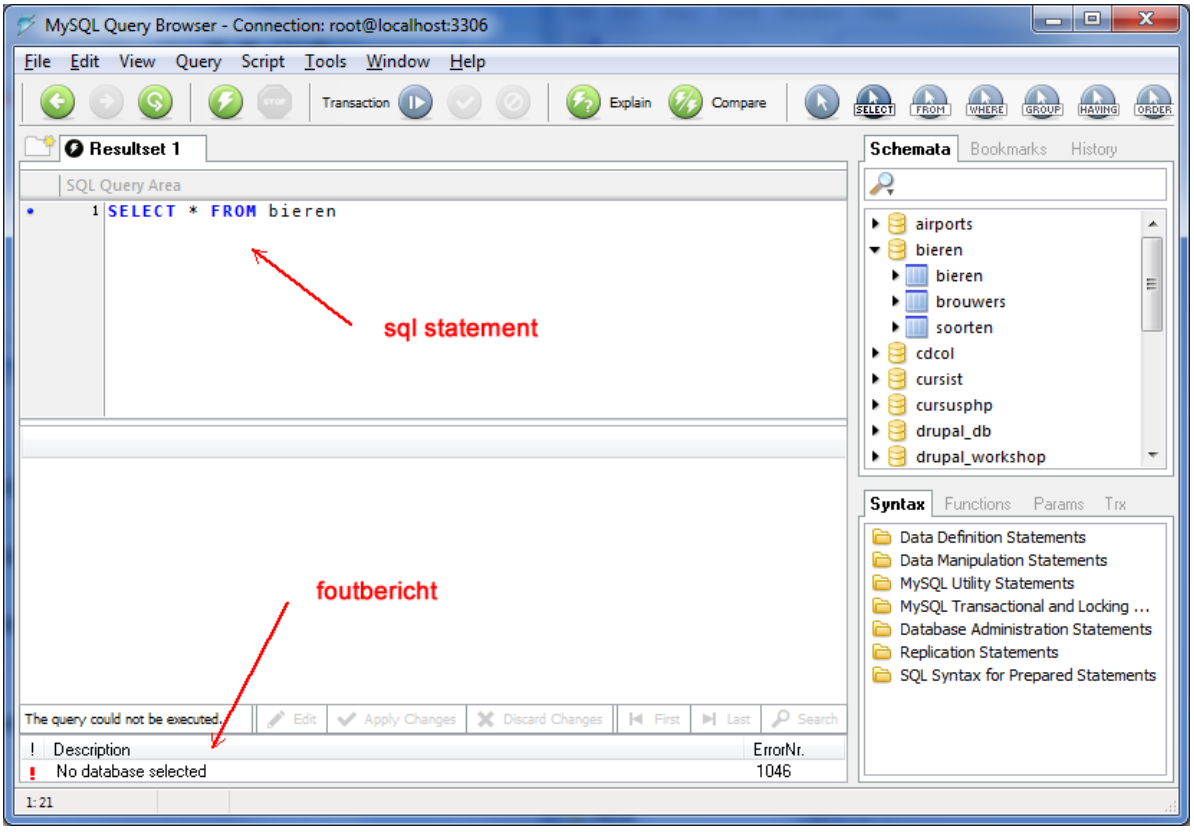


Het scherm bestaat uit 4 panelen:

1. de *SQL Query area* bovenaan, waar je je sql statement typt
2. eronder het resultaten paneel, waar de resultaten en berichten komen. In geval van een fout in de query wordt dit ook hier getoond.
3. rechts de Schemadata waar je de aanwezige databases (schema's) ziet

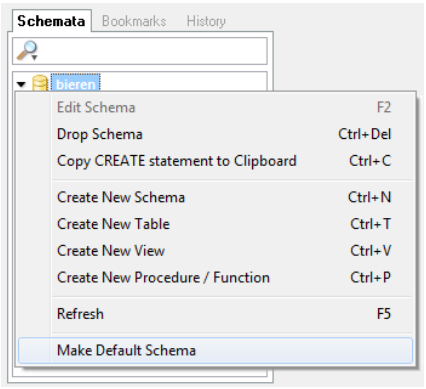
4. rechts onderaan de Syntax help

Een voorbeeld van een uitgevoerde query:



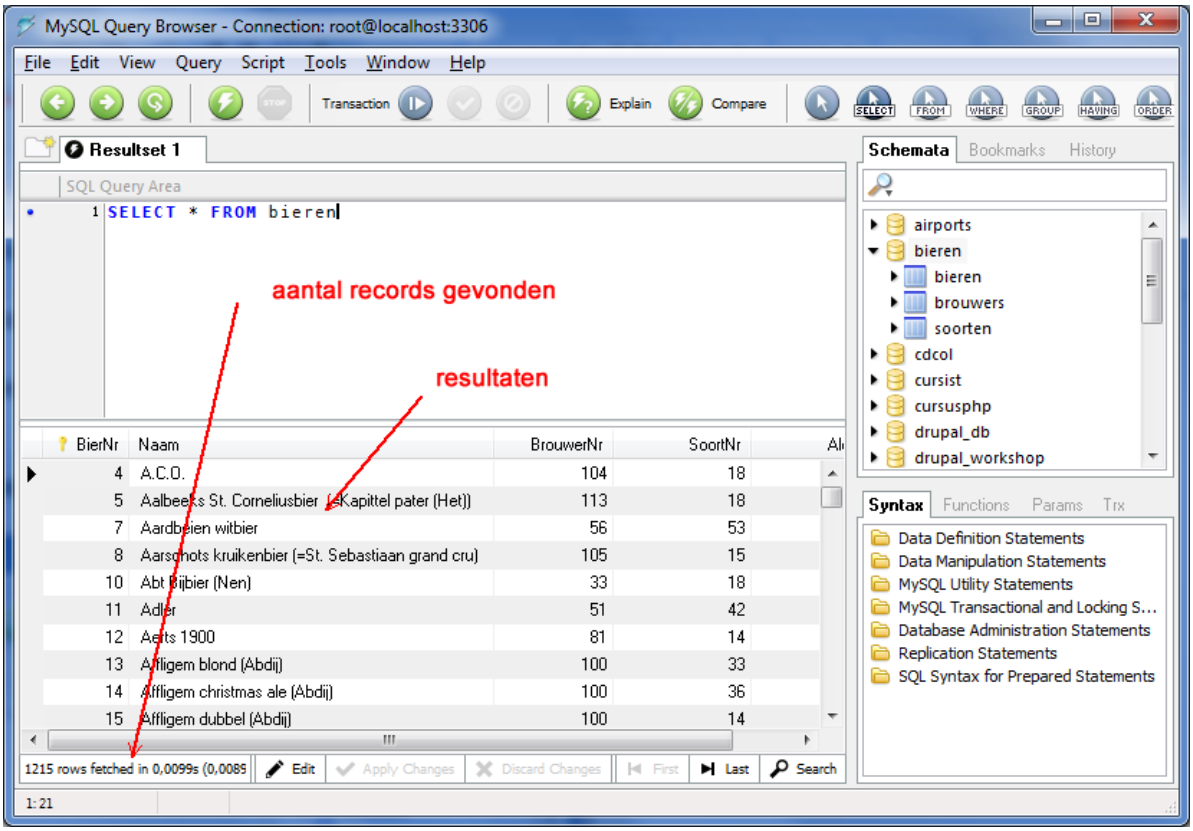
Alhoewel het sql statement correct is, krijgen we toch de foutmelding "*No database selected*". De server weet niet op welke db hij het statement moet gaan uitvoeren:

- ofwel klik je met de rechter muisknop op het *bieren* icoontje en kies je "*Make default schema*".



- ofwel verbeter je het sql statement naar "*SELECT * FROM bieren.bieren*", waarmee je aanduidt dat de tabel *bieren* zich in de database *bieren* bevindt.

Wij kiezen hier voor de eerste mogelijkheid en bekomen uiteindelijk volgend resultaat:


MySQL Query Browser - Connection: root@localhost:3306
File Edit View Query Script Tools Window Help
Transaction
Explain
Compare
Resultset 1
SQL Query Area
1 SELECT * FROM bieren
aantal records gevonden
resultaten
BierNr Naam BrouwerNr SoortNr
4 A.C.O. 104 18
5 Aalbeeks St. Corneliusbier (=Kapittel pater (Het)) 113 18
7 Aardbeien witbier 56 53
8 Aarschots kruikenbier (=St. Sebastiaan grand cru) 105 15
10 Abt Bijbier (Nen) 33 18
11 Adler 51 42
12 Aerts 1900 81 14
13 Affligem blond (Abdij) 100 33
14 Affligem christmas ale (Abdij) 100 36
15 Affligem dubbel (Abdij) 100 14
1215 rows fetched in 0,0099s (0,0085
Edit
Apply Changes
Discard Changes
First
Last
Search
1: 21
Schemata
Bookmarks
History
airports
bieren
bieren
brouwers
soorten
cdcol
cursist
cursusphp
drupal_db
drupal_workshop
Syntax
Functions
Params
Trx
Data Definition Statements
Data Manipulation Statements
MySQL Utility Statements
MySQL Transactional and Locking S...
Database Administration Statements
Replication Statements
SQL Syntax for Prepared Statements

4 DE DATABASES

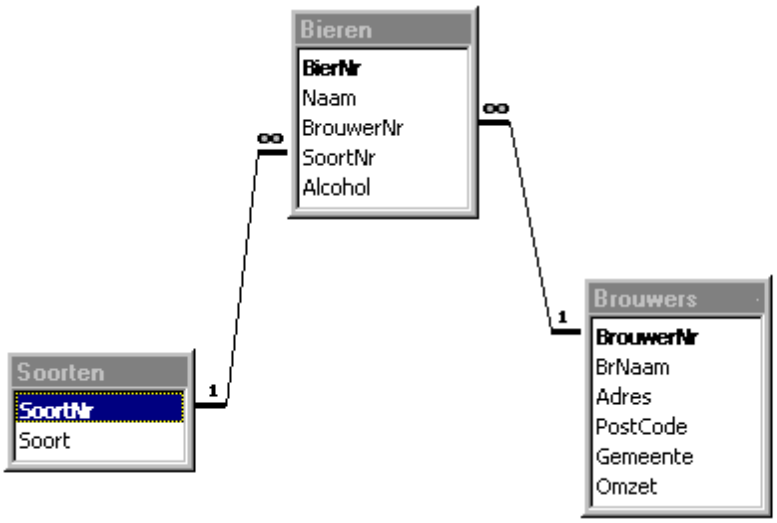
In deze cursus wordt er gebruik gemaakt van drie databases: bieren, plantv en presiden.
Alle voorbeelden uit de cursus zijn gebaseerd op de database bieren. Voor de oefeningen maak je gebruik van de database plantv. Tot slot maakt de eindoefening gebruik van de database presiden.

4.1 De database “bieren”

De database Bieren bestaat uit drie tabellen, nl.:

Bieren	Bevat een lijst van alle bieren, met de velden *Biernr, Naam, Brouwernr, Soortnr, Alcohol*. *Biernr* is de Primary Key. Het veld *Brouwernr* verwijst naar de brouwerij uit de tabel *Brouwers*. Het veld *Soortnr* verwijst naar de biersoort uit de tabel *Soorten*.
Brouwers	Bevat een lijst van alle brouwerijen, met de velden *Brouwernr, Brnaam, Adres, Postcode, Gemeente, Omzet*. *Brouwernr* is de Primary Key.
Soorten	Bevat een lijst van alle biersoorten, met de velden *Soortnr*, *Soorten*. *Soortnr* is de Primary Key.

Het schema van de database Bieren:

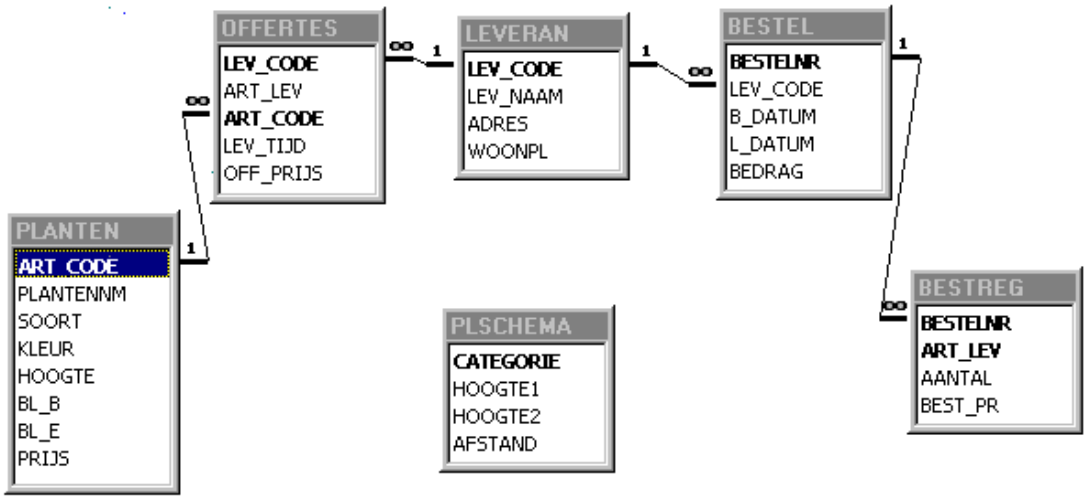


4.2 De database “plantv”

De database plantv bestaat uit 6 tabellen, nl:

Planten	Bevat de gegevens van de planten waarbij het veld art_code de primary key is, plantennm bevat de naam van de plant, bl_b en bl_e bevat de maanden wanneer de bloeiperiode van de plant begint en eindigt.
Offertes	Bevat de catalogusgegevens van onze leveranciers. lev_code en art_code vormen samen de primary key. art_lev is de code die de leveranciers geven aan de plant.
Leveran	Bevat de gegevens van de leveranciers. Lev_code is de primary key.
Bestel	Bevat de bestellingen die wij geplaatst hebben bij de leveranciers. B_datum is de besteldatum en l_datum is de gewenste leveringsdatum.
Bestreg	Bevat de detaillijnen van de bestelling. De primary key wordt bepaald door de velden bestelnr en art_lev.
Plschema	Bevat de gegevens van de verschillende soorten categorieën van planten.

