Databases & SQL

for OOP developers

Author: Patrick Geudens

0. Databases

0.0. Wat is een database?

Definitie: Een database is een digitale verzameling van gerelateerde gegevens.

Typisch voor databases is dat ze georganiseerd zijn rond aspecten uit de realiteit zodat de processen die deze informatie nodig hebben ondersteund worden en niet moeten worden veranderd.

Voorbeelden:

- De personeelsgegevens van een bedrijf
- De product-gegevens van een webshop
- Alle gegevens (personeel, produkt, klanten) die een bedrijf nodig heeft om de goede werking te waarborgen.

0.1. Database management systemen

Zoals hierboven beschreven zijn databases enkel en alleen gegevens. Deze gegevens moeten natuurlijk ook beheerd worden. Hiervoor hebben we Database Management Systemen of DBMS. Het DBMS bestaat uit een geintegreerde set van software die gebruikers toegang geeft tot de data(base) en deze te manipuleren.

Vanwege de hechte relatie tussen database en DBMS wordt de term database heel dikwijls (door niet Database Administrators) gebruikt voor het zowel de gegevens als de software die ze moet beheren.

Buiten de wereld wordt de term ook gebruikt voor allerhande gegevensverzamelingen, spreadsheets, MS Access bestanden. In deze cursus gaan we enkel rekening houden met databases waarvan de grootte het gebruik van een DBMS rechtvaardigd. Waar deze grens ligt wordt in principe bepaald aan de hand van budgetten en dergelijke, dit topic is ook out of scope voor deze cursus.

De functies die DBMSen aanbieden kunnen meestal verdeeld worden in 4 groepen:

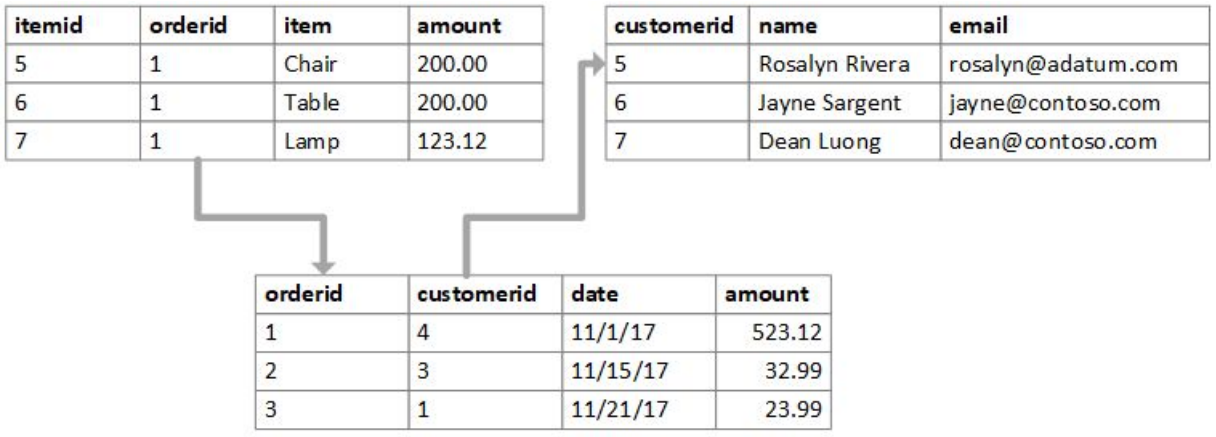
- Data definitie - Definieren van de datastructuur
- Data retrieval - Ophalen van gegevens
- Data manipulation - Aanpassen van gegevens
- Administratie - Beheren van toegang tot de gegevens e.d.

0.2. DBMS_1 != DBMS_2

In de wereld van Database Management Systemen is niet elk systeem hetzelfde. Ze zullen in grote lijnen wel dezelfde functionaliteit aanbieden, maar de manier waarop ze met de gegevens omgaan, de taal die we moeten gebruiken, ... verschilt van systeem tot systeem.

Binnen de DBMS wereld zijn er verschillende soorten systemen:

- Relationeel DBMS: gegevens worden voorgesteld door middel van gegevens EN relaties. De oudste en meest gebruikte vorm van DBMS.



itemid	orderid	item	amount
5	1	Chair	200.00
6	1	Table	200.00
7	1	Lamp	123.12

customerid	name	email
5	Rosalyn Rivera	rosalyn@adatum.com
6	Jayne Sargent	jayne@contoso.com
7	Dean Luong	dean@contoso.com

orderid	customerid	date	amount
1	4	11/1/17	523.12
2	3	11/15/17	32.99
3	1	11/21/17	23.99

- Hierarchisch DBMS: Binnen de gegevens is er een duidelijke “parent-child” verhouding. Het DBMS gaat de gegevens ook zo opslaan.
- Network DBMS: Is een afgeleide van de relationele database met meer mogelijkheden naar de relaties toe.
- Object Oriented DBMS: Niet te vergelijken met Object Oriented Programming. Maar deze databases zijn gespecialiseerd in het opslaan van verschillende media-types (Fotografie, tekst,…)
- Document Oriented database: Dit DBMS is georienteerd naar semi-structured databases. Het is niet een van de hoofdtypes van DBMS maar in de huidige ICT wereld niet meer weg te denken. Dit DBMS is meestal geoptimaliseerd naar snelheid en niet naar consistency.

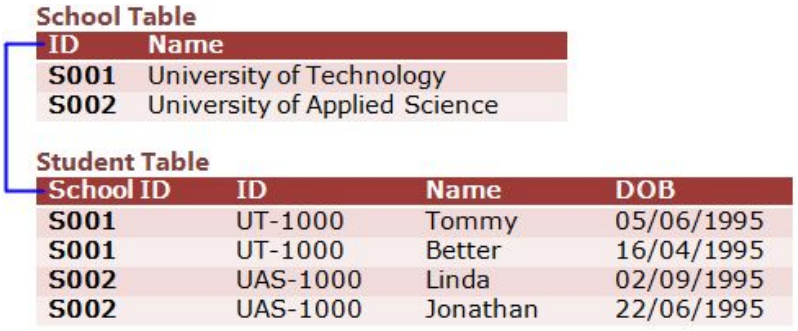
In deze cursus zullen wij ons concentreren op de meest gebruikte vorm van het database management systeem, namelijk: het relationele DBMS.

0.3. Relational Database Management Systems – RDBMS

Het relationele model organiseert zijn gegevens in tabellen of relaties deze bestaan op hun beurt uit rijen of records met een unieke sleutel of key die elke record uniek kan identificeren.

Om een vergelijking te maken met Object Oriented Programming:

RDBMS	OOP
Table	Class
Column	Property
Record	Instance of class
Key	Memory Reference



School Table

ID	Name
S001	University of Technology
S002	University of Applied Science

Student Table

School ID	ID	Name	DOB
S001	UT-1000	Tommy	05/06/1995
S001	UT-1000	Better	16/04/1995
S002	UAS-1000	Linda	02/09/1995
S002	UAS-1000	Jonathan	22/06/1995

Terminology

Row (Tuple – Record)

Een set van gegevens die 1 item voorstelt of beschrijft.
Vb: { ID: 001, Name: Tommy, DateOfBirth: 01/01/1901 }

Column (Attribute – Field)

Een gelabeled attribuut of eigenschap van een record.

Vb: Name

Tabellen (Tables)

Een set van records die dezelfde attributen hebben.

Students Table

Student	ID*
John Smith	084
Jane Bloggs	100
John Smith	182
Mark Antony	219

Activities Table

ID*	Activity*	Cost
084	Swimming	$17
084	Tennis	$36
100	Squash	$40
100	Swimming	$17
182	Tennis	$36
219	Golf	$47
219	Swimming	$15
219	Squash	$40

Database (Schema)

Een verzameling van tabellen die gerelateerd zijn. Sommige DBMSen laten toe om binnen een database ook nog organisatie niveaus te hebben, namelijk: schema's.

Primary Key (PK)

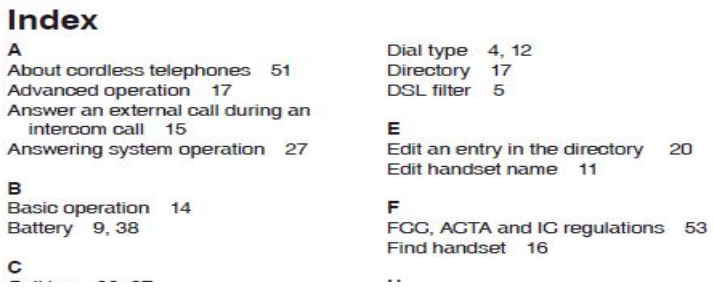
De sleutel van de tabel, een unieke waarde per record die de andere attributen uit de tabel identificeert.

Foreign Key (FK)

Dit is een attribuut in de tabel die verwijst naar een andere tabel waar deze attribuut het sleutelveld (PK) is. bv. klantnr in tabel facturen verwijst naar het klantnr in de tabel klanten. Het klantnr van de tabel klantnr is de PK van de tabel.

