Tabellen

Een database bestaat uit genormaliseerde tabellen. Normalisatie is een onderwerp die we later zullen bekijken. Het komt er op neer dat iedere tabel zijn eigen specifieke data heeft en gerelateerd kan zijn met andere tabellen in de database.
Voorbeeld:
De tabel met gebruikers kan gerelateerd zijn met de tabel van adressen.
Een gebruiker kan dus één of meerdere adressen hebben.

Datatypes

In een tabel bestaat elk **veld (kolom)** uit een bepaald **datatype**. Het datatype bepaalt welke soort gegevens in dat veld kunnen worden opgeslagen: getallen, tekst, datums, enz. Het correct kiezen van een datatype is essentieel voor **prestaties, opslagruimte en datakwaliteit**.

1. Numerieke datatypes

Type	Bereik (signed)	Bereik (unsigned)	Typische toepassing
TINYINT	–128 tot 127	0 tot 255	Leeftijd, kleine codes
SMALLINT	–32.768 tot 32.767	0 tot 65.535	Jaar, aantal stuks
MEDIUMINT	–8,3 miljoen tot 8,3 miljoen	0 tot 16,7 miljoen	Populatie, telling
INT	–2,1 miljard tot 2,1 miljard	0 tot 4,2 miljard	ID's, primaire sleutels
BIGINT	–2^63 tot 2^63 – 1	0 tot 2^64 – 1	Grote tellingen, financiële transacties

Gebruik UNSIGNED wanneer negatieve waarden onmogelijk zijn.

2. Decimale en floating-point getallen

Type	Precisie	Typische toepassing
FLOAT	7 cijfers (benadering, single precision)	Wetenschappelijke berekeningen
DOUBLE	15–16 cijfers (benadering, double precision)	Statistiek, meetwaarden
DECIMAL	Tot 65 cijfers voor de komma en 30 erna (exact)	Geldbedragen, prijzen

Belangrijk:

- FLOAT en DOUBLE zijn **benaderingen** → kans op afrondingsfouten.
- DECIMAL is exact → altijd gebruiken voor **geld en facturen.**

3. Tekst- en stringdatatypes

Type	Eigenschappen	Typische toepassing
CHAR(n)	Vaste lengte, max. 255 tekens	Postcode, landcode
VARCHAR(n)	Variabele lengte, max. 65.535 tekens (afhankelijk van charset en opslag)	Namen, e-mails
TEXT	Lange tekst (tot 4 GB)	Artikels, beschrijvingen
ENUM	Eén waarde uit een vaste lijst	Status: 'active', 'inactive'
BLOB	Binaire data, max. 4 GB	Afbeeldingen, audio, bestanden

Gebruik CHAR alleen voor velden met **altijd vaste lengte** (bv. landcode: "BE"). Voor alles wat variabel is, gebruik je VARCHAR.

4. Datum- en tijdsdatatypes

Type	Formaat	Voorbeeld	Typische toepassing
DATE	YYYY-MM-DD	2025-09-29	Geboortedatum
TIME	hh:mm:ss	14:35:00	Tijdstippen
DATETIME	YYYY-MM-DD hh:mm:ss	2025-09-29 14:35:00	Geboortedatum + tijd
TIMESTAMP	YYYY-MM-DD hh:mm:ss	2025-09-29 14:35:00	Automatische logtijden
YEAR	YYYY	2025	Kalenderjaar

Verschil **DATETIME vs TIMESTAMP**:

- DATETIME → onafhankelijk van tijdzone.
- TIMESTAMP → wordt automatisch aangepast aan de ingestelde tijdzone en kan zichzelf updaten bij wijzigingen.

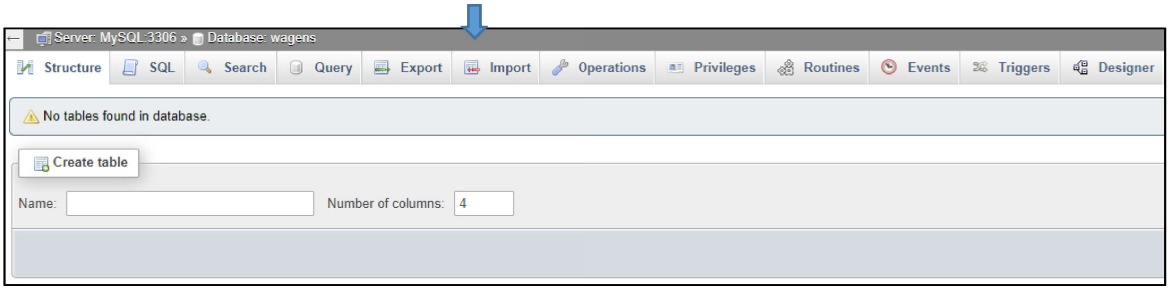
5. Praktische keuzes

In de praktijk gebruik je meestal:

- **INT UNSIGNED** → primaire sleutels (id).
- **DECIMAL(10,2)** → geld (10 cijfers totaal, 2 na de komma).
- **VARCHAR(255)** → tekst zoals namen, e-mails.
- **TEXT** → langere beschrijvingen.
- **DATETIME** of **TIMESTAMP** → wanneer je ook tijd nodig hebt.

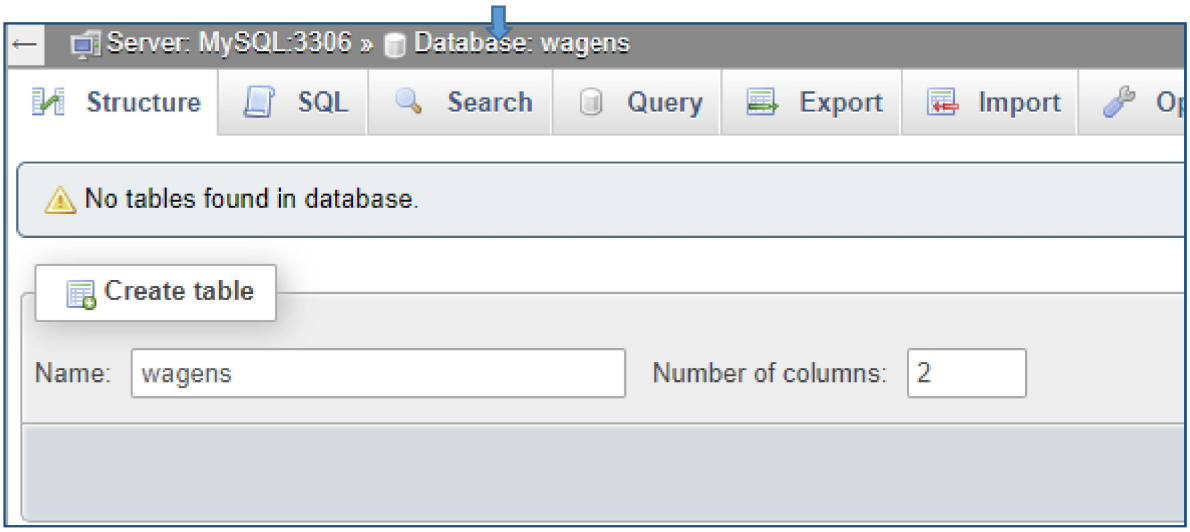
Nieuw

Een nieuwe tabel aanmaken is eenvoudig. Je dient eerst je database te selecteren. Je kan steeds controleren of je in de correcte database zit bovenaan PHPmyadmin. Net zoals in windows verkenner is dit een folder structuur die vertrekt van het server niveau.

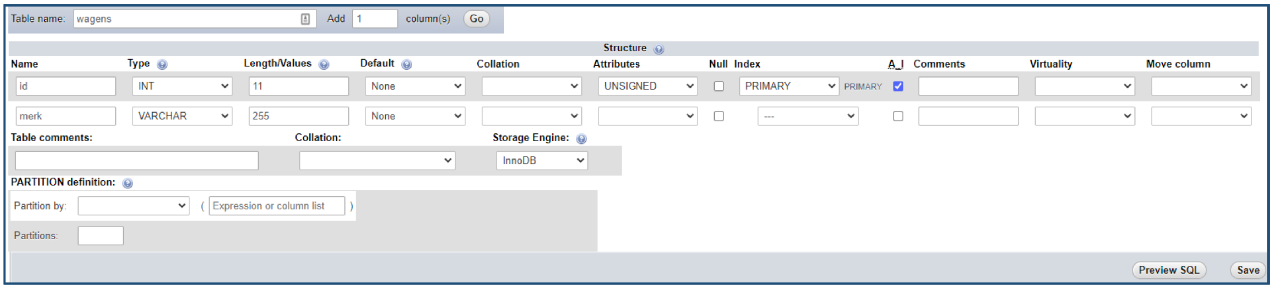


In ons voorbeeld hebben we het over wagens. Wagens bestaan uit merken en modellen. Hiervoor creëeren we dus 2 tabellen.

De eerste tabel wagens zal 2 velden bevatten: id, merk.
Vul de **naam** van de tabel in en duid hiervoor **2** kolommen aan voor het aantal velden. Klik dan op **Go**.



Conventie bij het schrijven van veldnamen: ALLEMAAL KLEINE LETTERS! GEEN CAMELCASE zoals variabelen in PROGRAMMEERTALEN.
Vul in zoals onderstaande afbeelding en klik op SAVE. Je hebt nu een lege tabel zonder data aangemaakt in de database van wagens.



Veld instellingen

Hier zullen we kort alle mogelijke instellingen bespreken tijdens het maken van velden in een tabel.

Name

Hier worden de kolomkoppen of veldnamen ingegeven. De eerste **veldnaam** die wij bijna altijd zullen invullen is het **id** van de tabel. Het **id** is ook meestal een oplopende nummering die start van 1 met een **auto_increment.** Auto_increment zorgt dus m.a.w. automatisch dat een volgend **record** automatisch verder wordt genummerd. Dit id is **UNIEK** en wordt de **PRIMARY KEY** van een tabel genoemd.

Een **record** is één enkele data-lijn in de database met ingevulde data per veld.

Type

Het type per veld is het datatype die we dienen te bepalen. Aangezien een id een oplopende nummering is zal dit van het type integer zijn. Hier kiezen we **INT.**

Een VARCHAR kan bijvoorbeeld gebruikt worden voor een string, DATETIME voor een datum, ...

Length/values

Aangezien het datatype bijvoorbeeld INT is kunnen we tot ca 2 billioen in cijfers laten oplopen**.** Dankzij lengte beperken we het aantal karakters toch. Veelal wordt er bij een id 11 karakters aan lengte toegevoegd die voor de meeste database meer dan voldoende is.

Het laatste id zou dus in principe 99999999999 kunnen zijn.

Default values

Hiermee kan je een standaard waarde meegeven wanneer een veld niet zou worden ingevuld door een gebruiker.

Er zijn 4 opties:

none = is default en blijft dus leeg

NULL = hier wordt een nullable in het veld geschreven. Je zal dan het woord NULL zien staan in het veld.

As defined: hier verschijnt een extra veld waar je zelf een default waarde kan schrijven. Deze default waarde wordt in het veld van de database weggeschreven wanneer de gebruiker niets zou hebben geschreven.

Current_time_stamp = Zal de datumtijdstempel wegschrijven van de pc.

Collation

Voor ieder veld kan je een aparte character set toevoegen indien je dit nodig zou hebben. In de meeste gevallen is de initiële character set van de database voldoende en blijft dit veld leeg.

Attributes

BINARY

De opslag van een ingetikte waarde zal binair gebeuren (1 en 0)

UNSIGNED

Een unsigned value kan enkel positieve getallen opslaan in de database.

Zie tabel datatypes (voorgaand).

UNSIGNED ZERO FILL

Zero fill zorgt ervoor dat alle ontbrekende characters met een 0 worden opgevuld.

Bijvoorbeeld: id(11)

id(11) kan een lengte van 11 getallen opslaan. Wanneer we nu het getal 1 opslaan dan wordt dit met zero fill: 00000000001

on update CURRENT_TIMESTAMP

Wanneer er een veld van het datatype datetime wordt gebruikt dan kunnen we hier ervoor zorgen dat automatisch de huidige datumtijdstempel van de pc wordt gebruikt wanneer een gebruiker een wijziging aanbrengt op dit record.

Null
Het veld mag leeg zijn wanneer aangevinkt.
Index
PRIMARY
Met deze optie kan je een veld de primaire sleutel maken van je tabel (PK = Primary KEY)
In combinatie met A-I (auto increment), maak je een primaire sleutel dus aan die daarnaast ook uniek is
UNIQUE
Wanneer je een veld UNIQUE zet, dan zal je dubbele waarden vermijden in een database. Dit veld kan wel een NULL value bevatten.
INDEX
MySQL gebruikt indexen om snel rijen met specifieke kolomwaarden te vinden. Zonder index moet MySQL de hele tabel scannen om de relevante rijen te lokaliseren. Hoe groter de tafel, hoe langzamer hij zoekt.
A_I
Wanneer aangevinkt zorgt A_I voor een oplopende nummering van een veld met een datatype integer.
COMMENTS
Hier kan je als developer een beschrijving geven van de bedoeling van een veld.

Code: CREATE TABLE

Een tabel kan je ook met code aanmaken. We maken dus hier gebruik van de MySQL-taal.
Oefening

Voorbeeld:

```
CREATE TABLE Klanten (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
voornaam VARCHAR(30) NOT NULL,
familienaam VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
)
```

Na het gegevenstype kunt u voor elke kolom andere optionele attributen specificeren:

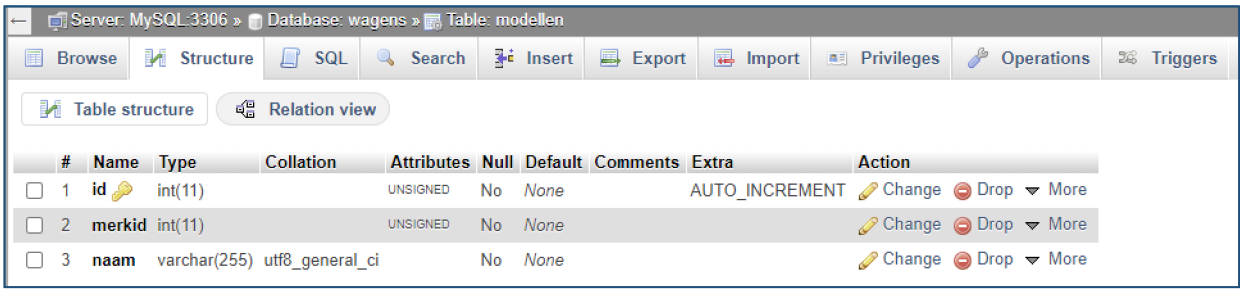
- NOT NULL - Elke rij moet een waarde voor die kolom bevatten, null-waarden zijn niet toegestaan
- DEFAULT-waarde - Stel een standaardwaarde in die wordt toegevoegd als er geen andere waarde wordt doorgegeven
- UNSIGNED - Gebruikt voor number datatypes , beperkt de opgeslagen gegevens tot positieve getallen en nul
- AUTO INCREMENT - MySQL verhoogt automatisch de waarde van het veld met 1 telkens wanneer een nieuw record wordt toegevoegd
- PRIMARY KEY - Wordt gebruikt om de rijen in een tabel uniek te identificeren. De kolom met PRIMARY KEY-instelling is vaak een ID-nummer en wordt vaak gebruikt met AUTO_INCREMENT

Opdracht

Maak een nieuwe tabel **modellen** aan in de tabel wagens. Zorg dat er 3 velden aanwezig zijn. Het eerste veld **id** is een integer die positieve primaire sleutel is met autonummering en een grootte van 11 heeft. Het tweede veld is **merkid** en is een positieve integer met een grootte van 11 en kan geen null bevatten.
Het derde veld is naam en zal alle modelnamen bevatten. Dit is een string met een grootte van 255 en kan geen null bevatten.