De prijzen in deze tabel zijn uitgedrukt in €. De hoogte is uitgedrukt in cm.

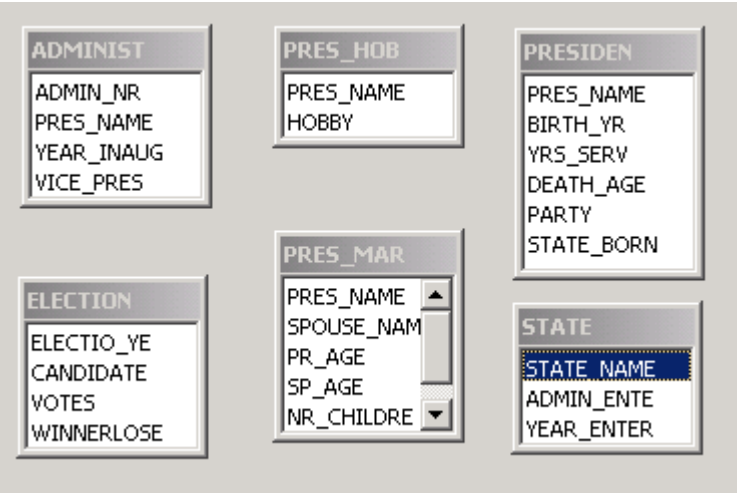


Deze database wordt gebruikt in de oefeningen.

4.3 De database “presiden”

De database PRESIDEN bevat 6 tabellen:

ADMINIST	Bevat de lijst van presidenten en hun vice-president met het beginjaar van hun ambtstermijn.
ELECTION	Bevat de presidentskandidaten.
PRES_HOB	Bevat de hobby's van de presidenten
PRES_MAR	Bevat de huwelijken van de presidenten
PRESIDEN	Bevat de lijst van presidenten en de duurtijd van hun ambtstermijn
STATE	Bevat een lijst van alle staten en wanneer zij tot de USA toegetreden zijn.



5 GEGEVENS SELECTEREN

5.1 SELECT Syntax

Met deze instructie geef je de RDBMS opdracht om gegevens uit de database in de vorm van een set records als resultaat te geven.

De SQL **sleutelwoorden** , zoals "select" en "from",
zijn NIET hoofdlettergevoelig, dus maakt het niet uit of je kleine of hoofdletters gebruikt. Heel dikwijls zie je echter de sleutelwoorden in hoofdletters gespeld, dit is enkel om ze duidelijk te doen uitkomen.

```
SELECT [predikaat] { * | tabel.* | [tabel.]veld1
[AS alias1] [, [tabel.]veld2 [AS alias2] [, ...]]}
FROM tabelexpressie [, ...] [AS alias]
[WHERE... ]
[GROUP BY... ]
[HAVING... ]
[ORDER BY... ]
[LIMIT ...]
```

De instructie SELECT bevat de volgende onderdelen:

Onderdeel	**Beschrijving**
`predikaat`	ALL of DISTINCT. Met een predikaat kan je het aantal records in het resultaat beperken. DISTINCT selecteert unieke records, ALL alles, ook dubbels. De standaardinstelling is ALL.
`*`	Bepaalt dat alle velden uit de opgegeven tabel of tabellen worden geselecteerd.
`tabel`	De naam van de tabel die de velden bevat waaruit records worden geselecteerd.
`veld1, veld2`	De namen van de velden waaruit gegevens worden opgehaald. Als je meer dan één veld opgeeft, worden deze opgehaald in de opgegeven volgorde.
`alias1, alias2`	De namen die je wilt gebruiken als kolomkoppen in plaats van de oorspronkelijke kolomnamen in tabel. AS is optioneel
`tabelalias`	Een naam die de echte tabelnaam vervangt. AS is optioneel
`tabelexpressie`	De naam van de tabel of de namen van de tabellen waaruit je gegevens wilt ophalen.

Voor het uitvoeren van deze bewerking doorzoekt de RDBMS de opgegeven tabel of tabellen, licht de opgegeven kolommen uit, selecteert de rijen die aan de voorwaarde(n) voldoen en sorteert of groepeert de resultaatrijen in de opgegeven volgorde.

Instructies met SELECT wijzigen de gegevens in de database niet.

5.2 Lijst opvragen

De minimale syntax voor een instructie SELECT luidt:

```
SELECT velden
FROM tabel
```

```
SELECT naam, alcohol
FROM bieren
```

geeft als resultaat een lijst met de naam en alcoholpercentage van alle bieren

```
SELECT brnaam
FROM brouwers
```

geeft als resultaat een lijst met alle namen van de brouwerijen

```
SELECT gemeente
FROM brouwers
```

geeft een lijst van alle woonplaatsen van de brouwerijen. Sommige gemeenten komen meerdere keren voor omdat er in deze gemeenten meerdere brouwerijen gevestigd zijn.
Om elke gemeente maar één keer te krijgen, gebruiken we DISTINCT. DISTINCT zorgt ervoor dat elke rij van het resultaat uniek is.
De SQL-instructie wordt dan:

```
SELECT DISTINCT gemeente
FROM brouwers
```

Met een sterretje (*) kan je alle velden in een tabel selecteren. In het volgende voorbeeld worden alle velden in de tabel bieren geselecteerd:

```
SELECT *
FROM bieren
```

5.3 Selectie maken m.b.v. WHERE

De WHERE clausule wordt gevolgd door een conditie. Een conditie is samengesteld uit:

```
attribuut operator constante | attribuut
```

De operator kan zijn: <, >, <=, >=, =, `LIKE`, `BETWEEN … AND …`, `IN (…)`

Een constante kan een getal zijn of een karakterreeks. Een karakterreeks moet altijd beginnen en eindigen met een enkele quote (**'**) of een dubbele quote (**"**).

```
SELECT naam
FROM bieren
WHERE alcohol < 5
```

geeft als resultaat een lijst met de naam van alle bieren met een alcoholpercentage lager dat 5%

```
SELECT brnaam
FROM brouwers
WHERE gemeente = 'Brussel'
```

geeft als resultaat een lijst van alle brouwerijen gelegen in Brussel

```
SELECT naam
FROM bieren
WHERE naam LIKE '%ale%'
```

geeft als resultaat een lijst van alle bieren waar het woord `ale` voorkomt in de naam.

Er zijn twee ***wildcard*** tekens: `%` (percent-teken) en _ (*underscore*).

Het percent-teken (%) duidt aan dat op die plaats een willekeurig aantal letters kunnen voorkomen.
Een underscore (_) neemt de plaats in van exact 1 willekeurig teken.
Bijvoorbeeld als de naam moet beginnen met Wit dan luidt de selectie als volgt: `naam LIKE 'wit%'`.

Aard van selectie	**Patroon**	**Waarden die in het patroon passen**	**Waarden die niet voldoen aan de voorwaarden**
Meerdere karakters	a%a %ab% ab%	aa, aBa, aBBBa abc, AABB, Xab abcdefg, abc	aBC aZb, bac cab, aab
Special teken	a[@]a	a@a	aaa
Eén karakter	a_a	aaa, a3a, aBa	aBBBa
Karakter moet voorkomen in de reeks	[a-z]	f, p, j	2, &
Karakter moet voorkomen in de benoemde lijst	[agm]	a, g, m	b, c, n, 4

Combinatie van hiervoor vermelde formaten	a[b-m]_	Ab9, af0	aacfd, a90

Opmerking:

```
SELECT naam
FROM bieren
WHERE alcohol BETWEEN 5 AND 7
```

geeft een lijst van alle bieren met een alcoholpercentage vanaf 5% tot en met 7%. Merk op dat dit inclusief de grenswaarden is.

```
SELECT naam
FROM bieren
WHERE alcohol IN (0, 5, 8)
```

geeft een lijst van alle bieren met een alcoholpercentage van 0%, 5% of 8%.

```
SELECT brnaam
FROM brouwers
WHERE gemeente IN ('Leuven', 'Genk', 'Antwerpen', 'Dendermonde',
'Wevelgem')
```

geeft een lijst van alle brouwerijen gevestigd in de gemeenten Leuven, Hasselt, Genk, Antwerpen, Dendermonde en Wevelgem.

```
SELECT naam
FROM bieren
WHERE alcohol IS NULL
```

geeft een lijst van alle bieren waarvan de kolom alcohol niet ingevuld is.
De operator IS NULL geeft lege velden.
Om de kolommen te controleren die niet leeg zijn gebruik je IS NOT NULL.

Je kunt voorwaarden combineren met elkaar met AND, OR en NOT. Bij AND moeten beide voorwaarden waar zijn omdat het resultaat waar zou zijn. Bij OR is het voldoende dat één van de voorwaarden waar is. Bij NOT gebeurt de negatie: waar wordt onwaar en omgekeerd.

Let op

```
alcohol IS NULL                wordt  alcohol IS NOT NULL

alcohol BETWEEN 5 AND 7        wordt  ALCOHOL NOT BETWEEN 5 AND 7

naam LIKE '%wit'               wordt  naam NOT LIKE '%wit%'

gemeente IN ('genk', 'leuven') wordt  gemeente NOT IN ('genk','leuven')
```

```
SELECT * FROM bieren
WHERE naam LIKE '%wit%' AND alcohol > 5
```

geeft een lijst van alle bieren met een naam die het woord wit bevat en alcoholpercentage hoger dan 5

```
SELECT * FROM bieren
WHERE naam NOT LIKE '%wit%'
```

geeft een lijst van alle bieren met een naam die het woord wit niet bevat.

Let op

Let op het aantal bieren dat getoond wordt:

instructie1 `SELECT * FROM bieren WHERE alcohol > 5`

instructie2 `SELECT * FROM bieren WHERE alcohol <= 5`

De som van het aantal bieren dat getoond wordt door de instructie 1 en 2 is niet gelijk aan het totale aantal bieren. De rijen met een alcoholwaarde `NULL` worden genegeerd.

In de database plantenv kunnen we de tabel “bestel” gebruiken om met datums te werken.

```
SELECT *
FROM bestel
WHERE b_datum>'1999-01-17'
```

Selecteert alle bestellingen later dan de opgegeven datum.

Hier tussen twee datums:

```
SELECT *
FROM bestel
WHERE b_datum BETWEEN '1999-02-17' AND '1999-03-17'
```

Het enige geldige datum formaat is ‘YYYY-MM-DD’ en je geeft het op als een string. Er bestaan echter in alle RDBMS Datum-Tijd functies die een datum in een andere opmaak kunnen omzetten of andere bewerking ermee kunnen doen. Deze functies gaan echter buiten de scope van deze cursus.

5.4 Een gesorteerde lijst opvragen m.b.v ORDER BY

```
SELECT naam, alcohol
FROM bieren
ORDER BY naam ASC
```

geeft een lijst van alle bieren (naam en alcoholpercentage) alfabetisch gesorteerd op op naam.

```
SELECT naam, alcohol
FROM bieren
ORDER BY brouwernr DESC, naam ASC
```

geeft een lijst van alle bieren (naam en alcoholpercentage) gesorteerd op brouwernr (dalend) en vervolgens alfabetisch op naam (stijgend).

```
SELECT naam, alcohol
FROM bieren
ORDER BY 1 ASC
```

geeft een lijst van alle bieren (naam en alcoholpercentage) alfabetisch gesorteerd op op naam. Maar hier wordt het kolomnummer gebruikt. De naam staat als eerste in de select, gevolgd door de kolom alcohol.

5.5 Het aantal records beperken met LIMIT

De LIMIT clausule in SELECT laat ons toe het aantal records te beperken. LIMIT kan gebruikt worden met één of twee argumenten:

```
SELECT *
FROM TABLE
LIMIT aantalrecords
```

of

```
SELECT *
FROM TABLE
LIMIT voorbijRij aantalrecords
```

Voorbeelden:

```
SELECT *
FROM bieren
ORDER BY alcohol DESC
LIMIT 5
```

geeft de top vijf van meest alcoholische bieren.

LIMIT kan je ook met twee argumenten gebruiken waarbij het eerste argument de rij is vanaf waar getoond zal worden (niet inclusief) en het tweede argument het aantal rijen vanaf die plaats.
De allereerste rij van een recordset heeft de index 0.

```
SELECT *
FROM bieren
ORDER BY alcohol DESC
LIMIT 5, 10
```

geeft 10 bieren beginnend vanaf record 6.

Om alle records vanaf een bepaald record tot het einde te krijgen, gebruik je een zeer groot getal als tweede argument:

```
SELECT *
FROM bieren
ORDER BY alcohol DESC
LIMIT 25, 9999999999999
```

geeft alle bieren vanaf het 26ste record

Opmerking:

- LIMIT kan ook gebruikt worden in DELETE … LIMIT en UPDATE … LIMIT. in deze "actie queries " is het LIMIT argument nooit een positie, maar een aantal

5.6 Oefeningen SELECT deel 1 (database plantv)

1. Geef een overzicht, met alle gegevens, van de leveranciers uit AALSMEER.

2. Geef een alfabetisch overzicht op plantennaam, met de artikelcode, plantennaam en prijs, van alle planten uit de tabel PLANTEN.

3. Welke planten beginnen in de maand maart te bloeien? Druk artikelcode, plantennaam, en begin bloeimaand af.

4. Toon alle bestellingen (BESTEL) vanaf 14 maart 1999

5. Maak een overzicht uit de tabel OFFERTES, gesorteerd op artikelcode en binnen artikelcode een sortering op artikelcode van de leverancier. Alleen de gegevens artikelcode, artikelcode van de leverancier en leverancierscode tonen.

6. Maak een gesorteerd overzicht van alle waterplanten. Sorteer op HOOGTE, grootste voorop.

7. Maak een lijst van de verschillende kleuren die bij de planten uit de tabel PLANTEN horen.

8. Maak een lijst van alle planten waarvan de kolom KLEUR niet ingevuld is.

9. Toon de verschillende soorten planten in de tabel PLANTEN.

10. Geef een overzicht van alle vaste planten welke geelkleurige bloemen geven. Van iedere plant toon je de volgende gegevens: artikelcode, plantennaam, hoogte en bl_b.