Index

Indexen in een database kan je best vergelijken met de index van een boek (het deeltje dat je meestal helemaal op het einde vindt)



Index

De technische uitwerking is uiteraard anders dan een boek, maar de index maakt het mogelijk om records in de database snel terug te vinden.

Query

Een query is de technische term die we gebruiken voor de instructies die we naar de server sturen.

Normalisatie

Het proces dat van de oplijsting van de gegevensbehoefte een werkbaar database model maakt. Dit proces is ook out of scope voor deze cursus.

0.4. ACID

Wanneer we spreken over databases, kunnen we niet aan de term ACID voorbijgaan. ACID beschrijft een set van eigenschappen aan welke een goed DBMS moet voldoen.

ACID:

- Atomicity
- Consistency
- Isolation
- Durability

Transactie

Voor we bovenstaande termen kunnen uitleggen moeten we eerst stilstaan bij transacties. Wat is dit mythische beest?

Een transactie symboliseert de kleinste eenheid van werk in een database. Een voorbeeld: stel dat we de volgende situatie hebben in een bank

RekNr	Voornaam	FamNaam	RekeningStand
BE12 1234 1234 1234	Al	BeBack	1500
BE23 2345 2345 2345	Jack	Mehov	1250

Al heeft 100 Euro geleend van Jack en wil deze terug overschrijven. Aangezien databases enkel data manipuleren en geen ‘functionaliteit’ aanbieden moet deze overschrijving opgedeeld worden in 2 instructies:

1. De rekening van Al moet verminderd worden met 100 euro.
→ Of: De nieuwe rekeningstand van Al is 1400

2. De rekening van Jack moet verhoogd worden met 100 euro.
→ Of: De nieuwe rekeningstand voor Jack is 1350.

Deze 2 acties horen bij elkaar en een DBMS geeft ons ook de mogelijkheid om 2 (of meerdere) acties te groeperen in een transactie zodat het DBMS het geheel van de 2 instructies ziet als 1 ‘unit of work’.

Atomicity

Wat zou er kunnen gebeuren als Atomicity niet zou gegarandeerd worden in het bovenstaande voorbeeld:

Zowel de rekening van Al en Jack moeten een nieuwe rekeningstand krijgen. De rekeningstand van Al wordt aangepast en vlak daarna crashed ons programma. De wijziging op Jack’s rekening is nog niet doorgevoerd. Al heeft op dit moment 100 euro minder, Jack heeft zijn geld niet gekregen.

Atomicity garandeert dat elke transactie (ongeacht deze bestaat uit 1 of meerdere instructies) zal behandeld worden als een geheel. De transactie zal dus ook als een geheel succesvol zijn of failen zelfs in het geval van bijvoorbeeld errors of electriciteitspannes.

Consistency

Consistency zorgt ervoor dat een transactie de database enkel van een ‘valid state’ naar een nieuwe valid state kan brengen. De data die naar de database geschreven wordt moet dus voldoen aan alle opgestelde voorwaarden binnen het design, inclusief custom rules, constraints, triggers of combinaties hiervan.

Een voorbeeld: Het zal niet mogelijk (mogen) zijn om de rekeningstand te wijzigen naar ‘AAAA’.

Isolation

Database servers werken dikwijls voor meerdere gebruikers tegelijk. Het is dan ook belangrijk dat deze ‘concurrency’ niet voor problemen gaat zorgen. Isolation moet er dan ook voor zorgen dat transacties die tegelijkertijd werden uitgevoerd hetzelfde resultaat geven als dat de transacties sequentieel waren.

Durability

Durability garandeert dat transacties die gecommit zijn (of bevestigd zijn door de server) effectief ook uitgevoerd zullen worden door de server, zelfs in het geval van een power failure of een hardware probleem.

0.5. Relationships

Het laatste deel van de database theorie dat we nog moeten behandelen zijn de soorten van relaties die we kunnen terugvinden in een relationele database.

1-1 (one to one) relationship

De one-to-one relation is de meest voorkomende relatie in een database. Een voorbeeld van een one-to-one relatie is:

- 1 persoon kan 1 medisch dossier hebben. En dat medisch dossier hoort bij welgeteld 1 persoon.
- Een iets minder duidelijk voorbeeld: Een persoon heeft 1 rijksregisternummer en dat nummer hoort tegelijk ook bij 1 persoon.

Een nuance die we zeker ook even willen aanhalen is het feit dat een rijbewijs een eigenlijke entiteit is, terwijl een rijksregisternummer een eigenschap is van een persoon.

In essentie: elke tabel zal een verzameling zijn van one-to-one relaties tussen het uniek identificerende attribuut (key) en de andere gegevens. Een tabel wordt in de database-theorie dan ook een relatie genoemd, nl. de relatie tussen de (primary) key en de andere attributen.

Persoon

id	naam	dossier
123	Jan Wandelaar	D001

Medisch Dossier

id	persoon	dokter
D001	123	...

De one-to-one relatie wordt in tabellen voorgesteld door de Primary Key van de verwezen tabel te gebruiken als Foreign Key in de gerelateerde tabel. Bij de one-to-one relatie is het mogelijk om zowel een unidirectionele als een bidirectionele relatie aan te leggen.

1-n (one to many) relationship

De one-to-many relatie is een relatie tussen 2 entiteiten waar entiteit A hoort bij 1 instantie van entiteit B, maar entiteit B kan gerelateerd worden met meerdere instanties van entiteit A.

Bijvoorbeeld:

- De Volvo S40 wordt gemaakt door het bedrijf Volvo. Het bedrijf Volvo maakt meerdere modellen (Volvo S40, S60, V80, …)
- Een factuur staat op naam van 1 persoon of bedrijf. De persoon of het bedrijf kan meerdere facturen op zijn naam hebben.

Klant

id	naam
123	Jan Wandelaar

Factuur

id	klant	totaal
F001	123	1.23 Euro
F002	123	12.3 Euro
F003	123	123 Euro

Om de one-to-many relatie voor te stellen wordt er vanuit de many kant van de relatie (many facturen) de referentie naar de one kant gelegd (one klant). Dus de PK van de one wordt gebruikt als FK aan de many-kant. Bij de one-to-many is het (theoretisch gezien) niet mogelijk om een bidirectionele relatie te creeren.

m-n (many to many) relationship

De many-to-many relatie is een relatie waar mogelijk meerdere instanties van entiteit A bij meerdere instanties van entiteit B horen en omgekeerd.

Bijvoorbeeld

- Een factuur kan meerdere producten bevatten en die producten kunnen op hun beurt ook voorkomen op meerdere facturen.
- Een schrijver kan meerdere boeken schrijven. Boeken op hun beurt kunnen ook geschreven worden door meerdere auteurs.

Het lastige aan de many-to-many relatie is de manier waarop deze wordt voorgesteld in een relationele database. Stel dat er een many-to-many relatie is tussen entiteiten A en B. We kunnen geen attribuut toevoegen aan entiteit A, aangezien we in de meeste gevallen niet weten hoeveel feitelijke relaties er gemaakt moeten worden (hoeveel is many?). Hetzelfde geldt voor de omgekeerde relatie.

De oplossing voor dit probleem is echter zeer elegant en deze bestaat uit het opdelen van de m-n relatie in een 1-m en een n-1 relatie. De m-n relatie wordt dan voorgesteld door records toe te voegen in een *tussentabel* of ook *intermediaire* tabel.

Product Table

id	Product
P001	Appels
P002	Bananen

Invoice Table

id	Totaalprijs
I123	10 Euro
I124	12 Euro

productID	InvoiceID	Amount
P001	I123	1
P002	I123	2
P001	I124	1
P002	I124	4

Uit bovenstaande tabel kunnen we nu opmaken dat factuur I123 bestaat uit:

- 1 Appel (P001)
- 2 Bananen (P002)

De minimum informatie in de intermediaire tabel is altijd een combinatie van de keys uit de gerelateerde tabellen. Deze tabel krijgt normaal gezien ook geen eigen PK, maar de unieke identificatie van de records gebeurd door de combinatie van beide FK's. We spreken in dit geval over een *composite key*.

Self-referencing relationship

De laatste relatie die we moeten vermelden in deze cursus is de self-referencing relationship. Deze relatie kan bestaan uit elk van de drie basis relationships (1-1, 1-m of m-n).

Een voorbeeld is bijvoorbeeld de verantwoordelijke in een bedrijf. In het personeelsbestand kan dan elk personeelslid een verantwoordelijke hebben. Maar deze verantwoordelijke is op zijn beurt (in normale omstandigheden) ook een personeelslid van datzelfde bedrijf. Dus er bestaat een relatie tussen verschillende instanties van de entiteit *personeelslid*.

Deze relaties worden op dezelfde manier voorgesteld als hierboven uitgelegd met het verschil dat aan beide kanten van de relatie dezelfde tabel terug te vinden is.

