Databases en SQL

- Samenvatting Database Design
- SQL - deel 1 (SELECT WHERE)
- Opgaven SQL deel 1
- SQL - deel 2 (joins)
- Opgaven SQL deel 2
- SQL - deel 3 (aggregate functions)
- Opgaven SQL deel 3

Samenvatting Database Design

De 5 basisregels

1. Een **entiteit** is een persoon, ding of gebeurtenis. Een getal of bedrag (bijvoorbeeld gewicht) is nooit een entiteit, maar altijd een attribuut (=eigenschap) van een entiteit.

2. Elke entiteit heeft precies één **PK (primary key)**. De primary key maakt de entiteit uniek (bijvoorbeeld kenteken van een auto).

3. Entiteiten hebben de volgende **relaties** 1:1, 1:N, N:1 of N:M.
   - 1:1 relaties bestaan bijna niet, als ze voorkomen dan kun je de relaties samenvoegen.
   - 1:N en N:1 is eigenlijk hetzelfde en komen het meest voor.
   - N:M kun je alleen via een koppeltabel maken.

4. Een **1:N** relatie verbind je met een lijntje met een 'harkje'. Het **lijntje** staat aan de 1-kant en het '**harkje**' staat aan de meer-kant.

5. Bij elk 'harkje' hoort precies één FK. De FK verwijst naar de PK van de table waarmee deze is verbonden.

Video Workshop

De volgende video is een samenvatting van de Workshop Databases (1)

https://www.youtube.com/embed/MT-gW0Uo_IQ

https://youtu.be/MT-gW0Uo_IQ

Datatypes

De meest voorkomende datatypes zijn:

Datatype	Voorbeeld
int	-2 147 648 - 2 147 649
varchar(), bijv. varchar(20)	"Big Boss 12"
date	2022-04-01
datetime	2022-04-01 18:43:12
decimal(6,2)	1250,95

Relaties (1:N) tussen entiteiten

Een relatie tussen twee entiteiten is vrijwel altijd een 1-op-meer relatie en heeft daardoor aan één kant een 'harkje'.

> image-1643803461861.png
> ('harkje')

Voorbeelden

- één persoon bezit meer (één of meer) auto's en niet anders om een auto heeft geen twee eigenaren
- één klas bestaat uit meer studenten en niet een student zit in meer klassen
- één huiswerkopdracht is meerdere keren beantwoord (door één student, denk aan Canvas) en niet één antwoord hoort bij meerdere huiswerkopdrachten.
- één voetbalclub heeft meer spelers en niet één speler hoort bij meerdere voetbalclub
- één school heeft meerdere studenten en niet één student zit op meerdere scholen

Het gaat bij al deze zaken niet of het echt niet kan. Natuurlijk zou een student op meerdere scholen kunnen zitten, maar dat is niet de regel. Zou dit wel 'normaal' zijn, dan zouden we de relatie ook als meer-meer kunnen zien. Eén school heeft meer studenten en één student zit op meerdere scholen. Een meer-op-meer relatie wordt in DB Design Level 2 besproken.

Neem student en studiecoach. Welke regel denk je dat van toepassing is?

één studiecoach	meer-studenten
eén student	meer studiecoaches

OK in dit geval is het dus de eerste regel. Dat betekent dat de meer kant het 'harkje' krijgt. De lijn tussen de twee entiteiten in het ERD heeft dus het harkje aan de meer-kant. Bij het harkje hoort ook de FK.

	één - kant	meer - kant
PK	één unieke PK	één unieke PK
Lijntje	geen 'harkje' (alleen streep)	'harkje'
FK	geen FK	één FK verwijst naar de PK van de één kant

Het ERD wordt dan.

image-1643791334313.png

Beide entiteiten hebben een unieke PK en FK en harkje staan aan dezelfde kant.

Samengevat: het harkje staat aan de meer kant en bij elk harkje hoort een FK. Of nog korter:

> **HARKJE = MEER = FK**

N:M (veel-op-veel)

Stel je voor je hebt de entiteit, *product* en *klant*. De relatie tussen deze twee entiteiten is N:M, veel-op-veel. Eén klant kan immers meer producten kopen en een product kan door meerdere klanten worden gekocht.

Om dit in een database te zetten moet je een koppeltabel aanmaken. Dit is een extra entiteit. Dat ziet er zo uit:

image-1664052926462.png

(voor het gemak zijn in dit ERD de datatypen even weggelaten).

De entiteit product_klant is de koppeltabel en deze verbind het product en de klant aan elkaar zodat er een N:M relatie ontstaat.

Via de combinatie van de FK's worden producten aan klanten gekoppeld. Stel er is een klant met het id 101 en een klant met id 102 en stel je hebt een product met id 10 en 11. Stel klant 101 en klant 102 hebben allebei product 10 en 11 gekocht. Dan staat er in de koppel tabel de volgende informatie:

id (PK)	klant_id (FK)	product_id (FK)
1	101	10
2	102	11

3	101	10
4	102	11

Stappenplan maken ERD

Nog even alle stappen die je moet uitvoeren om een ERD te maken op een rijtje:

1. Bepaal alle entiteiten. Dit zijn personen, dingen of gebeurtenissen waar je gegevens over wilt vastleggen.

2. Bepaal van alle entiteiten de attributen (wat je vastleggen).

3. Bepaal de datatypes van alle attributen.

4. Zorg ervoor dat elke entiteit een PK krijgt (bij twijfel gebruik je 'id').

5. Bepaal de relaties tussen de entiteiten. Bij een N:M relatie heb je een koppeltabel nodig.

6. Teken de relaties. Het harkje staat aan de meer-kant.

7. Bij elk harkje hoort een FK, de FK verwijst naar de PK met de entiteit waarmee de relatie bestaat.

8. Lees het verhaal nog een keer door en controleer of je alles wat er in het verhaal staat kunt vastleggen in je databaseontwerp.

Checklist, wat gaat er vaak fout?

1. Heeft elke entiteit precies één PK (*Primar yKey*)?

2. Is elke PK uniek, dus kan er maar één van voorkomen?
   (Achternaam kan bijvoorbeeld *geen* PK zijn, omdat er meer mensen zijn met dezelfde achternaam).
   Tip: meestal zijn PK's int en vaak heten ze gewoon id.

3. Heeft elke attribuut een datatype?

4. Telefoonnummer is geen int want dan valt de eerst 0 weg, immers 0612341234 wordt 612341234

5. Datum is altijd datatype date.

6. Datum plus tijd is datatype datetime.

7. De relatie heeft maximaal één 'harkje'. Het harkje staat aan de 'meer' kant. Dus een *student* 'hoort' bij één *klas* en bij een *klas* 'horen' meerdere *studenten*. Het harkje staat in dit voorbeeld dan aan de student kant.

8. Bij elk 'harkje' hoort precies één FK. De FK verwijst naar de PK van de table waarmee deze is verbonden.

9. De PK en FK die bij elkaar horen hebben hetzelfde datatype.

10. int heeft een vaste lengte, het is dus int en niet int(11).

11. varchar heeft altijd een lengte, dit is de *maximaal* lengte die kan voorkomen. varchar(5) als plaatsnaam is dus fout.

12. String datatypes zijn er in verschillende vormen in Database-land. De meest gebruikte zijn: char en varchar; char heeft een vaste lengte, varchar heeft een maximale lengte. *String* bestaat niet in MySQL.

Powerpoint

Twee powerpoints:

1. Database I
2. Database II

--

Perma-link: https://www.roc.ovh/link/459

SQL - deel 1 (SELECT WHERE)

Inleiding

We kijken naar een database met drie entiteiten; **student**, **vak** en een **koppeltabel**.

De relatie *student-vak* is **veel-op-veel (N:M)** want één student kan meer vakken hebben en één vak kan bij meer studenten horen.

We hebben dus een N:M relatie en hebben daarbij een **koppeltabel** nodig.

Als we het **cijfer** van een student voor een vak willen opslaan dan moet dat in de koppeltabel. Het cijfer gaat namelijk over de combinatie *student-vak*. Alles wat over de combinatie gaat moet in de koppeltabel.

We hebben dus het volgende ERD.



Database maken

Start MariaDB (MySQL) in XAMPP en ga naar http://localhost/phpmyadmin

Je kunt ook op de admin in XAMPP drukken:



- In phpmyadmin, maak je een nieuwe database en noem die **student**.

  

- Download de file student.sql en importeer deze.

  

- Controleer of de database is geïmporteerd. De koppeltabel in heet in deze database *cijfer* (dat is omdat in de koppeltabel het cijfer is opgeslagen; het is en blijft ook de *koppeling* tussen *student* en *vak*)

  

- Klap de drie entiteiten (cijfer, student en vak) open en kijk of je alle kolommen ziet.

Select

Om gegevens uit de database te halen is een aparte taal bedacht. Dit heet SQL. Een SQL-programmaatje wordt een query genoemd en telt meestal maar een paar regel.

De meeste query's halen gegeven uit de database en de meest eenvoudige query haalt alle gegevens uit één entiteit. Stel, je wilt alle informatie van alle studenten zien, dan is dit de query:

```
SELECT * FROM student
```

> SELECT betekent; haal de gegevens op.
>
> * betekent *alle* kolommen
>
> FROM betekent dat er een tabelnaam (entiteitsnaam) volgt en *student* is dan de naam van de tabel.

Voer de query uit in *phpmyadmin* onder het tabje SQL.

> image-1676379711258.png

Zorg er voor dat je de juiste database hebt geselecteerd (hebt aangeklikt).

Je hebt nu een * in de query gebruikt. Dat betekent dat je alle velden laat zien. Stel dat je alleen de voornaam en het email adres wilt afdrukken dan kan je dat als volgt.

```
SELECT voornaam, email FROM student
```

Aliasen (AS)

Stel dat je nu de kolomnamen wilt veranderen dan kan je aliassen gebruiken. Stel de kolomnaam *voornaam* veranderen in *Naam*, en *email* in *eMail* dan kan dat als volgt.

```
SELECT voornaam as 'Naam', email as 'eMail' FROM student
```

WHERE

Stel je wilt alle gegevens van de student met de voornaam *Grazia* zien. Dat doe je door een WHERE te gebruiken.

De standaard query wordt als volgt opgebouwd

```
SELECT kolomnaam1, kolomnaam2, .... of * voor alles
FROM tabelnaam
WHERE de conditie
```

De query die alle gegeven van de student *Grazia* opzoekt wordt dan

```
SELECT * FROM `student`
WHERE voornaam='Grazia'
```

Let op dat *Grazia* een string is en daarom tussen '(singel quotes) moet worden gezet.

Wildcards (LIKE)

Een conditie kan ook een zogenaamde wildcard bevatten. Een wildcard is bijvoorbeeld alle namen die met een a beginnen. Daarvoor gebruik je in plaats van de = een like en je gebruikt bijvoorbeeld 'a%' om aan te geven dat de naam met een a moet beginnen. Of bijvoorbeeld '%t' om aan te geven dat de naam met een t moet eindigen.

OK, stel willen alle gegevens van alles studenten zien van wie de voornaam met een a begint.

```
SELECT * FROM student
WHERE voornaam LIKE 'a%'
```

Hoeveel studenten hebben een voornaam die met een a begint?

Nog een voorbeeld, stel willen alle gegevens van alles studenten met een email adres dat met *.com* eindigt. Hoe doe je dat?

```
SELECT * FROM student
WHERE email  like '%.com'
```

AND en OR

Stel we willen alle gegevens van alle studenten waarvan de voornaam met een a of een b of een c begint. Dan kunnen we met OR (net als in PHP) deze condities combineren.

```
SELECT * FROM student
where voornaam like 'a%' OR voornaam like 'b%' OR voornaam like 'c%'
```

Stel dat je in deze bovenstaande query per ongeluk een AND had gebruikt, wat zou er dan gebeuren? Als je wilt kan je het uitproberen!

Stel we willen alle gegevens van alles studenten van wie de voornaam met een a begint én van wie het e-mailadres op *.com* eindigt.

Daarvoor hebben we een AND nodig om de twee condities te combineren.

```
SELECT * FROM student
where voornaam like 'a%' AND email like '%.com'
```

Nu een lastige. Stel we willen de studenten van wie de voornaam met een a, b of c begint en van wie het e-mailadres op *.com* eindigt. We moeten nu de AND en OR gaan combineren.

Als je dat doet dan is er een regel en dat is dat je de OR tussen haakjes moet zetten omdat die bij elkaar hoort.

Dus de gecombineerde query wordt dan;

```
SELECT * FROM student
where ( voornaam like 'a%' OR voornaam like 'b%' OR voornaam like 'c%' )
       AND email like '%.com'
```

Opgave 1

Maak een query waarin alle voornamen, achternamen en e-mailadressen van alle studenten worden getoond waarvan de voornaam met een *m* begint en waarvan het email adres eindigt met *.gov* of *.org*

Als je het goed doet krijg je drie resultaten. Welke?

Inleveren

Screenshot van je resultaten. Maak een screenshot van je gehele browser.

JOIN

FUNCTIONS

INSERT

UPDATE

DELETE

---

image-1676378580643.png

image-1676378514972.pngc22

Opgaven SQL deel 1

Gebruik, bij deze opgaven de studentendatabase. In de vorige les wordt beschreven hoe je deze kunt importeren.

In de vorige les wordt ook alles uitgelegd met voorbeelden.

Opgave 1

Maak een query waarin je alleen de voornaam en achternaam van alle studenten met een achternaam *Seaborne* laat zien.

Inleveren

Screenshot van je resultaten. Maak een screenshot van je gehele browser.

Opgave 2

Maak een query waarin alle voornamen, achternamen en e-mailadressen van alle studenten worden getoond waarvan waarvan het email adres eindigt met *.gov*

Inleveren

Screenshot van je resultaten. Maak een screenshot van je gehele browser.

Opgave 3

Maak een query waarin alle voornamen, achternamen en e-mailadressen van alle studenten worden getoond waarvan de voornaam met een *m* begint.

Inleveren

Screenshot van je resultaten. Maak een screenshot van je gehele browser.

Opgave 4

Maak een query waarin alle gegevens van alle studenten worden getoond waarvan de voornaam waarvan het email adres eindigt met *.gov* of *.org*

Inleveren

Screenshot van je resultaten. Maak een screenshot van je gehele browser.

Opgave 5

Maak een query waarin alle voornamen, achternamen en e-mailadressen van alle studenten worden getoond waarvan de voornaam met een *m* begint en waarvan het email adres eindigt met *.gov* of *.org*

Als je het goed doet krijg je drie resultaten. Welke?

Inleveren

Screenshot van je resultaten. Maak een screenshot van je gehele browser.

SQL - deel 2 (joins)

We gaan door met de student pagina uit deze les.

ERD

Deze database heeft drie tabellen.

image-1676378321011.png

We gaan kijken naar de koppel tabel. In de database heet deze tabel cijfer.

image-1676378544439.png

Tabel cijfer

We gaan nu alle gegevens selecteren uit de cijfertabel waarbij de student een 10 gehaald heeft.

Let op dat de cijfers in de database staan als getal tussen 0 en 100. 50 betekent een 5, 85 betekent een 8.5 en een 100 betekent dus een 10.

Voer de volgende query uit om alle 10's uit de cijfers tabel te selecteren.

```
SELECT *
FROM cijfer
WHERE cijfer = 100
```

Als het goed is krijg je 10 regels terug.

Op één van deze regels staat het volgende:

image-1676395284323.png

We zien hier dus dat student met id=10 een cijfer 100 (dat is dus een 10) heeft gehaald voor vak met id=2.

Laten we eens kijken welke student dit is.

```
SELECT *
FROM student
WHERE id = 10
```

En voor welk vak is dit?

```
SELECT *
FROM vak
WHERE id = 2
```

INNER JOIN

Wat we nu met deze drie queries hebben gedaan is het samenvoegen van de informatie van de drie tabellen.

Het *student_id* en *vak_id* uit de *cijfer* tabel zijn FK's (foreign keys) en die verwijzen naar de PK's (primary keys) van de gekoppelde tabellen *student* en *vak*.

Laten we de tabellen automatisch gaan combineren. Latne we eerst de cijfer tabel met de student tabel combineren.

```
SELECT *
FROM cijfer
INNER JOIN student ON student.id = cijfer.student_id
WHERE cijfer = 100
```

We hebben verschillende soorten JOIN maar standaard gebruiken we een INNER JOIN. Later wordt nog uitgelegd welke andere joins er zijn.

We maken dus een verbinding met de tabel student door het commando INNER JOIN. Dan moeten we nog wel vertellen hoe de tabellen zijn gekoppeld, wat is de FK en welke PK hoort daarbij? Dat zetten we achter de ON.

Zie je dat er *student.id* staat, dat is de kolom *id* uit de tabel *student* en dat is de PK van deze tabel. Deze koppelen we aan de FK uit de *cijfer* tabel, *cijfer.student_id*.

Je ziet nu dus ook dat als er meer tabellen zijn het gebruikelijk is om de tabelnaam voor de kolomnaam te zetten. Dit voorkomt verwarring (achter de WHERE zou je ook *cijfer.cijfer* mogen schrijven).

We gaan nu de tabel *vak* ook koppelen aan *cijfer* tabel.

```
SELECT *
FROM cijfer
INNER JOIN student on student.id = cijfer.student_id
INNER JOIN vak on vak.id = cijfer.vak_id
WHERE cijfer = 100
```

We zien nu alle velden uit alle tabellen, dat is wat onoverzichtelijk. Laten we alleen de naam, vak en cijfer afdrukken en laten we gelijk nette aliassen gebruiken. Die aliassen zijn in het vorige deel uitgelegd.

```
SELECT student.voornaam 'Voornaam',
  student.achternaam 'Achternaam',
        vak.naam 'naam',
        round(cijfer.cijfer/10,1) 'cijfer'
FROM cijfer
INNER JOIN student on student.id = cijfer.student_id
INNER JOIN vak on vak.id = cijfer.vak_id
WHERE cijfer = 100
```

Op regel 4 delen we het cijfer door 10 (cijfer.cijfer/10) en ronden af op één decimaal met de *round* functie. Dit werkt ongeveer hetzelfde als in PHP.

En bij welk vak zijn de meeste 10-en gevallen?

Soorten JOINS

Twee tabellen

Student				**Cijfer**		
ID (**PK**)	voornaam	achternaam		ID (PK)	student_ID (**FK**)	cijfer
1	Paddie	Durban		1	1	5
2	Rosemonde	Surr		2	2	6
4	Kenyon	Russilll		3	4	9
6	Shaylyn	Bilbrey		4	12	7
7	Terrijo	Anear		5	13	8
				6	14	7

(INNER) JOIN

Resultaat is alleen de rijen die een match hebben, dus waarbij de FK en PK hetzelfde is.

Paddie	Durban	5
Rosemonde	Surr	6

Kenyon	Russillll	9
Shaylyn	Bilbrey	7

LEFT (OUTER) JOIN

Resultaat is alle rijen uit de 'linker' tabel. De linker table is de tabel die direct na de WHERE staat.

Paddie	Durban	5
Rosemonde	Surr	6
Kenyon	Russillll	*NULL*
Shaylyn	Bilbrey	7
Terrijo	Anear	*NULL*

RIGHT (OUTER) JOIN

Resultaat is alle rijen uit de 'rechter' tabel. De rechter table is de tabel die in de JOIN staat.

Paddie	Durban	5
Rosemonde	Surr	6
Kenyon	Russillll	9
NULL	*NULL*	7
NULL	*NULL*	8
NULL	*NULL*	7

Opgaven SQL deel 2

Gebruik, bij deze opgaven de studentendatabase. In de eerste les wordt beschreven hoe je deze kunt importeren.

In de vorige les wordt ook de INNER JOIN uitgelegd met voorbeelden. Deze INNER JOIN heb je nodig bij deze opgaven.

Opgave 1, een 1.7

Maak een query waarin je laat zien hoe vaak er een 1.7 voor een vak is gegeven. Je kijkt hiervoor alleen in de *cijfer* tabel.

Inleveren

Screenshot van je resultaten. Maak een screenshot van je gehele browser.

Opgave 2, koppel met student

Maak een query waarin je laat zien hoe vaak er een 1.7 voor een vak is gegeven en maak nu een koppeling met een INNER JOIN naar de student tabel. Je ziet nu dus alle studenten die een 1.7 hebben gehaald.

Inleveren

Screenshot van je resultaten. Maak een screenshot van je gehele browser.

Opgave 3, koppel met vak

Voeg aan de query van opgave 2 ook de gegevens van het vak toe. Je kan nu dus zien wie voor welk vak een 1.7 hebben gehaald.

Inleveren

Screenshot van je resultaten. Maak een screenshot van je gehele browser.

Opgave 4, alles netjes maken

Gebruik de query van opgave 3, maar laat nu alleen de voornaam, achternaam, vaknaam en cijfer zien. In plaats van cijfer druk je cijfer/10 af en rond af op één decimaal. Je krijgt op deze manier een nettere output. Gebruik aliassen zodat de kolomnamen er netjes uit zien.

Inleveren

Screenshot van je resultaten. Maak een screenshot van je gehele browser.

Opgave 5, 10 voor PHP?

Hoeveel studenten hebben een 10 gehaald voor het vak PHP?

Inleveren

Antwoord en een uitleg hoe je aan het antwoord bent gekomen.

Opgave 6, 10 voor PHP?

Laat iedereen zien die een cijfer hoger dan een 9.5 heeft gehaald voor het vak Nederlands.

Druk alleen de voornaam, achternaam, cijfer en vaknaam af. Gebruik aliassen en druk het cijfer al als een decimaal getal (dus bijvoorbeeld 9.8 en niet 98).

Inleveren

Screenshot van je resultaten. Maak een screenshot van je gehele browser.

SQL - deel 3 (aggregate functions)

Gebruik, bij deze opgaven de studentendatabase. In de eerste les wordt beschreven hoe je deze kunt importeren.

In de vorige les wordt de INNER JOIN uitgelegd met voorbeelden. De INNER JOIN gebruiken we in deze les.

Stel we drukken alle cijfers af die voor Engels zijn behaald, dat kan met de volgende query.

```
SELECT cijfer
FROM cijfer
INNER JOIN vak ON vak.id=cijfer.vak_id
WHERE vak.naam='Engels'
```

Je ziet nu de lijst met cijfers (zonder studentnamen) die zijn behaald.

Stel we willen het gemiddelde weten van alle cijfers die voor Engels zijn behaald. Dat kan met de (aggregate) functie *avg*.

```
SELECT vak.naam, avg(cijfer/10)
FROM cijfer
INNER JOIN vak ON vak.id=cijfer.vak_id
WHERE vak.naam='Engels'
```

Voor het gemak hebben we het cijfer ook (weer) door 10 gedeeld omdat het als getal tussen 0 en 100 in de database staat. We hebben ook de vaknaam afgedrukt (beetje onlogisch want we hebben het hier alleen over Engels.

Je ziet nu dat het gemiddelde voor Engels een 6.9 is.

OK stel dat we de WHERE nu weglaten zodat we het gemiddeld van alle vakken krijgen wat gebeurt er dan?

Probeer maar, we krijgen het gemiddelde van (maar) één vak? Hoe kan dat?

Dat kom omdat we in de query moeten vertellen dat we per vak(naam) een gemiddelde willen zien. Dit doen we met de GROUP BY.

```
SELECT vak.naam, avg(cijfer/10)
FROM cijfer
INNER JOIN vak ON vak.id=cijfer.vak_id
GROUP BY vak.naam
```

Laten we ook nog sorteren op vak.naam, voeg aan het eind `ORDER BY vak.naam` toe.

.....

.....

.....

Opgaven SQL deel 3

...

...

...