11. Geef een overzicht van alle planten met een prijs boven de 10 € die niet tot de soort bomen behoren.

12. Maak een lijst van alle planten die in juni beginnen te bloeien en gele bloemen geven en tevens van alle planten die in augustus voor het eerst bloeien en een rode bloemen hebben. Plaats alle beschikbare gegevens in het overzicht.

13. Welke planten met gemengde bloeikleuren worden maximum 60 cm hoog? Druk artikelcode, plantennaam en hoogte af.

14. Geef een overzicht van alle leveranciers die niet in Hillegom wonen.

15. Van welke planten is zowel de kleur als de hoogte onbekend? Geef artikelcode, plantennaam, kleur en hoogte.

16. Welke planten bloeien in ieder geval in de periode augustus tot en met oktober? Geef artikelcode en plantennaam.

17. Welke planten bloeien in ieder geval in de maand september? Geef artikelcode en plantennaam.

18. Geef een overzicht van alle vaste planten met een prijs tussen 3 € en 5 €.

19. Geef een overzicht van alle planten die in maart, april, september of oktober beginnen te bloeien.

20. Bij welke planten komt het woord BOOM voor? Geef artikelcode en plantennaam.

21. Geef de artikelcode en plantennaam van alle planten die als derde letter een N hebben.

22. Welke 1- en 2-jarige planten staan er in de tabel PLANTEN?

23. Geef een overzicht van alle planten, behalve de bomen en de heesters, die tussen de 100 en 200 cm hoog zijn, rode of blauwe bloemen geven, en vóór augustus beginnen te bloeien. Alle gegevens behalve de prijs zijn belangrijk. Sorteer de lijst op soort en binnen soort op plantennaam.

24. Bij welke planten komt het woord KRUID voor in hun plantennaam, maar behoren niet tot de soort KRUID? Geef artikelcode en plantennaam.

25. Geef de artikelcode en de plantennaam van alle planten die beginnen met de letter L en eindigen met de letter E.

26. Welke planten hebben een plantennaam van precies 5 letters lang? Geef artikelcode en plantennaam.

27. Welke planten hebben een plantennaam van minimum 5 letters lang? Geef artikelcode en plantennaam.

28. Geef de 10 duurste planten

5.7 Aggregate functies

De volgende functies maken berekeningen op rijen:

`COUNT()`	aantal
`SUM()`	totaal
`AVG()`	gemiddelde
`MAX()`	grootste waarde
`MIN()`	kleinste waarde
`STD() of STDDEV()`	standaard deviatie

```
SELECT COUNT(*)
FROM bieren
```

geeft het totaal aantal bieren uit de tabel. In het resultaat zien we geen kolomnamen.
Om toch een kolomnaam te geven aan het resultaat, gebruiken we het sleutelwoord `AS` om een alias toe te kennen zoals in het volgende voorbeeld:

```
SELECT COUNT(*) AS aantal
FROM bieren
```

```
SELECT AVG(alcohol) AS gemidd, MAX(alcohol) AS maxi, MIN(alcohol)
AS mini, SUM(alcohol) AS som
FROM bieren
```

geeft het gemiddelde, maximum , minimum, totaal alcoholpercentage uit de tabel bieren

5.8 Berekeningen maken

```
SELECT brnaam, omzet * 0.9118 AS omzet_dollar, omzet * 116.6 AS
omzet_yen
FROM brouwers
```

geeft een lijst van alle brouwerijen met hun omzet in dollar en yen. De mogelijke werkingen zijn optellen(**+**), aftrekken(**-**), vermenigvuldigen(******), delen(***/***) en machtsverheffing(**^**). Het gebruik van haakjes **()** in de bewerking is toegelaten.

5.9 Oefening SELECT deel 2

1. Hoeveel leveranciers telt ons tuincentrum?
2. Geef de gemiddelde prijs van alle waterplanten.
3. Wat is de maximale hoogte van de bomen in de tabel PLANTEN?
4. Geef de gemiddelde, de laagste en de hoogste offerteprijs van leverancier 013.
5. Wat is de laagste offerteprijs voor artikelcode 123?
6. Geef bestelnummer, artikelcode van de leverancier en het totale bestelbedrag per bestelrij uit de Tabel BESTREG.
7. Maak een overzicht van de heesters uit de tabel PLANTEN, waarbij de prijzen met 5% zijn verhoogd.
8. Wat is het totale bedrag aan bestellingen bij leverancier 004?
9. Hoeveel stuks van de leveranciers-artikelcode B111 zijn besteld?

10. Wat is het totale bestelbedrag (exclusief korting) voor de leveranciers-artikelcode B331?

5.10 Gegevens groeperen

```
SELECT brouwernr, AVG(alcohol) AS gemidd
FROM bieren
GROUP BY brouwernr
```

berekent het gemiddelde alcoholpercentage per brouwernr

In de lijst van de te tonen kolommen mogen enkel bewerkingen met een aggregate functie staan en kolommen die vermeld staan na de group by.

```
SELECT art_code, art_lev, AVG(off_prijs)
FROM offertes
GROUP BY art_code
```

is niet toegelaten omdat bij art_lev geen aggregaat functie gebruikt wordt of omdat art_lev niet na de group by staat

```
SELECT LEFT(naam,1), AVG(alcohol)
FROM bieren
GROUP BY LEFT(naam,1)
```

toont ons de gemiddelde alcoholpercentage gegroepeerd volgens de eerste letter van de biernaam. Als je wilt groeperen op de beginletter, dan moet je de bewerking in het group by clausule herhalen.

```
SELECT brouwernr, MIN(alcohol) AS mini
FROM bieren
GROUP BY brouwernr
HAVING MIN(alcohol)<5
```

bepaalt het minimum alcoholpercentage per brouwernr, de lijst toont enkel de brouwernr's en percentages die kleiner zijn dan 5%. Je gebruikt "having" indien de selectie gebaseerd is op het resultaat van een bewerking met een aggregaat functie. In alle andere gevallen gebruik je "where".

```
SELECT brouwernr, AVG(alcohol) AS mini
FROM bieren
GROUP BY brouwernr
HAVING COUNT(*)>10
```

toont het gemiddelde alcoholpercentage per brouwernr voor alle brouwers die minimum 10 bieren produceren.

In een GROUP BY (en ook ORDER BY) mag je verwijzen naar een kolomnaam, een kolom alias of een kolom positie:

```
SELECT  soortnr, count(*) AS aantal, min(alcohol) AS laagste,
max(alcohol) AS hoogste
```

```
FROM bieren
GROUP BY soortnr
```

en ook:

```
SELECT  soortnr, count(*) AS aantal, min(alcohol) AS laagste,
max(alcohol) AS hoogste
FROM bieren
GROUP BY 1
```

en ook:

```
SELECT  soortnr AS soort, count(*) AS aantal, min(alcohol) AS
laagste,  max(alcohol) AS hoogste
FROM bieren
GROUP BY soort
```

5.11 Oefeningen SELECT deel 3

1. Tel het aantal planten per plantensoort uit de tabel PLANTEN.
2. Tel het aantal bestelregels per bestelling.
3. Wat is de gemiddelde prijs per plantensoort.
4. Hoeveel planten zijn er per plantensoort-kleurgroep in de tabel PLANTEN?
5. Vervaardig een overzicht waaruit visueel kan worden afgeleid welke kleur van de vaste planten de hoogste gemiddelde prijs heeft.
6. Laat per leverancierscode het aantal artikelen zien dat de leverancier aanbiedt onder voorwaarde dat de levertijd van het artikel minder dan 18 dagen bedraagt.
7. Wat is de gemiddelde prijs per plantensoort, exclusief de geelbloemige planten?
8. Maak een overzicht met de laagste en de hoogste offerteprijs per artikelcode.
9. Wat is de gemiddelde prijs per plantensoort, voor soorten met minstens 10 exemplaren in de tabel PLANTEN.
10. Hebben de planten met korte levertijden in het algemeen een hogere gemiddelde offerteprijs?
11. Maak een overzicht met de laagste en de hoogste bestelprijs per artikelcode van de leverancier.
12. Geef een overzicht van het aantal beschikbare planten per bloeibegin-maand/hoogte/kleur groep.
13. Wat is de laagste prijs per plantensoort van de planten die in ieder geval bloeien in de periode mei t/m juni?

14. Tel het aantal planten per aantal maanden dat de planten bloeien.
    Bijvoorbeeld

Duurtijd	Aantal
0	29
1	50
2	24
3	13
…	

15. De functie ROUND(getal, 0) rondt het getal af naar een geheel getal. Tel het aantal planten per prijs (na afronding).

5.12 Gegevens selecteren uit meerdere tabellen tegelijkertijd

Met deze bewerking kan je records uit twee tabellen combineren wanneer een gemeenschappelijk veld overeenkomstige waarden bevat.

```
FROM tabel1 INNER JOIN tabel2
  ON tabel1.veld1 operator tabel2.veld2
```

De instructie INNER JOIN bevat de volgende onderdelen:

Onderdeel	**Beschrijving**
`tabel1, tabel2`	De namen van de tabellen waaruit records worden gecombineerd.
`veld1, veld2`	De namen van de velden die worden gekoppeld. Als het geen numerieke velden zijn, moet het gegevenstype van de velden gelijk zijn en moeten ze dezelfde soort gegevens bevatten. Ze hoeven echter niet dezelfde naam te hebben. Indien dit wel zo is moet je wel een onderscheid maken door de tabelnamen te gebruiken op de volgende wijze: `tabel.veld`
`operator`	Een willekeurige relationele vergelijkingsoperator: "**=**", "**<**", "**>**", "**<=**", "**>=**" of "**<>**".

Opmerking: als één van de twee INNER JOIN kolommen een `NULL` waarde bevat, zal die nooit in het resultaat voorkomen, ook niet als beide kolommen een `NULL` bevatten.

Je kan een bewerking INNER JOIN toepassen in elke component FROM. Dit is het meest gebruikte type koppeling. Met INNER JOIN worden records uit twee tabellen gecombineerd op basis van gelijke waarden in een gemeenschappelijk veld.

Je kan INNER JOIN gebruiken met de tabellen Brouwers en Bieren om van alle bieren hun brouwerij te selecteren. Wil je daarentegen alle brouwerijen (met inbegrip van brouwerijen zonder bieren), dan kan je een bewerking LEFT JOIN of RIGHT JOIN gebruiken om een outer join (koppeling aan de buitenkant) te maken.

In het volgende voorbeeld worden de tabellen bieren en brouwers gekoppeld via het veld brouwernr:

```
SELECT Naam, Brnaam
FROM Bieren INNER JOIN Brouwers
  ON Bieren.Brouwernr  = Brouwers.Brouwernr
```

Hoewel Brouwernr in dit voorbeeld het gekoppelde veld is, wordt het niet opgenomen in de query-uitvoer omdat het niet is opgenomen in de instructie SELECT. Als je het gekoppelde veld wel wilt opnemen, geef je de veldnaam op in de instructie SELECT (in dit geval: Bieren.Brouwernr).

Met de volgende syntax kan je ook verschillende componenten ON koppelen in een instructie JOIN:

```
SELECT velden
FROM tabel1 INNER JOIN tabel2
  ON [(] tabel1.veld1 operator tabel2.veld1
  [ AND tabel1.veld2 operator tabel2.veld2)]
  [ OR tabel1.veld3 operator tabel2.veld3)]
```

Deze syntax heb je nodig wanneer je twee tabellen wilt koppelen met een samengestelde sleutel (dit is een sleutel gebaseerd op meerdere velden).

Tevens is het mogelijk met de volgende syntax meerdere tabellen met JOIN te nesten:

```
SELECT velden
FROM tabel1 INNER JOIN
  (tabel2 INNER JOIN [( ]tabel3
  [INNER JOIN [( ]tabelx [INNER JOIN ...)]
  ON tabel3.veld3 operator tabelx.veldx)]
  ON tabel2.veld2 operator tabel3.veld3)
  ON tabel1.veld1 operator tabel2.veld2
```

Bijvoorbeeld:

```
SELECT Bieren.Naam, Brouwers.BrNaam, Soorten.Soort
FROM Soorten INNER JOIN (Brouwers INNER JOIN Bieren
  ON Brouwers.BrouwerNr  = Bieren.BrouwerNr)
  ON Soorten.SoortNr  = Bieren.SoortNr
```

Geeft een lijst van alle bieren met de velden naam, brouwerij en soort uit de tabellen bieren, brouwers en soorten.

5.13 Outer join

Met deze bewerking combineer je records van de brontabel in een component FROM.

```
FROM tabel1 [ LEFT | RIGHT ] JOIN tabel2
  ON tabel1.veld1 vergopr tabel2.veld2
```

De bewerkingen LEFT JOIN en RIGHT JOIN bevatten de volgende onderdelen:

Onderdeel	Beschrijving

`tabel1, tabel2`	De namen van de tabellen waaruit records worden gecombineerd.
`veld1, veld2`	De namen van de velden die worden gekoppeld. De velden moeten hetzelfde gegevenstype hebben en dezelfde soort gegevens bevatten. Ze hoeven echter niet dezelfde naam te hebben.
`operator`	Een willekeurige relationele vergelijkingsoperator: "=", "<", ">", "<=", ">=" of "<>".

Met een bewerking LEFT JOIN maak je een left outer join. In een left outer join worden alle records uit de eerste (linker-) tabel opgenomen, zelfs als er geen overeenkomende waarden in de tweede (rechter-) tabel zijn.

Met een bewerking RIGHT JOIN maak je een right outer join. In een right outer join worden alle records uit de tweede (rechter-) tabel opgenomen, zelfs als er geen overeenkomende waarden in de eerste (linker-) tabel zijn.

Je kan bijvoorbeeld in de tabellen Afdelingen (linkertabel) en Werknemers (rechtertabel) met LEFT JOIN alle afdelingen selecteren, inclusief afdelingen zonder personeelsleden. Als je alle personeelsleden wil selecteren, dus met inbegrip van personeelsleden die niet werkzaam zijn op een bepaalde afdeling, gebruik je RIGHT JOIN.