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 drie categorieën : Data Manipulation Language, Data Definition Language, Data Control Language en Transaction Control.

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

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

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.

TCL 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) en bestaat onafhankelijk van het gebruikte platform/taal.

Belangrijk om te onthouden: SQL is een gestandaardiseerde taal en dus bruikbaar voor elke soort relationele database. Hou er wel rekening mee dat elke SQL-server zijn eigen SQL-dialect spreekt wat meestal resulteert in andere syntaxen en in uitzonderlijke gevallen verschillende query-resultaten.

Case Sensitivity

Voor MySQL: case sensitivity is hier afhankelijk van het onderliggende OS. Voor Windows zal MySQL niet case sensitive reageren, draaien we de server op een *nix OS zal het wel zo zijn.
Voor MSSQL: Niet case sensitive.

Ga er altijd van uit dat uw database systeem case sensitive is.

De redenering hiervoor is:

We willen onze statements/code systeem onafhankelijk schrijven. Misschien is uw achterliggend database systeem niet case sensitive, maar je weet nooit op welk systeem/os de database morgen zal draaien.

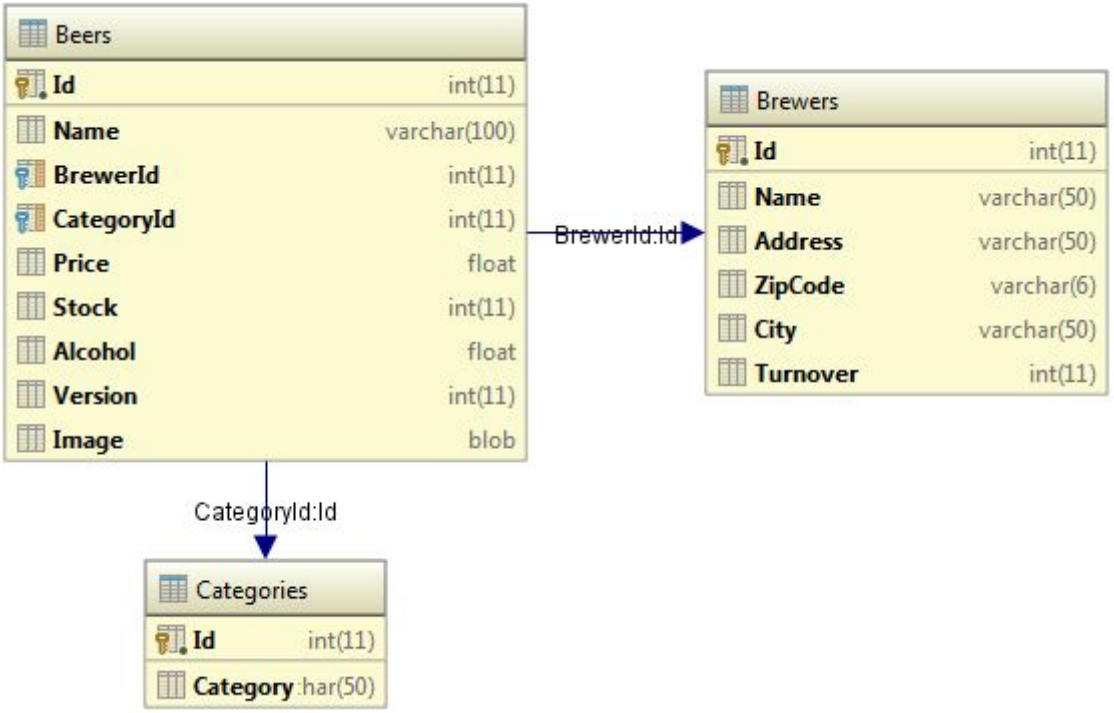
De DATABASE beersDB

In deze cursus wordt er gebruik gemaakt van de beersDB.

De database beersDB bestaat uit drie tabellen, namelijk:

Beers	Bevat een lijst van alle beers, met de velden Id, name, brewerId, categoryId, price, stock, alcohol, version, image. PK: Id Het veld brewerid verwijst naar de brouwerij uit de tabel Brewers. Het veld categoryId verwijst naar de biersoort uit de tabel Category.
Brouwers	Bevat een lijst van alle brouwerijen, met de velden Id, name, zipcode, city, turnover. PK: Id
Soorten	Bevat een lijst van alle biersoorten, met de velden Id, Category. PK: Id

Het schema van de database beers :



2. DML – GEGEVENS MANIPULEREN

2.1 Gegevens selecteren – SELECT

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

```
SELECT <option> * | table.* | [table.]col1 [AS alias1] [,
[table.]col2 [AS alias2] [, ...]]}
FROM table-expression  [, …]
[WHERE <condition>]
[GROUP BY  <column> ]
[HAVING <condition> ]
[ORDER BY <column> [ASC] [DESC] ]
[LIMIT] int [OFFSET] int
```

Onderdeel	**Beschrijving**
option	Een van de volgende predikaten: ALL, DISTINCT of TOP. Met een predikaat kan je het aantal records in het resultaat beperken. De standaardinstelling is ALL als geen predikaat is opgegeven. TOP wordt niet door alle relationele databases ondersteund.
*	Bepaalt dat alle velden uit de opgegeven table of tabellen worden geselecteerd.
table	De naam van de tabel die de velden bevat waaruit records worden geselecteerd.
col1, col2	De namen van de velden waaruit gegevens worden opgehaald. Als u meer dan één veld opgeeft, worden deze opgehaald in de opgegeven volgorde.
alias1, alias2	De namen die je wil gebruiken als kolomkoppen in plaats van de oorspronkelijke kolomnamen in tabel.
table-expression	De naam van de tabel of de namen van de tabellen waaruit je gegevens wil 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.

SELECT-instructies wijzigen de gegevens in de database niet. Merk ook op dat het resultaat van een query op zijn beurt een tabel is.

De minimale syntax voor een instructie SELECT luidt:

```
SELECT col | col, col | * FROM tabel
```

SELECT name, alcohol FROM beers;

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

SELECT name FROM brewers;

geeft als resultaat een lijst met alle namen van de brouwerijen

SELECT city FROM brewers;

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

SELECT * FROM beers;

2.2 Selectie voorwaarden - 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 (').

```
SELECT name
FROM beers
WHERE alcohol < 5;
```

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

```
SELECT name
FROM brewers
WHERE city = 'Brussel';
```

geeft als resultaat een lijst van alle brouwerijen gelegen in Brussel

```
SELECT name
FROM beers
WHERE name LIKE '%ale%';
```

geeft als resultaat een lijst van alle beers waar het woord ale voorkomt in de naam. Het procentteken duidt aan dat op die plaats een willekeurig aantal letters kunnen voorkomen. vb. 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	[agm]	a, g, m	b, c, n, 4

voorkomen in de benoemde lijst			
Combinatie van hiervoor vermelde formaten	a[b-m]_	Ab9, af0	aacfd, a90

Opmerking

```
SELECT name FROM beers
WHERE alcohol BETWEEN 5 AND 7;
```

geeft een lijst van alle beers met een alcoholpercentage dat gelegen is tussen 5% en 7%.

```
SELECT name FROM beers
WHERE alcohol IN (0, 5, 8);
```

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

```
SELECT name
FROM brewers
WHERE city in ('Leuven', 'Genk', 'Antwerpen', 'Dendermonde',
'Wevelgem');
```

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

```
SELECT name FROM beers WHERE alcohol is Null;
```

geeft een lijst van alle beers waarvan de kolom alcohol niet ingevuld is. Om de kolommen te controleren die niet leeg zijn gebruik je “IS NOT null”.

Je kunt voorwaarden combineren 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
name LIKE ‘%wit’	wordt name NOT LIKE ‘%wit%’
city IN (‘genk’,’leuven’)	wordt city NOT IN (‘genk’,’leuven’)

```
SELECT * FROM beers WHERE name 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 beers WHERE name NOT LIKE '%wit%';
```

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

Let op

Let op het aantal beers dat getoond wordt:

```
instructie1          SELECT * FROM beers WHERE alcohol > 5

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

De som van het aantal beers dat getoond wordt door de instructie 1 en 2 is niet gelijk aan het totale aantal beers. Null-waarden worden genegeerd bij selectie op gedefinieerde values. NULL is dus NIET hetzelfde als 0. Maar dat wisten we al.

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

```
SELECT name, alcohol FROM beers ORDER BY name ASC;
```

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

```
SELECT name, alcohol FROM beers ORDER BY brewerid DESC, name ASC;
```

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

```
SELECT name, alcohol FROM beers ORDER BY 1 ASC;
```

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

2.4 LIMIT … OFFSET …

Indien we onze zoekopdracht willen limiteren op het aantal resultaten, om bijvoorbeeld onze data te 'pagineren' kunnen we aan het SELECT statement een LIMIT toevoegen.

