Inhoudsopgave.

Inleiding.

Als er grote hoeveelheden data permanent opgeslagen moeten worden dan kun je hier natuurlijk zelf een programma voor schrijven dat zelf de files waarin de data is opgeslagen beheerd. Waarschijnlijk ben je er bij het H1 project al achter gekomen dat dit niet zo eenvoudig is. Vooral niet bij meerdere bestanden met gekoppelde gegevens. In dit geval kun je beter gebruik maken van een *Database Management System* (DBMS). Er zijn verschillende manieren om een database op te bouwen: hiërarchies, relationeel en object oriënted. Relationele Databases (RDBMS) zijn op dit moment het meest populair. Bekende relationele databases zijn Oracle, DB2, MySQL en Microsoft SQL Server.

Als je informatie uit een database wilt opvragen dan kun je gebruik maken van de standaard “vraagtaal” SQL (*Structured Query Language*). De SQL query:
```
SELECT ALL WHERE NAME = "Broeders" AND AGE > 40
```
vraagt bijvoorbeeld alle records op waarbij het veld `NAME` de waarde Broeders heeft en het veld `AGE` groter is dan 40.

De C&D studenten die hebben gekozen voor de TI minor komen uitgebreid in aanraking met relationele databases en SQL. Ook voor de overige C&D studenten is het echter van belang om enige kennis te hebben van databases. In dit dictaat vind je een basiscursus SQL die gedeeltelijk afkomstig is van http://www.sqlcourse.com en gedeeltelijk afkomstig is uit het dictaat Databases van Henk van den Bosch (docent TI op onze school).

Alle tabellen en SQL code uit deze basiscursus kun je downloaden van http://bd.thrijswijk.nl/sopx3/ zowel voor een Microsoft Access database als voor een Paradox database. Microsoft Acces is op alle studentwerkplekken beschikbaar en de Paradox database is te benaderen met het DataBase Desktop programma dat met Borland C++ Builder wordt meegeleverd.

Borland C++ Builder bevat een groot aantal standaard componenten waarmee vrijwel elk RDBMS te benaderen is via SQL. Kijk maar eens onder de tabs Data Access en Data Control.

1 Databases en Data Base Management Systems.

Een database is een verzameling van samenhangende gegevens opgeslagen op een achtergrondgeheugen. Een Data Base Management System (DBMS) is het software systeem dat dient voor het gebruik en het beheer van een database. Er zijn verschillende manieren bedacht waarop de logische gegevenstructuur van een database vastgelegd kan worden:

- hiërarchische databases.
- netwerk databases.
- relationele databases.
- semantische databases.
- object oriënted databases.

Op dit moment gebruiken bijna alle DBMS'en het relationele model. Deze worden RDBMS'en genoemd.

2 Tabellen.

Een relationeel database systeem bevat één of meer tabellen. De data of informatie die in de database is opgeslagen, is opgeslagen in deze tabellen. Elke tabel heeft een unieke naam en bestaat uit kolommen en rijen. Een kolom heeft een kolom naam (die binnen de tabel uniek moet zijn), een datatype en eventueel nog een aantal andere attributen. Een rij (ook wel record genoemd) bevat de in de kolommen gedefi-

nieerde data. Hieronder is een voorbeeld tabel gegeven genaamd leverancier. LevNr, Naam, Kwaliteit, en Plaats zijn de kolommen. De rijen bevatten de data van deze tabel:

Tabel: leverancier

LevNr	Naam	Kwaliteit	Plaats
1	Simon	20	Den Haag
2	Janssen	10	Utrecht
3	Bernard	30	Utrecht
4	Janssen	20	Den Haag
5	Bout	30	Leiden

De tabelnaam wordt ook wel entiteitstype genoemd. Een entiteit is de beschrijving van een “object” uit de werkelijkheid waarvan we de gegevens in de database willen opslaan. Elke regel van een tabel bevat een entiteit. De kolomnamen worden ook wel atributen genoemd. Elk “hokje” in de tabel bevat een zogenaamde attribuutwaarde van een entiteit. Om de verschillende entiteiten (regels) van een entiteitstype (tabel) te kunnen onderscheiden is het begrip sleutel (key) gedefinieerd. De sleutel van een entiteitstype is een attribuut (of combinatie van attributen) van dit entiteitstype waarvan de waarde elke entiteit uniek identificeert. Elke tabel heeft een key nodig omdat de entiteiten anders niet onderscheiden kunnen worden.

Soms zijn meerdere attributen geschikt om als sleutel te gebruiken. Als we voor het entiteitstype student b.v. de atributen studentnummer, naam, woonplaats, geboortedatum, SOFI-nummer en paspoortnummer definiëren dan kunnen we het attribuut studentnummer als key gebruiken maar ook het atribuut SOFI-nummer. De key we uiteindelijk kiezen om een entiteit mee te identificeren wordt de primaire sleutel (primary key) genoemd. De atribuut paspoortnummer is overigens niet als sleutel voor student te gebuiken omdat niet elke student een paspoort heeft.

Soms is geen enkele atribuut (of combinatie van atributen) geschikt om als sleutel te gebuiken. De oplossing is simpel: we voegen dan gewoon een geschikte atribuut toe. Dit is de reden dat de atribuut LevNr in de tabel leverancier is opgenomen.