In het volgende voorbeeld worden de tabellen Soorten en Bieren gekoppeld op het veld Soortnr. De select-instructie genereert een lijst van alle Soorten, inclusief biersoorten waarvan er geen bieren aanwezig zijn in de tabel Bieren:

```
SELECT Naam, Soort
FROM Soorten LEFT JOIN Bieren
  ON Soorten.Soortnr  = Bieren.Soortnr
```

In het volgende voorbeeld worden de tabellen Brouwers en Bieren gekoppeld op het veld Brouwernr. De select-instructie genereert een lijst van alle Brouwers, inclusief de brouwerijen waarvan er geen bieren aanwezig zijn in de tabel Bieren:

```
SELECT BrNaam, Naam
FROM Bieren RIGHT JOIN Brouwers
  ON Brouwers.Brouwernr  = Bieren.Brouwernr
```

Opmerkingen:

- Als je een query wilt maken waarin alleen records met identieke gegevens in de gekoppelde velden worden opgenomen, gebruikt je een bewerking INNER JOIN.
- LEFT JOIN of RIGHT JOIN kan worden genest binnen INNER JOIN, maar INNER JOIN kan niet worden genest binnen LEFT JOIN of RIGHT JOIN.
- Je kan meerdere componenten ON koppelen.

5.14 Self Join

Een self join is een inner join of een outer join waarbij je twee keer dezelfde tabel gebruikt. Om dit goed te laten verlopen moet je de tabellen een aliasnaam geven. Als je de tabellen een aliasnaam geeft, moet je bovendien in de select, on, group by en where clausules verwijzen naar de tabel met de aliasnaam ipv van de tabelnaam.

```
SELECT b1.brnaam, b2.brnaam
FROM brouwers AS b1 INNER JOIN brouwers AS b2
ON b1.gemeente = b2.gemeente AND b1.brouwernr < b2.brouwernr
```

geeft een lijst van brouwers die in dezelfde gemeente wonen. Aan de tabel brouwers wordt telkens een aliasnaam gegeven. Als je een kolomnaam gebruikt, ben je verplicht om de aliasnaam te gebruiken.

5.15 UNION

Hiermee maak je een samenvoegquery, waarmee de resultaten van twee of meer onafhankelijke query's of tabellen worden gecombineerd.

```
SELECT ...
UNION [ALL | DISTINCT]
SELECT ...
```

De UNION-bewerking bevat de volgende onderdelen:

Onderdeel	Beschrijving
query1-n	Een SELECT-instructie, de naam van een opgeslagen query of de naam van een opgeslagen tabel die vooraf wordt gegaan door het sleutelwoord TABLE.

Nadere informatie

In een enkele UNION-bewerking kan je in elke willekeurige combinatie de resultaten samenvoegen van twee of meer query's, tabellen of SELECT-instructies.

```
SELECT * FROM bieren WHERE soortnr  = 1
UNION
SELECT * FROM bieren WHERE soortnr  = 5
```

selecteer alle bieren met soortnr 1 of 5

```
SELECT * FROM bieren
UNION
SELECT * FROM bieren
```

toont alle gegevens van de tabel bieren slechts éénmaal.

Standaard worden geen dubbele records als resultaat gegeven Als je een UNION-bewerking gebruikt. UNION DISTINCT is de default waarde en is dus hetzelfde als UNION alleen. Met UNION ALL wordt wel alle records getoond, ook de dubbele. De

query wordt dan ook sneller uitgevoerd. Het is echter mogelijk UNION ALL en UNION DISTINCT in één query samen te gebruiken.

Alle query's in een UNION-bewerking moeten hetzelfde aantal velden opvragen. Het is echter niet zo dat de velden ook dezelfde lengte of hetzelfde gegevenstype moeten hebben.

Aliassen kan je alleen gebruiken in de eerste SELECT-component, omdat ze worden genegeerd in alle overige. In de ORDER BY-component moet je naar velden verwijzen met dezelfde namen die worden gebruikt in de eerste SELECT-component.

Opmerkingen:

- Je kan een GROUP BY- en/of HAVING-component opnemen in elk query-argument om de als resultaat gegeven gegevens te groeperen.

- Aan het eind van het laatste query-argument kan je een ORDER BY-component opnemen om de gegevens in het resultaat in een bepaalde volgorde weer te geven.

5.16 Oefeningen SELECT deel 4

1. Maak een overzicht van de bestellingen met het bestelnummer, de naam van de leverancier, de leverdatum en het bestelde bedrag.

2. Welke planten hebben een levertijd van maximum 10 dagen ? Plaats in het overzicht de gegevens: artikelcode, leveranciers-artikelcode, plantennaam en levertijd.

3. Geef een overzicht waaruit blijkt bij welke leverancier(s) de planten besteld kunnen worden. In het overzicht moeten de volgende gegevens staan: artikelcode, plantennaam, leverancierscode, naam van de leverancier, levertijd en offerteprijs. Sorteer het overzicht op plantennaam.

4. Maak een overzicht van die planten die een prijs hebben welke minstens 50% boven de offerteprijs ligt. Geef artikelcode, plantennaam, prijs van de plant, leverancierscode en offerteprijs.

5. Hoeveel verschillen de gehanteerde bestelprijzen met de huidige offerteprijzen . Presenteer in de lijst de gegevens bestelnummer, artikelcode van de leverancier en het positieve of negatieve verschil.

6. In welke plaatsen kan het tuincentrum vaste planten bestellen?

7. Maak een overzicht van alle rode planten geleverd door leveranciers die niet in Aalsmeer wonen. Vermeld de gegevens: artikelcode, plantennaam, soort, naam van de leverancier, woonplaats. Sorteer op soort en plantennaam.

8. Bepaal voor iedere artikelcode de laagste offerteprijs. Vermeld in het overzicht de volgende gegevens: artikelcode, plantennaam en de bijbehorende offerteprijs. Sorteer het overzicht op plantennaam.

9. Zoek de bestellingen welke een besteldatum hebben die gelijk is aan de leverdatum van één of meer andere bestellingen.
   Geef bestelnummer 1e bestelling, besteldatum 1e bestelling, bestelnummer 2e bestelling, leveringsdatum 2e bestelling.

10. Welke artikelcodes van het tuincentrum en artikelcodes van een leveranciers zijn hetzelfde ongeacht de plant?
    Geef leverancierscode 1e artikel, artikelcode, leverancierscode 2e artikel, artikelcode van de leverancier.

11. Op een gegeven moment (1 april 1999) moet een overzicht worden vervaardigd, waarop is aangegeven welke bestellingen geleverd zijn. Voor de bestellingen die geleverd zijn moet in een extra kolom de opmerking 'GELEVERD' worden geplaatst; voor de andere bestellingen wordt in die kolom een aantal streepjes geplaatst.
    Sorteer het overzicht op bestelnummer.
    Geef bestelnummer, leveringsdatum, bedrag, bericht 'GELEVERD' of '-------'.
    (In SQLtryout gebruik je #1999-4-1# als getalnotatie)

12. Het tuincentrum wil graag een lijst waarop is aangegeven welke bomen in Aalsmeer en welke buiten Aalsmeer verkrijgbaar zijn. Op het overzicht moeten de volgende gegevens verschijnen: artikelcode, plantennaam, artikelcode van de leverancier, alsmede een aanduiding 'AALSMEER' of 'BUITEN AALSMEER'.

5.17 Subqueries

Bij een subquery wordt het resultaat van een select-instructie gebruikt in een andere sql-instructie.

```
SELECT naam
FROM bieren
WHERE alcohol = (SELECT MAX(alcohol) FROM bieren)
```

geeft een lijst van alle bieren met het hoogste alcoholpercentage. In dit voorbeeld mag de subquery slechts 1 waarde als resultaat gegeven.
Indien je de operatoren =, >, <, >=, <= gebruikt mag de subquery slechts **één** waarde als resultaat gegeven.
Alleen de operator `IN` kan met een set van waarden werken zoals in het volgende voorbeeld getoond wordt.

```
SELECT naam
FROM bieren
WHERE brouwernr IN (SELECT brouwernr FROM brouwers WHERE gemeente
LIKE 'brugge')
```

geeft een lijst van alle bieren die in Brugge gebrouwen zijn.

In deze voorbeelden worden de subqueries gebruikt in de `WHERE`-clausule van de select-instructie. Je kan ook subqueries combineren met de INSERT en UPDATE instructies.

Subqueries kan je ook gebruiken in de `FROM` clausule van een `SELECT`. Eerst wordt dan de subquery uitgevoerd en dan pas de andere `SELECT`-instructie. MySQL vereist echter dat de subquery een alias krijgt met de syntax.

```
SELECT ... FROM (subquery) [AS] name ...
```

Een voorbeeld:

```
SELECT t1.soortnr
FROM (SELECT DISTINCT soortnr, brouwernr FROM bieren) AS t1
GROUP BY soortnr HAVING COUNT(*)  = 1
```

geeft het soortnr van de soorten die maar door 1 brouwerij gebrouwen worden.

```
SELECT soorten.soort, r1.gemiddelde FROM
  (SELECT soortnr, AVG(alcohol) AS gemiddelde
  FROM bieren GROUP BY soortnr) AS r1 INNER JOIN soorten
  ON r1.soortnr = soorten.soortnr
```

In de subquery wordt eerst het gemiddelde alcoholpercentage berekend per soortnr. Gemiddelde is een aliasnaam uit de subquery die gebruikt wordt als kolomnaam in de buitenste query. Ook hier is een aliasnaam verplicht.

5.18 Gecorreleerde subqueries

Een gecorreleerde subquery is een subquery waarin een kolom gebruikt wordt die tot een tabel behoort die in een ander queryblok gespecificeerd is.

De opdracht “Maak een lijst van alle bieren met een alcoholpercentage lager dan het gemiddelde van de eigen soort” luidt als volgt:

```
SELECT b1.*
FROM bieren b1
WHERE b1.alcohol < (
     SELECT AVG(b2.alcohol)
     FROM bieren b2
     WHERE b2.soortnr  = b1.soortnr
     )
```

Hier wordt wel tweemaal gebruik gemaakt van de tabel Bieren. Daarom wordt er aan de tabel een alias toegekend. Om aan te duiden tot welke tabel de velden horen gebruiken we hier de aliasnaam i.p.v. de naam Bieren.

5.19 Oefeningen SELECT deel 5

1. Welke planten zijn hoger dan de gemiddelde hoogte van alle planten tezamen? Druk alle gegevens af.

2. Welke planten zijn duurder dan de gemiddelde prijs van de bomen? Druk alle gegevens ervan af.

3. Maak een overzicht van de leveranciers (alle gegevens) waar nog bestellingen uitstaan met een leverdatum die vóór 1 april 1999 ligt. (In SQLtryout gebruik je #1999-4-1# als getalnotatie)

4. Welke rijen hebben, de laagste offerteprijs van alle offertes in de tabel OFFERTES? Geef alle gegevens.

5. Welke planten zijn lager dan de laagste vaste plant? (alle gegevens geven) Planten waar de hoogte = 0 worden niet meegerekend.

6. Welke planten zijn hoger dan de gemiddelde hoogte van vaste planten en tevens goedkoper dan de gemiddelde prijs van vaste planten? Geef alle gegevens.

7. Welke planten hebben een prijs die tussen de laagste en hoogste prijs van de klimplanten ligt? Geef alle gegevens.

8. Welke bestellingen hebben meer dan 5% korting gekregen? Betrek de tabel BESTREG bij de oplossing.
   Geef alle gegevens.

9. Maak een overzicht van alle artikelcodes die een lagere offerteprijs hebben dan de gemiddelde offerteprijs voor de betreffende artikelcode. Plaats de gegevens artikelcode, leveranciersnaam en offerteprijs op het overzicht, en sorteer op artikelcode.

10. Maak een overzicht van bestelde planten die een bestelprijs hebben welke hoger is dan de maximum offerteprijs voor zo'n plant. Plaats de volgende gegevens op het overzicht: bestelnummer, artikelcode van de leverancier, plantennaam en bestelprijs.

6 GEGEVENS UIT DE DATABASE AANPASSEN

6.1 INSERT

Met deze instructie kan je één of meerdere records toevoegen aan een tabel.

```
INSERT INTO doel [(veld1[, veld2[, ...]])]
VALUES (waarde1[, waarde2[, ...])
```

De instructie INSERT INTO bevat de volgende onderdelen:

Onderdeel	**Beschrijving**
`doel`	De naam van de tabel of query waaraan records worden toegevoegd.
`bron`	De naam van de tabel of query waaruit de records worden gekopieerd. zie verder bij INSERT INTO ... SELECT
`veld1, veld2`	Bij het argument doel de namen van de velden waaraan gegevens worden toegevoegd en bij het argument bron de namen van de velden waaruit de gegevens afkomstig zijn.
`tabelexpressie`	De naam van de tabel of de namen van de tabellen waaruit de toegevoegde records afkomstig zijn. Dit argument kan één tabelnaam zijn of een samenstelling die het resultaat is van een bewerking INNER JOIN, LEFT JOIN of RIGHT JOIN, of een opgeslagen query. zie verder bij INSERT INTO ... SELECT
`waarde1, waarde2`	De waarden die in bepaalde velden van de nieuwe record worden ingevoegd. Elke waarde wordt ingevoegd in het veld dat overeenkomt met de positie van de waarde in de lijst: waarde1 wordt ingevoegd in veld1 van de nieuwe record, waarde2 in veld2, enzovoort. Je moet elke waarde scheiden van de volgende met een komma. Aanhalingstekens zijn enkel nodig voor Strings

Je kan met de instructie INSERT INTO een record toevoegen aan een tabel

- ofwel door sommige/alle veldnamen op te geven:

```
INSERT INTO Soorten (Soortnr, Soort)
VALUES (30, 'Extra donker')
```

voegt een nieuwe soort toe aan de tabel Soorten. Als je niet elk veld opgeeft, wordt in de ontbrekende kolommen de **standaardwaarde** of `NULL` ingevoegd.
Op deze manier kan je ook de volgorde van de velden zelf bepalen:

```
INSERT INTO Soorten (Soort, Soortnr)
VALUES ('Extra donker',30)
```

Als de primaire sleutel van een tabel een Autonummerveld (*Auto increment*) is, veronderstel bijvoorbeeld Soortnr, dan geven we dit veld niet op

```
INSERT INTO Soorten (Soort)
VALUES ('Extra donker')
```

het nummer zal automatisch ingevuld worden.