```
SELECT * FROM Beers LIMIT 100;
```

Zal het query-resultaat beperken tot 100 records te beginnen van de eerste record. Indien we niet bij het eerste record willen beginnen moeten we de OFFSET gebruiken in combinatie met LIMIT

```
SELECT * FROM Beers LIMIT 25 OFFSET 50;
```

Zal 25 bieren tonen te beginnen met het 51ste record.

In MySQL kunnen we de vorige statement enigszins inkorten:

```
SELECT * FROM Beers LIMIT 50, 25;
```

2.5 Oefeningen SELECT deel 1

Het getal achter de opgave zijn het aantal records dat gezocht wordt.

(a) Geef een lijst van alle biercategorieen. (38)

(b) Toon de lijst van categorieen in dalende alfabetische volgorde zonder de categorie-id's. (38)

(c) Toon een lijst van alle brouwerijen die meer dan 5000 Euro turnover hebben. (54)

(d) Toon nu enkel de naam en de stad van de brouwerijen die minder dan 5000 Euro turnover, maar niet 0. Sorteer de lijst op basis van de turnover. (53)

(e) Geef een lijst van alle mogelijke alcoholgehaltes in de beers tabel. Dus geen dubbels. En gesorteerd van groot naar klein. (14)

(f) Toon alle namen van bieren waarvan de naam "wit" bevat zonder dubbels, alfabetisch gesorteerd. (30)

(g) Toon alle bieren met meer alcohol dan 3 en minder dan 7 zonder gebruik te maken van logische operatoren (<, >, …) (568)

(h) Geef de top 3 van de sterkste bieren in onze database. (3)

(i) Doordenker: Maak een lijst van de naam, straat, postcode en

stad voor alle brouwers in 3 kolommen voor een adressenlijst.
M.a.w. combineer de postcode en de stad in 1 kolom.

2.6 Aggregate functions en groeperen – GROUP BY

In SQL hebben we ook een hele hoop ingebouwde functies die we kunnen loslaten op onze data. Van tellen van het aantal records die een query teruggeeft tot het omvormen van alle letters naar hoofdletters in de query.

Herinnering: het SELECT statement zal geen data in de database veranderen. Dus alle aggregate functions zullen uitgevoerd worden op het resultaat van de query, NIET op de data in de database.

Voor deze cursus zullen we 5 functies gebruiken. Voor alle andere mogelijkheden verwijzen we je door naar de documentatie van het DBMS dat gebruikt wordt. De mogelijkheden (en syntax) zullen ook veranderen naargelang het gekozen DBMS. De vermelde functies in onze cursus zijn universeel genoeg om tot de SQL taal gerekend te worden.

```
SELECT COUNT(*) FROM beers;
```

geeft het totaal aantal bieren uit de tabel. In het resultaat zien we COUNT(*) als kolomnaam. Om een meer bruikbare kolomnaam te geven aan het resultaat, gebruiken we de AS om een alias toe te kennen zoals in het volgende voorbeeld:

```
SELECT COUNT(*) AS aantal FROM beers;
```

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

geeft het gemiddelde, maximum , minimum, totaal alcoholpercentage uit de tabel beers.

Let op:

voor bovenstaande agregaat functies (maar niet alle) worden NULL values genegeerd indien het resultaat van de query een NULL value in elke kolom van het resultaat bevat. Dus indien we COUNT(alcohol) gebruiken i.p.v. COUNT(*) bekomen we een verschillend resultaat.

De volgende stap in ons SQL avontuur is het groeperen van de gegevens. Met de GROUP BY clause krijgen we een soortgelijk resultaat als met de DISTINCT optie. Echter op de achtergrond worden de records anders behandelt en krijgen we een hoop meer opties.

Bekijken we de volgende SQL instructie:

```
SELECT brewerid, AVG(alcohol) AS gemiddelde
FROM beers
GROUP BY brewerid;
```

berekent het gemiddelde alcoholpercentage per brewerid. Op de achtergrond worden alle records met dezelfde brewerid gegroepeerd, daarna geteld, het totaal alcohol opgeteld en vervolgens het gemiddelde alcoholgehalte berekend (per brewerid).

Belangrijk om te weten is dat de records gegroepeerd worden op de achtergrond op basis van de gekozen kolom. Alle andere kolommen worden in essentie genegeerd. Ze zullen nog wel getoond worden, maar de gegevens die getoond worden voor de kolommen waarop niet gegroepeerd wordt zijn de eerste records die het DBMS tegenkomt in de fysieke database (op de harde schijf) en dus niet altijd relevant.

```
SELECT AVG(Alcohol), Name
FROM Beers
GROUP By BrewerId;
```

is mogelijk, maar de gegevens gevonden in name zullen niet representatief zijn voor de gegroepeerde gegevens.

```
SELECT LEFT(name,1), AVG(alcohol)
FROM beers
GROUP BY LEFT(naam,1);
```

toont ons de gemiddelde alcoholpercentage gegroepeerd volgens de eerste letter van de biernaam.

```
SELECT brewerid, MIN(alcohol) AS mini
FROM beers
GROUP BY brewerid
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 brewerid, AVG(alcohol) AS mini
FROM beers
GROUP BY brewerid
```

```
HAVING count(*) > 10;
```

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

2.7 Berekeningen maken

Het is ook mogelijk om simpele berekeningen te maken in SQL.

```
SELECT name,
     turnover * 0.9118 AS omzet_dollar,
     turnover * 116.6 AS omzet_yen
FROM brewers;
```

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.

2.8 Oefening SELECT deel 2

(a) Hoeveel verschillende brouwers zitten er in de database?
(118)

(b) Bereken de gemiddelde turnover van alle brouwers.
(114302.1525)

(c) Geef het laagste, gemiddelde en hoogste alcoholgehalte uit de beers tabel in 1 instructie.
(0, 5.8706, 15)

(d) Bereken de gemiddelde turnover van alle brouwers in de provincie Brabant (postcodes die beginnen met een 1) maar negeer de brouwerij ‘Palm’.
(39525.7143)

(e) Bereken het gemiddelde alcohol per categorie(id).
(114 records)

(f) Toon de hoogste bier prijs per categorie maar negeer alle bieren die niet in stock zijn. Sorteer het resultaat op categorieid. Zorg er ook voor dat we enkel prijzen zien die hoger zijn dan 3.
(5 records)

2.9 Gegevens selecteren uit meerdere tabellen

INNER JOIN

Er zijn op dit ogenblik twee gangbare methodes. De eerste is de clausule INNER JOIN, die wordt niet erkend door alle soorten relationele databases maar wel door de meest gebruikte.

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

```
FROM tabel1
INNER JOIN tabel2 ON tabel1.col1 operator tabel2.col2
```

De instructie INNER JOIN bevat de volgende onderdelen:

Onderdeel	**Beschrijving**
tabel1, tabel2	De namen van de tabellen waaruit records worden gecombineerd.
col1, col2	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 moet je wel een onderscheid maken door de tabelnamen te gebruiken op de volgende wijze : tabel.veld
operator	Een relationele vergelijkingsoperator: "=", "<", ">", "<=", ">=" of "<>".

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 Brewers en Beers om alle van alle beers hun brouwerij te selecteren. Wil je daarentegen alle brouwerijen (met inbegrip van brouwerijen zonder beers), 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 beers en brouwers gekoppeld via het veld brewerid:

```
SELECT beers.name, brewers.name
FROM beers
INNER JOIN Brewers ON beers.brewerid = brewers.id
```

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 wil opnemen, geef je de veldnaam op in de instructie SELECT (in dit geval: beers.Brouwernr).

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

```
SELECT velden
FROM tabel1
INNER JOIN tabel2 ON tabel1.col1 operator tabel2.col1
AND tabel1.col2 operator tabel2.col2
OR tabel1.veld3 operator tabel2.veld3
```

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

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

```
SELECT velden
FROM tabel1
INNER JOIN tabel2 ON tabel1.veldx = tabel2.veldx
INNER JOIN tabel3 ON tabel1.veldy = tabel3.veldy;
```

```
SELECT beers.name, brewers.name, categories.category
FROM beers
INNER JOIN brewers ON beers.BrewerId = brewers.Id
INNER JOIN categories ON beers.CategoryId = categories.Id;
```

Geeft een lijst van alle beers met de velden name, brouwerij-naam en soort uit de tabellen beers, brewers en categories.

Deze methode wordt niet door alle relationele databases ondersteund. De tweede methode gaat als volgt:

```
SELECT velden FROM tabel1, tabel2
WHERE tabel1.col1=tabel2.col2;
```

Onderdeel	**Beschrijving**
tabel1, tabel2	De namen van de tabellen waaruit records worden gecombineerd.

col1, col2	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 moet je wel een onderscheid maken door de tabelnamen te gebruiken op de volgende wijze : tabel.veld

In het volgende voorbeeld worden de tabellen beers en brewers gekoppeld via het veld brewerid:

```
SELECT beers.name, brouwers.name
FROM beers, brewers
WHERE beers.brewerid = brewers.id;
```

Algemene syntax om meer dan twee tabellen te koppelen wordt:

```
SELECT velden
FROM tabel1, tabel2, …tabelN
WHERE tabel1.col1 = tabel2.col2
AND tabel2.col2 = …
AND tabel….veld… = tabeln.veldn
```

SELECT Naam, Brnaam, Soort FROM beers, Brouwers, Soorten WHERE beers.brouwernr=brouwers.brouwernr and beers.soortnr=soorten.soortnr

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

Outer join

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

```
FROM tabel1
[LEFT | RIGHT] JOIN tabel2
ON tabel1.col1 vergopr tabel2.col2
```

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

Onderdeel	Beschrijving
tabel1, tabel2	De namen van de tabellen waaruit records worden gecombineerd.
col1, col2	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 Categories en Beers gekoppeld op het veld CategoryId. De SELECT-instructie genereert een lijst van alle Soorten, inclusief biersoorten waarvan er geen beers aanwezig zijn in de tabel beers:

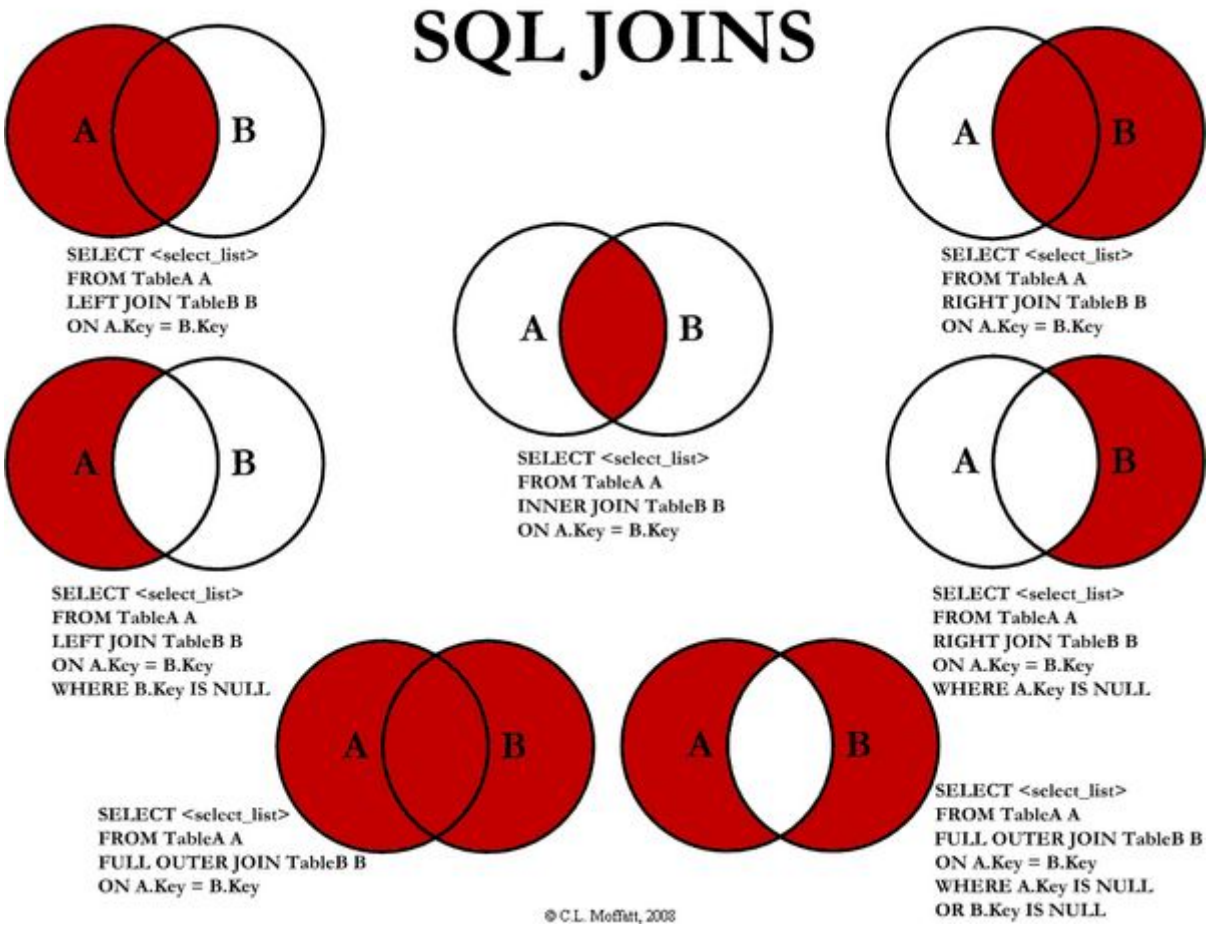
```
SELECT name, category
FROM Categories
LEFT JOIN beers ON categories.id = beers.categoryid;
```

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

Inner Join versus Outer Join

Om het verschil tussen de inner join en de outer join aan te tonen gebruiken we de volgende 'simpele' tabellen.

Tabel A	**Tabel B**		**INNER JOIN**			LEFT JOIN			RIGHT JOIN	
Col a	Col b		ON A.a = B.b			ON A.a = B.b			ON A.a = B.b	
1	3		3	3		1	null		3	3
2	4		4	4		2	null		4	4
3	5					3	3		null	5
4	6					4	4		null	6



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 clauses verwijzen naar de tabel met de aliasnaam ipv van de tabelnaam.

```
SELECT b1.name, b2,name
FROM brewers AS b1 INNER JOIN brewers AS b2
ON b1.city = b2.city AND b1.id < b2.id;
```

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.

2.10 UNION

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

```
Query1
UNION [ALL]
query2
UNION [ALL]
queryn
```

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 beers WHERE id = 1
UNION
SELECT * FROM beers WHERE id = 5;
```

selecteert alle beers met id 1 of 5.

```
SELECT * FROM beers
UNION
SELECT * FROM beers;
```

toont alle gegevens van de tabel beers slechts éénmaal. Standaard worden geen dubbele records als resultaat gegeven als je een UNION-bewerking gebruikt. Met UNION ALL wordt wel alle records getoond, ook de dubbele. De query wordt dan ook sneller uitgevoerd.

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.

2.11 Oefeningen SELECT deel 4

(a) Toon een lijst van alle biernamen en hun categorie-naam van alle bieren met 0% alcohol. (31)

(b) Toon een lijst van alle brouwers met de prijs van hun duurste bier.
(records 113 - gedeeltelijk resultaat:
Achouffe - 3.06
Alken - 4.5)

(c) Toon ‘alle’ bieren met een id tussen 1500 en 1600 met hun eventuele brouwer(naam). (41)

(d) Toon de top 3 bieren (id en naam) met het meeste alcohol samen met de naam van de brouwer en de categorienaam.

```
529 - Fitt           - Wieze        - Alcoholarm
229 - Bush de Noel   - Dubuisson    - Gerstewijn
511 - Fantome brewery (The best of) - Fantome - Massieve Ale
```

2.12 Subqueries

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

```
SELECT name
FROM beers
WHERE alcohol = (SELECT MAX(alcohol) FROM beers);
```

geeft een lijst van alle beers met het hoogste alcoholpercentage. In dit voorbeeld mag de subquery slechts een waarde als resultaat gegeven. Indien u 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 volgdende voorbeeld getoond wordt.

```
SELECT name
FROM beers
WHERE brewerid IN (
 SELECT brewerid
 FROM brewers
 WHERE city LIKE 'Brussel'
);
```

geeft een lijst van alle beers die in Brussel gebrouwen zijn. In deze voorbeelden worden de subquery gebruikt in de where-clausule van de select-instructie. Je kan ook subqueries combineren met de INSERT en UPDATE instructies.