In deze cursus zullen we nog twee andere tabellen gebruiken:

Tabel: artikel

ArtNr	Naam	Kleur	Gewicht
1	moer	rood	12
2	bout	groen	17
3	schroef	blauw	17
4	schroef	rood	14
5	veer	blauw	12
6	as	rood	19

De primary key van deze tabel is ArtNr. Primary keys kunnen in andere tabellen als verwijs sleutel (foreign key) gebruikt worden. Om deze reden kun je een primary key niet zo maar veranderen!

Tabel: bestelling

LevNr	ArtNr	Aantal
1	1	300
1	2	200
1	3	400
1	4	200
1	5	100
1	6	100
2	1	300
2	2	400
3	2	200
4	2	200
4	4	300
4	5	400

Het atribuut LevNr is een foreign key die verwijst naar een entiteit uit de tabel leverancier. Het atribuut ArtNr is een foreign key die verwijst naar een entiteit uit de tabel artikel. De atributen LevNr en ArtNr vormen samen de primary key voor de tabel bestelling.

3 Wat is SQL?

SQL (speek uit: “ess-que-el”) is de afkorting van Structured Query Language. SQL wordt gebruikt om met een database te communiceren. De SQL taal is gestandaardiseerd door ANSI (American National Standards Institute) en wordt door ieder relationeel database management systeem ondersteund. SQL statements (of commando’s) kunnen bijvoorbeeld gebruikt worden om data in de database bij te werken of om data uit de database op te vragen. Een aantal bekende relationele database management systemen zijn: Oracle, DB2, Sybase, Microsoft SQL Server, Access, Ingres, MySQL, etc. Al deze database systemen gebruiken SQL, maar de meeste hebben hun eigen uitbreidingen aan de standaard SQL toegevoegd (zogenaamde “proprietary extensions”). Met behulp van de standaard SQL commando’s zoals `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE`, en `DROP` kunnen we alles doen wat nodig is om een database te beheren. Dit hoofdstuk geeft een inleiding in het gebruik van de belangrijkste SQL statements.

4 Gegevens opvragen (deel 1).

Het `SELECT` statement wordt gebruikt om informatie uit de database op te vragen. Z’n vraag wordt een “query” genoemd. Bij het opvragen van de informatie kun je bepaalde selectiecriteria opgeven. Er wordt dan gezocht naar de informatie die aan deze selectiecriteria voldoet. Het formaat van een eenvoudig `SELECT` statement is als volgt:

```
SELECT kolomnaam[, kolomnaam, ...]
FROM tabelnaam [WHERE selectiecriterium];
```

`[ ]` = optioneel

- Het resultaat van een `SELECT` statement wordt gegeven in de vorm van een (tijdelijke) tabel.
- Na het `SELECT` keyword volgt een lijst met kolomnamen. Deze lijst bepaald welke kolommen in de resultaat tabel voorkomen. Je kunt zoveel kolommen selecteren als je wilt. Je kunt alle kolommen selecteren door een `*` te gebruiken.
- Na het keyword `FROM` volgt een tabelnaam. Deze tabel wordt doorzocht om de informatie te vinden.

- De `WHERE` clausule (clause) kan worden gebruikt om het selectiecriterium op te geven. Dit selectiecriterium (ook wel conditie genoemd) heeft als eenvoudigste vorm:
  `kolomnaam operator waarde`

  Operatoren die in een `WHERE` clause gebruikt kunnen worden:

`=`	gelijk aan
`>`	groter dan
`<`	kleiner dan
`>=`	groter dan of gelijk aan
`<=`	kleiner dan of gelijk aan
`<>`	niet gelijk aan
`LIKE`	De `LIKE` operator is een erg krachtige operator die je in staat stelt om rijen te selecteren die lijken op de `waarde` die je opgeeft. Het procent teken `%` kan gebruikt worden als een “joker” die overeenkomt met één of meer willekeurige karakters.

Voorbeeld 1:

```
SELECT Gewicht, Naam
FROM artikel;
```

Het resultaat van dit `SELECT` statement is de volgende tabel:

Gewicht	Naam
12	moer
17	bout
17	schroef
14	schroef
12	veer
19	as

Voorbeeld 2:

Geef de naam en het gewicht van alle rode artikelen.

```
SELECT Naam, Gewicht
FROM artikel
WHERE Kleur='rood';
```

Resultaat:

Naam	Gewicht
moer	12
schroef	14
as	19

Voorbeeld 3:

Geef alle gegevens van de leveranciers waarvan de naam met een B begint.

```
SELECT *
FROM leverancier
WHERE Naam LIKE 'B%';
```

Resultaat:

LevNr	Naam	Kwaliteit	Plaats
3	Bernard	30	Utrecht
5	Bout	30	Leiden

4.1 Boolean operatoren.

Eenvoudige selectiecriteria kunnen met `AND` en `OR` worden gecombineerd tot een samengesteld selectie-criterium.