- ofwel door geen veldnamen op te geven:

```
INSERT INTO Brouwers
VALUES (99, 'Brouwerij Vaattappers', 'Interleuvenlaan 2', 3000,
'Heverlee', NULL)
```

voegt een nieuwe brouwerij toe aan de tabel brouwers.

In dat geval ben je **verplicht de volgorde te respecteren** zoals de velden gedefinieerd zijn. Als je geen waarde voor een veld hebt moet je een `NULL` of een lege string ‘‘ doorgeven, in elk geval iets.

Merk ook op dat MySQL Query Browser de volgende boodschap terugstuurt:
"*1 row affected by the last command, no resultset returned*".
Dergelijke "actie-queries" geven geen set van records terug, enkel een getal dat zegt hoeveel rijen beïnvloed zijn door de query.

Het is ook perfect mogelijk met één INSERT statement meerdere records toe te voegen:

```
INSERT INTO Soorten (Soortnr, Soort)
VALUES
(90, 'witbier'),
(91, 'Ice bier'),
(93, 'honing bier')
```

zorg enkel voor komma’s om de verschillende datasets te scheiden.

Merk op dat een sql backup (bv: *bieren_origineel.sql*) grotendeels bestaat uit dergelijke statements.

6.2 INSERT INTO …SELECT … FROM

Je kan met INSERT INTO ook een set records uit een andere tabel of view selecteren en toevoegen aan een doeltable met de component SELECT ... FROM.

In dat geval geeft de component SELECT aan welke velden worden toegevoegd aan de tabel die wordt aangegeven door doel.

```
INSERT INTO doel [(veld1[,veld2[,...]])]
SELECT [bron.]veld1[, veld2[, ...]
FROM tabelexpressie
```

De bron- of doeltabel kan een bestaande tabel of een *updatable* view zijn (meer informatie over *updatable view* vind je in Hoofdstuk 8 Views). Als een *updatable view* is opgegeven, worden de records door de RDBMS toegevoegd aan alle tabellen die zijn opgegeven in de query.

Het is in MySQL **niet** mogelijk met deze syntax een **nieuwe tabel** te genereren.

Veronderstel dat een tabel *tafelbieren* met dezelfde structuur als *bieren* bestaat (zie verder bij CREATE voor het aanmaken), dan kunnen we met

```
INSERT INTO tafelbieren
SELECT bieren.* FROM bieren
INNER JOIN soorten
ON bieren.soortnr = soorten.soortnr
WHERE soort='tafelbier'
```

de juiste bieren ernaartoe kopiëren. Merk op dat de "tafelbieren" daarmee niet uit *bieren* verwijderd zijn.

6.3 INSERT …ON DUPLICATE KEY UPDATE

Als je ON DUPLICATE KEY UPDATE toevoegt aan een INSERT statement dat je gebruikt op een tabel die een primaire sleutel of een index heeft die enkel unieke waarden toelaat, dan wordt het unieke record bijgewerkt. De syntax is

```
INSERT INTO doel [(veld1[, veld2[, ...]])]
VALUES (waarde1[, waarde2[, ...])
ON DUPLICATE KEY UPDATE
```

of

```
INSERT INTO doel [(veld1[,veld2[,...]])]
SELECT [bron.]veld1[, veld2[, ...]
FROM tabelexpressie
ON DUPLICATE KEY UPDATE
```

Zonder het predikaat “ON DUPLICATE KEY UPDATE” weigert MySQL de INSERT bij een dubbelwaarde.

Dit is nuttig voor situaties waar je wil "*invoegen zoniet bijwerken*".

Veronderstel dat je een tabel *tags* hebt, die bijhoudt hoeveel keer een woord gebruikt wordt .

```
CREATE TABLE tags(
  tag VARCHAR(80) PRIMARY KEY,
  aantal INTEGER
)
```

dan kunnen we het record ofwel invullen ofwel bijwerken met

```
INSERT INTO tags (tag, aantal) VALUES ('php',1)
ON DUPLICATE KEY UPDATE aantal = aantal+1
```

De eerste keer voegt deze query in, alle volgende keren verhoogt hij de teller *aantal*

6.4 Oefening INSERT

1. Het tuincentrum breidt zijn leverancierskring uit met GROEN BV. De volgende gegevens zijn bekend:

Lev_code	045
Lev_naam	GROEN BV.
Adres	ONDER DE LINDE 234
WOONPL	AALSMEER

Voeg deze nieuwe leverancier toe aan de tabel LEVERAN.

2. Op 23 april 1999 wordt een nieuwe bestelling 0205 geplaatst bij leverancier 013. De volgende items werden besteld en moeten in de tabel BESTREG opgenomen worden:

BESTELNR	ART_LEV	AANTAL	PRIJS
0205	C051	10	8.15
0205	B101	200	0.40
0205	B111	25	2.30
0205	G001	50	1.30

De leverdatum wordt op één week na de besteldatum gezet.
De leverancier geeft 8% korting op het bruto bestelbedrag.
Werk dit af in zo weinig mogelijk sql statements en in de juiste volgorde.

6.5 UPDATE

Met deze instructie wijzig je de waarden in velden in een opgegeven tabel op basis van opgegeven criteria.

Syntax voor één tabel:

```
UPDATE tabelexpressie
SET col_name = nieuwewaarde [,col_name = nieuwewaarde]
```

```
[ORDER BY ...]
[WHERE criteria]
[LIMIT rijaantal]
```

Syntax voor meerdere tabellen:

```
UPDATE tabelexpressie
SET col_name = nieuwewaarde [,col_name = nieuwewaarde]
[WHERE criteria]
```

De instructie UPDATE bevat de volgende onderdelen:

Onderdeel	**Beschrijving**
`Tabelexpressie`	De naam van de tabellen (met inner join) met de gegevens die worden gewijzigd en uit geselecteerd worden.
`Nieuwewaarde`	Een expressie die bepaalt welke waarde wordt ingevoegd in een bepaald veld in de bijgewerkte records of `default`, een verwijzing naar de standaardwaarde van die kolom
`Criteria`	Een expressie die bepaalt welke records worden bijgewerkt. Alleen records die voldoen aan de expressie, worden bijgewerkt.
`rijaantal`	maximum aantal rijen dat wordt bijgewerkt

UPDATE is vooral bruikbaar als je een groot aantal records wilt wijzigen of als de te wijzigen records zich in meerdere tabellen bevinden.

Je kan diverse velden in één keer wijzigen. In het volgende voorbeeld worden het adres, postcode en gemeente van de brouwerij met het nummer 99 gewijzigd:

```
UPDATE Brouwers
SET Adres='Keizerslaan 111', Postcode = 1000, Gemeente='Brussel'
WHERE Brouwernr = 99
```

Het alcoholpercentage wordt verhoogd met 0.5 voor bieren die gebrouwen worden in de brouwerij "Artois".

```
UPDATE Bieren INNER JOIN Brouwers
ON Bieren.BrouwerNr = Brouwers.BrouwerNr
SET Alcohol = Alcohol + 0.5
WHERE Brouwers.BrNaam = 'Artois'
```

Je kan deze update instructie ook met een subquery oplossen:

```
UPDATE Bieren
SET alcohol = alcohol+0.5
WHERE BrouwerNr IN
(SELECT BrouwerNr FROM Brouwers
WHERE Brouwers.BrNaam = 'Artois')
```

Belangrijk!

- net als andere "actie-queries" genereert UPDATE geen resultaatset. Je krijgt enkel een bericht met het aantal records dat bijgewerkt werd.

- Als je records hebt bijgewerkt met de UPDATE-instructie, **kan je de bewerking niet meer ongedaan maken**.
  Het is altijd aan te raden vooraf een gewone SELECT-instructie te maken met dezelfde criteria, om te controleren welke records in aanmerking zullen komen! Wijzig de query pas daarna in een UPDATE-instructie.

  In het voorbeeld hierboven is het mogelijk een andere UPDATE query te maken die de prijsverhoging ongedaan maakt door opnieuw 0.5 alcohol af te trekken voor de Artois bieren.

  Maar voeren we echter deze update uit:

```
UPDATE Bieren
SET BrouwerNr = 3
```

  dan heb je het brouwernummer voor **alle** bieren op 3 gezet en kan je met geen enkele mogelijkheid meer achterhalen wat hun vroegere waarde was…

6.6 Oefeningen UPDATE

1. Wijzig in de tabel BESTEL de leverdatum in 5 april 1999 voor het bestelnummer 0191.

2. Verhoog alle offerteprijzen van de bolgewassen in de tabel OFFERTES met 10%.

6.7 DELETE

Met deze instructie verwijder je de records uit een of meer tabellen die zijn opgegeven in de component FROM en die voldoen aan de component WHERE.

Eén tabel syntax:

```
DELETE
FROM tabel
WHERE criteria
[LIMIT rijaantal]
```

Syntax voor meerdere tabellen:

```
DELETE tabel1[.*][,tabel2[.*]]
FROM Tabelexpressie
WHERE criteria
```

De instructie DELETE bevat de volgende onderdelen:

Onderdeel	Beschrijving
`tabel`	De naam van de tabel waaruit records worden verwijderd.
`tabelexpressie`	Expressie die de tabellen bevat waaruit records worden verwijderd.
`criteria`	Een expressie die aangeeft welke records worden verwijderd.
`rijaantal`	maximum aantal rijen dat wordt gewist

DELETE gebruik je om een aantal records te verwijderen uit de tabel zonder de tabel zelf te vernietigen.

Gebruik de instructie DROP als je een volledige tabel wilt verwijderen uit de database. De structuur gaat echter verloren als je de tabel verwijdert.
Als je daarentegen DELETE gebruikt, worden alleen de gegevens verwijderd. De tabelstructuur en alle tabeleigenschappen, zoals veldkenmerken en indexen, blijven bestaan.

Je kan met DELETE records verwijderen uit tabellen die een één-op-veel-relatie hebben met andere tabellen. Ook hier wordt de referentiële integriteit (zie verder) bewaakt. Als er een relatie bestaat tussen de twee tabellen, dan kan je enkel een record verwijderen uit de tabel Brouwers als er geen records voorkomen in de tabel Bieren met dat Brouwernr.

Met een DELETE worden volledige records verwijderd en niet alleen de gegevens in bepaalde velden. Als je waarden in een bepaald veld wilt verwijderen, kan je een UPDATE instructie geven waarmee de waarden worden gewijzigd in `Null`.

```
DELETE FROM bieren
WHERE biernr = 52
```

verwijdert het record met het nummer 52 uit de tabel Bieren

```
DELETE brouwers.*
FROM brouwers
WHERE brouwernr IN
  (SELECT brouwernr FROM BIEREN WHERE alcohol > 20)
```

verwijdert alle brouwerijen die een bier produceren met een alcoholpercentage dat hoger is dan 20%. Mbv de subquery

Merk op dat je deze query korter kan schrijven als

```
DELETE FROM brouwers
WHERE brouwernr IN
(SELECT brouwernr FROM BIEREN WHERE alcohol > 20)
```

Hetzelfde resultaat kon ook bereikt worden met een JOIN ipv van een subquery

```
DELETE brouwers.* FROM brouwers INNER JOIN bieren
  ON brouwers.brouwernr = bieren.brouwernr
WHERE alcohol > 20
```

verwijdert alle brouwerijen die een bier produceren met een alcoholpercentage dat hoger is dan 20%.

```
DELETE brouwers.*, bieren.*
FROM brouwers INNER JOIN bieren
  ON brouwers.brouwernr = bieren.brouwernr
WHERE brnaam = 'DECA'
```

verwijdert de brouwerij 'DECA' **en** alle bieren die in deze brouwerij geproduceerd worden.

Opmerkingen!

- Als je records hebt verwijderd met DELETE, kan je de bewerking **niet meer ongedaan maken**.
  Als je wilt weten welke records zullen worden verwijderd, kan je eerst het resultaat van een SELECT instructie met dezelfde voorwaarden onderzoeken en vervolgens de DELETE instructie uitvoeren.

- gebruik LIMIT om het aantal records te beperken. LIMIT is bijvoorbeeld aan te raden in DELETE statements die via scripting (php) met een parameter uitgevoerd worden. Veronderstel dat de gebruiker de mogelijkheid geboden wordt om een bier via de naam te wissen, maar ipv van een naam geeft hij een % teken door, dan kan het statement resulteren in

  ```
  DELETE FROM bieren WHERE naam LIKE '%'
  ```

  dan zijn nu alle bieren gewist! Om dit te vermijden schrijf je best:

  ```
  DELETE FROM bieren WHERE naam LIKE 'Mijnbier' LIMIT 1
  ```

  zo kan er maximaal 1 record per keer gewist worden.

6.8 Oefening DELETE

1. Verwijder alle rijen uit de tabel OFFERTES die betrekking hebben op de heesters van leverancier 021

2. Verwijder alle gegevens over de waterplanten in zowel de tabel PLANTEN als OFFERTES. Let op de volgorde van de behandeling.

7 BEHEER VAN TABELLEN EN RELATIES

7.1 CREATE TABLE

Algemene syntax:

```
CREATE TABLE [IF NOT EXISTS] TABLE
(velddefinities)
[tabelopties]
```

bemerk dat alle velddefinities in ronde haakjes staan. Eventuele tabelopties volgen daarna. Meer specifieke syntax:

```
CREATE TABLE [IF NOT EXISTS] tabel
(
veld1 type [(grootte)] [NOT NULL] [AUTO_INCREMENT][index1]
[, veld2 type [(grootte)] [NOT NULL] [index2]
[, ...]]
[,meervoudigeindex]
)
[tabelopties]
```

De instructie CREATE TABLE bevat de volgende onderdelen:

Onderdeel	**Beschrijving**
`tabel`	De naam van de tabel die je wilt maken.
`veld1, veld2`	De naam van het veld of de velden die je in de tabel wilt opnemen. Er moet minstens één veld worden gemaakt.
`type`	Het gegevenstype van het veld in de nieuwe tabel.
`grootte`	De veldgrootte in tekens (alleen voor tekst- en binaire velden).
`index1, index2`	Een component CONSTRAINT die een index met een enkelvoudig veld definieert.
`Meervoudigeindex`	Een component CONSTRAINT die een index met meervoudige velden opgeeft.