De subqueries kan je ook gebruiken in de from clausule van de select. Eerst wordt dan de subquery uitgevoerd en dan pas de andere select-instructie.

```
SELECT categoryid
FROM (SELECT DISTINCT categoryid, brewerid FROM beers)
GROUP BY categoryid
HAVING COUNT(*) = 1;
```

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

```
SELECT category, average
FROM (
 SELECT categoryid, AVG(alcohol) AS average
 FROM beers
 GROUP BY categoryid) AS r1
INNER JOIN categories 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. Om het resultaat van de subquery te kunnen joinen met een andere tabel moet je aan aliasnaam geven aan de subquery.

2.13 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 beers die een lager alcoholpercentage dan die van zijn eigen soort” luidt als volgt :

```
SELECT b1.*
FROM beers b1
WHERE b1.alcohol < (SELECT AVG(b2.alcohol)
                    FROM beers b2
                    WHERE b2.categoryid=b1.categoryid);
```

Hier wordt wel tweemaal gebruik gemaakt van de tabel beers. 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 beers.

2.14 Oefeningen SELECT deel 5

(a) Selecteer de brouwers die een turnover hebben die hoger dan het gemiddelde is. (16)

(b) Toon alle bieren die het minimum of maximum alcohol gehalte hebben. (33)

(c) Toon alle bieren met een alcohol hoger dan het gemiddelde en waarvan de brouwers een turnover hebben dat boven het gemiddelde ligt. (60)

(d) Doordenker zonder search: Toon een lijst van alle brouwers met de prijs en naam van hun duurste bier. Het is mogelijk dat er meerdere bieren per brouwer geselecteerd worden. (266)

Resultaat:

N'Ice Chouffe	Achouffe	3.06
Noel Tripel	Alken	4.5
Rendux ambree	Ambly	2.75
St. Monon ambree	Ambly	2.75
Rendux brune	Ambly	2.75
Toison d'or (triple)	Anker	2.75
Oliver Twist (Nen)	Anker	2.75
Gouden Carolus	Anker	2.75
Julius	Artois	2.86
Campbell's christmas	Artois	2.86
Petrus (Triple)	Bavik	2.75
Kuurnse witte	Bavik	2.75
Kuurnse bruine	Bavik	2.75

2.15 GEGEVENS UIT DE DATABASE AANPASSEN

INSERT - Record(s) toevoegen

Eén record:

```
INSERT INTO target [(col1, col2, …)]
VALUES (value1, value2, …);
```

Meerdere records:

```
INSERT INTO target [(col1, col2, …)]
SELECT [source.]col1, col2[, …]
FROM table(s);
```

Onderdeel	**Beschrijving**
target	De naam van de tabel of query waaraan records worden toegevoegd.
source	De naam van de tabel of query waaruit de records worden gekopieerd.
col1, col2	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.
table(s)	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.
value1, value2	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 col1 van de nieuwe record, waarde2 in col2, enzovoort. U moet elke waarde tussen aanhalingstekens (‘ ‘) plaatsen en de afzonderlijke waarden scheiden met komma's.

Je kan met de instructie INSERT INTO één record toevoegen aan een tabel met de eerder beschreven syntax voor het toevoegen van één

record. In dit geval geeft de code de naam en de waarde voor elk veld in de record op. Je moet elk veld in de record opgeven waaraan een waarde moet worden toegewezen en tevens de waarde zelf. Als je niet elk veld opgeeft, wordt in de ontbrekende kolommen de default value of Null ingevoegd. Records worden aan het einde van de tabel toegevoegd.

```
INSERT INTO Categories (Id, Category)
          VALUES (30, 'Extra donker');
```

```
INSERT INTO Brewers
VALUES (99, 'Brouwerij Vaattappers', ‘Interleuvenlaan 2’, 3000, ‘Heverlee’, 1000);
```

Indien je records toevoegt aan de tabel wordt er steeds gecontroleerd of de referentiële integriteit niet aangetast wordt. Je kan bijvoorbeeld geen records toevoegen aan de tabel Beers met een categoryId of brewerId dat niet voorkomt in de respectievelijke tabellen omdat in de tabel Beers BrewerId en categoryId als Foreign Key gedefinieerd zijn.

Indien je niet zeker bent in welke volgorde je de velden moet plaatsen kan je steeds het “DESCRIBE <table>” commando gebruiken om de 'natuurlijke' volgorde van de kolommen te achterhalen.

Als de doeltabel een primaire sleutel bevat, moet je ervoor zorgen unieke, niet-Null-waarden toe te voegen aan het primaire sleutelveld of de primaire sleutelvelden. Als je dat niet doet, zal de RDBMS geen records toevoegen.

Een kleine nuance voor MySQL gebruikers, bij het gebruik van AUTO_INCREMENT in de PK column: indien je de kolomnamen niet meegeeft in het INSERT statement, zal je ook de PK kolom zelf moeten invullen. Wil je de PK door het RDBMS laten invullen, moet je dit aangeven door de andere in te vullen kolommen specifiek te benoemen in het INSERT commando.

Voorbeeld MET PK:

```
INSERT INTO Beers
VALUES (6, "TestBierNaam", 2, 2, 10, 1, 1, 1, null);
```

Voorbeeld zonder PK:

```
INSERT INTO Beers (Name, BrewerId, CategoryId, Price, Stock, Alcohol, Version) VALUES ("TestBierNaam", 2, 2, 10, 1, 1, 1);
```

De VALUES clause moet wel de volgorde respecteren van
- ofwel de doeltabel
- ofwel de volgorde gespecifieerd na de tabelnaam.

INSERT INTO ... SELECT ... FROM ...

Je kan met INSERT INTO ook records copieeren vanuit een tabel naar een andere tabel. Ipv de VALUES clause gebruiken we dan een SELECT FROM om te bepalen welke velden gecopieerd worden.

Stel we hebben een tabel BeerCopy die structureel hetzelfde is als Beers dan kunnen we met het volgende statement records copieeren van de Beers tabel naar de tabel BeerCopy:

```
INSERT INTO BeerCopy
SELECT * FROM Beers WHERE Id < 100;
```

Indien de doeltabel een andere structuur heeft als de sourcetabel zullen we in de SELECT clause de kolommen moeten meegeven in de volgorde van de doeltabel.

```
INSERT INTO BeerTemp (Name, Alcohol)
SELECT Name, Alcohol FROM Beers WHERE Id < 100;
```

LET OP: Bij dit commando, net als bij elk andere INSERT moet aan de referentiële integriteit voldaan zijn van de doeltabel. (in dit geval BeerCopy). Anders krijgen we een SQL error.

UPDATE

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

```
UPDATE table
SET col1 = value1, col2 = value2
WHERE criteria;
```

Onderdeel	**Beschrijving**
table	De naam van de tabellen (met inner join) met de gegevens die worden gewijzigd en uit geselecteerd worden.
Col = value	Een expressie die bepaalt welke waarde wordt ingevoegd in een bepaald veld in de bijgewerkte records.
Criteria	Een expressie die bepaalt welke records worden bijgewerkt. Alleen records die voldoen aan de expressie, worden bijgewerkt.

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 Brewers
SET Address = 'Keizerslaan 111',
     Zipcode = 1000,
     Gemeente = 'Brussel'
WHERE Brouwernr = 99;
```

Het alcoholpercentage wordt verhoogd met 0.5 voor bieren die gebrouwen worden in een brouwerij met een omzet > 1000000.

```
UPDATE Beers
SET alcohol = alcohol + 0.5
FROM Beers INNER JOIN Brewers ON Beers.BrewerId = Brewers.Id
WHERE turnover > 1000000;
```

Je kan deze update instructie ook met een subquery oplossen:

```
UPDATE Beers
SET alcohol = alcohol + 0.5
WHERE BrewerId IN
     (SELECT Id FROM Brewers WHERE turnover > 1000000);
```

LET OP – WHERE clause

Gebruik steeds een WHERE clause in je UPDATE statements.

Het volgende voorbeeld:

```
UPDATE Brewers SET Zipcode = 1000;
```

zal zeker werken. Echter net als we bij SELECT gezien hebben dat zonder de WHERE clause alle records geselecteerd zullen worden, zullen bij UPDATE **alle** records geupdate worden!!

Echter als we later zelf SQL statements naar de server sturen vanuit onze code zullen deze zonder meer uitgevoerd worden. Let hier zeker voor op als je een custom query schrijft.

LET OP – Referentiële Integriteit

Pas zeker op bij het updaten van FK waarden.
We gaan hier dieper op in in 5.4.Foreign Key Constraints
ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }

DELETE

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

```
DELETE [tabel] FROM tabelexpressie WHERE criteria
```

Onderdeel	Beschrijving
tabel	De optionele naam van de tabel waaruit records worden verwijderd.
tabel	De naam van de tabel waaruit records worden verwijderd.
criteria	Een expressie die aangeeft welke records worden verwijderd.

Je kan met DELETE records verwijderen uit tabellen die een één-op-veel-relatie hebben met andere tabellen. Ook hier wordt de referentiële integriteit bewaakt. Bij een delete opdracht gelden een paar extra regels die ingesteld worden op tabel-niveau.

ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }

Voor meer informatie hierover: zie 5.4.Foreign Key Constraints

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

```
DELETE FROM Beers WHERE id = 52;
```

verwijdert het record met het nummer 52 uit de tabel Beers.

```
DELETE FROM brewers
WHERE id IN (SELECT brouwernr from BIEREN WHERE alcohol > 20);
```

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

```
DELETE FROM brewers INNER JOIN beers ON brewers.id = brewerid
WHERE alcohol > 20;
```

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

```
DELETE Brewers.*, Beers.*
FROM Brewers INNER JOIN Beers ON Brewers.id = Beers.Brewerid
WHERE Brewers.name LIKE 'DECA';
```

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

LET OP
Als je records hebt verwijderd met DELETE, kunt 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.

2.16 Foreign Key Constraints (in MySQL)

Een van de belangrijkste onderdelen van een RDBMS is het beschermen van de referentiële integriteit. Als we naar een record in andere tabel refereren d.m.v. een FK is het natuurlijk de bedoeling dat deze record ook effectief bestaat.