Voorbeeld 4:

Geef alle bestellingen met LevNr 1 waarbij het aantal groter of gelijk aan 300 is.

```
SELECT *
FROM bestelling
WHERE LevNr = 1 AND Aantal >= 300;
```

Resultaat:

LevNr	ArtNr	Aantal
1	1	300
1	3	400

De vergelijkingsoperatoren hebben een hogere prioriteit dan de boolean opterators. Haakjes zijn dus in dit geval niet nodig maar als je het duidelijker vindt dan mag je haakjes toevoegen:

```
SELECT *
FROM bestelling
WHERE (LevNr = 1) AND (Aantal >= 300);
```

4.2 `IN` en `BETWEEN` operatoren.

De `IN` and `BETWEEN` operatoren kunnen in een `WHERE` clause als volgt gebuikt worden:

```
SELECT lijst-met-kolomnamen
FROM tabelnaam
WHERE kolomnaam IN (lijst-met-waarden);

SELECT lijst-met-kolomnamen
FROM tabelnaam
WHERE kolomnaam BETWEEN waarde AND waarde;
```

De `IN` operator selecteert een rij als de data in de kolom voorkomt in de lijst met waarden.

Voorbeeld 5:

```
SELECT *
FROM leverancier
WHERE Plaats IN ('Den Haag', 'Leiden');
```

Resultaat:

LevNr	Naam	Kwaliteit	Plaats
1	Simon	20	Den Haag
4	Janssen	20	Den Haag
5	Bout	30	Leiden

Een selectiecriterium waarin de `IN` operator wordt gebruikt kan worden herschreven met behulp van de `=` en `OR` operatoren:

```
SELECT *
FROM leverancier
WHERE Plaats = 'Den Haag' OR Plaats = 'Leiden';
```

Maar de `IN` operator is korter en eenvoudiger te lezen zeker als je wilt testen op meer dan twee waarden.

Je kunt de operator `NOT IN` gebruiken om rijen waarvan de data niet in de kolom voorkomt te selecteren.

De `BETWEEN` operator kun je gebruiken om te testen of de data “tussen” de twee opgegeven waarden ligt. De rij wordt ook geselecteerd als de data in de opgegeven kolom gelijk is aan een van de twee opgegeven waarden. Je kunt `BETWEEN` *`waarde`* `AND` *`waarde`* dus beter lezen als “van waarde tot en met waarde”.

Voorbeeld 6:

```
SELECT Naam
FROM artikel
WHERE Gewicht BETWEEN 14 AND 17;
```

Dit statement zal de kolom Naam van de tabel artikel laten zien waarbij alle rijen geselecteerd worden waarbij de waarde in de kolom Gewicht tussen 14 en 17 is (inclusief 14 en 17).

Resultaat:

Naam
bout
schroef
schroef

Dit statement kan worden herschreven met behulp van de `>=`, `<=` en `OR` operatoren:

```
SELECT Naam
FROM artikel
WHERE Gewicht >= 14 AND Gewicht <= 17;
```

Je kunt de `NOT BETWEEN` operator gebruiken om te testen of de data niet “tussen” de twee opgegeven waarden ligt.

4.3 Rekenkundige operatoren.

Standaard ANSI SQL-92 definieert de eerste vier van de onderstaande rekenkundige operatoren:

rekenkundige operatoren

`+`	optellen
`-`	aftrekken
`*`	vermenigvuldigen
`/`	delen
`%`	modulo (rest)

De modulo operator bepaald de rest van een integer deling. Deze operator is niet gedefinieerd in ANSI SQL, maar de meeste databases ondersteunen deze uitbreiding.

4.4 Wiskundige functies.

In de onderstaande tabel staan enkele wiskundige functies. Deze functies zijn niet opgenomen in de ANSI SQL-92 standaard. Deze functies kunnen in alle bekende database systemen gebruikt worden.

`ABS(`*x*`)`	geeft de absolute waarde van *x*
`SIGN(`*x*`)`	geeft het teken van de waarde *x* (-1, 0, of 1)
`MOD(`*x*`,`*y*`)`	geeft de rest van de deling *x*`/`*y* (gelijk aan *x*`%`*y*)
`FLOOR(`*x*`)`	geeft de grootste integer waarde kleiner dan of gelijk aan *x*
`CEILING(`*x*`)` of `CEIL(`*x*`)`	geeft de kleinste integer waarde groter dan of gelijk aan *x*
`POWER(`*x*`,`*y*`)`	geeft the waarde van *x* tot de macht *y*
`ROUND(`*x*`)`	geeft de waarde van *x* afgerond op de dichtsbijzijnde integer
`ROUND(`*x*`,`*d*`)`	geeft de waarde van *x* afgerond op `d` plaatsen achter de decimale punt
`SQRT(`*x*`)`	geeft de wortel van *x*

De rekenkundige operatoren en wiskundige functies mogen in de `WHERE` clause van een `SELECT` gebruikt worden om een selectiecriterium op te geven. De rekenkundige operatoren en wiskundige functies kun je ook gebuiken om bewerkingen op hele kolommen uit te voeren.

Voorbeeld 7:

```
SELECT LevNr, LevNr + Kwaliteit
FROM leverancier;
```

Resultaat:

LevNr	LevNr + Kwaliteit
1	21
2	12
3	33
4	24
5	35

Het `SELECT` statement heeft nog vele andere mogelijkheden maar daar komen we later op terug.

5 Tabellen aanmaken.

Het `CREATE TABLE` statement kun je gebruiken om een nieuwe tabel aan te maken. Het formaat van een eenvoudig `CREATE TABLE` statement is als volgt:

```
CREATE TABLE tabelnaam
(kolomnaam datatype[, kolomnaam datatype, ...]);
```

Bij elke kolom kun je ook bepaalde “constraints” (voorwaarden of beperkingen) opgeven:

```
CREATE TABLE tabelnaam
(kolomnaam datatype [constraints]
[, kolomnaam datatype [constraints], ...]);
```

`[ ]` = optioneel

Voorbeeld 8:

```
CREATE TABLE artikel
(ArtNr INTEGER,
Naam VARCHAR(20),
Kleur VARCHAR(20),
Gewicht INTEGER);
```

Resultaat:

Tabel: artikel

ArtNr	Naam	Kleur	Gewicht

Zoals je ziet bevat de tabel na de `CREATE TABLE` instructie nog geen data. De tabel kan later met behulp van de instructie `INSERT` worden gevuld.

Je kunt dus een nieuwe tabel aanmaken met het `CREATE TABLE` commando gevolgd door de tabelnaam, gevolgd door een haakje openen. Hierna geef je per kolom de kolomnaam, gevolgd door het datatype van deze kolom, eventueel gevolgd door constraints voor deze kolom. De definities van de verschillende kolommen worden van elkaar gescheiden door een komma. De laatste kolomdefinitie wordt afgesloten door een haakje sluiten. Het `CREATE TABLE` commando wordt, net zoals elk SQL commando afgesloten met een punt-komma.

De tabelnaam en kolomnamen moeten beginnen met een letter. In deze namen mogen alleen letters, cijfers en underscores voorkomen. Een naam mag niet langer zijn dan 30 karakters. Een SQL keyword zoals `SELECT`, `CREATE`, `INSERT`, enz mag niet als naam worden gebuikt.

Het datatype geeft aan wat voor soort gegevens in een kolom kunnen worden opgeslagen. Als je bijvoorbeeld de kolom genaamd “Naam” wilt gebruiken om namen in op te slaan dan moet deze kolom van het datatype `VARCHAR` (variable-length character) zijn.

De meest gebruikte datatypen.

`CHARACTER(`*`size`*`)`	Fixed-length character string. De *`size`* geeft het aantal karakters wat in deze kolom moet worden ingevuld. De maximale *`size`* is 255.
`VARCHAR(`*`size`*`)`	Variable-length character string. De *`size`* geeft het aantal karakters wat maximaal in deze kolom mag worden ingevuld. De maximale *`size`* is 255.
`INTEGER`	Geheel getal.
`FLOAT`	Floating point getal.
`DATE`	Datum.
`MONEY`	Geld bedrag.
`BOOLEAN`	Boolean.
`AUTOINC`	Autoincrement geheel getal.
`BLOB`	Binary large object. Dit datatype kan een muzieknummer, een plaatje, een film, een programma, een tekst in HTML, een WORD document, enz. bevatten. SQL bemoeit zich niet met de inhoud van een `BLOB`.

Wat zijn constraints? Bij het aanmaken van een tabel kun je bij elke kolom bepaalde constraints opgeven. Een constraint is een voorwaarde waaraan de data die in de kolom wordt opgeslagen moet voldoen. Zo kun je bijvoorbeeld met behulp van de `UNIQUE` constraint aangeven dat elke waarde hoogstens één keer in de betreffende kolom mag voorkomen. Alle waarden moeten uniek zijn. De RDBMS zal bij het invoeren van gegevens in de tabel controlleren of aan alle constraints wordt voldaan. Als dit niet het geval is zal het RDBMS een passende foutmelding geven. Enkele andere voorbeelden van constraints zijn `NOT NULL` en `PRIMARY KEY`. `NOT NULL` geeft aan dat in deze kolom in elke rij een waarde moet worden ingevuld. De `PRIMARY KEY` constraint definieerd dat deze kolom als primary key van de tabel gebruikt wordt. We hebben al gezien dat de primary key voor elke rij (elk record) uniek moet zijn. Elke tabel is geoptimaliseerd om te zoeken op de primary key.

Voorbeeld 9:

Je hebt net een eigen bedrijf opgericht en je gaat voor het eerst personeel aannemen. Je wilt de gegevens van je personeelsleden in een database opnemen. Omdat je verwacht in de toekomst uit te groeien tot een multinational sla je alle gegevens op in het Engels. Je wilt de volgende gegevens bijhouden: firstname, lastname, title, age, en salary.