Met de instructie CREATE TABLE kan je een nieuwe tabel inclusief velden en beperkende voorwaarden voor velden opgeven.

De naam van een tabel, kolom of ander schema object (index, etc.) is vrije keuze, ze mag een _ (underscore) of een $ (dollar) bevatten. Ze mag ook andere speciale karakters zoals een spatie bevatten, maar dan moet je de naam in aanhalingstekens zetten.
Het standaard aanhalingsteken in MySQL is de **backtick** ` (accent grave).
Zo zijn bijvoorbeeld `*mijn kolom*` en *$_kolom* toegelaten. Bedenk wel dat je nadien

zult moeten blijven de aangehaalde naam gebruiken om de spatie te "omzeilen". Gereserveerde woorden zoals COUNT en INTERVAL kunnen op die manier ook gebruikt worden als Schema Objectnaam.

Het attribuut IF NOT EXISTS verhindert een foutbericht als er reeds een tabel van die naam voorkomt in de database. MySQL controleert helemaal niet of die ook dezelfde structuur heeft en overschrijft die ook niet.

Als NOT NULL is opgegeven voor een veld moet de inhoud van het veld steeds opgevuld zijn.

Je kan een bepaalde kolom de primaire sleutel van de tabel maken met PRIMARY KEY. Door meer dan één veld in te geven maak je een meervoudige primaire sleutel. Een primaire sleutel is een UNIQUE, INDEX, NOT NULL veld

Het attribuut AUTO_INCREMENT kan gebruikt worden op een integer of een floating-point veld. Daarmee maak je er een autonummering veld van waarbij het veld automatisch de vorige waarde + 1 krijgt. Er kan slechts één veld AUTO_INCREMENT krijgen en het moet ook geïndexeerd worden.

Er zijn heel wat **tabelopties**, we beperken ons tot ééntje: ENGINE. Het sleutelwoord TYPE is een alias voor ENGINE. Hiermee kunnen we de *storage engine* kiezen waarmee een tabel gemaakt/behandeld wordt. Bijvoorbeeld:

```
CREATE TABLE klanten (
  klantnr INT NOT NULL,
  klnaam VARCHAR (30) NOT NULL
)ENGINE=INNODB
```

Hiermee wordt de tabel *klanten* van het type InnoDB. De default engine is MyISAM. Het is onnodig in deze cursus hier verder op uit te wijden.

7.1.1 De voornaamste data types in MySQL.

De verschillende types kunnen mét of zonder argument gebruikt worden, dus bv. BIT of BIT(20). Zonder argumenten krijgen ze de standaardwaarde mee.

Numerieke data types: gebruikt om getallen op te slaan.

Naam	Opslagwijze	Betekenis
BIT	1 bit	Een bitwaarde veld. Standaard 1 bit en kan dan enkel de waarde 0 of 1 bevatten.
BOOL, BOOLEAN	1 byte	is een synonym voor TINYINT(1). Gebruikt voor `true/false` waarden. 0 staat gelijk aan `false`, alle andere waarden worden omgezet naar 1 dat gelijk staat aan `true`.

INT, INTEGER	4 bytes	Een geheel getal. 2^{32} mogelijke waarden. Kan een getal bevatten met een waarde tussen -2147483648 en 2147483647. Variaties op INT zijn: TINYINT (1byte, 2^8 waarden), SMALLINT(2 bytes, 2^{16} waarden), MEDIUMINT(3 bytes, 2^{24} waarden), BIGINT(8 bytes, 2^{64} waarden)
FLOAT		Decimaal getal enkele precisie. Kan een getal bevatten met een waarde tussen $1.18x10^{-38}$ tot $3.40x10^{+38}$, of negatief. Een dergelijk getal is ongeveer precies tot 7 decimalen
DOUBLE		Decimaal getal dubbele precisie. Kan een getal bevatten met een waarde tussen $2.23x10^{-308}$ tot $1.80x10^{+308}$, of negatief. Een dergelijk getal is ongeveer precies tot 15 decimalen

Tekst data types (String): gebruikt om tekst op te slaan.

Naam	**Opslagwijze**	**Betekenis**
CHAR(M)		Een tekst van vaste lengte. Als je geen lengte `M` opgeeft, krijg je standaard CHAR(1). In een CHAR(1) kan je dus slechts één letter opslaan. De maximumlengte is 255. Dit datatype wordt steeds opgeslagen met de vaste lengte die je specificeerde. CHAR is dus niet efficiënt op gebied van ruimte, daarentegen is het wel sneller om te lezen dan VARCHAR.
VARCHAR(M)		Een tekst van variabele lengte met een maximum van `M`. In een VARCHAR(10) kan je dus tekst van maximaal 10 letters opslaan. Kleiner maar trager dan CHAR.
TEXT		gebruikt om grote stukken tekst op te slaan. Maximaal 64Kb
BLOB		gebruikt om binaire data op te slaan, zoals beelden. Wordt opgeslagen als een binair TEXT veld. Maximaal 64KB.

		Er bestaan van TEXT en BLOB ook varianten die minder of meer kunnen opslaan.
ENUM		Een enumeratie. Dat is een lijst van mogelijke waarden, waaruit een keuze gemaakt zal moeten worden. Bijvoorbeeld ENUM('vrouw','man'). Dit type kan ook een `Null` waarde hebben.
SET		Identiek aan ENUM, maar de gebruiker kan een aantal van de waarden kiezen, niet slechts één.

Datum/Tijd types.

Naam	**Opslagwijze**	**Betekenis**
DATE		Een datum in de ISO *jaar-maand-dag* volgorde. Datums worden standaard getoond in `YYYY-MM-DD` formaat
TIME		Een tijd in het `HH:MM:SS` formaat.
DATETIME		Een combinatie van de vorige twee. Formaat `YYYY-MM-DD HH:MM:SS`
TIMESTAMP		bewaart de datum-tijd waarop de record de laatste maal gewijzigd werd. Wordt getoond in het DATETIME formaat.
YEAR		Bewaart een jaartal.

Enkele voorbeelden:

```
CREATE TABLE klanten (
  klantnr INT NOT NULL, klnaam VARCHAR (30) NOT NULL,
  kladres VARCHAR (40),
  klpost VARCHAR(4),
  klgemeente VARCHAR (40)
)
```

Creëert een nieuwe tabel *klanten* met de velden *klantnr, klnaam, kladres, klpost,* klgemeente. De velden *klantnr* en *klnaam* moeten een waarde bevatten.

```
CREATE TABLE klanten (
  klantnr INT AUTO_INCREMENT PRIMARY KEY,
  klnaam VARCHAR(30) NOT NULL,
  kladres VARCHAR(40),
  klpost VARCHAR(4),
```

```
  klgemeente VARCHAR(40)
)
```

Maakt eenzelfde tabel maar het veld *klantnr* heeft nu autonummering en is gedefinieerd als primaire sleutel.

7.2 NULL versus NOT NULL

NULL en NOT NULL maken deel uit van de kolomdefinitie. Hiermee bepaal je of een veld leeg gelaten mag worden.

Constraint	Omschrijving
NULL	de waarde NULL is toegelaten.
NOT NULL	de kolom moet steeds ingevuld zijn, de waarde NULL is niet toegelaten.

In het voorbeeld hierboven zie je hoe het veld *klnaam* verplicht ingevuld moet worden.

7.3 Default waarde

Als er een `DEFAULT` gedefinieerd wordt voor een kolom, wordt die waarde toegekend aan die kolom, als bij toevoeging van de rij de kolom **leeg** gelaten werd. Je kan een constante, een berekening of een functie gebruiken bij de definitie van het *default*. Het resultaat moet wel een constante(waarde) zijn.

Als je de DEFAULT onmiddellijk wilt definiëren bij een CREATE TABLE instructie:

```
CREATE TABLE bestellijn
(
  bestelnr INT NOT NULL,
  biernr INT NOT NULL REFERENCES bieren (biernr),
  aantal INT NOT NULL DEFAULT 1,
  PRIMARY KEY (bestelnr, biernr)
) ENGINE=INNODB
```

Als aantal niet ingevuld wordt bij het toevoegen van een record, wordt de waarde van aantal 1.

Je kan de default waarde ook nadien specificeren met
ALTER TABLE … ALTER kolom SET DEFAULT:

```
ALTER TABLE bestellijn
ALTER aantal SET DEFAULT 1
```

Je kan de default waarde ook terug verwijderen:

```
ALTER TABLE bestellijn
ALTER aantal DROP DEFAULT
```

7.4 Primary key

De PRIMARY KEY, of de **primaire sleutel** van een tabel, is één kolom of een combinatie van kolommen die samen een unieke index vormen. De velden van een *primary key* (PK) moeten UNIQUE en NOT NULL zijn. Met andere woorden, ze moeten een waarde bevatten die uniek is. Een table kan maar één PK hebben, terwijl hij meerdere gewone indexen kan bevatten.

Een PK kan gedefinieerd worden tijdens de aanmaak van een tabel. Als de PK één veld bevat, kan je dat specificeren bij de kolom:

```
CREATE TABLE klanten
(
  klantnr INT NOT NULL PRIMARY KEY,
  naam VARCHAR(50) NOT NULL,
  adres VARCHAR(50) NOT NULL,
  postcode CHAR(4) NOT NULL,
  woonplaats VARCHAR(50) NOT NULL
) ENGINE = INNODB
```

De kolom *klantnr* is de sleutel.

Heeft de tabel een samengestelde PK, dan specificeer je die achteraan de kolomdefinities:

```
CREATE TABLE bestellijn
(
  bestelnr INT NOT NULL,
  biernr INT NOT NULL,
  aantal INT NOT NULL,
  PRIMARY KEY (bestelnr, biernr)
) ENGINE=INNODB
```

Bestelnr en biernr moeten steeds ingevuld zijn en vormen de PK.

Je kan de PK ook nadien definiëren in een ALTER TABLE instructie:

```
ALTER TABLE bestellijn
ADD PRIMARY KEY (bestelnr)
```

Bestelnr wordt de PK van deze tabel.

7.5 Indexen

Een index is een structuur die de zoeksnelheid op een tabel veel kan versnellen.

Indien je veel zoekt op een bepaald veld is het aan te raden om er een index op te plaatsen. Een index kan ook een combinatie van meerdere velden bevatten.

Indexen worden aangemaakt:

- tijdens de aanmaak van een tabel met INDEX of KEY (synoniem)

```
CREATE TABLE klanten(
  klantnr INT NOT NULL PRIMARY KEY,
  naam VARCHAR(50)  NOT NULL,
```

```
  adres VARCHAR(50) NOT NULL,
  postcode CHAR(4) NOT NULL,
  woonplaats VARCHAR(50) NOT NULL,
  INDEX(naam)
) ENGINE = INNODB
```

De tabel klante*n* krijgt naast een primary key ook een index op het veld *naam*.

- nadien, met de instructie CREATE INDEX

```
CREATE INDEX indexnaam ON tabelnaam(kolomnaam, kolomnaam, ...)
```

- nadien, met de instructie ALTER TABLE

```
ALTER TABLE tbl_naam ADD INDEX indexnaam(kolomnaam, ...)
```

In elk van die gevallen krijgt de index een indexnaam - in het eerste geval wordt dat eenvoudigweg de kolomnaam.
In MySQL is het sleutelwoord KEY een synoniem voor INDEX.

Er zijn ook een aantal speciale indexen, waaronder UNIQUE.

Een UNIQUE index maakt een beperking (constraint, zie verder) waardoor de waarden in het veld uniek moeten zijn. Als we voorgaande voorbeelden aanpassen krijgen we

```
CREATE TABLE klanten(
  klantnr INT NOT NULL PRIMARY KEY,
  naam VARCHAR(50)  NOT NULL,
  adres VARCHAR(50) NOT NULL,
  postcode CHAR(4) NOT NULL,
  woonplaats VARCHAR(50) NOT NULL,
  UNIQUE(naam)
) ENGINE = INNODB
```

7.6 CREATE INDEX, DROP INDEX

De volledige syntax van CREATE INDEX is

```
CREATE [ UNIQUE ] INDEX INDEX_veld
ON tabel (veld [ASC|DESC][, veld [ASC|DESC], ...])
```

De instructie CREATE INDEX bevat de volgende onderdelen:

Onderdeel	**Beschrijving**
`index`	De naam van de index die je wilt maken.
`tabel`	De naam van de bestaande tabel waarvoor de index wordt gemaakt.
`veld`	De naam van het veld of de velden die je in de index wilt opnemen. Als je een enkelvoudige index wilt maken, plaats je de veldnaam tussen haakjes achter de tabelnaam. Als je een meervoudige index wilt maken, geef je de naam op van elk veld dat in de index moet worden opgenomen. Als je een aflopende

	index wilt maken, neem je het gereserveerde woord DESC op, anders wordt een oplopende index gemaakt.

Dubbele waarden in het geïndexeerde veld of de geïndexeerde velden van verschillende records kan je voorkomen met het gereserveerde woord UNIQUE.

In MySQL wordt dit statement infeite vertaalt naar een ALTER TABLE statement, dat wat meer mogelijkheden biedt.

In MySQL kan je geen primary key maken met CREATE INDEX, gebruik daarvoor ALTER TABLE.

```
CREATE INDEX indexnaam ON klanten (naam)
```

Creëert een index *indexnaam* op het veld *naam* van de tabel *klanten*

Met de instructie DROP INDEX kan je de index verwijderen.

```
DROP INDEX indexnaam ON klanten
```

Verwijdert de index *indexnaam* van de tabel *klanten*.

7.7 CREATE TABLE …SELECT

Met deze syntax maak je een **nieuwe tabel** aan op basis van geselecteerde records uit een andere tabel.

```
CREATE TABLE nieuw_tabel SELECT * FROM orig_tabel
```

Een voorbeeld:

```
CREATE TABLE alcoholarm
SELECT naam, brnaam
FROM bieren, brouwers, soorten
WHERE bieren.brouwernr=brouwers.brouwernr
AND bieren.soortnr=soorten.soortnr
AND soorten.soort='Alcoholarm'
```

Selecteert de kolommen *naam* en *brnaam* van alle bieren die van de soort "Alcoholarm" zijn en maakt een nieuwe tabel *alcoholarm* aan.