Maar wat nu als we een record willen verwijderen uit tabel A, maar in tabel B wordt er naar deze record verwezen. We kunnen bij elke UPDATE/DELETE gaan nagaan of we geen verweesde records gaan achterlaten. Gelukkig zal MySQL (en de meeste andere SQL RDBM Systemen) ons hierbij een handje helpen.

Voorlopig zullen we ons beperken tot de gevolgen van deze constraints. Het aanmaken ervan zullen we later zien (6.BEHEER VAN TABELLEN EN RELATIES).

Foreign Key Constraints bestaan uit 2 delen. De UPDATE constraint en de DELETE constraint. Beide zien er gelijkaardig uit maar zullen enkel maar werken voor hun respectievelijke bewerkingen.

```
ON UPDATE { RESTRICT | NO ACTION | CASCADE | SET NULL }
```

```
ON DELETE { RESTRICT | NO ACTION | CASCADE | SET NULL }
```

RESTRICT : Dit is de default waarde. Indien we een record willen verwijderen uit de parent tabel die in een child tabel als FK wordt gebruikt of als we de PK van deze record willen wijzigen, zal de statement worden geweigerd. Restrict is MySQL specifiek.

NO ACTION : Zelfde als Restrict. Maar is de standaard SQL uitdrukking.

CASCADE : Bij het UPDATEN van de PK waarde in de parent zullen de records in de child tabel geupdate worden met de nieuwe waarde. Bij een DELETE zullen de child records samen met de parent verwijderd worden.

In praktijk: Indien we de brouwer met id 99 verwijderen zullen alle bieren die deze brouwer maakt ook verwijderd worden.

Indien we de brouwer met id 101 updaten naar id 156 zullen alle child records ook mee aangepast worden.

SET NULL : Bij UPDATE/DELETE van de parent record zullen de child records hun FK waarde op Null gezet krijgen.

In praktijk: We verwijderen de brouwer met id 7, dan zullen alle bieren gemaakt door deze brouwer als brewerId 'Null' krijgen. Hetzelfde zal gebeuren als we de PK van deze brouwer updaten.

SET DEFAULT : Deze constraint is niet gekend in elk RDBMS en zelfs binnen MySQL wordt deze niet herkend door de InnoDB engine.

.

In praktijk: Als we een brouwer verwijderen zullen de brewerId velden van de betreffende bieren op de default waarde van het veld brewerId gezet worden.

0. Beheer van tabellen en relaties

CREATE TABLE

Om in de wijde wereld van de create table te kunnen duiken moeten we eerst een woordje uitleg geven over de verschillende data-types die in SQL beschikbaar zijn. Merk wel op dat deze datatypes over de verschillende SQL-servers een andere invulling kunnen krijgen, zo zijn er beduidende verschillen tussen MySQL- en MSSQL-server.

Datatype	
CHAR(size)	String met vaste lengte (Max 255 : 2^8 - 1)
VARCHAR(size)	String met variabele lengte (Max 255 : 2^8 - 1)
TEXT	String (Max 65.535 : 2^16 - 1)
MEDIUMTEXT	String (Max 16.777.215 : 2^24 - 1)
LONGTEXT	String (Max 4.294.967.295 : 2^32 - 1)
ENUM(x,y,…)	Lijst van mogelijke waarden (max 2^16-1 values) Max 1 geselecteerde waarde
SET	Zelfde als enum (Max 64 opties) Meerdere geselecteerde waarden mogelijk
INT(size)	Integer (signed of unsigned) Signed: -128 tot 127 Unsigned: 0 tot 255 Size: aantal digits
FLOAT(size, d)	Floating Decimal Point Size: Max aantal digits d : Precisie (aantal digits rechts van de komma)
DOUBLE(size, d)	

Oefeningen

Voor je begint:

We gaan een tijdelijke tabel aanmaken met records uit de 'employees' tabel. Alle bewerkingen in deze oefenreeks worden uitgevoerd op deze tabel: empSubset.

```
-- Aanmaken tabel 'empSubset' aan.

CREATE TABLE empSubset LIKE employees;
ALTER TABLE empSubset
CHANGE COLUMN emp_no emp_no INT(11) NOT NULL AUTO_INCREMENT,
ADD UNIQUE INDEX emp_no_UNIQUE (emp_no ASC);



-- Verwijderen tabel.

DROP TABLE empSubset;
```

(a) Maak een nieuwe werknemer 'Jos den Os' aan in de tabel 'empSubset'. Zorg ook dat alle relevante waarden ingevuld worden.

(b) Copieer de werknemers met id tussen 10000 en 11000 naar de tabel empSubset

(c) Copieer birth_date, hire_date, first_name, last_name, gender, kids voor de werknemers met id tussen 11001 en 12000;

(d) Zet de favoriete kleur voor de werknemers zonder favoriete kleur op 'WHITE'

(e) Doordenker: Genereer een willekeurige schoenmaat voor de werknemers. Uiteraard zijn de maten voor de vrouwelijke collega's kleiner (35-38) dan die van de mannelijke (39-44).

(f) Verwijder alle werknemers met minder dan 2 kinderen.

(g) Verwijder alle werknemers met een schoenmaat die kleiner is dan het gemiddelde.

(a) Geef een overzicht, met alle gegevens, van alle departementen.

```
SELECT * FROM [werkDB.]departments;
```

(b) Geef een alfabetisch overzicht (zonder departementsnummer) van alle departementen.

```
SELECT dept_name FROM [werkDB.]departments ORDER BY dept_name;
```

(c) Toon alle vrouwelijke werknemers geboren na 1976. (Data worden in het formaat 'YYYY-MM-DD' geschreven)

```
SELECT * FROM [werkDB.]employees
WHERE gender like 'F' AND birth_date >= '1977-01-01';
```

(d) Welke werknemers zijn aangenomen in januari 2013. (Data worden in het formaat 'YYYY-MM-DD' geschreven)

```
SELECT first_name, last_name
FROM [werkDB.]employees
WHERE hire_date >= '2013-01-01' AND hire_date < '2013-02-01';
```

(e) Maak een overzicht van alle werknemers alfabetisch gesorteerd op voornaam en vervolgens omgekeerd alfabetisch op achternaam. Enkel de gegevens first_name, last_name, emp_no en birth_date tonen

```
SELECT first_name, last_name, emp_no, birth_date
FROM [werkDB.]employees
ORDER BY first_name ASC, last_name DESC;
```

(f) Maak een overzicht van de verschillende titels in het bedrijf. Geen dubbels!!!

```
SELECT DISTINCT title FROM titles;
```

(g) Maak een lijst van de werknemers met emp_no van 10900 tot en met 11000. (zonder <, <=, >=, >)

```
SELECT * FROM employees WHERE emp_no BETWEEN 10900 AND 11000;
```

(h) Maak een lijst van alle werknemers die geen a in hun voornaam hebben.

```
SELECT * FROM employees WHERE first_name NOT LIKE '%a%';
```

(i) Lijst alle verschillende favoriete kleuren op.

```
SELECT DISTINCT fav_colour FROM employees;
```

(j) Toon de eerste 100 werknemers zonder kinderen en als favoriete kleur “White”.

```
SELECT *
FROM employees
WHERE kids = 0 and fav_colour LIKE "WHITE" LIMIT 100;
```

(k) Geef een alfabetisch overzicht van alle vrouwen met kinderen.

```
SELECT * FROM employees
WHERE kids <> 0 and gender like 'f'
ORDER BY last_name, first_name;
```

(l) Toon een lijst van werknemers die van de kleur wit, blauw of purper houden

```
SELECT * FROM employees
WHERE fav_colour IN ("WHITE", "BLUE", "PURPLE");
```

(m) Doordenker: Creeer een lijst van alle volledige namen en noem de kolom: “FullName”. Doe dit voor alle mensen waarvan de voornaam of achternaam begint met een 'b'.

```
SELECT concat(first_name, " ", last_name) AS FullName
FROM employees
WHERE first_name LIKE 'b%' OR last_name LIKE 'b%';
```

(n) Doordenker: Geef een lijst van alle 30- en 40-jarigen.

```
SELECT first_name, last_name
FROM employees
WHERE FLOOR(DATEDIFF(CURDATE(), birth_date) / 365) IN (30, 40);
```

(a) Hoeveel verschillende departementen telt ons bedrijf?

```
SELECT COUNT(*) FROM departments;
```

(b) Bereken het gemiddelde salaris uitbetaald aan de werknemers.

```
SELECT AVG(salary) FROM werkdb.salaries;
```

(c) Geef de gemiddelde, de laagste en de hoogste salaris voor de werknemer 10052.

```
SELECT MIN(salary), MAX(salary), AVG(salary)
FROM salaries
WHERE emp_no = 10052;
```

(d) Bereken het gemiddelde salaris van werknemer 10423 in SpaceEuros als je weet dat 1 Euro = 2 SpaceEuro?

```
SELECT AVG(salary) * 2
FROM salaries
WHERE emp_no = 10423;
```

(e) Zoek op van welke datum tot welke datum werknemers 16222 loon heeft ontvangen.

```
SELECT MIN(from_date), MAX(to_date) FROM salaries
WHERE emp_no = 16222;
```

(f) Bereken het totaal aantal kinderen van onze vrouwelijke werknemers.

```
SELECT SUM(kids)
FROM employees
WHERE gender = 'F'
```

(a) Tel het aantal mannen en vrouwen die werken/gewerkt hebben in ons bedrijf. Gebruik 1 commando en geef ook duidelijk aan welk cijfer van welke groep afkomstig is.

```
SELECT gender, COUNT(*)
FROM employees
GROUP BY gender;
```

(b) Tel het aantal werknemers per departement (table: dept_emp) die momenteel nog in dienst zijn. (Mensen die nog in dienst zijn hebben als einddatum: 9999-01-01)

```
SELECT dept_no, COUNT(*)
FROM dept_emp
WHERE to_date = '9999-01-01'
GROUP BY dept_no;
```

(c) Ga na of er een verband is tussen favoriete kleur en het (gemiddelde) aantal kinderen voor onze werknemers.

```
SELECT fav_colour, AVG(kids)
FROM employees
GROUP BY fav_colour;
```

(a) Maak een volledig overzicht (geen specifieke kolommen) van alle managers met hun persoonsgegevens samen met de gegevens uit dept_manager.

```
SELECT *
FROM dept_manager
INNER JOIN employees ON dept_manager.emp_no = employees.emp_no;
```

(b) Toon de werknemers met id tussen 10500 en 10600 met hun huidige titel.

```
SELECT * FROM employees
INNER JOIN titles ON employees.emp_no = titles.emp_no
WHERE employees.emp_no BETWEEN 10500 AND 10600
AND to_date = '9999-01-01';
```

(c) Herschrijf (a) om enkel de huidige managers te tonen. Toon alleen het departementsnummer en de naam (voor- en achternaam) van de manager.

```
SELECT dept_no, first_name, last_name
FROM dept_manager
INNER JOIN employees ON dept_manager.emp_no = employees.emp_no
WHERE to_date = '9999-01-01';
```

(d) Herschrijf (b) zodat je de huidige manager toont van elk departement met de departementsnaam (departements).

```
SELECT dept_name, first_name, last_name
FROM dept_manager
INNER JOIN employees ON dept_manager.emp_no = employees.emp_no
INNER JOIN departments ON dept_manager.dept_no =
departments.dept_no
WHERE dept_manager.to_date = '9999-01-01';
```

(e) Bereken de gemiddelde schoenmaat per departement en plaats dit in een overzicht samen met de naam van het departement.

```
SELECT departments.dept_name, AVG(shoesize)
FROM employees
INNER JOIN dept_emp ON dept_emp.emp_no = employees.emp_no
INNER JOIN departments ON dept_emp.dept_no = departments.dept_no
GROUP BY dept_emp.dept_no;
```

(f) Toon een overzicht van de huidige werknemers die werken in 'Development' werken.

```
SELECT *
FROM employees
INNER JOIN dept_emp ON employees.emp_no = dept_emp.emp_no
INNER JOIN departments ON departments.dept_no = dept_emp.dept_no
WHERE dept_emp.to_date = '9999-01-01' AND dept_name LIKE 'DEV%';
```

(g) Doordenker: Toon de departementen waar de gemiddelde leeftijd van de huidige werknemers minder dan 40 jaar is.

```
SELECT departments.dept_name, departments.dept_no,
AVG(FLOOR(DATEDIFF(CURDATE(), birth_date) / 365)) AS avg_age
FROM employees
INNER JOIN dept_emp ON employees.emp_no = dept_emp.emp_no
INNER JOIN departments ON departments.dept_no = dept_emp.dept_no
WHERE dept_emp.to_date = '9999-01-01'
GROUP BY dept_emp.dept_no
HAVING avg_age < 40;
```

(a) Toon alle werknemers die het meeste kinderen hebben.

```
SELECT *
FROM employees
WHERE kids = (SELECT MAX(kids) FROM employees);
```

(b) Toon de werknemers die een hoger salaris hebben dan het gemiddelde loon. Zorg dat de namen niet dubbel getoond worden.

```
SELECT first_name, last_name
FROM employees
WHERE emp_no IN (SELECT DISTINCT emp_no
                 FROM salaries
                 WHERE salary > (SELECT AVG(salary)
                                 FROM salaries));
```

(c) Toon het gemiddelde loon van elke huidige departementsmanagers (tevens ook de namen van deze managers) en toon in 1 keer ook de som van deze lonen.

```
SELECT first_name, last_name, avg(salary)
FROM employees as e
INNER JOIN dept_manager as dm ON e.emp_no = dm.emp_no
INNER JOIN salaries as s ON s.emp_no = e.emp_no
WHERE dm.to_date = '9999-01-01'
GROUP BY e.emp_no
UNION
SELECT "", "Totaal: ", SUM(avg_sal)
FROM ( SELECT avg(salary) as avg_sal
FROM employees as e
INNER JOIN dept_manager as dm ON e.emp_no = dm.emp_no
INNER JOIN salaries as s ON s.emp_no = e.emp_no
WHERE dm.to_date = '9999-01-01'
GROUP BY e.emp_no) AS temp;
```

(d) Verdeel de salarissen in 4 gelijke intervallen en bereken voor elk interval het gemiddelde loon.

```
SELECT 'Interval 1:' AS 'Interval', AVG(salary) AS Gemiddelde FROM
salaries
WHERE salary < (SELECT MIN(salary) FROM salaries) + (SELECT
(MAX(salary) - MIN(salary)) / 4 FROM salaries)
UNION
SELECT 'Interval 2:' AS 'Interval', AVG(salary) FROM salaries
WHERE salary >= (SELECT MIN(salary) FROM salaries) + (SELECT
(MAX(salary) - MIN(salary)) / 4 FROM salaries)
AND salary < (SELECT MIN(salary) FROM salaries) + 2 * (SELECT
(MAX(salary) - MIN(salary)) / 4 FROM salaries)
UNION
SELECT 'Interval 3:' AS 'Interval', AVG(salary) FROM salaries
WHERE salary >= (SELECT MIN(salary) FROM salaries) + 2 * (SELECT
(MAX(salary) - MIN(salary)) / 4 FROM salaries)
AND salary < (SELECT MIN(salary) FROM salaries) + 3 * (SELECT
(MAX(salary) - MIN(salary)) / 4 FROM salaries)
UNION
SELECT 'Interval 4:' AS 'Interval', AVG(salary) FROM salaries
WHERE salary >= (SELECT MIN(salary) FROM salaries) + 3 * (SELECT
(MAX(salary) - MIN(salary)) / 4 FROM salaries);
```

(a) Maak een nieuwe werknemer 'Jos den Os' aan in de tabel 'empSubset'. Zorg ook dat alle relevante waarden ingevuld worden.

```
INSERT INTO empSubset
VALUES (1, "1980-01-01", "Jos", "Den Os", 'M', '2015-01-01',
'WHITE', 2, 46);
```

(b) Copieer de werknemers met id tussen 10000 en 11000 naar de tabel empSubset

```
INSERT INTO empSubset
VALUES (1, "1980-01-01", "Jos", "Den Os", 'M', '2015-01-01',
'WHITE', 2, 46);
```

(c) Copieer birth_date, hire_date, first_name, last_name, gender, kids voor de werknemers met id tussen 11001 en 12000;

```
INSERT INTO empSubset (birth_date, hire_date, first_name,
last_name, gender, kids)
SELECT birth_date, hire_date, first_name, last_name, gender, kids
FROM employees WHERE emp_no BETWEEN 11001 AND 12000;
```

(d) Zet de favoriete kleur voor de werknemers zonder favoriete kleur op 'WHITE'

```
UPDATE empSubset SET fav_colour = 'WHITE' WHERE fav_colour IS
NULL;
```

(e) Doordenker: Genereer een willekeurige schoenmaat voor de werknemers zonder shoenmaat. Uiteraard zijn de maten voor de vrouwelijke collega's kleiner (35-38) dan die van de mannelijke (39-44). (Tip: 2 commando's)

```
UPDATE empSubset SET shoesize = FLOOR( 35 + RAND( ) * 3 )
WHERE gender = 'F' AND shoesize IS NULL;
UPDATE empSubset SET shoesize = FLOOR( 39 + RAND( ) * 5 )
WHERE gender = 'M' AND shoesize IS NULL;
```

(f) Verwijder alle werknemers met minder dan 2 kinderen.

```
DELETE FROM empSubset WHERE kids < 2;
```

(g) Verwijder alle werknemers met een schoenmaat die groter is

dan het gemiddelde.

```
DELETE FROM empSubset
WHERE shoesize > (SELECT AVG(shoesize) FROM empSubset);
```