Oplossing:

```
CREATE TABLE employee
(firstname VARCHAR(30),
lastname VARCHAR(30) NOT NULL,
title VARCHAR(10),
age INTEGER,
salary MONEY);
```

Resultaat:

firstname	lastname	title	age	salary

6 Gegevens invoeren.

Het INSERT statement kun je gebruiken om een rij gegevens aan een tabel toe te voegen.

```
INSERT INTO tabelnaam [(kolomnaam, kolomnaam, ...)]
VALUES (waarde, waarde, ...);
```

[ ] = optioneel

Om een rij gegevens (een record) in een tabel toe te voegen moet je het keyword INSERT INTO gebruiken. Gevolgd door de tabelnaam, gevolgd door een haakje openen, gevolgd door een lijst van kolomnamen gescheiden met komma's, gevolgd door een haakje sluiten, gevolgd door het keyword VALUES, gevolgd door een haakje openen, gevolgd door een lijst met waarden, gevolgd door een haakje sluiten. De waarden die je opgeeft worden opgeslagen in één rij in de tabel in de kolommen die je hebt opgegeven. Als je geen kolomnamen hebt opgegeven dan worden de waarden "gewoon" van links naar rechts weggeschreven. Strings moeten tussen enkele aanhalingstekens worden opgegeven.

Voorbeeld 10:

```
INSERT INTO employee
(firstname, lastname, age, salary)
VALUES ('Luke', 'Duke', 45, 45000.00);
```

Resultaat:

firstname	lastname	title	age	salary
Luke	Duke		45	45000.00

De uitkomst van een SELECT statement kan ook gebruikt worden als invoer voor een INSERT statement.

```
INSERT INTO tabelnaam [(kolomnaam, kolomnaam, ...)]
SELECT ... FROM ... WHERE ...
```

[ ] = optioneel

Voorbeeld 11:

Maak een tabel genaamd goedeLeveranciers met de namen en vestigingsplaatsen van alle leverancies die een kwaliteit hebben >15.

Oplossing:

```
CREATE TABLE goedeLeverancier
(GoedeLevNr AUTOINC PRIMARY KEY,
Naam VARCHAR(30),
VestigingsPlaats VARCHAR(30));

INSERT INTO goedeLeverancier
(Naam, Vestigingsplaats)
SELECT Naam, Plaats
FROM leverancier
WHERE Kwaliteit > 15;
```

Resultaat:

Tabel: goedeLeverancier

GoedeLevNr	Naam	Vestigingsplaats
1	Simon	Den Haag
2	Bernard	Utrecht
3	Janssen	Den Haag
4	Bout	Leiden

7 Gegevens wijzigen.

Het `UPDATE` statement kun je gebruiken om een rij (record) in een tabel te wijzigen met een `WHERE` clause kun je een selectiecriterium opgeven. Alleen de rijen die aan dit criterium voldoen worden gewijzigd.

```
UPDATE tabelnaam
SET kolomnaam = waarde [,kolomnaam = waarde, ...]
WHERE selectiecriterium;
```

`[ ]` = optioneel

Omdat met de `UPDATE` instructie van hoogstens 1 tabel de gegevens kunnen worden gewijzigd treden er integriteits problemen op als meerdere tabellen gewijzigd moeten worden. In de meeste RDBMS'en is het mogelijk om zogenaamde transactions te definiëren. In een transactie kunnen dan meerdere SQL instructies worden samengevoegd die allemaal òf wel òf niet uitgevoerd.

Voorbeeld 12:

```
UPDATE artikel
SET Gewicht = 25
WHERE Kleur = 'blauw';

UPDATE artikel
SET Kleur = 'groen', Gewicht = 15
WHERE Naam = 'moer' AND Gewicht < 15;
```

Resultaat:

Tabel: artikel

ArtNr	Naam	Kleur	Gewicht
1	moer	groen	15
2	bout	groen	17
3	schroef	blauw	25
4	schroef	rood	14
5	veer	blauw	25
6	as	rood	19

8 Gegevens verwijderen.

Het `DELETE` statement kun je gebruiken om rijen (records) uit een tabel te verwijderen.

```
DELETE FROM tabelnaam
[WHERE selectiecriterium];
```

`[ ]` = optioneel

Om één of meer rijen uit een tabel te verwijderen kun je het `DELETE FROM` commando gebruiken, gevolgd door de tabelnaam, eventueel gevolgd door het keyword `WHERE`, gevolgd door het selectiecriterium waaraan de te verwijderen regels moeten voldoen. Als je geen `WHERE` clause gebruikt worden alle regels verwijderd.

Voorbeeld 13:

```
DELETE FROM employee;
```

Resultaat:

firstname	lastname	title	age	salary

```
DELETE FROM bestelling
WHERE Aantal = 100;
```

Resultaat:

Tabel: bestelling

LevNr	ArtNr	Aantal
1	1	300
1	2	200
1	3	400
1	4	200
2	1	300
2	2	400
3	2	200
4	2	200
4	4	300
4	5	400

Het is nu mogelijk dat er integriteitsproblemen ontstaan.

Voorbeeld 14:

Verwijder de leveranciers met een kwaliteit < 15.

```
DELETE FROM leverancier
WHERE Kwaliteit < 15;
```

Resultaat:

Tabel: leverancier

LevNr	Naam	Kwaliteit	Plaats
1	Simon	20	Den Haag
3	Bernard	30	Utrecht
4	Janssen	20	Den Haag
5	Bout	30	Leiden

Leverancier met LevNr 2 is nu uit de database verwijderd maar er staan nog verwijzingen naar deze leverancier in de tabel bestelling.