De nieuwe table bevat dus twee kolommen.

7.8 CREATE TABLE LIKE

Met deze syntax maak je een lege, nieuwe tabel aan op basis van de structuur van een andere tabel. Je kopieert dus enkel de structuur, niet de records.

```
CREATE TABLE oudeklanten LIKE klanten
```

maakt een nieuwe tabel *oudeklanten* met dezelfde structuur als *klanten*.

7.9 Oefeningen CREATE TABLE

1. Creëer de tabel klachten met de volgende velden:

veld	data-type	Lengte
klachtnr	numeriek	4
art_code	karakter	3
datum	datum	
klacht	karakter	100
status	karakter	2

   Vul de tabel KLACHTEN met de klacht nr. 1 die betrekking heeft op artikel-code 316. De klacht luidt: "*planten (lev_code 019) verkocht op 14-1-1999 vertoonden bruine plekken na circa 2 maanden*".
   De status is GL (gemeld aan leverancier) en de klacht wordt ingebracht op 15-3-1999.

2. Er is behoefte aan een tabel AANBIEDINGEN waaruit snel de goedkoopste leveranciers van de planten uit de tabel PLANTEN kunnen worden opgezocht. De tabel heeft de volgende kolommen: ART_CODE, PLANTENNM, LEV_CODE en OFF_PRIJS.
   Maak en vul deze tabel met één statement vanuit de PLANTEN en OFFERTES tabellen.

3. Maak een nieuwe, maar lege tabel OUDEBESTELLINGEN met dezelfde structuur als BESTEL

7.10 DROP TABLE

Deze instructie verwijdert een bestaande tabel uit een database.

```
DROP TABLE tabel
```

De instructie DROP bevat de volgende onderdelen:

Onderdeel	Beschrijving
`Tabel`	De naam van de tabel die je wilt verwijderen

Voordat je een tabel kunt verwijderen, moet je de tabel sluiten.

```
DROP TABLE klanten
```

verwijdert de tabel klanten uit database.

7.11 ALTER TABLE

Met deze instructie wijzig je het ontwerp van een tabel die reeds is aangemaakt.

```
ALTER TABLE tabel
{
ADD {[COLUMN] veld [(column_definition)]
    | CONSTRAINT index} |
DROP {[COLUMN] veld
    | CONSTRAINT index} |
MODIFY [COLUMN] veld column_definition
}
```

De instructie ALTER TABLE bevat de volgende onderdelen:

Onderdeel	Beschrijving
`Tabel`	De naam van de tabel die je wilt wijzigen.
`Veld`	De naam van het veld dat je wilt toevoegen aan of verwijderen uit de tabel.
`Column_definition`	De specificaties voor het veld zoals het type, de grootte, etc…

De mogelijkheden van ALTER TABLE zijn zeer uitgebreid, we belichten slechts enkele mogelijkheden:

- Een nieuw veld toevoegen aan de tabel met ADD COLUMN.

  U geeft de veldnaam, het gegevenstype en (voor tekst- en binaire velden) een optionele grootte op. De volgende instructie voegt bijvoorbeeld aan de tabel Brouwers een veld van 25 karakters met de naam Opmerkingen toe:

```
ALTER TABLE Brouwers ADD COLUMN Opmerkingen CHAR(25)
```

- Een veld verwijderen met DROP COLUMN.
  U hoeft alleen de naam van het veld op te geven.

```
ALTER TABLE Brouwers DROP COLUMN Opmerkingen
```

- De velddefinitie van een kolom wijzigen met MODIFY:

```
ALTER TABLE klanten MODIFY COLUMN naam VARCHAR(100) NOT NULL
```

- Je kan ook een index definiëren op een veld met ADD INDEX

```
ALTER TABLE klanten ADD INDEX (naam), ADD UNIQUE (adres)
```

Opmerkingen:

- Je kan per keer slechts één veld of index toevoegen of verwijderen.

- Met de instructie CREATE INDEX kan je een index met een enkelvoudig veld of met meervoudige velden toevoegen aan een tabel. Met de instructie ALTER TABLE of DROP kan je een index verwijderen die is gemaakt met ALTER TABLE of CREATE INDEX.

- Je kan NOT NULL gebruiken in een enkelvoudig veld of binnen een benoemde component CONSTRAINT die geldt voor een benoemde CONSTRAINT met ofwel een enkelvoudig veld, dan wel met meervoudige velden. De beperking NOT NULL kan je echter slechts één keer toepassen op een veld, anders treden er runtime-fouten op.

- bij het wijzigen van een kolom, MOET je het datatype en alle andere attributen opnieuw meegeven, want anders bestaat de kans dat het datatype wijzigt of dat de attributen achterwege gelaten worden

7.12 Oefeningen Alter

1. Breid de tabel PLANTEN uit men een kolom VOORRAAD (numeriek 4 cijfers) voor het aantal stuks dat het tuincentrum van de desbetreffende plant nog in voorraad heeft.

2. Definieer als standaardkleur voor planten de kleur groen.

7.13 Constraints

Een ***constraint*** is een beperkende voorwaarde voor één of meerdere kolommen. Indexes zijn *constraints*, NOT NULL en UNIQUE zijn *constraints*, maar je kan er evengoed een relatie met een andere tabel tot stand mee brengen en meer.

Met het sleutelwoord CONSTRAINT in de instructies ALTER TABLE en CREATE TABLE kan je dergelijke voorwaarden instellen. We moeten echter direct opmerken dat het sleutelwoord CONSTRAINT optioneel is in veel gevallen: zo hebben we reeds gezien dat o.a. het instellen van een primary key of een index op eenvoudige wijze kan zonder CONSTRAINT te gebruiken.

De syntax met CONSTRAINT is wat formeler en laat toe de voorwaarde te benoemen, zodat je die later ook makkelijk kunt verwijderen.

Er zijn twee typen CONSTRAINT-componenten: één om een beperkende voorwaarde in te stellen op een enkelvoudig veld, en één om een beperkende voorwaarde in te stellen op meerdere velden.

Beperkende voorwaarde voor één veld:

```
CONSTRAINT naam {PRIMARY KEY | UNIQUE | NOT NULL | REFERENCES
refererendetabel [(refererendveld1)]}
```

Beperkende voorwaarde voor meerdere velden:

```
CONSTRAINT naam
{PRIMARY KEY (primair1[, primair2 [, ...]]) |
UNIQUE (uniek1[, uniek2 [, ...]]) |
```

```
FOREIGN KEY (verw1[, verw2 [, ...]])
REFERENCES refererendetabel [(refererendveld1[, refererendveld2[,
...]])]}
```

De component CONSTRAINT bevat de volgende onderdelen:

Onderdeel	**Beschrijving**
`Naam`	De naam van de beperkende voorwaarde die je wilt maken.
`Primair1, primair2`	De naam van het veld of de velden die je wilt aanwijzen als de primaire sleutel(s).
`Uniek1, uniek2`	De naam van het veld of de velden die je wilt aanwijzen als unieke sleutel(s).
`Verw1, verw2`	De naam van een of meer refererende sleutelvelden die verwijzen naar velden in een andere tabel.
`Refererendetabel`	De naam van de refererende tabel met het veld of de velden die worden aangegeven door refererendveld.
`Refererendveld1, refererendveld2`	De naam van het veld of de velden in refererendetabel die worden aangegeven door verw1, verw2. Je kan deze component weglaten als het veld waarnaar wordt verwezen, de primaire sleutel is van refererendetabel.

De syntax voor een beperkende voorwaarde voor één veld gebruik je in de velddefinitie van een instructie ALTER TABLE of CREATE TABLE.

Bijvoorbeeld om de primary key en een index aan te geven:

```
CREATE TABLE klanten
(
  klantnr INT NOT NULL,
  naam VARCHAR(50)  NOT NULL,
  adres VARCHAR(50) NOT NULL,
  postcode CHAR(4) NOT NULL,
  woonplaats VARCHAR(50) NOT NULL,
  CONSTRAINT pk_klanten PRIMARY KEY (klantnr),
  CONSTRAINT i_naam UNIQUE (naam)
) ENGINE = INNODB
```

Omdat het sleutelwoord CONSTRAINT hier optioneel is, mag je het eenvoudigweg achterwege laten en de verkorte syntax gebruiken die we al eerder uitlegden.

7.13.1 FOREIGN KEY constraints

Om relaties tussen tabellen te leggen kunnen we ook CONSTRAINT gebruiken.
Opmerking: enkel `INNODB` tabellen ondersteunen foreign key constraints.

```
[CONSTRAINT [symbol]]
FOREIGN KEY[INDEX_name] (INDEX_col_name, ...)
REFERENCES tbl_name (INDEX_col_name,...)
[ON DELETE (RESTRICT | CASCADE | SET NULL | NO ACTION)]
[ON UPDATE (RESTRICT | CASCADE | SET NULL | NO ACTION)]
```

Met FOREIGN KEY kan je een veld (in de *parent table*) aanwijzen als refererende sleutel.

Met REFERENCES geef je het/de veld(en) aan van de refererende tabel (*child table*) waarnaar wordt verwezen. Als het veld of de velden waarnaar wordt verwezen, de primaire sleutel van de refererende tabel vormen, hoef je de velden niet op te geven.

De foreign key en reference veld(en) moeten van een vergelijkbaar datatype zijn.

De optionele clausule ON DELETE bepaalt wat er gebeurt als het record in de *parent table* verwijderd wordt, ON UPDATE bepaalt wat er gebeurt als de foreign key waarde in de *parent table* gewijzigd wordt. De mogelijke settings zijn:

- RESTRICT: de `DELETE` of `UPDATE` voor de parent table wordt geweigerd, er gebeurt niets.
  Dit is ook de standaardinstelling als de clausules ONDELETE/UPDATE niet gespecificeerd zijn.

- NO ACTION: in MySQL is dit hetzelfde als RESTRICT

- SET NULL: bij een `DELETE` of `UPDATE` van de *parent table* wordt de waarde van de gerefereerde kolom(men) in de *child table* op `NULL` gezet. Deze kolommen mogen uiteraard geen NOT NULL ingesteld hebben

- CASCADE: bij een `DELETE` van een record in de *parent table* wordt dit record verwijderd en automatisch ook de gerelateerde rijen van de child table. Bij een update van een record in de *parent table* wordt dit record gewijzigd en automatisch ook de gerelateerde rijen van de child table bijgewerkt.

Een *foreign key constraint* bepaalt de **referentiële integriteit**: de RDBMS gaat de ingegeven waarde van een *foreign key* controleren bij het *reference* veld: als die waarde daar niet voorkomt wordt de ingave geweigerd.
Bijvoorbeeld: als een bestelling ingegeven wordt met een klantNr (FK) die niet voorkomt als PK in de tabel klanten, wordt de ingave geweigerd. Op dezelfde wijze wordt een UPDATE en een DELETE van de PK aan de regels onderworpen hierboven gespecificeerd.

Voorbeeld 1: de 1-N relatie tussen klant en bestellingen.aanmaak klanten:

```
CREATE TABLE klanten
(
  klantnr INT NOT NULL PRIMARY KEY,
  naam VARCHAR(50)  NOT NULL,
  adres VARCHAR(50) NOT NULL,
  postcode CHAR(4) NOT NULL,
  woonplaats VARCHAR(50) NOT NULL
) ENGINE = INNODB
```

we kunnen de relatie onmiddellijk inbouwen in de aanmaak van bestellingen:

```
CREATE TABLE bestellingen
(
  bestelnr INT(11) NOT NULL PRIMARY KEY,
  klantnr INT(11) NOT NULL,
  besteldatum datetime NOT NULL,
  CONSTRAINT fk_klantnr FOREIGN KEY (klantnr)
  REFERENCES klanten(klantnr)
  ON DELETE CASCADE
  ON UPDATE CASCADE
) ENGINE=InnoDB
```

of korter

```
CREATE TABLE bestellingen
(
  bestelnr INT(11) NOT NULL PRIMARY KEY,
  klantnr INT(11) NOT NULL REFERENCES klanten(klantnr)
  ON DELETE CASCADE ON UPDATE CASCADE,
  besteldatum datetime NOT NULL
) ENGINE=INNODB
```

Veronderstel dat we de relatie niet onmiddellijk voorzien:

```
CREATE TABLE bestellingen
(
  bestelnr INTEGER NOT NULL PRIMARY KEY,
  klantnr INTEGER NOT NULL,
  besteldatum datetime NOT NULL
)ENGINE = INNODB
```

Om nu de 1-N relatie toe te voegen:

```
ALTER TABLE bestellingen
ADD CONSTRAINT fk_klanten FOREIGN KEY (klantnr)
REFERENCES klanten (klantnr)
ON DELETE CASCADE
ON UPDATE CASCADE
```

of

```
ALTER TABLE bestellingen
ADD FOREIGN KEY (klantnr)
```

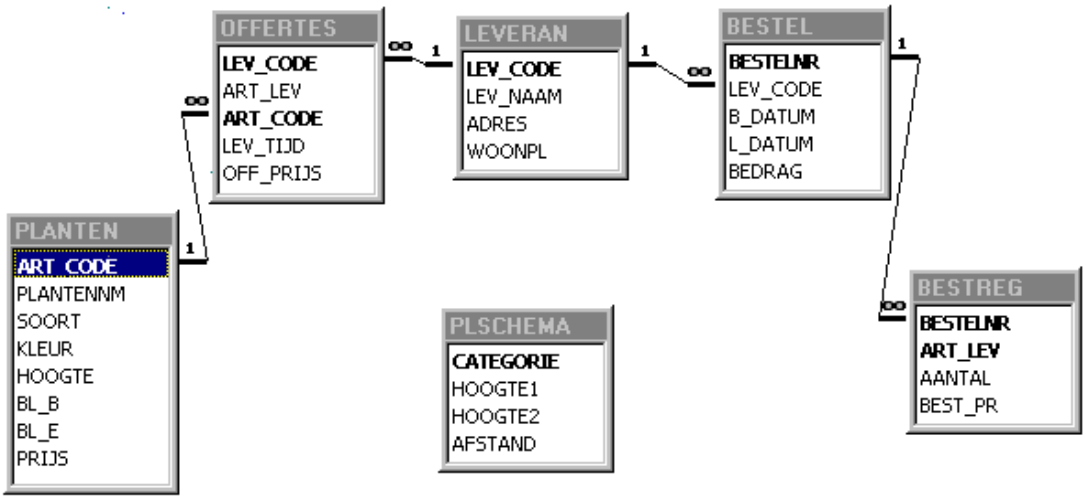
```
REFERENCES klanten (klantnr)
ON DELETE CASCADE
ON UPDATE CASCADE
```

Zoals je ziet hoeven we CONSTRAINT niet te gebruiken. Het heeft echter wel één voordeel: de *constraint* heeft nu een naam '*fk_klanten*', en als we die willen wissen is het eenvoudig:

```
ALTER TABLE bestellingen DROP FOREIGN KEY fk_klanten
```

7.14 Oefeningen integriteit en indexen

1. Definieer de nodige primary keys en de relaties op de tabellen van de database planten. Gebruik hiervoor het schema dat je in het begin van de cursus tegenkwam.



