van data naar info

Domein H: databases

INHOUD

**“A database administrator walks into a NoSQL bar,
but he turns and leaves because he can’t find a table.”**

— ERLEND OFTEDAL

**“Smart data structures and dumb code work
a lot better than the other way around.”**

— ERIC S. RAYMOND

**“About 15% of all Google search queries are queries that no one
has ever searched before. That means every day, humans around the world
are googling millions of unprecedented queries.”**

— QUINCY LARSON

“Don’t ask SQL developers to help you move furniture. They drop tables.”

— CARLA NOTAROBOT (TWITTER ACCOUNT)

**“The word SEQUEL turned out to be somebody’s trademark.
So I took all the vowels out of it and turned it into SQL.
That didn’t do too much damage to the acronym.
It could still be the Structured Query Language.**

— DONALD CHAMBERLIN (IBM | MEDEBEDENKER VAN SQL)

H1 DATAWERELD

1.1 Inleiding

In 2019 meldde Google dat het 1,1 miljard euro wilde investeren in nieuwe datacenters in Nederland zoals in de Eemshaven (Groningen; zie figuur 1.1). Hiermee wordt de totale investering van alleen Google al twee-en-half miljard! Wat wordt er in deze datacenters bewaard? En hoe wordt het bewaard? En waarom heeft het zoveel waarde?

FIGUUR 1.1

Dit zijn vraagstukken die we in deze module gaan behandelen, beginnend met de basisvraag: wat is **data**? Data zijn *ruwe* gegevens, zoals die bijvoorbeeld rechtstreeks uit een temperatuursensor komen. Een verzameling van data of gegevens wordt ook wel een **dataset** genoemd.

In figuur 1.2A zie je een dataset. Het probleem met deze data is, dat de gegevens onbruikbaar zijn, omdat ze niet interpreteerbaar zijn: je weet niet wat ze voorstellen. Gaat het hier b.v. om plekken op aarde? Er is geen context of betekenis, waardoor de data geen waarde heeft.

Peru	205	133	63
Sienna	160	82	45
Tan	210	180	140

A

kleur	R	G	B
Peru	205	133	63
Sienna	160	82	45
Tan	210	180	140

B

FIGUUR 1.2

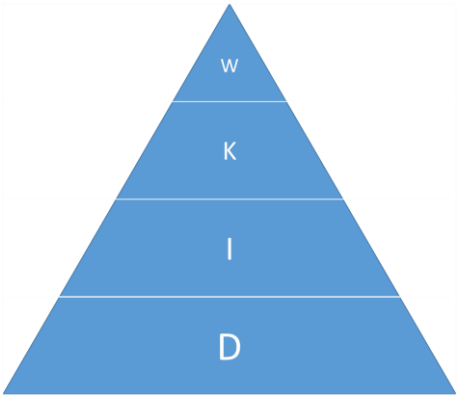
Om de data betekenis te geven, is meer data nodig. In figuur 1.2B is betekenis aan de originele ruwe data gegeven door **metadata** toe te voegen: extra data om andere data te beschrijven. De gegevens zijn nu herkenbaar geworden: er is sprake van **informatie**. Het zijn kleurnamen met hun RGB-kleurcode (rood, groen en blauw).

Google levert informatie via de bekende zoekmachine op basis van heel veel data. Met informatie kun je nader onderzoek doen, in de vorm van analyses. Door verschillende informatieonderdelen met elkaar te verbinden of patronen te ontdekken, ontstaat een nieuw begrip en inzicht in een onderwerp: er ontstaat **kennis**. Kennis is het begrijpen van de informatie. Overigens bestaan er meerdere definities van kennis. Soms wordt aan de reeks data, informatie en kennis nog **wijsheid** toegevoegd. Informatie is beschrijvend (Engels: *what is?*), kennis is vooral gericht op de informatie die we al hebben (verklarend: Engels: *why is?*) en wijsheid is gericht op verwachtingen en nieuwe keuzes (op basis van ervaring en patronen: *why do?*).

Opdracht 1 DIKW-piramide

Met betrekking tot de theorie, spreekt men over de **DIKW-piramide**.

1. Waar staan de letters van de afkorting DIKW voor?
2. Op het internet kun je plaatjes vinden van de DIKW-piramide, vergelijkbaar met figuur 1.3. Leg uit waarom hiervoor een piramidevorm wordt gebruikt.
3. Wat is het verschil tussen data en informatie?
4. Noem vijf bedrijven die data van jou in hun bezit hebben.
5. Op basis van vraag 4: welke informatie kunnen die bedrijven hieruit afleiden (Denk ook aan combinaties van data)?
6. Stel dat jij een ruim budget had om een grote dataset te kopen. Welke data zou je willen hebben? En met welk doel?



FIGUUR 1.3

Opdracht 2 Wat doen we met al die data?

Bekijk de video waarin, op licht-filosofische wijze, wordt gesproken over het gebruik van data, interpretatie en de wijsheid die daar uit zou kunnen volgen. De lezing bevat de nodige *oneliners*.

7. Noteer de drie uitspraken of zinnen die je het meest interessant of treffend vindt.

1.2 Data bewaren

Deze paragraaf gaat niet over het bewaren van data op een harde schijf of SD-kaart. Het gaat hier niet om de techniek van de fysieke opslag, maar om welke indeling of **datastructuur** je kunt gebruiken om vergelijkbare data altijd op dezelfde manier op te slaan. De centrale vraag daarbij is: *Wat wil je met de opgeslagen informatie doen?* Want bewaren is meestal geen doel op zich en soms verboden.

kleur	R	G	B
Peru	205	133	63
Sienna	160	82	45
Tan	210	180	140

FIGUUR 1.4

Er bestaat niet één datastructuur voor de opslag van data. Daarom kijken we in dit hoofdstuk naar meerdere aanpakken (of **paradigma**'s). De keuze voor een paradigma hangt in de praktijk af van vragen als:

- Hoeveel data is er of wordt er verwacht?
- Hoe vaak verandert de dataset? (b.v. veel lezen of schrijven?)
- Voor wie moet de data beschikbaar zijn?
- Welk doel heeft de opslag? Wat wordt er straks mee gedaan? (Wordt er bijvoorbeeld straks statistiek mee bedreven?)

kleur	Peru
R	205
G	133
B	63

kleur	Sienna
R	160
G	82
B	45

kleur	Tan
R	210
G	180
B	140

FIGUUR 1.5

In figuur 1.4 zie je nogmaals de data uit figuur 1.2. De data wordt weergegeven in een **tabel**. Het betreft een tabel met drie kleuren, waarbij aangegeven is wat de kleurnaam (*kleur*) is en met hoeveel rood, groen en blauw (de primaire keuren uit hun RGB-code) de kleur is opgebouwd. Rijen in dit soort tabellen, worden **records** genoemd.

In figuur 1.5 staat dezelfde data niet in een tabel, maar in de vorm van objecten. De term **object** ken je misschien al van object-georiënteerd programmeren. Vaak wordt één unieke eigenschap van het object gebruikt om het object te identificeren. In dit geval is dit de eigenschap of het **attribuut** *kleur*.

Bij de programmeerlessen heb je vast ook al kennis gemaakt met de **variabele** en de **array** of lijst. Het uitvoeren van programmeercode met daarin een variabele, array (lijst) of object zorgt voor de opslag van data in het werkgeheugen van de computer. Als je een hoeveelheid data langer wil bewaren in een extern geheugen zoals een harde schijf, dan gebruik je daarvoor een **bestand** of een **database**. Deze module gaat alleen over databases.

Databases zijn er in verschillende soorten, afhankelijk van de gekozen opslagstructuur ofwel het gekozen **databasemodel** (zie § 1.3). Databases die werken met tabellen zoals in figuur 1.4 heten **relationele databases**, omdat elk record bestaat uit datapunten die een **relatie** met elkaar hebben (bij elkaar horen).

Hoofdstuk 2 is volledig gewijd aan relationele databases. Daarnaast bestaan er databases waarin data op een andere manier wordt opgeslagen, zoals met de objecten in figuur 1.5. Deze niet-relationele databases heten in het algemeen **noSQL**-databases.

Opdracht 3 data in een object

In figuur 1.6 zie je nogmaals de data uit figuur 1.5. Aan de data is de bijbehorende hexadecimale kleurcode toegevoegd, die je misschien kent van het maken van websites. Deze hex-code kun je zelf afleiden (als je weet hoe) uit de gegeven hoeveelheden rood, groen en blauw.

Peru	
R	205
G	133
B	63
hex	CD853F

Sienna	
R	160
G	82
B	45
hex	A0522D

Tan	
R	210
G	180
B	140
hex	D2B48C

FIGUUR 1.6

8. Noem een argument om de hex-code niet in een database op te slaan en een argument om dit wel te doen.
9. In figuur 1.6 is behalve de hex-code nog meer informatie toegevoegd. Welke informatie is dat?
10. Welke metadata zie je in figuur 1.6?
11. Welke metadata zie je wel in figuur 1.4 maar niet in figuur 1.6?
12. Gebruik het internet om informatie te vinden over de kleur *Gainsboro*. Maak hiermee zowel een object als een record.

Opdracht 4 XML I: een dataset beschrijven

In figuur 1.6 is de data (en de metadata) gerepresenteerd als een plaatje (met tekst) van drie losse objecten. Een manier om data *echt* op te slaan, is door het als platte tekst te bewaren met een vaste structuur. Dit kan worden gedaan met behulp van **XML**.

In figuur 1.7 zie je een voorbeeld van een bestand in XML-formaat. Het zijn gegevens van twee dieren uit de praktijk van een dierenarts.

13. Waar staat XML voor?
14. Hoe kun je zien dat het om twee dieren gaat?
15. Op welke manier is ervoor gezorgd dat duidelijk is welke data bij welk dier hoort?
16. XML wordt zelfbeschrijvend (Engels: *self-descriptive*) genoemd. Wat wordt daarmee bedoeld?

```
<praktijk>
    <dier>
        <naam>Buck</naam>
        <klant>Al Bundy</klant>
        <soort>hond</soort>
        <geslacht>m</geslacht>
        <geboren>1983</geboren>
    </dier>
    <dier>
        <naam>Illbatting</naam>
        <klant>Pippi</klant>
        <soort>paard</soort>
        <geslacht>m</geslacht>
        <geboren>1961</geboren>
    </dier>
</praktijk>
```

FIGUUR 1.7

Opdracht 5 XML II: de structuur van XML

In deze opgave gaan we een XML-bestand bestuderen en aanpassen.

17. Open het XML-bestand *H1O05_dierenarts_1.0.xml* in je browser. Jouw browser geeft de data weer met behulp van de symbolen ► en ▼. Klik op deze symbolen: wat zie je?