9 Tabellen verwijderen.

Het `DROP TABLE` statement kun je gebruiken om een tabel (inclusief alle gegevens) te verwijderen uit de database.

```
DROP TABLE tabelnaam
```

Voorbeeld 15:

```
DROP TABLE employee;
```

Je kunt een hele tabel, inclusief alle rijen, verwijderen met het `DROP TABLE` commando gevolgd door de tabelnaam. `DROP TABLE` is niet hetzelfde als het verwijderen van alle rijen uit een tabel met het DELETE FROM commando. Bij het verwijderen van alle rijen met behulp van het DELETE FROM commando blijven de colomnamen en bijbehorende datatypen en constraints behouden. Het DROP TABLE commando verwijderd de volledige tabel definitie.

10 Gegevens opvragen (deel 2).

Je hebt al geleerd hoe je het `SELECT` statement kunt gebruiken om gegevens uit de database op te vragen. Het `SELECT` statement heeft een groot aantal opties die gebruikt kunnen worden om geavanceerde queries samen te stellen. Je kunt dan allerlei vragen over de gegevens beantwoorden, bijvoorbeeld: Geef het minimale aantal en het maximale aantal van de bestellingen voor elk artikel.

Het `SELECT` statement heeft vijf clauses (opties) die je kunt opgeven. De `FROM` clause is als enige verplicht. Elke clause heeft weer een hele lijst met mogelijke selectiecriteria, options, parameters, enz. De veschillende clauses zullen nu worden besproken.

Het uitgebreide formaat van het `SELECT` statement is als volgt:

```
SELECT [ALL | DISTINCT] lijst-met-kolomnamen
FROM lijst-met-tabelnamen
[WHERE selectiecriterium]
[GROUP BY lijst-met-kolomnamen]
[HAVING selectiecriterium]
[ORDER BY lijst-met-kolomnamen [ASC | DESC] ]
```

`[ ]` = optioneel

10.1 `FROM` & `WHERE` clause quick review.

De `FROM` en `WHERE` clauses zijn al eerder besproken.

Voorbeeld 16:

```
SELECT ArtNr, Naam, Gewicht
FROM artikel
WHERE kleur = 'rood' OR Gewicht > 15;
```

Resultaat:

ArtNr	Naam	Gewicht
1	moer	12
2	bout	17
3	schroef	17
4	schroef	14
6	as	19

`ALL` en `DISTINCT` zijn keywords die je kunt gebruiken om alle (default) of alleen orginele (unieke) rijen (records) in de resultaattabel op te nemen. Als je alleen unieke waarden wil zien in een specifieke kolom dan kun je het `DISTINCT` keyword voor de kolomnaam plaatsen.

Voorbeeld 17:

```
SELECT DISTINCT Gewicht
FROM artikel;
```

Resultaat:

Gewicht
12
17
14
19

Het `ALL` keyword zal alle rijen die aan het selectiecriterium voldoen opnemen in de resultaattabel inclusief eventuele duplicaten. Het `ALL` keyword wordt default gebruikt als er niets wordt opgegeven.

10.2 Aggregate functies.

Aggregate functies.

`MIN(`*`kolomnaam`*`)`	geeft de laagste waarde in een bepaalde kolom
`MAX(`*`kolomnaam`*`)`	geeft de hoogste waarde in een bepaalde kolom
`SUM(`*`kolomnaam`*`)`	geeft de som van alle integer waarden in een bepaalde kolom
`AVG(`*`kolomnaam`*`)`	geeft de gemiddelde waarde van een bepaalde kolom
`COUNT(`*`kolomnaam`*`)`	geeft het aantal rijen in een bepaalde kolom
`COUNT(*)`	geeft het aantal rijen in een tabel

Je kunt aggregate functies gebruiken om een bewerking met alle elementen van een bepaalde kolom uit te voeren. Door deze functies worden alle rijen van een bepaalde kolom “samengenomen” tot één resultaat. Het engelse “to aggregrate” betekent verenigen.

Voorbeeld 18:

```
SELECT AVG(Gewicht)
FROM artikel;
```

Resultaat:

AVERAGE OF Gewicht
15.17

Voorbeeld 19:

```
SELECT AVG(Gewicht)
FROM artikel
WHERE Kleur = 'rood';
```

Resultaat:

AVERAGE OF Gewicht
15.00

Voorbeeld 20:

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

Resultaat:

COUNT(*)
12

De `COUNT(*)` aggregate functie is de enige aggregate functie waarbij je geen kolomnaam hoeft op te geven. Het bovenstaande voorbeeld bepaalt het aantal rijen (records) in de tabel bestelling.

Voorbeeld 21:

Bepaal het maximum aantal van een bepaalde bestelling uit de tabel bestelling.

```
SELECT max(Aantal)
FROM bestelling;
```

Resultaat:

MAX OF Aantal
400

10.3 `GROUP BY` clause.

De `GROUP BY` clause syntax:

```
SELECT lijst-van-kolomnamen
FROM lijst-van-tabelnamen
GROUP BY lijst-van-kolomnamen;
```

De `GROUP BY` clause kun je gebruiken om alle rijen waarvan de waarde in een bepaalde kolom (of meerdere kolommen) gelijk zijn te verzamelen in groepen. Als er nu een aggregrate functie wordt gebuikt dan wordt deze functie voor elke groep rijen appart uitgevoerd. Dit is het snelst uit te leggen met een voorbeeld.

Voorbeeld 22:

Bepaal voor elke leverancier het maximum aantal van alle bestellingen bij die leverancier.

```
SELECT LevNr, max(aantal)
FROM bestelling
GROUP BY LevNr;
```

Resultaat:

LevNr	MAX OF Aantal
1	400
2	400
3	200
4	400

Voorbeeld 23:

Geef voor elk artikel het minimale bestelde aantal en het maximale bestelde aantal van de bestellingen van dat artikel.

```
SELECT ArtNr, min(Aantal), max(Aantal)
FROM bestelling
GROUP BY ArtNr;
```

Resultaat:

ArtNr	MIN OF Aantal	MAX OF Aantal
1	300	300
2	200	400
3	400	400
4	200	300
5	100	400
6	100	100

10.4 `HAVING` clause.

`HAVING` clause syntax:

```
SELECT lijst-van-kolomnamen
FROM lijst-van-tabelnamen
GROUP BY lijst-van-kolomnamen
HAVING selectiecriterium;
```

De `HAVING` clause kun je gebruiken om een selectiecriterium op te geven waaraan de rijen in de groepen moeten voldoen. De `HAVING` clause kan alleen na een `GROUP BY` clause gebruikt worden. De `HAVING` clause kan het best worden uitgelegd met een voorbeeld.

Voorbeeld 24:

In dit voorbeeld maken we weer gebruik van de tabel bestelling. Als je per leverancier het gemiddelde aantal van alle bestellingen bij die bepaalde levencier wilt bepalen dan kan dat als volgt:

```
SELECT LevNr, avg(Aantal)
FROM bestelling
GROUP BY LevNr;
```

Maar stel nu dat je dit gemiddelde aantal alleen wilt afdrukken als dit gemiddelde groter is dan of gelijk is aan 250:

```
SELECT LevNr, avg(Aantal)
FROM bestelling
GROUP BY LevNr
HAVING avg(Aantal) >= 250;
```

Resultaat:

LevNr	AVERAGE OF Aantal
2	350.00
4	300.00

10.5 ORDER BY clause.

`ORDER BY` clause syntax:

```
SELECT lijst-met-kolomnamen
FROM lijst-met-tabelnamen
ORDER BY lijst-met-kolomnamen [ASC | DESC];
```

`[ ]` = optioneel

De `ORDER BY` clause kun je gebruiken om de resultaat tabel te sorteren (in oplopende of in aflopende volgorde). De resultaat tabel wordt gesorteerd op de in de `ORDER BY` clause opgegeven kolommen.

`ASC`	Oplopende volgorde - default
`DESC`	Aflopende volgorde

Voorbeeld 25:

```
SELECT *
FROM bestelling
WHERE Aantal > 250
ORDER BY Aantal DESC;
```

Resultaat:

LevNr	ArtNr	Aantal
1	3	400
2	2	400
4	5	400
1	1	300
2	1	300
4	4	300

Voorbeeld 26:

```
SELECT *
FROM bestelling
WHERE Aantal > 250
ORDER BY Aantal DESC, ArtNr;
```

Resultaat:

LevNr	ArtNr	Aantal
2	2	400
1	3	400
4	5	400
1	1	300
2	1	300
4	4	300

11 Gegevens opvragen door meerdere tabellen te combineren.

Tot nu toe hadden alle queries betrekking op de gegevens uit één enkele tabel. Het wordt echter pas leuk als we de gegevens uit verschillende tabellen gaan combineren. De mogelijkheid om gegevens uit verschillende tabellen met elkaar te combineren is een van de meest waardevolle eigenschappen van relationele databases en SQL.

Je kunt de gegevens uit twee of meer tabellen met elkaar combineren door:
- een `SELECT` statement op de “join” van twee of meer tabellen uit te voeren.
- het resultaat van een `SELECT` statement te gebruiken in de `WHERE` clause van een ander `SELECT` statement. Dit wordt dan een geneste `SELECT` genoemd.

We zullen deze methoden één voor één bespreken. Natuurlijk kun je deze twee methoden ook weer met elkaar combineren. De kracht van relationele databases zit in deze mogelijkheden om de gegevens uit verschillende tabellen met elkaar te combineren. Op deze manier kun je gebruik maken van de relaties tussen de verschillende tabellen.

11.1 Gegevens combineren door middel van “join”.

Een `SELECT` statement waarbij de gegevens uit meerdere tabellen gecombineerd worden wordt een “Join” genoemd.

Met behulp van een “join” kun je de informatie uit twee of meer tabellen combineren in één resultaat tabel door middel van een `SELECT` statement. Je maakt een “join” door meerdere tabelnamen achter het `FROM` keyword te vermelden:

```
SELECT lijst-met-kolomnamen
FROM lijst-met-tabelnamen
WHERE selectiecriterium
```

Voorbeeld 27:

Geef alle details van alle bestellingen:

```
SELECT leverancier.Naam, leverancier.Kwaliteit, leverancier.Plaats,
artikel.Naam, artikel.Kleur, artikel.Gewicht, bestelling.Aantal
FROM bestelling, leverancier, artikel
WHERE bestelling.LevNr = leverancier.LevNr
AND bestelling.ArtNr = artikel.ArtNr;
```

Resultaat:

Naam	Kwaliteit	Plaats	Naam_1	Kleur	Gewicht	Aantal
Simon	20	Den Haag	moer	rood	12	300
Simon	20	Den Haag	bout	groen	17	200
Simon	20	Den Haag	schroef	blauw	17	400
Simon	20	Den Haag	schroef	rood	14	200
Simon	20	Den Haag	veer	blauw	12	100
Simon	20	Den Haag	as	rood	19	100
Janssen	10	Utrecht	moer	rood	12	300
Janssen	10	Utrecht	bout	groen	17	400
Bernard	30	Utrecht	bout	groen	17	200
Janssen	20	Den Haag	bout	groen	17	200
Janssen	20	Den Haag	schroef	rood	14	300
Janssen	20	Den Haag	veer	blauw	12	400

Deze “join” wordt ook wel “Inner Join” of “Equijoin” genoemd.

De kolomnamen worden nu voorafgegaan door de tabelnaam en een punt. Dit is alleen maar noodzakelijk als een bepaalde kolomnaam in meerdere tabellen voorkomt. Voor de duidelijkheid is het echter aan te bevelen om bij alle kolomnamen de tabelnaam waarin deze kolom voorkomt te specificeren.

Voorbeeld 28:

```
SELECT Naam, SUM(Gewicht * Aantal)
FROM bestelling, artikel
WHERE artikel.ArtNr = bestelling.ArtNr
GROUP BY Naam;
```

Resultaat:

Naam	SUM OF Gewicht * Aantal
as	1900
bout	17000
moer	7200
schroef	13800
veer	6000

Ook is het mogelijk om gegevens te verkrijgen uit de combinatie van een relatie (table) met zichzelf. Er ontstaat nu een probleem: Hoe moet je dezelfde attributen uit dezelfde tabel onderling onderscheiden? Dit probleem is opgelost door het invoeren van een alias. Deze alias moet aan de tabelnaam in de tabel lijst van het `SELECT` statement worden toegevoegd (gescheiden van de tabelnaam met een spatie).

Voorbeeld 29:

Geef alle mogelijke combinaties van 2 artikelen met dezelfde kleur.

Oplossing:

```
SELECT x.Naam, y.Naam
FROM artikel x, artikel y
WHERE x.Kleur = y.Kleur AND x.ArtNr < y.ArtNr;
```

De voorwaarde `x.ArtNr < y.ArtNr` is om twee redenen opgenomen:
- artikel combinaties van een artikel met zichzelf worden daardoor uitgesloten.
- artikel combinaties die simpel een verwisseling zijn van een andere combinatie worden niet meegenomen.

Resultaat:

Naam	Naam_1
schroef	veer
moer	schroef
moer	as
schroef	as

11.2 Gegevens combineren door middel van geneste selecties.

Voor sommige vragen is het nodig om geneste selecties toe te passen. Dit kun je doen door in de voorwaarde van de `WHERE` clause van een `SELECT` statement een ander `SELECT` statement op te nemen. Dit kan op 3 manieren:
- Door middel van de `IN` of `NOT IN` operatoren.
- Door middel van de `EXISTS` of `NOT EXISTS` operator.
- Door middel van een vergelijkingsoperator (`=`, `<>`, `<`, `>`, `<=` of `>=`).

Voorbeeld 30:

Geef het ArtNr, de Naam en de Kleur van bestellingen waarvan het Aantal > 300 is.

Oplossing:

```
SELECT ArtNr, Naam, Kleur
FROM artikel
WHERE ArtNr IN (
     SELECT ArtNr
     FROM bestelling
     WHERE Aantal > 300
);
```

Resultaat:

ArtNr	Naam	Kleur
2	bout	groen
3	schroef	blauw
5	veer	blauw

Alternatieve oplossing:

```
SELECT ArtNr, Naam, Kleur
FROM artikel
WHERE EXISTS (
     SELECT *
     FROM bestelling
     WHERE artikel.ArtNr = ArtNr AND Aantal > 300
);
```

Voorbeeld 31:

Geef de nummers en de namen van de leveranciers waarbij ten minste 1 bestelling loopt voor een rood artikel.

Oplossing:

```
SELECT LevNr, Naam
FROM leverancier
WHERE LevNr IN (
     SELECT LevNr
     FROM bestelling
     WHERE ArtNr IN (
          SELECT ArtNr
          FROM artikel
          WHERE Kleur = 'rood'
     )
);
```

Resultaat:

LevNr	Naam
1	Simon
2	Janssen
4	Janssen

Voorbeeld 32:

Geef de nummers van de leveranciers die een kwaliteit hebben die hoger is dan de huidige gemiddelde kwaliteit van de leveranciers.

Oplossing:

```
SELECT LevNr
FROM leverancier
WHERE Kwaliteit > (
     SELECT AVG(Kwaliteit)
     FROM leverancier
);
```

Resultaat:

LevNr
3
5