   Het is mogelijk dat de reeds aanwezige gegevens het aanmaken van sommige relaties verhinderen. Hoe pak je dit probleem aan?

2. Specificeer een index BSRIDX01 op de samengestelde sleutel bestelnr en art_lev van de tabel BESTREG.

3. De tabel OFFERTES wordt regelmatig geJOINed met de tabellen PLANTEN, LEVERAN en BESTREG. Welke indexen zijn waardevol voor deze tabel OFFERTES? maak ze

8 VIEWS

Een view is een **virtuele tabel** als resultaat van een query. Een view kan je beschouwen als een opgeslagen query.

Ze zijn interessant omdat ze alle mogelijkheden bieden van een query, maar met aparte gebruikersrechten. Zo kan je gebruikers slechts een beperkt zicht op de gegevens bieden, terwijl de tables zelf afgeschermd worden. Bijvoorbeeld in een personeelsdatabase, kan je met een view de kolommen 'salaris' en 'adres' onzichtbaar maken voor gebruikers of met een view kan je **alle** gegevens voor de personeelsgroep 'spionnen' uitsluiten…

Omdat views net als tabellen zijn, kan je ook queries uitvoeren op views.

Afhankelijk van de soort `SELECT`-instructie die gebruikt wordt, heeft een view de mogelijkheid om gegevens toe te voegen, te wijzigen en te wissen, een *updatable view*, of niet: een *read-only view*.

Om *updatable* te zijn moet een *view* aan de volgende criteria beantwoorden:

- gebouwd op 1 tabel, de *primary key* inbegrepen
- geen GROUP BY, HAVING, UNION, SELECT DISTINCT clausules
- geen aggregate functions (AVG(), SUM(), …)
- geen berekende kolommen
- geen subquery in de select
- geen joins

8.1 Create view

De syntax om een view aan te maken

```
CREATE VIEW viewnaam
AS
SELECT-instructie
```

Enkele voorbeelden:

Views worden soms gebruikt om de toegang van gebruikers tot bepaalde gegevens te verhinderen:

```
CREATE VIEW BrouwersBeperkt
AS
SELECT B.BrouwerNr, B.BrNaam, B.Adres, B.PostCode,B.Gemeente
FROM brouwers B
```

maakt een (*updatable*) view die alle velden van Brouwers toont met uitzondering van het veld *omzet*

Views worden dikwijls gemaakt om gebruikers berekende gegevens te bieden zoals de voorbeelden hieronder verduidelijken.

Voorbeeld 1:

```
CREATE VIEW toptien
AS
SELECT brouwernr, brnaam, omzet FROM brouwers
ORDER BY omzet DESC LIMIT 10
```

maakt een (*read-only*) view die een lijst van de 10 brouwers met de meeste omzet selecteert.
Omdat we gebruik maken van een “ORDER BY” is de view niet updatable.

Voorbeeld 2:

```
CREATE VIEW brouwerOmzetPerGemeente
AS
SELECT gemeente, COUNT(*) AS aantal, SUM(omzet) AS totaal FROM
  brouwers
GROUP BY gemeente
ORDER BY totaal DESC
```

maakt een (*read-only*) view die de brouwers omzet per gemeente toont.
Omwille van onder andere de “GROUP BY” is deze view niet updatable.

```
CREATE VIEW AantalBierenPerSoort
AS
SELECT S.Soort AS 'soort bier', COUNT(B.BierNr) AS aantal
FROM soorten S
INNER JOIN bieren B
  ON S.SoortNr=B.SoortNr
GROUP BY S.Soort
ORDER BY S.Soort ASC
```

maakt een (*read-only*) view die het aantal bieren per soort toont.
De view is *read-only* omwille van het gebruik van “GROUP BY” , “INNER JOIN” …

Opmerkingen:

- een view wordt "bevroren" tijdens zijn aanmaak: veranderingen aan de onderliggende tabellen worden niet automatisch opgenomen in de view (bv. het toevoegen van een kolom)

8.2 Een view gebruiken

Een view kan gebruikt worden zoals een table. Je kan er een query op maken:

```
SELECT * FROM aantalbierenpersoort WHERE aantal>100
```

of

```
INSERT INTO BrouwersBeperkt (BrNaam,Adres,Postcode,Gemeente)
VALUES('De Maffe Kezen', 'laantje 6',9999, 'NooitGedacht')
```

vermits ‘brouwersBeperkt’ *updatable* is

8.3 Drop view

Verwijdert de view uit de database.

```
DROP VIEW viewnaam
```

Voorbeeld:

```
DROP VIEW toptien
```

verwijdert de view toptien uit de database

8.4 Oefeningen

1. Definieer een view VASTLAAG waarin alle gegevens van alle vaste planten uit de tabel PLANTEN voorkomen met een hoogte van maximaal 15 cm.

2. Definieer een view GEM_O_PR met de kolommen art_code, MIN_OFF, MAX_OFF en GEM_OFF, waarin respectievelijk de artikelcode en per artikelcode de laagste, de hoogste en gemiddelde offerteprijs is vermeld.

3. Definieer een view LEV014 waarin alleen van leverancier 014 de volgende gegevens staan: ART_CODE, PLANTENNM, ART_LEV, OFF_PRIJS, PRIJS.

4. Maak een view BOMEN met de volgende gegevens: ART_CODE, PLANTENNM, HOOGTE, PRIJS. Uiteraard alleen de bomen selecteren.

5. Maak een view PLANTEN_IB waarin de volgende gegevens komen: BESTELNR, ART_LEV en PLANTENNM. Per samengestelde sleutel van de BESTREG tabel moet dus de plantennaam worden toegevoegd.

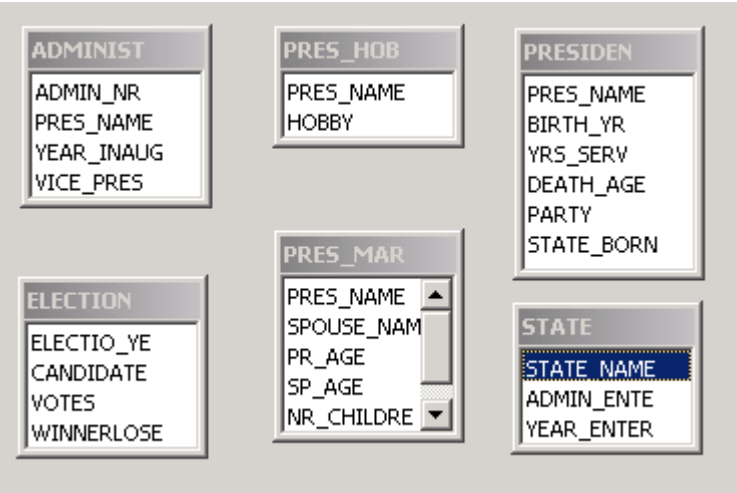
6. Maak een view ZOMERPL waarmee de gegevens ART_CODE, PLANTENNM, SOORT en PRIJS zijn te benaderen van alle planten die in de maanden juni, juli en augustus beginnen te bloeien.

7. Maak een view waardoor alleen de offertegegevens van de leveranciers uit LISSE zijn te selecteren.

9 EINDOEFENING (NIEUWE VERSIE)

De database PRESIDEN bevat 6 tabellen:

ADMINIST	Bevat de lijst van presidenten en hun vice-president met het beginjaar van hun ambtstermijn.
ELECTION	Bevat de presidentskandidaten.
PRES_HOB	Bevat de hobby's van de presidenten
PRES_MAR	Bevat de huwelijken van de presidenten
PRESIDEN	Bevat de lijst van presidenten en de duurtijd van hun ambtstermijn
STATE	Bevat een lijst van alle staten en wanneer zij tot de USA toegetreden zijn. (51: één foutieve staat: New Georgia)



Voer deze opdrachten uit. Bewaar je SQL statements in een tekstbestand en noteer het aantal records erbij dat getoond of verwerkt werd.
Kolomnamen in "aanhalingstekens" moeten zo genoemd worden.

1. Druk de namen van alle presidenten af met hun geboortejaar.
2. Druk de namen van de staten af waarin minstens 1 president geboren is.
3. Toon de namen van de staten waarvan de naam ofwel begint met "*New*" ofwel eindigt met "*ia*".
4. De tabel *state* bevat de foutieve staat "*New Georgia*". Verwijder dit record.
5. Druk de top 10 af van de presidenten met het grootste leeftijdsverschil met hun echtgenotes, enkel voor zij die even oud of ouder zijn dan hun echtgenote.
   Geef presidentsnaam, naam echtgenote, "leeftijdsverschil"
6. Druk de naam, het "geboortejaar" en het "sterfjaar" af van de presidenten die vrijgezel bleven.
7. Druk de namen van **alle** staten af met het "aantal" presidenten dat daar geboren werden, geordend volgens dalend aantal
8. Toon de namen van de presidenten die zelf ooit vice-president waren, samen met het "aantal keer vice-president"

9. Wie was president in 1937? (of eender welk ander jaar). Toon zijn naam, het "beginjaar" en "eindjaar" van zijn termijn.
10. Selecteer de naam van de president (noem dit veld president), zijn "geboortejaar", zijn "huwelijksjaar", zijn "verkiezingsjaar", zijn "overlijdensjaar", de naam van zijn "vrouw"(en), het aantal "kinderen", voor **alle** presidenten (ook zij die niet trouwden), alfabetisch gerangschikt en plaats deze records in een nieuwe tabel "familie". Doe dit met 1 SQL statement!
11. Bij deze laatste actie zagen we iets over het hoofd: als het veld "death_age" in presiden de waarde 0 heeft, dan gaan we ervan uit dat hij nog niet overleden is: zorg ervoor dat de waarde van "overlijdensjaar" in familie – voor deze presidenten **leeg** is.
12. Voeg aan de tabel familie een veld "pres_id" toe van het data type integer dat een autonummering heeft en primary key is, voeg met hetzelfde SQL statement een index "i_namen" toe op het veld "president"
13. Maak een nieuwe, **lege** tabel "hobbies" met dezelfde structuur als pres_hob
14. Vul deze tabel met alle records van pres_hob
15. In deze tabel (hobbies) zitten enkele typfouten in de namen.
    Als we er vanuit gaan dat de tabel "familie" de correcte namen van alle presidenten bevat, zoek dan de fout getypte presidenten in hobbies.
    Corrigeer daarna met enkele sql statements de foute namen.
16. Voeg nu ook een nieuwe kolom "pres_id" toe aan hobbies van het data type integer. Vul deze daarna met de juiste overeenkomende pres_id uit familie.
17. Leg nu een 1-M relatie tussen de tabellen familie en hobbies op de pres_id velden. Als een record van familie gewist wordt moet het verwijderen geweigerd worden als er een overeenkomend record in hobbies bestaat. Als de pres_id van een record van familie gewijzigd wordt, moeten de overeenkomende records in hobbies ook gewijzigd worden.
    Test dit uit door te proberen het record van president "Adams J Q" te wissen in familie. Welk bericht krijg je?
18. Maak een view "hobbies_overzicht" waarbij we per hobby het aantal presidenten zien in dalende volgorde.
19. Druk een lijst af van de gebeurtenissen:

jaar	geboorte	naam	state
jaar	overlijden	naam	state
jaar	huwelijk	naam	spouse
jaar	winnaar verkiezingen	naam	party
jaar	verliezen verkiezingen	naam	party
jaar	toetreding state bij	state	huidige president
jaar	geboorte echtgenote	naam	echtgenoot

    De lijst is gesorteerd volgens jaar.
20. Welke presidenten werden geboren in het sterfjaar van een andere president? Gebruik hiervoor de tabel familie. Toon president + geboortejaar en president + overlijdensjaar

10APPENDIX A

De MySQL SQL syntax kan je natrekken op:

http://dev.mysql.com/doc/refman/5.0/en/sql-syntax.html

Opsomming van de **verschillen** met de standaardcursus SQL:

SELECT	• het predicaat TOP bestaat niet in MySQL, vervang door LIMIT • het statement "IN externedatabase" bestaat niet voor SELECT • de syntax SELECT … INTO of SELECT … INTO TABLE waarbij een **nieuwe tabel** gemaakt wordt, wordt niet ondersteund. Vervang door CREATE TABLE … SELECT
SUBQUERIES	• een subquery in een FROM of die deel uitmaakt van een JOIN, **moet** een alias krijgen
INSERT	• de syntax "IN externedatabase" wordt niet ondersteund • één INSERT statement kan meerdere waarden tezelfdertijd invoegen
CREATE TABLE	• CREATE TABLE… SELECT kan een nieuwe tabel aanmaken op basis van een SELECT • CREATE TABLE… LIKE maakt een nieuwe, lege tabel met dezelfde structuur als de voorbeeldtabel

11COLOFON

Sectorverantwoordelijke: Ortaire Uyttersprot

Cursusverantwoordelijke: Jean Smits

Didactiek: Werkgroep Informatica Basis

Lay-out: Leyman Eugène

Medewerkers: Brigitte Van Ceulebroeck
Jan Vandorpe
Jan De Tavernier
Siska Corneillie

Versie: 06/01/2015

Nummer dotatielijst: