

van data naar info

Domein H: databases

H1 DATAWERELD

H2 SQL-DATABASES

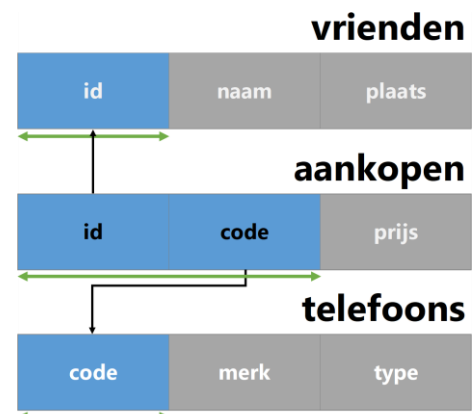
2.1 Inleiding

In hoofdstuk 1 heb je kennis kunnen maken met SQL-databases: **relationele databases** op basis van één of meerdere tabellen met rijen of **records** met dataelementen die een relatie met elkaar hebben.

In figuur 2.1 zie je een **strokendiagram** van een relationele database op basis van de casus *mobiel*:

Alan is geïnteresseerd in smartphones. Hij heeft een kleine database gemaakt waarin hij kan bijhouden welke telefoons er op de markt verschijnen. Van deze telefoons wil hij het merk en type bijhouden.

Omdat hij wil bijhouden wie van zijn vrienden welke telefoon heeft, houdt hij alle aankopen bij met de op dat moment betaalde prijs. Van zijn vrienden noteert hij alleen de naam en de woonplaats.



FIGUUR 2.1

In dit hoofdstuk komt een aantal databases vaker terug in de oefeningen. De database *mobiel* hoort daar bij. Hieronder zie je de drie tabellen met alle records:

vrienden			telefoons			aankopen		
id	naam	plaats	code	merk	type	id	code	prijs
1	Alan	Groningen	a1	Apple	iPhone 13	1	a1	819
2	Bob	Assen	a2	Apple	iPhone 14	6	a1	849
3	Christel	Heerlen	a3	Apple	iPhone SE	4	a2	929
4	Daphne	Groningen	g1	Google	Pixel 6a	8	g2	589
5	Eve	Heerlen	g2	Google	Pixel 7	9	o1	869
6	Frits	Delft	o1	OnePlus	11	2	s1	319
7	Gonny	Groningen	s1	Samsung	Galaxy A23	3	s1	299
8	Hajar	Emmen	s2	Samsung	Galaxy S23	8	s1	329
9	Ingo	Assen	s3	Samsung	Z flip 4	9	s3	929
			x1	Xiaomi	12 Lite	7	x1	415
			x2	Xiaomi	12X	8	x2	499

FIGUUR 2.2

Als in een opgave wordt verwezen naar de casus *mobiel*, dan kun je deze bladzijde gebruiken bij het beantwoorden van vragen. De andere databases uit dit hoofdstuk worden beschreven in de opgave zelf of in een bijlage.

LET OP: in dit hoofdstuk staat niet alle theorie binnen de paragrafen uitgelegd. Je doet ook kennis en vaardigheid op door de opgaven te maken.

2.2 Een database maken met SQL

SQL staat voor *Structured Query Language*. Het is een standaardtaal die gebruikt wordt in een databasemanagementsysteem voor relationele databases. De SQL-commando's zijn op te delen in meerdere categorieën. In dit hoofdstuk behandelen we:

- DDL (*Data Definition Language*) voor het maken van databases en tabellen
- DML (*Data Manipulation Language*) voor het toevoegen, bewerken en verwijderen van records
- DQL (*Data Query Language*) voor het opvragen van data via zoekopdrachten

Het inrichten van een database binnen het databasemanagementsysteem begint bij het maken van een nieuwe, lege database. Voor de casus *mobiel* uit de inleiding hoort hierbij het **DDL**-commando *CREATE*:

```
CREATE DATABASE `mobiel`;           // maak een nieuwe database met de naam mobiel
```

Met onderstaande instructieregels wordt nu de tabel *vrienden* toegevoegd aan de database:

```
USE `mobiel`;                       // gebruik de database mobiel
CREATE TABLE `vrienden` (          // maak een tabel met de naam vrienden
    `id` int AUTO_INCREMENT PRIMARY KEY, // maak een primaire sleutel met de naam id
    `naam` text NOT NULL,           // maak een attribuut met de naam naam
    `plaats` text NOT NULL,         // maak een attribuut met de naam plaats
);
```

In bovenstaande SQL-code is te zien dat aan elk attribuut een **datatype** wordt meegegeven. Dat is verplicht. Het type *int* staat voor integer (geheel getal) en *text* voor tekst. Met *varchar(16)* maak je een attribuut dat kan bestaan uit cijfers, letters en andere symbolen met een maximale lengte die je zelf kunt kiezen (in dit geval 16).

De (niet noodzakelijke) toevoeging *NOT NULL* betekent dat het veld niet leeg mag blijven als records worden toegevoegd. Het databasemanagementsysteem controleert daar dan op. Het toevoegen van een primaire sleutel gaat met *PRIMARY KEY*. De bedoeling is dat het veld *id* niet alleen uniek is, maar ook dat de nummering bij het toevoegen van een nieuw record automatisch wordt verhoogd. Met *AUTO_INCREMENT* kan het databasemanagementsysteem dit zelf bijhouden.

Om de relaties tussen de tabellen te beschrijven, gebruiken we ook referentiesleutels. Hoe je dat doet leer je in een opgave in deze paragraaf. Het databasemanagementsysteem bewaakt dan de referentiële integriteit als records worden verwijderd, toegevoegd of gewijzigd.

Het toevoegen van records kan met het **DML**-commando *INSERT*:

```
INSERT INTO `vrienden` (`id`, `naam`, `plaats`) VALUES
(1, 'Alan', 'Groningen'),
(2, 'Bob', 'Assen');
```

Met bovenstaande SQL-code worden twee van de records in figuur 2.2 gemaakt. Merk op dat er voor het attribuut *id* geen aanhalingstekens worden gebruikt bij de ingevoerde waarden (*values*). Bij een getal doe je dat niet, bij een ander datatype wel. Natuurlijk kun je niet alleen data toevoegen, maar zijn er ook DML-commando's om data aan te passen of te verwijderen. Dat leer je bij het maken van de opdrachten.

Het belangrijkste **DQL**-commando is *SELECT*, waarover je juist meer in de volgende paragrafen leert:

```
SELECT *                          // selecteer alle attributen van alle records
FROM `aankopen`                  // uit de tabel aankopen
```

Opdracht 1 DDL: een database met tabellen maken

In de inleiding van dit hoofdstuk staat de casus *mobiel* beschreven. In de theorie kun je lezen hoe met SQL een database *mobiel* kan worden gemaakt met een tabel *vrienden*.

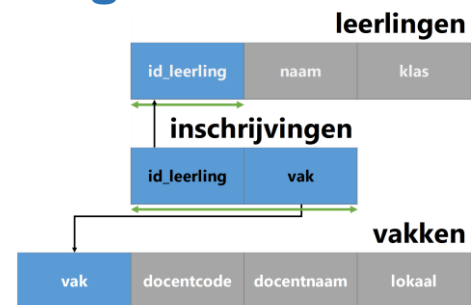
1. Geef de SQL-code waarmee de tabel *telefoons* wordt gemaakt (zie figuur 2.1). Zorg dat de attributen *code* en *type* het datatype *varchar* krijgen met maximaal drie respectievelijk 16 tekens. Het veld *merk* krijgt het tekstformaat.
2. Voeg SQL-code toe waarmee de eerste twee records van de tabel *telefoons* worden toegevoegd.

De database *mobiel* is onderdeel van de online omgeving. We kunnen daarom het SQL-bestand waarmee de volledige database is gemaakt bekijken.

3. Open *mobiel.sql* (in de map *databases*) in je editor en controleer jouw antwoorden op bovenstaande twee vragen.
4. De tabel *aankopen* bevat een samengestelde sleutel. Met welke SQL-regel is deze gemaakt?
5. In figuur 2.1 zie je twee referentiesleutels (*foreign key*). Met welke SQL-regels zijn die gemaakt?

Opdracht 2 DDL & DML: examentraining

In figuur 2.3 zie je het eindresultaat van de database *examentraining* volgens het strokendiagram van hoofdstuk 1.



FIGUUR 2.3

6. Open het bestand *examentraining.sql* aan in de map *databases* van de werkomgeving.
7. Voeg een coderegel toe waarmee de database *examentraining* kan worden gecreëerd.
8. Gebruik de volgende regels om in de database *examentraining* de tabel *vakken* aan te maken:

```
USE `examentraining`;  
CREATE TABLE `vakken` (  
    `vak` varchar(16) PRIMARY KEY,  
    `docentcode` varchar(3) NOT NULL,  
    `docentnaam` text NOT NULL,  
    `lokaal` tekst  
);
```

9. Voeg regels toe, zodat de tabel *leerlingen* wordt gemaakt. Bedenk zelf voor elk veld het juiste datatype. Vergeet niet op de primaire sleutel toe te kennen.
10. Ga naar de startpagina van de werkomgeving en kies bij databasebeheer voor *Importeren*.
11. Ga nu naar de *Querier*. Controleer of de database *examentraining* in de lijst staat.
12. Selecteer de database en controleer of de tabellen *vakken* en *leerlingen* correct worden getoond.
13. Voeg SQL-coderegels toe om de tabel *inschrijvingen* te maken. Kies daarna opnieuw voor *Importeren* en controleer of nu alle drie de tabellen correct worden weergegeven.

Stel dat jij je voor twee vakken hebt ingeschreven voor de examentraining bij jou op school.

14. Voeg SQL-regels (DML) toe aan *examentraining.sql* waarmee alle benodigde data aan de drie tabellen wordt toegevoegd.
15. Ga naar de *Querier*. Vul bij de *Invoer* de query `SELECT * FROM vakken` in en klik op *Uitvoeren*.
16. Gebruik query's om ook de inhoud van de andere twee tabellen op te vragen met de *Querier*.
17. Voer in: `DESC inschrijvingen`. Wat is het resultaat?

Opdracht 3 DML: data manipuleren

In de theorie heb je kennis gemaakt met *INSERT*, waarmee je data in een tabel kunt invoegen. In deze opgave leer je om aanpassingen te doen aan data in de database, of gegevens te verwijderen.

18. Open de database *mobiel* in de *Querier*.
19. De *Querier* beschermt de database tegen het per ongeluk wijzigen of verwijderen van data en databases. Zorg dat het vinkje voor *Alleen lezen* uitstaat, zodat we wijzigingen kunnen aanbrengen.

In de theorie wordt een *INSERT*-opdracht getoond, waarbij Alan en Bob een *id* met een waarde meekrijgen. Echter, doordat bij de definitie van de tabel gebruik gemaakt is van *AUTO_INCREMENT*, is het niet nodig om dit nummer mee te geven. In plaats daarvan mag je *null* invullen.

20. Wat is de betekenis van *null*? Gebruik eventueel hoofdstuk 1 of het internet.
21. Voeg jezelf toe aan de database *vrienden*. Vul bij het veld *id* geen waarde in, maar gebruik *null*.
22. Gebruik *SELECT* om de inhoud van de tabel *vrienden* te tonen. Welk *id* heb je gekregen?
23. Voeg jouw mobiele telefoon toe aan de tabel *telefoons* en controleer met *SELECT* of dat gelukt is.

Je hebt nu jezelf en jouw telefoon toegevoegd aan de database. Maar de informatie dat jij de eigenaar bent van die telefoon ontbreekt nog.

24. Voeg deze ontbrekende data toe. Controleer of de data goed is toegevoegd aan de tabel.

Je hebt nu data toegevoegd aan de database. Maar wat nu als je een foutje had gemaakt? Ook daarvoor bestaan DML-commando's. Hieronder zie je twee regels voor het wijzigen respectievelijk verwijderen van data:

```
UPDATE vrienden SET naam = 'Jos', plaats = 'Amersfoort' WHERE id = 10
```

```
DELETE FROM aankopen WHERE id = 10
```

MERK OP: tot nu toe hebben we tabel- en veldnamen altijd met aanhalingstekens zoals ``telefoons`` geschreven. Hoewel dit gebruikelijk is, is het niet perse noodzakelijk. Voor het gemak laten we het van nu af aan dan ook weg. De aanhalingstekens bij teksten zoals `'Amersfoort'` moeten wel blijven!

25. Hierboven staan twee SQL-regels. Leg voor beide in je eigen woorden uit wat ze betekenen.
26. Voer de *UPDATE*-regel uit en controleer of het resultaat overeenkomt met jouw uitleg.
27. Voer de *DELETE*-regel uit. Controleer ook nu of het overeenkomt met jouw voorspelling.
28. Alan is verhuisd van Groningen naar Zwolle. Verwerk dit met behulp van een SQL-opdracht.
29. Verwijder alle records uit de tabel *aankopen* met een prijs onder de 500 euro.

Je hebt nu veel wijzigingen aangebracht in de database *mobiel*. Gelukkig kunnen we de database in zijn oorspronkelijke staat herstellen.

30. Ga naar de startpagina van de werkomgeving en klik op *Herstel standaard databases*.

★ Opdracht 4 DDL: de databasestructuur bewerken

In de vorige opgave werd data bewerkt, maar uiteraard kun je ook de tabellen van de database zelf bewerken. Daarmee definieer je de database (en dus ook de records die je erin kunt plaatsen): DDL.

Gebruik het internet (b.v. [w3schools](http://w3schools.com)) om zelf uit te zoeken hoe je de volgende opdrachten uit kunt voeren:

31. Voeg het attribuut *werkgeheugen* toe aan de tabel *telefoons* met datatype *int*. Het veld mag leeg blijven.
32. Verander het attribuut *naam* in de tabel *vrienden* naar *voornaam*.
33. Verwijder alle records (met één instructieregel) uit de tabel *aankopen*.
HINT: gebruik *TRUNCATE*.
34. Ga naar de startpagina van de werkomgeving en klik op *Herstel standaard databases*.

2.3 Data opvragen uit de database: query's

In de vorige pagina hebben we vooral gekeken naar de vraag hoe je met SQL een database kunt aanmaken, de databasestructuur kunt bewerken en data kunt toevoegen, aanpassen of verwijderen. Veel data is opgeslagen in een database, omdat je de data later weer wil gebruiken. Je wil de data dan kunnen opvragen en soms data combineren om tot nieuwe kennis te komen. Daarom zijn de komende paragrafen gewijd aan **DQL**-zoekopdrachten, zoals het voorbeeld uit de vorige paragraaf:

```
SELECT *                // selecteer alle attributen van de records
FROM aankopen           // uit de tabel aankopen
```

Als bovenstaande zoekopdracht of **query** wordt uitgevoerd, wordt de volledige tabel *aankopen* getoond. Dat is niet altijd praktisch. Vaak wil je een beperkt aantal attributen laten zien van een beperkt aantal **records**. Je stelt dan een eis *waaraan* de getoonde records moeten voldoen met de toevoeging **WHERE**:

```
SELECT id, naam          // maak een resultaat tabel met de kolomnamen id en naam
FROM vrienden            // uit de tabel vrienden
WHERE plaats = 'Groningen' // met records waarvoor geldt dat de plaats Groningen is
```

De combinatie van **SELECT**, **FROM** en **WHERE** is tamelijk standaard bij het opstellen van een query. Voor de overzichtelijkheid zetten we de drie delen hier op drie aparte regels, maar dat is niet perse noodzakelijk.

De voorbeeldquery hierboven bevat een enkele eis. Met **logische operatoren** zoals **AND**, **OR** en **NOT** kun je meerdere eisen met elkaar combineren, zoals met de volgende query

```
SELECT type
FROM telefoons
WHERE merk = 'Xiaomi' OR merk = 'Google'
```

waarmee een overzicht wordt gemaakt van alle types telefoons in de database die van het merk Xiaomi of Google zijn. In deze paragraaf staan opdrachten om hier meer mee te oefenen.

Opmerkingen tot slot: bij de komende opdrachten gebruiken we steeds de *Querier* tenzij anders aangegeven. Dit niet steeds opnieuw bij de opdracht vermeld.

We kiezen ervoor om alle SQL-commando's met hoofdletters te schrijven, zodat het onderscheid tussen SQL en tabelnamen en -attributen nog duidelijk wordt, maar kleine letters mag ook!

Opdracht 5 mobiel: basisquery's

Bij deze opgave gebruiken we de database *mobiel* en bekijken we de volgende query's:

- `SELECT naam FROM vrienden WHERE plaats = 'Assen'`
- `SELECT code, type FROM telefoons WHERE merk = 'OnePlus' OR merk = 'Xiaomi'`
- `SELECT code, type FROM telefoons WHERE merk = 'OnePlus' AND merk = 'Xiaomi'`
- `SELECT naam FROM vrienden WHERE plaats = 'Groningen' AND id > 4`
- `SELECT naam FROM vrienden WHERE plaats = 'Groningen' AND id <= 4`
- `SELECT naam FROM vrienden WHERE plaats = 'Groningen' AND NOT id < 4`
- `SELECT id, code FROM aankopen WHERE prijs IS NULL`

35. Beschrijf in een Nederlandse zin voor elk van de query's welke vraag ze aan de database stellen.
36. Gebruik de tabellen in figuur 2.2 om (dus zonder *Querier*) uit te zoeken wat de uitkomst is.
37. Controleer jouw antwoord op de vorige vraag door de query's via de *Querier* uit te voeren.

Opdracht 6 top 2000: basisquery's

In deze opdracht maken we voor het eerst kennis met de top-2000 database. Hiervan bestaan twee versies. In deze opgave werken we met versie 1. Tenzij anders vermeld is het de bedoeling dat je een query gebruikt om de vraag te beantwoorden. Als antwoord noteer je uiteindelijk de gebruikte query.

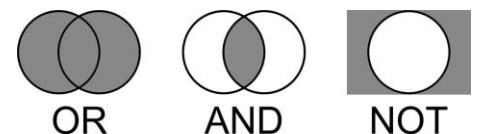
38. Lees de beschrijving van de database *top 2000 versie 1* in de bijlage.
39. Selecteer *top 2000 v1* in de *Querier* en bekijk de inhoud van de database door de query
`SELECT * FROM top2000` uit te voeren. Uit hoeveel records bestaat deze database?
40. Selecteer de titel en de naam van de artiest van het nummer dat in 2020 op positie 1 stond.
41. Welke nummers van Adele (figuur 2.4) die gemaakt zijn in 2011 hebben een notering gehad in de top 2000? Zorg dat in de resultaattabel alleen de titel en het jaar van uitgave te zien zijn.
42. Selecteer alle artiestennamen die een track met de titel *Sorry* in de top 2000 hebben gehad.
43. Gebruik query *g* van de vorige opgave om alleen die artiesten te selecteren die wel een liedje met de titel *Sorry* hebben, maar in de editie van 2018 niet in de top 2000 stonden.
44. Keer de selectie van de vorige vraag om: welke artiesten stonden juist wel in 2018 in de lijst met een nummer dat *Sorry* heet?



FIGUUR 2.4

Opdracht 7 top 2000: logische operatoren

In de theorie en de vorige opdrachten heb je al gebruik moeten maken van *AND*, *OR* en *NOT* om een combinatie van eisen aan de data die je selecteert. In figuur 2.5 is de betekenis van deze operatoren met Venndiagrammen weergegeven.



FIGUUR 2.5

Naast de hierboven genoemde woorden, heb je in de theorie kunnen zien dat je ook gebruik kunt maken van wiskunde tekens zoals $>$ (groter dan), \leq (kleiner dan of gelijk aan) en andere varianten hierop.

45. Beschrijf in woorden wat in algemene zin (dus geen records benoemen) de uitkomst is van de query
`SELECT titel, artiest FROM top2000 WHERE ed2021 <= 10 AND ed2020 > 10`
46. Controleer jouw beschrijving door de query uit te voeren.
47. De query
`SELECT titel, artiest FROM top2000 WHERE ed2019 <= 10 OR ed2018 > 10`
zorgt voor een resultaat tabel met 2000 records. Controleer dit en leg vervolgens uit of dit aantal vooraf te voorspellen zou zijn geweest.
48. Selecteer alle records met liedjes van voor 1970 die in de editie van 2019 wel in de top 2000 stonden maar een jaar later niet.
49. Geef een overzicht van alle nummers van de artiesten *Harry Styles* en *Taylor Swift*.
50. Maak een resultaat tabel met de kolommen *artiest*, *titel* en *uit* van de nummers van *Harry Styles* en van *Taylor Swift* die zijn uitgekomen in 2015 of later. Kijk goed naar het resultaat: staan er geen liedjes van voor 2015 in jouw lijst? (Zie eventueel de volgende opdracht.)

Opdracht 8 top 2000: werken met haakjes

Gokje: bij de laatste vraag van opdracht 7 heb je (ongeveer) de volgende query geprobeerd:

```
SELECT artiest, titel, uit FROM top2000
WHERE uit >= 2015 AND artiest = 'Harry Styles' OR artiest = 'Taylor Swift'
```

Als dat zo is, heb je kunnen zien dat het nummer *Shake it Off* van *Taylor Swift* tot de resultaten behoort. Dit nummer is uit 2014 en dat mocht niet! Hoe kan het dat deze toch in de resultaat tabel staat? Het huidige resultaat geeft *alle nummers uit 2015 of later van Harry Styles of nummers van Taylor Swift*. Als we haakjes toevoegen aan de query is dit beter te zien:

```
SELECT artiest, titel, uit FROM top2000
WHERE (uit >= 2015 AND artiest = 'Harry Styles') OR artiest = 'Taylor Swift'
```

Er geldt nu niet voor alle liedjes dat ze uit 2015 of later moeten komen. Dat geldt wel als we zeggen:

```
SELECT artiest, titel, uit FROM top2000
WHERE uit >= 2015 AND (artiest = 'Harry Styles' OR artiest = 'Taylor Swift')
```

51. Voer de laatste query hierboven uit en stel vast dat er nu geen records verschijnen van voor 2015.
52. Geef de titels van alle tracks van de artiest *Queen* die in 2019 of 2020 in de top 50 van de top 2000 stonden.
53. Geef de titel en de positie in de top 2000 van 2018 van alle tracks van *The Beatles* die in 1963, 1964 of 1965 zijn uitgebracht.
54. Geef de titel en de positie in de top 2000 van 2018 van alle tracks van *The Beatles* die niet in 1963, 1964 of 1965 zijn uitgebracht.

★ Opdracht 9 top 2000: LIKE met jokers / wildcards

Tot nu toe hebben we query's gemaakt waarbij we steeds de exacte zoekterm (zoals de artiestennaam) weten. Maar wat nu als je een naam niet exact weet? In dat geval gebruik je de operator *LIKE* in combinatie met de **wildcard**-symbolen `%` of `_`. De twee tekens `%` en `_` worden ook wel **jokers** genoemd.

55. Voer de query `SELECT artiest FROM top2000 WHERE artiest LIKE 'y%'` uit. Gebruik de resultaat tabel om uit te leggen wat de betekening van `LIKE 'y%'` is.
56. Beschrijf in je eigen woorden wat de uitkomst van de query `SELECT artiest FROM top2000 WHERE artiest LIKE '%y%'` zal zijn. Voer de query uit en controleer je beschrijving.
57. Hoeveel liedjes met de term *Christmas* in de titel staan er in de database?
58. Zijn er records waarvoor geldt dat zowel de artiestennaam als de titel de letter *q* bevat? Zo ja, hoeveel?
59. Voer de query `SELECT artiest FROM top2000 WHERE artiest LIKE '__u_'` uit. Gebruik de resultaat tabel om uit te leggen wat het verschil is tussen `%` en `_`.
60. Vervang het gedeelte na `LIKE` uit de vorige vraag door `'__u%'`. Wat is het resultaat?
61. Hoeveel artiestennamen met precies twee symbolen staan er in de database?
62. Khalid is fan van *Ed Sheeran* en vraagt zich af of er meer artiesten zijn met als voornaam *Ed*. Hij gebruikt `'Ed%'`. Leg uit dat dit niet tot het gewenste resultaat leidt. Hoe los je dit probleem op?
63. Met welke collega artiesten heeft de artiest *Snelle* (figuur 2.6) nummers in de top 2000 staan?



FIGUUR 2.6