Data wordt in XML beschreven met **elementen** zoals `<naam>Buck</naam>` die zijn gemaakt met een *start-tag* (`<naam>`), een *eind-tag* (`</naam>`) en de data (**Buck**).

Boven de data staat in de meeste browsers (zie figuur 1.8):

This XML file does not appear to have any style information associated with it. The document tree is shown below.

18. Wat wordt bedoeld met deze boomstructuur (Engels: *tree*)?
19. Met betrekking tot de structuur van data, kom je vaak de Engelse termen *parent* en *child* tegen. Wat wordt daarmee bedoeld? Gebruik eventueel het internet.
20. Noem een element dat zowel een *parent* als een *child* is.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

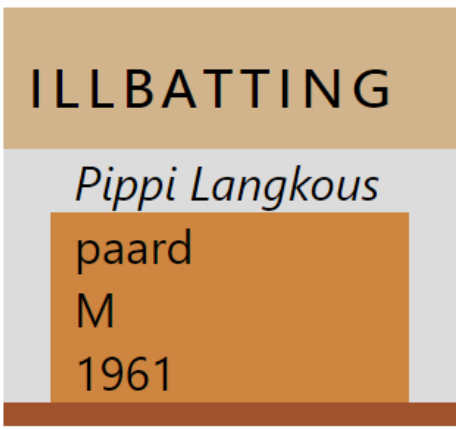
<praktijk>
<dier>
<naam>Buck</naam>
<klant>Al Bundy</klant>
<soort>hond</soort>
<geslacht>m</geslacht>
<geboren>1983</geboren>
</dier>
<dier>
<naam>Illbatting</naam>
<klant>Pippi</klant>
<soort>paard</soort>
<geslacht>m</geslacht>
<geboren>1961</geboren>
</dier>
</praktijk>

FIGUUR 1.8

Opdracht 6 XML III: opmaak

In XML wordt alleen data opgeslagen. Je kunt die data meer leesbaar maken door er vormgeving aan toe te voegen, zoals in figuur 1.9 voor één van de dieren uit de praktijk in figuur 1.7 is gedaan.

21. Open het XML-bestand *H1O06_dierenarts_1.1.xml* in je browser en bekijk het resultaat.
22. Open het bestand in een *editor*: op welke manier is er voor gezorgd dat de browser het bestand met opmaak laat zien?
23. Maak zelf een XML-bestand met de naam *H1O06_kleuren_1.xml* en verwerk hierin de data van figuur 1.4.
24. Voeg nu een bestand toe, zodanig dat jouw eigen xml-bestand met een nette vormgeving wordt weergegeven.



FIGUUR 1.9

Opdracht 7 data in een tabel

In de vorige opdrachten is een kleine dataset beschreven van dieren uit de praktijk van een dierenarts.

25. Maak een tabel van deze dataset, inclusief de gegeven metadata.
26. Wat is een record? Hoeveel records zitten er in deze dataset?
27. Waarom heet een relationele database *relationeel*? Wat is hier de betekenis van dat woord?
28. Leg uit waarom het beter is om het geboortejaar van een dier te registreren dan de leeftijd.
29. Je vindt de dieren in deze database terug op basis van hun naam. Leg uit waarom dit in een grote dierenpraktijk al gauw een probleem wordt.
30. Bedenk een oplossing voor het probleem uit de vorige vraag.

Opdracht 8 meerdere tabellen

In deze opgave bekijken we een uitgebreidere dataset van de administratie van een dierenarts in Excel. De dataset bestaat uit drie tabellen: *dieren*, *klanten* en *afspraken* (zie figuur 1.10).
Uit de vraagstelling van de vorige opdracht kun je concluderen dat er aan de dataset van figuur 1.7 nog wel zaken te verbeteren zijn.

datum	tijd	dier	notitie
3-01-22	10:00	2	eet slecht
3-01-22	11:00	9	
4-01-22	13:30	4	controle
4-01-22	10:30	6	castratie
4-01-22	13:30	1	sterilisatie
4-01-22	14:30	2	diabetes
7-01-22	10:00	4	

FIGUUR 1.10

31. Open het Excel-bestand *H1O08_dierenarts_2.0.xlsx*. Bekijk de inhoud van de drie tabbladen *dieren*, *klanten* en *afspraken*.
32. Hoe is het probleem waar vraag 29 naar verwijst hier opgelost?
33. In de tabel *afspraken* zie je in het eerste record in de kolom dier het nummer 2 staan. Hoe heet dat dier?
34. Wat is de woonplaats van dit dier?
35. In vraag 27 werd gevraagd waarom een relationele database zo heet. Leg met vraag 33 en 34 uit dat er nog een manier is waarop er een relatie in een relationele database kan zijn.

De manier van opslaan zorgt ervoor dat je vragen kunt beantwoorden op basis van data die in verschillende tabellen staan. Zo'n vraag is eigenlijk een zoekopdracht. Beantwoord de volgende vragen:

FIGUUR 1.11 ANGEL DE ENGELVIS

36. Welk dier is het jongst?
37. Welk levend dier is het oudst?
38. Hoeveel katten staan er in de database?
39. Hoeveel katten horen bij een klant uit Groningen?
40. Zijn er klanten die nog nooit een afspraak hebben gemaakt? Zo ja, wat is of zijn hun namen?

Opdracht 9 kattendata

De kunstenaar Owen Mundy heeft de website *I know where your cat lives* gemaakt, om te laten zien hoe eenvoudig het is om gegevens te verzamelen die mensen al dan niet bewust op internet achterlaten.

41. Klik op het icoontje hiernaast om naar de website te gaan.
42. Kun je een kat vinden in jouw buurt of van iemand die je kent?
43. Welke data is nodig om deze website te maken?

Mensen plaatsen vaak foto's van hun huisdieren op het internet met daarbij de naam van het dier.

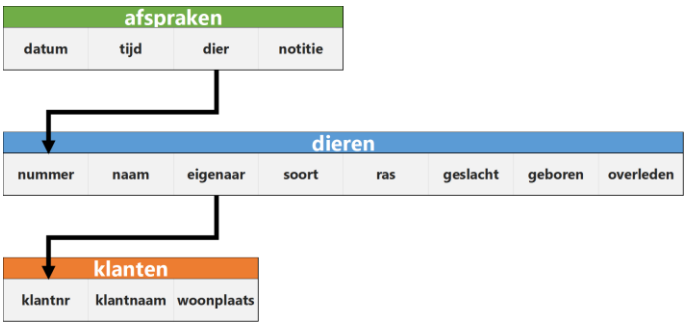
FIGUUR 1.12 BRIECK DE KAT

44. Leg uit hoe je met behulp van een foto de locatie (soms) kunt achterhalen.
45. Leg uit dat de gevonden locatie niet altijd klopt.

1.3 Databasemodellen

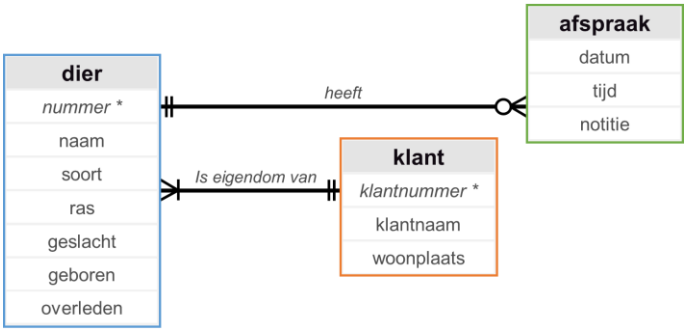
In de vorige paragraaf hebben we twee databasemodellen of -paradigma's gezien om data gestructureerd op te slaan. Als je een nieuwe database maakt, is de keuze voor een databasemodel zoals een **tabel** de eerste stap. Maar binnen dat databasemodel of -paradigma moet je nog steeds zelf structuur aanbrengen.

In figuur 1.13 is de database van de dierenartsenpraktijk uit opdracht 8 schematisch weergegeven in een **strokendiagram**. Merk op dat er geen data te zien is, maar alleen metadata. Het diagram toont de structuur waarbinnen de data wordt opgeslagen. Alle tabellen worden beschreven met daarbij de attributen van de datapunten van de recor ds in die tabel. De zwarte pijlen in figuur 1.13 tonen de relaties tussen de tabellen van de database. Deze heten **referenties**.



FIGUUR 1.13

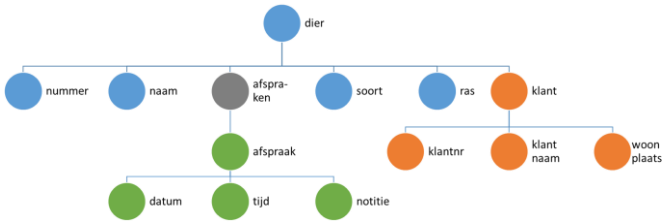
Als we dezelfde data van de dierenartsenpraktijk niet met tabellen maar met het **objectmodel** willen beschrijven, ontstaat een diagram zoals in figuur 1.14. De drie objecten worden beschreven op basis van een klasses met onderlinge relaties. Op de gebruikte symbolen bij die relaties komen we terug.



FIGUUR 1.14

De gegevens van een dier uit de objectdatabase vind je in figuur 1.14 terug door gebruik te maken van het attribuut *nummer*. Het *nummer* geeft je de toegang tot de bijbehorende data van het dier en wordt daarom de **primaire sleutel** genoemd. Een dier terugvinden lukt alleen als dit nummer **uniek** is. We zeggen dan dat voor het attribuut *nummer* **uniciteit** geldt.

In figuur 1.13 zien we een strokendiagram met drie tabellen. In het objectdiagram van figuur 1.14 zien we een diagram met drie objecten. Als we het dier centraal stellen, kunnen we zeggen dat een dier in de administratie van de dierenarts eigenschappen heeft, waaronder mogelijke afspraken en een eigenaar (klant). De klant heeft (net als b.v. een afspraak) zelf ook weer eigenschappen.



FIGUUR 1.15

In figuur 1.15 is dit getekend als een **boomstructuur** met meerdere vertakkingen die de hiërarchie tussen de elementen laat zien. Het beschrijven van data met een boomstructuur is een specifiek voorbeeld van het databasemodel **graaf** dat we in de opgaven verder zullen bekijken. In een boomstructuur of **boom** heten de elementen **datapunten** of **knopen**. Knopen zijn met elkaar verbonden door **takken**. Het bovenste datapunt heet de **wortel** van de boom: in figuur 1.15 is dat *dier*.

Bij het beschrijven van een graaf worden ook in Nederland vaak Engelse termen gebruikt. Een boom is een *tree* met *branches* (vertakkingen) die begint bij de *root*. Een element is een *node*. Een *node* die zich lager in de *tree* bevindt, heet een *child*: *klant* is een *child* van *dier*. Andersom is *dier* de *parent* van *klant*. Een link tussen twee *nodes* wordt behalve een *branch* ook een *edge* genoemd.

Opdracht 10 modellen voor een kleurendatabase

In deze opdracht kijken we naar een dataset met kleurennamen. Daarbij kijken we in eerste instantie naar een database op basis van een tabel.

46. Open het Excel-bestand *H1O10_kleuren_1.xlsx* en bekijk de data op het tabblad *kleuren*.
47. Zijn er attributen in de tabel waarvoor uniciteit geldt? Zo ja: welke?
48. Bekijk nu de data op het tabblad *kleuren_uitgebreid*. Zijn hier attributen met uniciteit?

Het attribuut *hex* op het tabblad *kleuren_uitgebreid* is berekend met de data in de kolommen B, C en D. De data in kolom E heet daarom **afgeleide data**. Om dat te benadrukken hebben we kolom E grijs gemaakt.

49. Noem een voordeel en een nadeel van het toevoegen van afgeleide data aan een database.

De data die je nu in het tabel-formaat ziet, kan ook met het objectmodel worden beschreven.

50. Wat is in die dataset dan de *primary key* of de primaire sleutel?
51. Beschrijf de data van *kleuren_uitgebreid* met het objectmodel, door een ontwerp voor een object te tekenen voor een kleur, vergelijkbaar met de aanpak in figuur 1.14.
52. Beschrijf de data ook met een graaf. Teken een boomstructuur (*tree*), op de manier van figuur 1.15.

Opdracht 11 formats en het ontbreken van data: null

Als je een database wilt maken, kies je eerst voor het databasemodel. Vervolgens beschrijf je de data en hun onderlinge relaties en denk je na over mogelijke beperkingen, zoals het format van de data:

FIGUUR 1.16

53. Leg uit dat data voor het attribuut *postcode* aan hele precieze eisen moet voldoen.
54. Noem nog drie voorbeelden van attributen waarvoor het verstandig is de inhoud qua format te beperken.
55. Leg uit dat een attribuut dat als primaire sleutel dient altijd een waarde moet hebben.

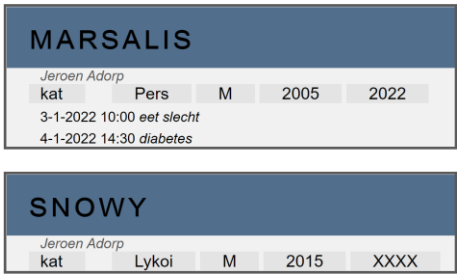
Het is voor een datapunt (*node*) of een attribuut van een *record* niet altijd nodig dat er een waarde is. Soms blijft een dataveld leeg.

56. Bekijk de video over het ontbreken van data.
57. Verklaar met behulp van de video wat *null island* is en waar op aarde het zich bevindt.
58. Leg in eigen woorden uit wat *null* betekent.

In de theorie bij deze paragraaf hebben we drie modellen gezien voor de database van een dierenarts.

59. Open (nogmaals!) het Excel-bestand *H1O08_dierenarts_2.0.xlsx* met de relationele database op basis van tabellen.
60. Zijn er velden met de waarde *null*? Hoe zie je dat?
61. Bekijk de tabel *afspraken*. Zijn er kolommen waarvan de bijbehorende waarde van het record *null* zou mogen blijven? Licht je antwoord toe.

In figuur 1.17 zie je een *screenshot* met twee objecten – de dieren Marsalis en Snowy – uit het administratieprogramma van de dierenarts op basis van het objectmodel.



FIGUUR 1.17

62. Vergelijk figuur 1.17 met figuur 1.14: welke data-attributen worden niet door het programma op het scherm getoond?
63. Bij Marsalis is er op twee manieren sprake van meer data dan bij Snowy. Welke twee manieren zijn dat?
64. Is voor beide van deze manieren sprake van een waarde *null*?

Opdracht 12 alternatieve boomstructuur

In de theorie bij deze paragraaf wordt een database voor de administratie van een dierenartsenpraktijk besproken. Stel dat jij zo'n database moet maken.

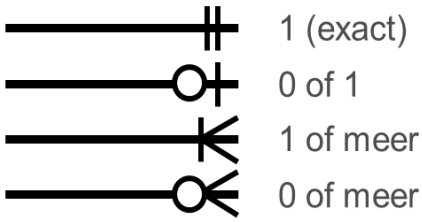
65. Wie stel je dan centraal? De *klant* of het *dier*? Geef minimaal één argument.

In figuur 1.15 is de database van de dierenarts als boom gegeven met *dier* als wortel (*root*) van de boom.

66. Teken een boom met *klant* als wortel.
67. Teken ook een boom met *afspraak* als wortel.
68. Is *afspraak* in jouw boom een *child* van *klant* of van *dier*?
69. Bekijk figuur 1.15 (en figuur 1.14): Is *afspraak* daarin een *child* van *klant* of van *dier*?
70. Zijn er elementen die in de ene boom een *parent* zijn van een ander *element* terwijl ze in de andere boom juist een *child* van dat element zijn?
71. Wat is volgens jou de beste keuze als wortel van de boom? Is dat *dier*, *klant* of *afspraak*? Waarom?

Opdracht 13 kardinaliteit I: soorten relaties

In de theorie van deze paragraaf worden relaties beschreven tussen tabellen, objecten en datapunten. Het aantal relaties kun je beschrijven op basis van de **kardinaliteit** van de relatie. Dit kan worden gedaan met de *kraaienpootnotatie* in figuur 1.18.



FIGUUR 1.18

Eerst een voorbeeld:
Een dier hoort (in de administratie) altijd bij één klant, maar een klant kan meerdere dieren hebben. In figuur 1.14 is dit aangegeven met symbolen, waarvan je in figuur 1.18 de bijbehorende betekenis ziet.

Kardinaliteit geeft aan hoeveel objecten van de ene soort gekoppeld kunnen zijn aan hoeveel objecten van de andere soort. Of voor tabellen: hoeveel records in de ene tabel een relatie kunnen hebben met hoeveel records uit een andere tabel.

Beantwoord de volgende vragen met behulp van figuur 1.14 en figuur 1.18:

72. Hoe zie je dat een dier niet gekoppeld kan worden aan meerdere klanten?
73. Is het mogelijk om een straatkat (zonder eigenaar) in de administratie op te nemen? Hoe zie je dat?
74. Hierboven (bij het voorbeeld) staat de relatie tussen *dier* en *klant* in een Nederlandse zin uitgelegd. Leg de relatie tussen *dier* en *afspraak* uit met een vergelijkbare zin.
75. Verklaar dat het bovenste symbool in figuur 1.18 alleen kan worden toegepast op objecten die een attribuut hebben met uniciteit.

☆ Opdracht 14 XML IV: XML versus HTML

Als je zelf al eens een website hebt gemaakt, is het je vast opgevallen dat XML-documenten erg lijken op HTML-documenten? Data wordt in XML beschreven met **elementen** zoals `<naam>Angel</naam>` die zijn gemaakt met een *start-tag* (`<naam>`), een *eind-tag* (`</naam>`) en de data (**Angel**). De elementen zijn **genest**: elementen bevinden zich binnen andere elementen (die soms zelf weer elementen bevatten).

HTML bevat ook elementen, maar deze zeggen niet perse iets over de data, maar iets over de rol in het document. In HTML is `<h1>Angel</h1>` een kop (van niveau 1; Engels: *header*). Dit vertelt niet dat het een naam is.

76. Open het XML-bestand *H1O14_klas_1.0.xml* in je browser en bekijk de geneste elementen.
77. Maak een boomstructuur bij de data in dit XML-document.
78. Open het HTML-bestand *H1O14_klas_1.1.html* in je browser en bekijk het resultaat.
79. Open nu het HTML-bestand *H1O14_klas_1.1.html* in je editor en bestudeer de HTML-code met haar geneste elementen.
80. Maak een boomstructuur bij de data in dit HTML-document.

1.4 Databases ontwerpen: normaliseren

1.5 NoSQL: een andere kijk op dataopslag

1.6 CAP theorem

H2 SQL-DATABASES

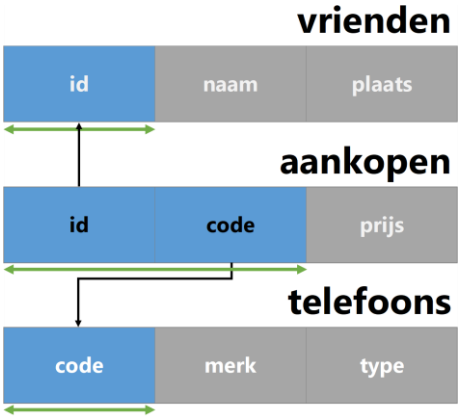
2.1 Inleiding

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:

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.

vrienden

id	naam	plaats
1	Alan	Groningen
2	Bob	Assen
3	Christel	Heerlen
4	Daphne	Groningen
5	Eve	Heerlen
6	Frits	Delft
7	Gonny	Groningen
8	Hajar	Emmen
9	Ingo	Assen

telefoons

code	merk	type
a1	Apple	iPhone 13
a2	Apple	iPhone 14
a3	Apple	iPhone SE
g1	Google	Pixel 6a
g2	Google	Pixel 7
o1	OnePLus	11
s1	Samsung	Galaxy A23
s2	Samsung	Galaxy S23
s3	Samsung	Z flip 4
x1	Xiaomi	12 Lite
x2	Xiaomi	12X

aankopen

id	code	prijs
1	a1	819
6	a1	849
4	a2	929
8	g2	589
9	o1	869
2	s1	319
3	s1	299
8	s1	329
9	s3	929
7	x1	415
8	x2	499

FIGUUR 2.2

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*.

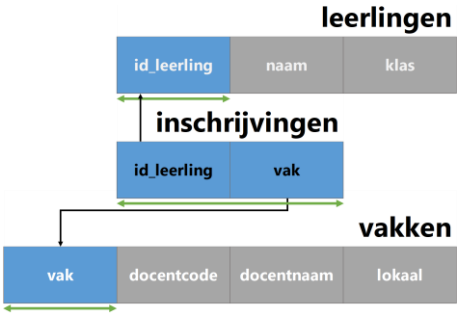
1. Geef de SQL-code waarmee de tabel *telefoons* wordt gemaakt (zie figuur 2.1). Zorgt dat de attributen *code* en *type* het datatype *varchar* krijgen met maximaal drie respectievelijk 16 tekens. Het veld *merk* krijgt het tekstformaat.
2. Voeg SQL-code toe waarmee de eerste twee records van de tabel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



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` text
);
```

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* beschermd 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) 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 paragaaf:

```
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 resultaattabel 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:

a. `SELECT naam FROM vrienden WHERE plaats = 'Assen'`
b. `SELECT code, type FROM telefoons WHERE merk = 'OnePlus' OR merk = 'Xiaomi'`
c. `SELECT code, type FROM telefoons WHERE merk = 'OnePlus' AND merk = 'Xiaomi'`
d. `SELECT naam FROM vrienden WHERE plaats = 'Groningen' AND id > 4`
e. `SELECT naam FROM vrienden WHERE plaats = 'Groningen' AND id <= 4`
f. `SELECT naam FROM vrienden WHERE plaats = 'Groningen' AND NOT id < 4`
g. `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

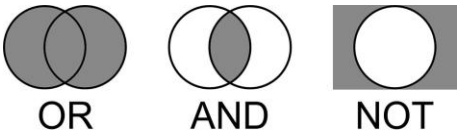
In deze opdracht maken we voor het eerst kennis met de top-2000 database. Hiervan bestaan twee versies. In deze opgave werken we met versie 1. Tenzij anders vermeld is het de bedoeling dat je een query gebruikt om de vraag te beantwoorden. Als antwoord noteer je uiteindelijk de gebruikte query.

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), <= (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 resultaattabel 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 resultaattabel 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 resultaattabel 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 resultaattabel om uit te leggen wat de betekening van `LIKE 'y%'` is.
56. Beschrijf in je eigen woorden wat de uitkomst van de query 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 resultaattabel 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

2.4 Resultaten ordenen en statistiek bedrijven

Je hebt kennis gemaakt met het schrijven van zoekopdrachten. Misschien heb je al ervaren dat de resultaattabellen er soms wat onhandig uitzien. Soms staan resultaten dubbel vermeld, of is de ordening onhandig. En bij opdracht 9 werd een aantal keer de vraag gesteld *hoeveel* (liedjes of artiesten) ergens van was. Daarbij kon je wel een zoekopdracht maken, maar moest je nog wel zelf tellen. Dat kan beter.

Stel dat we op basis van de database *mobiel* een overzicht willen hebben van alle woonplaatsen waarin vrienden wonen. Op basis van de vorige paragraaf is `SELECT plaats FROM vrienden` dan een optie. Het probleem hierbij is, dat er veel vrienden eenzelfde woonplaats hebben, zodat plaatsen dubbel worden vermeld. Bovendien staan de namen door elkaar. We willen een alfabetisch gesorteerde lijst. Dat kan met:

```
SELECT DISTINCT plaats        // selecteer unieke attribuutwaarden uit plaats
FROM vrienden                 // uit de tabel vrienden
ORDER BY plaats ASC           // sorteer de kolom plaats oplopend (alfabetisch)
```

Met de toevoeging DISTINCT kun je unieke items selecteren. Dit voorkomt dubbele identieke waarden in de resultaattabel. Het **ordenen** of **sorteren** van de resultaattabel doe je met ORDER BY. Je kiest dan een kolomnaam van de resultaat, gevolg door ASC (Engels: *ascending*: oplopend) of DESC (Engels: *descending*: aflopend). Het verbeterde resultaat zie je in figuur 2.7.

plaats
Assen
Delft
Emmen
Groningen
Heerlen

FIGUUR 2.7

Je kunt met behulp van SQL aan een databasemanagementsysteem vragen om statistiek te bedrijven op de data in een database, zoals voor de vraag: *hoeveel vrienden staan er in de database?* Om hier rechtstreeks antwoord op te krijgen, gebruik je COUNT om de bijbehorende records te tellen:

```
SELECT COUNT(*) AS aantal     // tel het aantal records
FROM vrienden                 // van de tabel vrienden
```

Merk op dat COUNT een **functie** is. Wat je telt moet daarom tussen haakjes. Als je records wilt tellen, kun je de asterix (*) gebruiken. Er is geen attribuut *aantal* in de database. Met behulp van de **alias** *aantal* verwijs je naar de uitkomst van de COUNT-functie en zorg je dat de resultaattabel een kolomnaam heeft.

Wat moeten de leerlingen halen om een voldoende te blijven staan op hun rapport? Hoeveel stappen heb ik vandaag gezet (figuur 2.8)? Wat is mijn gemiddelde hartslag als ik lig te slapen?

FIGUUR 2.8

Bij het gebruik van databases spelen berekeningen en statistiek vaak een centrale rol. Dit is een overzicht van SQL-functies voor statistiek:

- `COUNT` telt het aantal records van een selectie
- `SUM` telt attribuutwaarden van records bij elkaar op
- `AVG` neemt het gemiddelde van de attribuutwaarden
- `MAX` zoekt de grootste waarde in een selectie
- `MIN` zoekt de kleinste waarde in een selectie

Deze statistische functies kun je gebruiken op een volledige selectie van records, maar het is ook mogelijk om de selectie op te delen in meerdere groepen en daar statistiek op te bedrijven. Hoe dat werkt, leer je tijdens het maken van de opdrachten van deze paragraaf.

Opdracht 10 mobiel: basisquery's

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

1. `SELECT DISTINCT merk FROM telefoons ORDER BY merk ASC`
2. `SELECT MIN(prijs) AS best FROM aankopen WHERE code = 's1'`
3. `SELECT MAX(prijs) AS duur FROM aankopen WHERE NOT (code = 'a1' OR code = 'a2')`
4. `SELECT DISTINCT plaats FROM vrienden WHERE plaats LIKE '%en' ORDER BY plaats ASC`

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

Opdracht 11 top 2000: ORDER BY en DISTINCT

Voor deze opdracht gebruiken we *top 2000 versie 1*.

67. Toon alle door *Muse* (figuur 2.9) gemaakte liedjes met een notering in de top 2000, op volgorde van het jaar van uitgave. Het resultaat moet alleen de titel en het jaar bevatten, met het meest recente nummer bovenaan.
68. Geef alle door *Muse* gemaakte liedjes met een notering in de top 300 van de top-2000 van 2020. Geef alleen de titel en de positie, oplopend gesorteerd op positie.
69. Is het mogelijk om de query van de vorige vraag aan te passen, zodat wordt gesorteerd op het jaar van uitgave (in plaats van positie), zonder dat de kolom *uit* in de resultaattabel verschijnt?

FIGUUR 2.9

70. Maak een alfabetische lijst van alle artiesten die een notering in de top-20 van de top-2000 van 2018 hadden. Elke artiestennaam mag maar één keer voorkomen. Hoeveel verschillende artiestennamen vind je in de lijst?

Opdracht 12 mobiel: statistische functies

In de theorie staan een aantal functies waarmee je statistiek kunt bedrijven op een database. Gebruik deze functies bij de volgende opdrachten.

71. Tel het aantal vrienden dat in Groningen of Assen woont. Geef het resultaat in een tabel met de kolomnaam *aantal*.
72. Voorspel met een Nederlandse zin wat het resultaat zal zijn van de query:
    `SELECT COUNT(DISTINCT plaats) AS N FROM vrienden`
73. Voer bovenstaande query uit en gebruik het resultaat om vast te stellen of jouw voorspelling juist was.
74. Wat is de gemiddelde prijs die vrienden hebben betaald voor een iPhone? Gebruik de codes *a1* en *a2* in je query.
75. Hoeveel geld hebben alle vrienden samen aan telefoons uitgegeven volgens deze database? Zorg voor een resultaattabel met de kolom *totaal*.
76. Voorspel met een Nederlandse zin wat het resultaat zal zijn van de query:
    `SELECT SUM(prijs) / COUNT(DISTINCT id) AS antwoord FROM aankopen`
77. Voer bovenstaande query uit en gebruik het resultaat om vast te stellen of jouw voorspelling juist was.
78. De functies `MAX` en `MIN` geven de maximale respectievelijk minimale waarde van een attribuut in een dataset, zoals *de hoogste prijs*. Zoek uit of deze functies ook werken met attributen met tekst. Zo ja, leg dan uit hoe de functies in dat geval werken.

Opdracht 13 top 2000: ordenen en statistiek

Voor deze opdracht gebruiken we opnieuw *top 2000 versie 1*.

79. De zanger *Phil Collins* (figuur 2.10) maakt ook deel uit van de band *Genesis*. Maak een overzicht van alle titels die van *Phil Collins* en van *Genesis* in de database staan. De resultaattabel moet de naam van de artiest (alfabetisch gesorteerd), de titel van het liedje en het jaar waarin het liedje is uitgekomen geven.
80. Als het goed is heb je bij de vorige vraag voor de sortering gekozen voor `ORDER BY artiest ASC`. Breid deze regel uit naar `ORDER BY artiest DESC, uit ASC`. Beschrijf vervolgens het resultaat.
81. Voorspel wat er gebeurt wanneer je de -regel vervangt door `ORDER BY 1 DESC, 2 ASC`. Bekijk vervolgens het resultaat.

FIGUUR 2.10

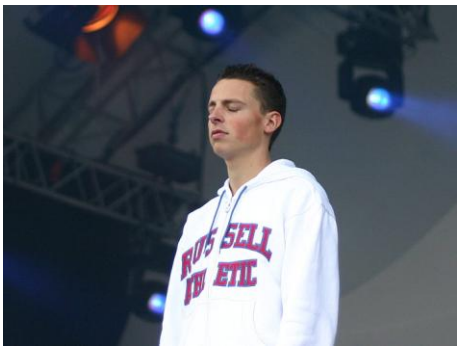
Bekijk de volgende query's:

```
SELECT COUNT(artiest) AS x FROM top2000 WHERE NOT ed2021 IS NULL
SELECT COUNT(DISTINCT artiest) AS y FROM top2000 WHERE NOT ed2021 IS NULL
```

82. Wat is de uitkomst van de waarden *x* en *y*? Als je denkt dat het niet precies te voorspellen is, mag je een schatting doen.
83. Wat is de betekenis van de uitkomsten *x* en *y* en dus van de query's?
84. Leg uit waarom de toevoeging `WHERE NOT ed2021 IS NULL` voor deze query's noodzakelijk is.

We willen graag weten welke liedjes het meest gestegen of gedaald zijn in de top-2000. We proberen de volgende query:

```
SELECT artiest, titel, (ed2020 - ed2019) AS verschil
FROM top2000 ORDER BY 3 DESC
```



FIGUUR 2.11

85. Bekijk het resultaat van deze query. Wat is de exacte betekenis van *verschil*? Wat kun je zeggen over het liedje (en de artiest; zie figuur 2.11) dat bovenaan staat?
86. Navigeer naar het einde van de resultaattabel. Daar zie je dat de 3e kolom voor heel veel liedjes geen waarde bevat. Geef hiervoor de verklaring.
87. Breid je zoekopdracht uit. Zorg dat de in de vorige opdracht bedoelde records niet meer worden getoond.

☆ Opdracht 14 oudste vrouwen: werken met datums

In deze paragraaf hebben we statistiek bedreven met getalswaarden en teksten en getallen gesorteerd. In speciaal dataformaat is de **datum**. Dat bestuderen we aan de hand van de casus *oudste vrouwen*.

88. Lees de beschrijving van de database *oudste vrouwen* in de bijlage.
89. Geef een overzicht van records van vrouwen die ouder dan 116 zijn geworden, gesorteerd op leeftijd met de oudste bovenaan.
90. Bepaal de gemiddelde leeftijd van de vrouwen in de database die voor 1880 zijn geboren.
91. Voer de volgende query uit:
    ```
    SELECT naam, DATEDIFF(sterfdatum,geboortedatum) AS L
    FROM vrouwen ORDER BY 2 DESC
    ```
    en bekijk het resultaat. Wat doet de functie DATEDIFF? Gebruik eventueel het internet.
92. Wat is de betekenis en eenheid van de kolom *L*?
93. Opvallend veel oude vrouwen komen uit Japan en Amerika. Selecteer alle vrouwen die een leeftijd van meer dan 116 met daarin de kolom *L* omgerekend naar jaren in de vorm 116.2 (jaar) die juist niet uit Japan of Amerika komen.

2.5 Meer statistiek: groeperen

In de theorie van de vorige paragraaf staat beschreven hoe je het totale aantal vrienden in de vriendentabel van de database *mobiel* kunt bepalen. Dat is een statistiek die betrekking heeft op alle records in de tabel.

Stel nu dat we *het aantal vrienden per woonplaats* willen weten. Dat kan door per woonplaats een query op te stellen (als je eerst hebt uitgezocht welke woonplaatsen er allemaal zijn), maar het kan ook in één keer. Het resultaat in figuur 2.12 is bereikt met de query:

```
SELECT plaats, COUNT(*) AS aantal
FROM vrienden
GROUP BY plaats              // maak groepjes per plaats
```

plaats	aantal
Groningen	3
Assen	2
Heerlen	2
Delft	1
Emmen	1

FIGUUR 2.12

Met de SQL-opdracht GROUP BY geef je aan dat het je records met dezelfde attribuutwaarden wilt groeperen. In dit geval: maak groepjes van records op basis van dezelfde plaats. Het tellen met de functie COUNT wordt vervolgens per groepje (dus per woonplaats) uitgevoerd.

Nu we dit resultaat hebben bereikt, kunnen we nog een stap verder gaan, zoals voor de zoekopdracht: *Geef een overzicht van alle plaatsen waar meer dan één vriend woont*. We hebben geleerd om eisen te stellen aan records met behulp van WHERE. Maar merk op: het aantal vrienden van een woonplaats is geen attribuut (-waarde) in de database. Deze waarde staat niet (rechtstreeks) in de database, maar is een eigenschap van de groepjes die we hebben gemaakt.

Om eisen te stellen aan de groepswaarden die worden getoond, gebruik je:

```
SELECT plaats, COUNT(*) AS aantal FROM vrienden
GROUP BY plaats              // maak groepjes per plaats
HAVING aantal > 1            // en toon groepen waarvoor geldt: aantal > 1
```

Het commando HAVING kun je dus uitleggen als *(groepjes) die de eigenschap hebben dat*. Vervolgens kun je tellen met de COUNT-functie zoals we hier hebben gedaan. Maar ook de overige statistische functies die je in de vorige paragraaf hebt geleerd, kun je toepassen op de groep.

Opdracht 15 mobiel: groeperen

Bij deze opgave gebruiken we de database *mobiel*.

94. Bekijk figuur 2.13. Geef de query waarmee deze resultaattabel is verkregen. Vergeet niet om ook de ordening aan te brengen.
95. Pas de query van de vorige vraag aan, zodanig dat alleen merken met precies twee modellen worden getoond.

merk	N
Apple	3
Samsung	3
Google	2
Xiaomi	2
OnePlus	1

FIGUUR 2.13

De volgende vragen gaan over de query:

```
SELECT code, MIN(prijs) AS bestBuy FROM aankopen
GROUP BY code ORDER BY bestBuy ASC
```

96. Leg in woorden uit (dus niet de resultaattabel geven) wat de uitkomst van deze query is.
97. Voer de query uit. (Had je gelijk?)
98. Maak een overzicht van de gemiddelde prijs en de code van mobieltjes waarvoor geldt dat het gemiddelde aankoopbedrag van dat mobieltje minder dan € 600,- is.

☆ Opdracht 16 top oudste vrouwen: groeperen

Voor deze opdracht gebruiken we de database *oudste vrouwen*.

Van de in 1997 overleden Française Jeanne Calment (figuur 2.14) wordt aangenomen dat ze de oudste mens is die ooit geleefd heeft (waarvan de geboorte- en sterdatum onomstreden zijn). Ze werd ruim 122 jaar. In de database staat slechts één vrouw uit Frankrijk.

FIGUUR 2.14

99. Geef een overzicht van landen met het aantal in de database opgenomen oudste vrouwen per land, gesorteerd op het land met de meeste vermeldingen.
100. Verfijn het resultaat door alleen landen op te nemen met meer dan één vermelding in de database.

Japan heeft wereldwijd (relatief gezien) de meeste mensen die meer dan 100 jaar oud worden.

101. Maak een query waarmee de gemiddelde leeftijd van de oudste vrouwen uit Japan kan worden berekend.

Je hebt bij de vorige vraag en eerdere opdrachten met het gemiddelde waarschijnlijk gemerkt, dat dit standaard met veel cijfers achter de komma wordt weergegeven. Gelukkig kunnen we resultaten netter weergeven, door deze af te ronden met de ROUND-functie. Voorbeeld: ROUND(1.174,1) geeft 1.2 als antwoord omdat er op één cijfer achter de komma wordt afgerond. ROUND(1.174,2) geeft 1.17.

102. Verbeter de query van de vorige vraag, door de gemiddelde leeftijd op één decimaal achter de komma af te ronden.
103. Geef een overzicht van de gemiddelde leeftijd van de oudste vrouwen per land, afgerond op twee decimalen, oplopend gesorteerd op de gemiddelde leeftijd voor landen met meer dan één vrouw in de database.

Opdracht 17 top 2000: groeperen

104. Maak een overzicht van artiestennamen met het aantal liedjes dat van hen in de top 2000 van 2019 stond. Natuurlijk sorteer je van hoog naar laag op aantal. Welke artiest (figuur 2.15) komt het meest voor?
105. Maak een overzicht van artiesten die met 20 of meer liedjes in de database staan vermeld.
106. Veel liedjes hebben een unieke titel, maar dat geldt niet voor alle liedjes. Maak een overzicht van titels die meer dan twee keer in de database voorkomen.

FIGUUR 2.15

Met de commando's die we hebben geleerd kunnen we complexe query's maken die veel instructies combineren, zoals:

```
SELECT artiest, AVG(ed2021) AS gem FROM top2000
WHERE ed2021 <= 100 GROUP BY artiest
HAVING COUNT(*) >= 4 ORDER BY gem ASC
```

MERK OP we gebruiken hier achter HAVING geen alias maar een andere functie (COUNT) dan de functie die we bij het selecteren hebben gebruikt (AVG).

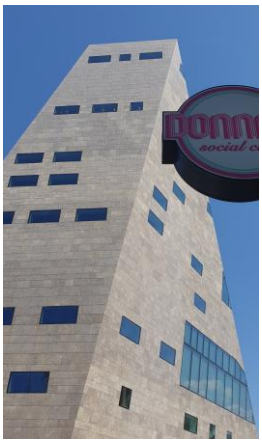
107. Beschrijf in algemene bewoordingen wat de uitkomst van de query zal zijn.
108. Voer de query uit en vergelijk jouw omschrijving met de uitkomst.
109. Onder de query staat een MERK OP over het gebruik van functies bij HAVING. Noem een reden waarom je deze techniek zou willen gebruiken.

2.6 Subquery's

Met WHERE filteren we onze selectie door voorwaarden te stellen aan het resultaat. Als we in de database *mobiel* alle vrienden willen vinden die in Groningen wonen, gebruiken we:

```
SELECT naam                  // selecteer het attribuut naam van records
FROM vrienden                // uit de tabel vrienden
WHERE plaats = 'Groningen'   // met een plaats gelijk aan Groningen
```

In paragraaf 2.1 kun je zien dat de resultaattabel van deze query de namen *Alan*, *Daphne* en *Gonny* zal bevatten. Bij deze query hebben we bij de WHERE gebruik gemaakt van de voorkennis dat we weten dat er vrienden in Groningen (figuur 2.16) wonen. Maar kijk eens naar de volgende vraag:
Welke mensen uit de tabel vrienden wonen in de plaats waar Alan woont?



FIGUUR 2.16

Als de woonplaats van *Alan* niet bekend is, zal deze eerst met SQL moeten worden bepaald. De **uitkomst** van deze **subquery** kan vervolgens worden gebruikt bij de WHERE van de hoofdquery. Dat gaat zo:

```
SELECT naam                  // selecteer het attribuut naam
FROM vrienden                // uit de tabel vrienden
WHERE plaats =               // voor records met plaats gelijk aan…
(
  SELECT plaats              // selecteer het attribuut plaats van
  FROM vrienden              // records uit de tabel vrienden
  WHERE naam = 'Alan'        // met een naam gelijk aan Alan
)
```

De subquery – het geel gemarkeerde deel hierboven – staat tussen haakjes en wordt daarom als eerste uitgevoerd. De uitkomst van deze subquery is `'Groningen'` en dit wordt vervolgens ingevuld achter `WHERE plaats =` . Daarmee ontstaat uiteindelijk de eerste query uit deze paragraaf.

De uitkomst van de subquery hierboven is één attribuutwaarde van één enkel record. Het is ook mogelijk om een subquery te maken met meerdere resultaten. We passen onze vraag aan naar:
Welke mensen uit de tabel vrienden wonen in de woonplaats van Alan of de woonplaats van Bob?

FIGUUR 2.17

De WHERE van de subquery moet nu worden uitgebreid tot: `WHERE naam = 'Alan' OR naam = 'Bob'`. *Alan* woont in *Groningen* en *Bob* woont in *Assen* (figuur 2.17). De subquery vindt nu dus meerdere records met bijbehorende attribuutwaarden voor *plaats*. We zoeken nu vrienden, waarvoor geldt dat hun woonplaats IN het lijstje *Groningen, Assen* valt. Daarvoor is het commando IN:

```
SELECT naam                  // selecteer het attribuut naam
FROM vrienden                // uit de tabel vrienden
WHERE plaats IN              // voor records waarvan de plaats voorkomt in…
(
  SELECT plaats              // selecteer het attribuut plaats van
  FROM vrienden              // records uit de tabel vrienden
  WHERE naam = 'Alan'        // met een naam gelijk aan Alan
  OR naam = 'Bob'            // of met een naam gelijk aan Bob
)
```

Een subquery kan ook een vraag stellen aan een andere tabel dan de hoofdquery. Dat zie je bij het maken van de vragen in de opgaven bij deze paragraaf.

Opdracht 18 mobiel: oefenen met subquery's

Bij deze opgave gebruiken we de database *mobiel*. Gebruik subquery's voor het beantwoorden van de vragen. Zorg dat alleen het antwoord op de vraag als resultaattabel verschijnt en bijvoorbeeld dus niet een grotere tabel waar het antwoord door sortering bovenaan staat.

110. Wat is de laagste prijs die door voor een *Samsung Galaxy A23* (figuur 2.18) is betaald? Geef de resultaattabel met alleen de prijs in een kolom met de naam *laagste*.
111. Hoeveel telefoons zijn er aangeschaft van de merken *Xiaomi* en *Oneplus*? Geef de resultaattabel de kolom *N* mee.
112. Hoeveel telefoons van het merk *Samsung* zijn er aangeschaft met een prijs onder de € 500,-?
113. Geef een overzicht van het merk en type van telefoons waarvan er exemplaren verkocht zijn onder de € 500,-. Zorg dat elk type maar één keer wordt genoemd, ook als er meerdere exemplaren van zijn verkocht.

FIGUUR 2.18

114. Geef het merk en type van alle telefoons die van het merk zijn, dat ook het type *Z flip 4* verkoopt.

Waarschijnlijk heb je bij de vorige vraag gekozen voor een oplossing waarbij de *Z flip 4* zelf ook als resultaat verschijnt. Het liefst willen we dat vermijden en alleen alle overige telefoons van het merk tonen.

115. Breid de query van de vorige vraag uit, zodat ook aan deze eis is voldaan.

☆ Opdracht 19 mobiel: dubbele subquery's

In de theorie hebben we gezien dat je het resultaat van een subquery kunt gebruiken in je hoofdquery. Soms is een zoekvraag zo complex dat je in de subquery nogmaals een subquery nodig hebt. Voorbeeld:
Wie heeft er een telefoon van het merk OnePLus gekocht?

Het antwoord kun je vinden met de query:

```
SELECT naam FROM vrienden WHERE id = (
      SELECT id FROM aankopen WHERE code = (
            SELECT code FROM telefoons WHERE merk = 'OnePlus')
)
```

116. Voer bovenstaande query uit en bekijk het resultaat.
117. Leg uit dat het = teken in `WHERE id =` als `WHERE code =` beter kan worden vervangen door `IN`.
118. Geef een overzicht van namen van vrienden met een telefoon van het merk *Xiaomi*.
119. Geef het merk en type van alle telefoons die door *Ingo* zijn gekocht.

Opdracht 20 top 2000: subquery's

Voor deze opdracht gebruiken we *top 2000 versie 1*. Gebruik subquery's voor jouw oplossing.

120. Geef de titel van het nummer van *R.E.M.* dat het hoogst in de top-2000 van de editie 2021 stond.
121. Geef alle titels van liedjes van *U2* die in 2020 hoger stonden dan het hoogst genoteerde nummer van *Maan* in dat jaar.
122. Geef alle titels en namen van de artiesten die in de in 2020 in de top-5 stonden (dus niet alleen het liedje uit die top-5), alfabetisch gesorteerd op artiest en vervolgens op titel.
123. Hetzelfde als de vorige vraag maar nu met het aantal liedjes per artiest in plaats van de titels.
124. Selecteer alle liedjes van artiesten in de top-1000 van 2020 die een liedje in de top-50 van 2020 hebben dat in 2019 helemaal niet in de top-2000 stond.
125. Welke artiesten stonden in 2019 wel in de top-100 van de top 2000, maar een jaar eerder nog niet in de top-100? Geef een overzicht van unieke artiestennamen.
126. Leg uit dat
    `SELECT DISTINCT artiest FROM top2000 WHERE ed2019 <= 100 AND NOT ed2018 <= 100`
    niet tot het juist antwoord leidt. (Was dat jouw antwoord bij de vorige vraag?)

Eindopdrachten met query's op basis van één tabel

De stof die je tot nu toe geleerd hebt in dit hoofdstuk hebben we steeds geoefend op drie bekende databases (*mobiel*, *top2000* en *oudstevrouwen*). Uiteindelijk moet je in staat zijn om het geleerde toe te passen op een nieuwe database.

Voor deze opdracht gebruiken we de database *voornamen*.

127. Lees de casusbeschrijving van de database *voornamen* aan het eind van deze module.
128. Geef de DDL-coderegels (zie paragraaf 2.2) waarmee zowel de database *voornamen* als de tabel *namen* kan worden gebruikt. Kies zelf geschikte datatypes voor de genoemde attributen.
129. Leg voor elk van de vier attributen in de database uit waarom deze niet geschikt is als primaire sleutel.
130. Is er volgens jou een combinatie van twee attributen die als samengestelde sleutel zou kunnen dienen? Zo ja, welke combinatie is dat?

Bij de onderstaande vragen kun je gebruik maken van de *querier*, maar het beste kun je proberen om de opdrachten eerst schriftelijk te maken en daarna pas met behulp van de computer te testen of jouw antwoorden goed zijn.

131. Geef de query waarmee, oplopend van 1 naar 25 de top-25 meisjesnamen wordt getoond. Zorg voor de kolommen *positie*, *naam* en *aantal*.
132. Leg in woorden uit wat het resultaat zal zijn van de volgende query:

```
SELECT naam, geslacht FROM namen
WHERE NOT (aantal <= 100000 OR aantal >= 250000)
```

133. Jos vraagt zicht af:
Als ik de top-100 van namen van vrouwen en mannen zou nemen. Hoeveel miljoen namen zijn dat bij elkaar per geslacht? In figuur 2.19 zie je zijn resultaat tabel. Geef de query waarmee dit resultaat is bereikt.

geslacht	N
V	3.1913
M	3.7705

FIGUUR 2.19

Er zijn namen die zowel voor jongens als voor meisjes worden gebruikt. Deze staan dus twee keer in de database.

134. Geef het aantal unieke namen in de database in een resultaattabel met de kolomnaam *aantal*.

Als de maker van deze opgave op zoek wil naar namen die zowel voor jongens als meisjes voorkomen, komt hij er achter dat er iets mis gaat bij de verwerking van de query in figuur 2.20.

```
SELECT naam, geslacht
FROM namen WHERE naam IN
(
  SELECT naam FROM namen
  GROUP BY naam
  HAVING COUNT(naam) > 1
  ORDER BY naam ASC
)
ORDER BY naam ASC
```

FIGUUR 2.20

135. Voer de query uit en bekijk de resultaattabel. Leg hiermee uit wat er mis gaat bij de zoektocht naar namen met een dubbele vermelding (voor zowel jongens als meisjes). Zorg dat uit je antwoord naar voren komt wat de uitkomst van de query zou moeten zijn.
136. Vincent vraagt zicht af:
Hoeveel meisjes- en jongensnamen zijn er die meer dan 50.000× zijn geregistreerd?
Geef de query die leidt tot een resultaattabel met de kolommen *geslacht* en het bedoelde *aantal*.

☆ Voor de volgende vragen heb je kennis van de LIKE-operator nodig.

137. Leg uit welke van onderstaande twee query's tot de meeste records in de resultaattabel zal leiden:

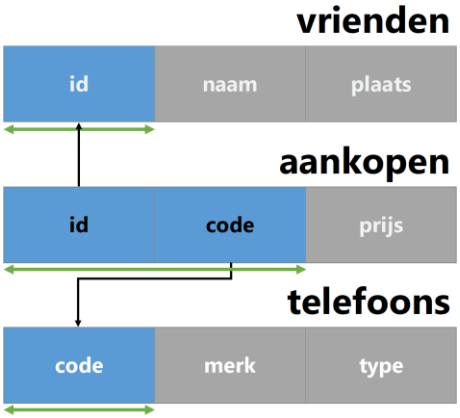
```
SELECT naam FROM namen WHERE naam LIKE 'Ren_'
SELECT naam FROM namen WHERE naam LIKE 'Ren%'
```

138. Bepaal het gemiddeld aantal registraties van namen in de database van jouw geslacht en die beginnen met de eerste letter van jouw voornaam. Rond de uitkomst af op hele cijfers.

2.7 Meerdere tabellen bevragen

In figuur 2.21 zie je nogmaals het strokendiagram van de database *mobiel* die bestaat uit drie tabellen. In de vorige paragraaf hebben we gezien dat we data uit meerdere tabellen kunnen combineren met subquery's, maar in veel gevallen is het beter dit op een andere manier te doen. We bekijken de volgende casusvraag:

Welke types telefoons van het merk Samsung zijn verkocht voor minder dan € 350,-? Geef een overzicht van de types met de bijbehorende verkoopprijzen.



FIGUUR 2.21

Voor deze zoekopdracht heb je twee tabellen nodig. Aan beide tabellen wordt een eis gesteld. Stel dat we eerst willen weten welke types telefoons in aanmerking komen. Dan kunnen we hiervoor de volgende query gebruiken:

```
SELECT code, type FROM telefoons WHERE merk = 'Samsung'
```

Deze query geeft drie resultaten (zie eventueel de volledige database in figuur 2.2): De *Galaxy A23*, de *Galaxy S23* en de *Z flip 4*.

Aan de tabel aankopen wordt de eis gesteld dat de prijs onder de € 250,- ligt:

```
SELECT code, prijs FROM aankopen WHERE prijs < 350
```

Ook dit levert drie resultaten op: 299, 319 en 329. Deze drie prijzen horen wel allemaal bij de code *s1* die hoort bij de *Galaxy A23*.

We doen nu een eerste (foute!) poging om deze query's te combineren tot één nieuwe zoekopdracht:

type	prijs
Galaxy A23	299
Galaxy A23	319
Galaxy A23	329
Galaxy S23	299
Galaxy S23	319
Galaxy S23	329
Z flip 4	299
Z flip 4	319
Z flip 4	329

FIGUUR 2.22

```
SELECT type, prijs
FROM telefoons, aankopen
WHERE merk = 'Samsung' AND prijs < 350
ORDER BY type ASC, prijs ASC
```

Het resultaat van deze query (figuur 2.22) is niet wat we willen: het databasemanagementsysteem heeft alle records van de tabel *telefoons* (die voldoen aan de eis) gecombineerd met alle records van de tabel *aankopen* tot (3 × 3 =) 9 resultaten. Dit is onjuist: de *Z flip 4* is bijvoorbeeld (zie de onderste regel van de resultaattabel in figuur 2.22) helemaal niet verkocht voor een prijs van 329.

Voor ons is het logisch dat de *code* van het type in de tabel telefoons dezelfde moet zijn als de *code* in de tabel *aankopen*, maar dat geldt niet voor het databasemanagementsysteem. Daarom gebruiken we de volgende toevoeging (gemarkeerd met geel):

code	type	prijs
s1	Galaxy A23	319
s1	Galaxy A23	299
s1	Galaxy A23	329

FIGUUR 2.23

```
SELECT telefoons.code, type, prijs
FROM telefoons, aankopen
WHERE telefoons.code = aankopen.code
AND merk = 'Samsung' AND prijs < 350
```

Het resultaat in figuur 2.23 is het antwoord op de vraag. Merk op dat we `telefoons.code` hebben toegevoegd aan de query. Omdat beide tabellen een attribuut code hebben, moet je (met *tabelnaam.*) aangeven uit welke tabel de informatie moet worden gehaald (ook al is de code in beide tabellen gelijk).

Opdracht 21 mobiel: meerdere tabellen

Bij deze opgave gebruiken we de database *mobiel*. Gebruik de techniek uit de theorie van deze paragraaf bij het oplossen van de volgende vraagstukken:

139. Geef de verschillende prijzen die zijn betaald voor een *Samsung Galaxy A23*.
140. Geef een overzicht van (unieke) telefoonmerken die mobiels hebben verkocht voor onder de € 500.
141. Geef het aantal mobieltjes dat verkocht is voor minder dan € 500.
142. Geef het totale bedrag dat vrienden uit Assen hebben uitgegeven aan telefoons.
143. Geef een overzicht van types telefoons die niet van Samsung of Apple zijn?
144. Hoeveel types telefoons zijn er verkocht die niet van Samsung of Apple zijn?

We kunnen ook vragen stellen, waarbij niet twee maar drie tabellen nodig zijn voor het antwoord. In dat geval moeten we de verbindingen tussen alle tabellen beschrijven, zoals in de volgende query:

```
SELECT naam, merk, type, prijs
FROM vrienden,telefoons, aankopen
WHERE vrienden.id = aankopen.id
AND telefoons.code = aankopen.code
AND plaats = 'Groningen'
```

145. Beschrijf de betekenis van bovenstaande query met een Nederlandse zin.
146. Controleer jouw antwoord bij de vorige vraag door de query uit te voeren.
147. Pas de query aan zodat in plaats van *merk* en *type* alleen de unieke *code* van de telefoon in de resultaat tabel wordt getoond.
148. Geef een overzicht van unieke merken en types telefoons die gekocht zijn door vrienden die niet uit Assen of Emmen komen, alfabetisch gesorteerd op merk en daarbinnen alfabetisch gesorteerd op type.

Opdracht 22 top 2000: verbeterde versie

In deze module heb je kennis gemaakt met de casus *Top 2000 versie 1* met bijbehorende database *top_2000_v1*. Vanaf gaan we aan de slag met een variant op deze database met meerdere tabellen: *Top 2000 versie 2* met bijbehorende database *top_2000_v2*.

149. Ga naar de casusomschrijving van *Top 2000 versie 2* achterin deze module. Lees de casustekst en bestudeer het bijbehorende strokendiagram.
150. Leg uit dat de tabel *notering* geen primaire sleutel heeft.
151. Is er een combinatie van attributen die als samengestelde sleutel zou kunnen fungeren?
152. Noem een argument waarom deze versie 2 gezien kan worden als een betere versie dan versie 1. Wat is er *beter* aan het op deze manier opslaan van de data?
153. In hoofdstuk 1 heb je kennis gemaakt met het begrip *redundantie*. Vergelijk *versie 1* en *versie 2* op dit punt en beschrijf kort je bevindingen.

We gaan de inhoud van de database verkennen door middel van een aantal databases.
Gebruik query's om de volgende vragen te beantwoorden:

154. Van hoeveel edities van de top 2000 is data opgenomen in deze database?
155. Hoeveel verschillende artiesten staan er in deze database?

Er is een foutje in deze database geslopen. Er is één artiest (figuur 2.24) die met twee verschillende waarden van *artiest_id* in de database staat.

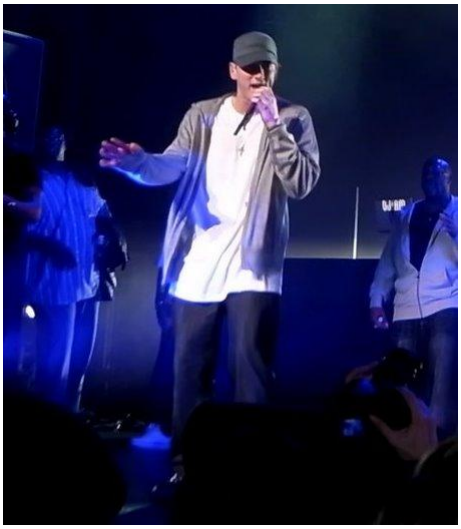
156. Onderzoek wie dat is en wat hiervan de oorzaak is. Gebruik hiervoor een query met HAVING.

FIGUUR 2.24

Opdracht 23 top 2000 v2: meerdere tabellen

Gebruik de database *top 2000 versie 2* en de techniek uit de theorie van deze paragraaf bij het oplossen van de volgende vraagstukken:

157. Geef een overzicht van alle titels van nummers van de artiest *Eminem* (figuur 2.25) met het jaar van uitgave, zodanig gesorteerd dat het meest recente nummer bovenaan staat.
158. Welk nummer van *Eminem* heeft het hoogst gestaan en in welk jaar was dat? Geef een overzicht van liedjes van *Eminem* met hun positie (met vermelding van de editie). Zorg dat de lijst zo geordend is dat de beste positie (laagste) bovenaan staat.
159. Geef de titel van de track die in 2005 op positie 37 stond.
160. Geef de top-10 (met artiest en titel) van de top-2000 van 2008.
161. Van welke artiest of artiesten is het oudste nummer in de database? En hoe heet dat nummer?



FIGUUR 2.25

Het komt voor dat artiesten een nummer uitbrengen dat in het jaar van uitgave ook al meteen in de top 2000 wordt gekozen. Hoe vaak zou dat al gebeurd zijn? voor de jaren die deze database bestrijkt)?

162. Ontwerp een query die bovenstaande vraag beantwoord door een lijst van artiesten met titels te maken die aan de beschreven eis voldoen. Zorg voor een sortering, zodanig dat de artiest die in deze situatie op de hoogste positie (is lage waarde) binnenkwam bovenaan staat.

☆Opdracht 24 top 2000 v2: complexere query's lezen

Gebruik de database *top 2000 versie 2*. In deze opgave moet je de stof uit de voorgaande paragrafen combineren om tot je antwoord te komen, zoals in de volgende query:

```
SELECT naam, COUNT(DISTINCT notering.track_id) as N
FROM artiest, track, notering
WHERE artiest.artiest_id = track.artiest_id
AND track.track_id = notering.track_id
GROUP BY naam HAVING N > 25
ORDER BY N DESC
```

163. Beschrijf de uitkomst van deze query met een Nederlandse zin.
164. Leg uit wat er verandert aan de betekenis van de uitkomst van deze query als het woord DISTINCT wordt weggelaten.

Bekijk de volgende query:

```
SELECT naam, titel, uit
FROM artiest, track
WHERE artiest.artiest_id = track.artiest_id
AND titel IN
(
	SELECT titel
	FROM track
	GROUP BY titel
	HAVING COUNT(titel) > 3
)
ORDER BY titel ASC, uit ASC
```

165. Beschrijf de uitkomst van deze query met een Nederlandse zin.
166. Controleer je antwoorden op de vragen uit deze opgave door de twee query's uit te voeren.

☆Opdracht 25 top 2000 v2: complexe query's schrijven

In deze opdracht krijg je een reeks uitdagende query's waarbij je alle kennis van dit hoofdstuk moet combineren.

uit	aantal
1973	1
1974	2
1975	3
1976	3
1977	3
1978	3
1979	5
1980	3
1981	2
1982	2

FIGUUR 2.26

167. Met hoeveel verschillende tracks staat de artiest *Ed Sheeran* in de database? Zorg dat alleen dit aantal verschijnt.
168. Welke artiesten anders dan *Ed Sheeran* hebben precies evenveel verschillende tracks in de database als *Ed Sheeran*?
169. ABBA staat met 27 verschillende liedjes in de database. In figuur 2.26 zie je een tabel met daarin de jaren waarin deze nummers uitkwamen en het aantal nummers van ABBA van dat jaar dat leidde tot een notering in de top-2000.
Geef de bijbehorende query.
170. Geef een overzicht van artiestennamen die in 2005 een liedje in de top-100 hadden maar in 2008 geen liedjes meer in de top-100 hadden.
171. Als je op plek 2000 staat, sta je dus nog nèt in de top-2000. Er zijn twee liedjes die twee keer precies op plek 2000 hebben gestaan. Geef een query waarmee de titels en bijbehorende artiesten verschijnen.

☆Opdracht 26 top 2000 v2: ALL en ANY

Bij veel zoekopdrachten zoeken we naar extreme waarden: *wie is het oudst? Welke mobiel is het duurste?* Voor een enkele dataset kun je daar gebruik maken van de MAX- of MIN-functie. Maar hoe vergelijk je meerdere selecties met elkaar?

De vraag
Welke artiesten hebben een nummer met de titel *Crazy* geschreven dat ouder is dan elk nummer met de titel *One*?
is te beantwoorden met de query:

FIGUUR 2.27

```
SELECT naam
FROM artiest, track
WHERE artiest.artiest_id = track.artiest_id
AND titel = 'Crazy'
AND uit < ALL
(
      SELECT uit
      FROM artiest, track
      WHERE artiest.artiest_id = track.artiest_id
      AND titel = 'One'
)
```

172. Maak (eerst) een overzicht van alle artiesten, titels en jaar van uitgave van liedjes met de titel *Crazy* of de titel *One*.
173. Laat het resultaat staan en open nogmaals de *querier* een tweede tabblad in je browser. Voer bovenstaande query uit. Bekijk het resultaat en onderzoek dit met het resultaat van de vorige vraag.
174. Ga naar https://www.w3schools.com/mysql/mysql_any_all.asp (of een website naar jouw keuze over het thema ALL en ANY) en bestudeer de betekenis van ALL en ANY.
175. Voorspel wat het resultaat van de query hierboven zal zijn als we het woord ALL vervangen door ANY. Controleer vervolgens je voorspelling.
176. Geef een alfabetisch overzicht van (unieke) artiesten waarvan de naam begint met een *y* die hoger hebben gestaan dan welke artiest dan ook van wie de naam begint met een *X*.

Eindopdrachten

De stof die je tot nu toe geleerd hebt in dit hoofdstuk hebben we steeds geoefend op drie bekende databases (*mobiel*, *top2000* en *oudstevrouwen*). Uiteindelijk moet je in staat zijn om het geleerde toe te passen op een nieuwe database.

Voor deze opdracht gebruiken we de database *wielrennen*.

177. Lees de casusbeschrijving van de database *wielrennen* aan het eind van deze module.
178. Open de database *wielrennen* in de *querier* en bestudeer alle attributen in alle tabellen. Maak op basis van de casusbeschrijving en de inhoud van de tabellen een strokendiagram van deze database. Kies daarbij ook een geschikte samengestelde sleutel voor de tabel *deelname*.

Bij de onderstaande vragen kun je gebruik maken van de *querier*, maar het beste kun je proberen om de opdrachten eerst schriftelijk te maken en daarna pas met behulp van de computer te testen of jouw antwoorden goed zijn. Tenzij anders vermeld is de opdracht om een query te schrijven.

179. Onderzoek voor welke kalenderjaren er informatie in deze database zit. Zorg voor een resultaattabel zonder dubbele resultaten die aflopend is.

Bauke Mollema (figuur 2.28) is een profwielrenner uit Zuidhorn. De volgende vragen gaan over hem.

FIGUUR 2.28

180. Maak een overzicht van alle deelnames van Bauke aan grote wielerrondes in deze database, oplopend gesorteerd op jaar.
181. Voor welk(-e) team(-s) reed Bauke in deze jaren? Breid de vorige query uit en maak gebruik van een *subquery*.
182. Bij elke ronde verdient Bauke *punten* voor deelnemers van het spel. Met welke opdracht krijg je alleen dit maximum te zien?
183. In welke ronde, inclusief jaartal, haalde Bauke de meeste punten voor deelnemers van het Tourspel? Zorg dat alleen deze informatie wordt getoond.

Voor de spelers van het spel is het aantal *punten* dat een renner haalt cruciaal. Echter, ook de *waarde* speelt een rol: betere renners hebben een hogere waarde. De totale waarde die een speler kan inzetten wanneer hij zijn eigen team opstelt in het spel, is beperkt. We gebruiken de volgende query:

```
SELECT rennerid, waarde, punten,
ROUND(10*punten/waarde)/10 AS ratio
FROM deelname WHERE waarde >= 10
HAVING ratio >= 25 ORDER BY ratio DESC
```

voornaam	achternaam	naam	waarde	wedstrijd	jaar	ratio
Jonas	Vingegaard	Jumbo - Visma	14	Tour	2022	41.0000
Remco	Evenepoel	Quick-Step - Alpha Vinyl Team	15	Vuelta	2022	40.9000
Wout	van Aert	Jumbo - Visma	14	Tour	2022	35.6000
Primož	Roglic	Jumbo - Visma	20	Vuelta	2020	34.7000
Primož	Roglic	Team Jumbo - Visma	18	Vuelta	2019	33.7000
Mads	Pedersen	Trek - Segafredo	11	Vuelta	2022	32.6000
Tadej	Pogacar	UAE Team Emirates	20	Tour	2021	32.1000
Egan	Bernal	INEOS Grenadiers	18	Giro	2021	31.8000
Tadej	Pogacar	UAE Team Emirates	20	Tour	2022	29.1000
Primož	Roglic	Jumbo - Visma	20	Vuelta	2021	27.8000
Tadej	Pogacar	UAE Team Emirates	18	Tour	2020	27.5000
Julian	Alaphilippe	Deceuninck - Quick-Step	17	Tour	2019	25.6000
Wilco	Kelderman	Team Sunweb	13	Giro	2020	25.0000

FIGUUR 2.29

184. Beschrijf in één of meerdere Nederlandse zinnen wat de uitkomst van bovenstaande query zal zijn.
185. De gebruikte query is uitgebreid om tot het resultaat in figuur 2.29 te komen. Wat was de query?

Het aantal vragen dat je op basis van deze database kunt stellen is legio. We geven nog wat voorbeelden:

186. Gebruik de LIKE-operator om te onderzoeken welke teams door het merk *Alpecin* zijn gesponsord.
187. Geef een overzicht van renners die door *Jumbo* zijn gesponsord.
188. Geef een overzicht van renners (met alleen voor en achternaam) die in één kalenderjaar drie grote wielerrondes reden.
189. Bedenk nu samen met je buurman of buurvrouw andere vraagstukken op basis van deze database. Probeer daarbij de stof uit alle paragrafen in te zetten. Uiteraard zorg je ook voor het antwoord!

CASUSSEN

Top 2000 versie 1

Elk jaar tussen Kerst en Oud & Nieuw kun je op NPO radio 2 luisteren naar de Top 2000 (https://www.nporadio2.nl/top2000) die wordt uitgezonden vanuit het Top 2000-cafè (zie foto). De top 2000 is een lijst met de favoriete liedjes van de luisteraars. Zij kunnen elk jaar opnieuw stemmen op hun favoriete nummers. De 2000 meest gekozen liedjes halen de jaarlijst.

In de database *top 2000 v1* staan alle liedjes die in 2017, 2018, 2019, 2020 of 2021 in de top 2000 hebben gestaan. Ze zijn verwerkt in één tabel met de *titel* van het liedje, de *artiest* die het liedje heeft uitgevoerd. Het veld *uit* geeft aan *uit* welk jaar het liedje komt, ofwel in welk jaar het liedje is gemaakt of uitgebracht. Er is elk jaar een nieuwe jaarlijst. De lijst van b.v. 2017 heet de editie 2017. Als een bepaald liedje in dat jaar in de top 2000 stond, bevat het record van het liedje voor het attribuut *ed2017* een getal dat de positie van dat liedje in dat jaar aangeeft. Als een liedje in een bepaald jaar niet in de lijst voorkomt, dan blijft het veld leeg (*NULL*).

Oudste vrouwen

Zo af en toe verschijnt het in het nieuws: de verjaardag of het overlijden van de oudste levende mens ter wereld. In de meeste gevallen, betreft dit een vrouw. Begin deze eeuw was de oudste levende mens op aarde een Nederlandse: Hendrikje *Henny* van Andel-Schipper (hiernaast op haar verjaardag) uit Hoogeveen. Henny werd liefst 115 jaar en 62 dagen oud.

Deze database bestaat uit één tabel met daarin de namen, geboorte- en sterfdatum, het land en de leeftijd van 20 van de oudste vrouwen die ooit geleefd hebben.

Voornamen

Het Meertens Instituut onderzoekt en documenteert taal en cultuur in Nederland en Nederlandse taal en cultuur in de wereld. Op hun website zijn verschillende databases (of *databanken* zoals zij ze noemen) te vinden, waaronder de Nederlandse Voornamenbank.

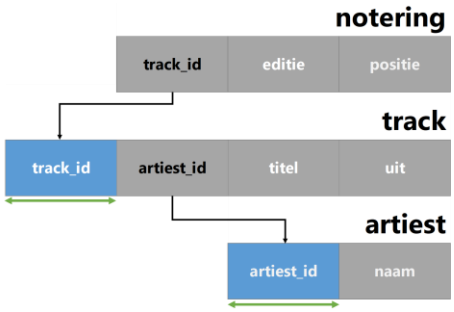
Deze Nederlandse Voornamenbank bevat 500.000 voornamen, waarvan de populariteit vanaf 1880 en de verspreiding over Nederland wordt getoond. Een wat beperktere set voornamen is te downloaden via de website. Deze dataset is gebruikt om een de database *voornamen* te maken met alleen namen die meer dan 500 keer zijn vergeven. Deze bestaat uit één tabel met vier attributen:

1. *positie* Dit zegt iets over hoe vaak een voornaam voorkomt ten opzichte van andere voornamen. Op positie 1 staat de meest voorkomende naam, positie 2 de één-na-meest voorkomende naam, etc. Omdat de ranking zowel voor meisjes- als voor jongensnamen wordt gemaakt, komen de posities 2× voor;
2. *naam* De voornaam behorende bij het record;
3. *aantal* Het aantal keren dat deze voornaam is geregistreerd;
4. *geslacht* Dit veld bevat *M* als het een jongensnaam betreft en *V* als het een meisjesnaam betreft.

Top 2000 versie 2

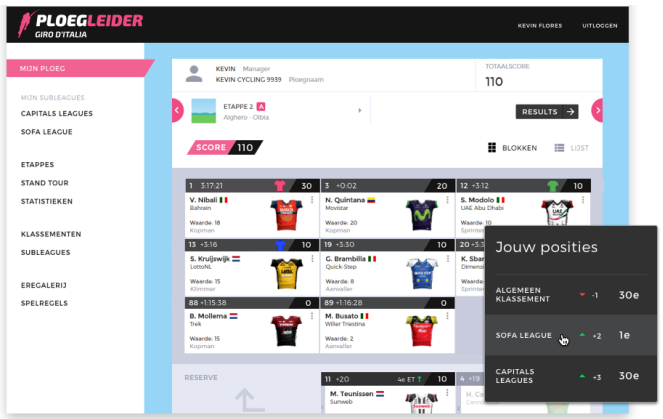
Elk jaar tussen Kerst en Oud & Nieuw kun je op NPO radio 2 luisteren naar de Top 2000 (https://www.nporadio2.nl/top2000) die wordt uitgezonden vanuit het Top 2000-cafè. De top 2000 is een lijst met de favoriete liedjes van de luisteraars. Zij kunnen elk jaar opnieuw stemmen op hun favoriete nummers. De 2000 meest gekozen liedjes halen de jaarlijst.

Hiernaast zie je het strokendiagram van de database met de tabellen *artiest*, *notering* en *track*. Artiesten hebben een naam en een uniek id. Zij kunnen met meerdere liedjes – die heten hier *tracks* – in de top 2000 staan. Als dat het geval is, heeft die track een *notering* in de top 2000-editie van dat jaar, waarin wordt verwezen naar het unieke *track_id* van het liedje. Het veld *editie* bevat dan het jaar waarin de track in de top 2000 stond en *positie* geeft aan op welke plek het liedje dat jaar in de lijst stond.



Wielrennen

Het profwielrennen kent drie grote wielerrondes: de Ronde van Frankrijk (*Tour*), de Ronde van Italië (*Giro*) en de Tonde van Spanje (*Vuelta*).



Wielerliefhebbers kunnen via www.ploegleider.nl een spel spelen, waarbij ze met een beperkt budget een aantal wielrenners kunnen *kopen*. Afhankelijk van de prestaties die deze gekozen wielrenners daarna halen, verdien je punten. Uiteraard wint degene die met de door hem of haar gekozen wielrenners in totaal de meeste punten heeft behaald.

De database bevat drie tabellen:

- *renner* met data over individuele renners
- *team* met data over wielerteams: een wielrenner neemt niet in zijn eentje deel aan een wedstrijd: een deelname is altijd namens een team.
- *deelname* geeft informatie over deze deelnames. Het is niet vanzelfsprekend dat elke wielrenner aan elke wedstrijd deelneemt. Ook kan het voorkomen dat een renner van team wisselt.