Oplossing:

```
CREATE TABLE merken (
id INT(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
merkid INT(11) UNSIGNED NOT NULL,
naam VARCHAR(255) NOT NULL
)
```



Momenteel hebben we dus een database met de naam wagens. In deze database hebben we 2 tabellen, nl. merken en modellen. Wat we nog NIET hebben is een relatie tussen deze 2 tabellen.

Relaties

We sommen eerst de meest voorkomende relaties op die mogelijk zijn in een database tussen 2 of meerdere tabellen.
- één-op-één
- één-op-veel
- veel-op-veel

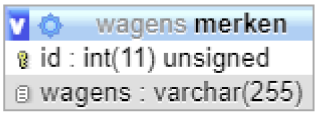
Relatie: één-op-veel

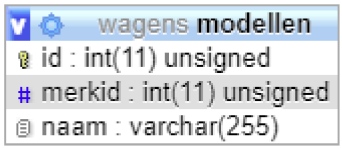
In ons voorbeeld hebben we bijvoorbeeld een model: AUDI
Audi heeft meerdere modellen in zijn gamma: A3, A4, A5,...

Dit noemen we een één-op-veel-relatie, want het merk audi (één) heeft meerdere modellen (veel).

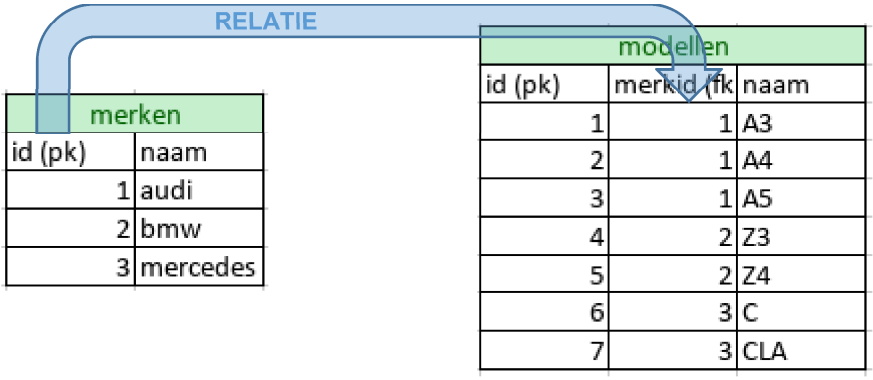
Een relatie dienen we dus ook te definiëren. Momenteel hebben we reeds alle velden in de tabellen gedefinieerd, maar nog niet de relatie.

Wanneer we de tabellen even schematisch bekijken in de designer tab op het niveau van de database dan kan je dit zien. Er is duidelijk geen link dus deze 2 tabellen. Deze lijken te 'zweven' naast elkaar





Hieronder zie je wat de bedoeling zou moeten zijn in een **één-op-veel** relatie datagewijs.



merken

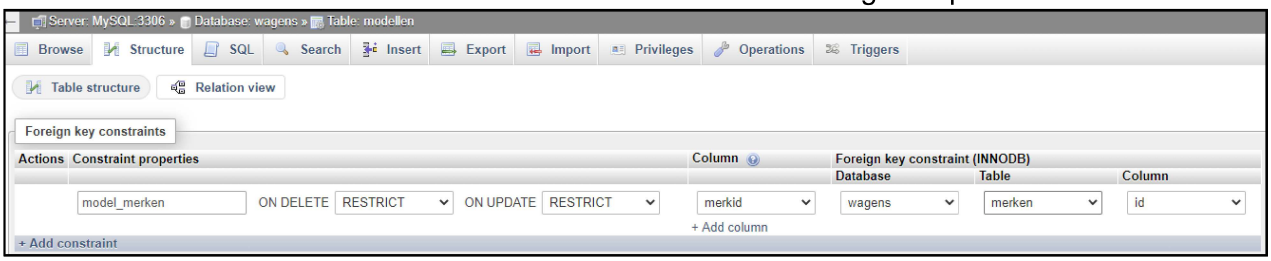
id (pk)	naam
1	audi
2	bmw
3	mercedes

modellen

id (pk)	merkid (fk	naam
1	1	A3
2	1	A4
3	1	A5
4	2	Z3
5	2	Z4
6	3	C
7	3	CLA

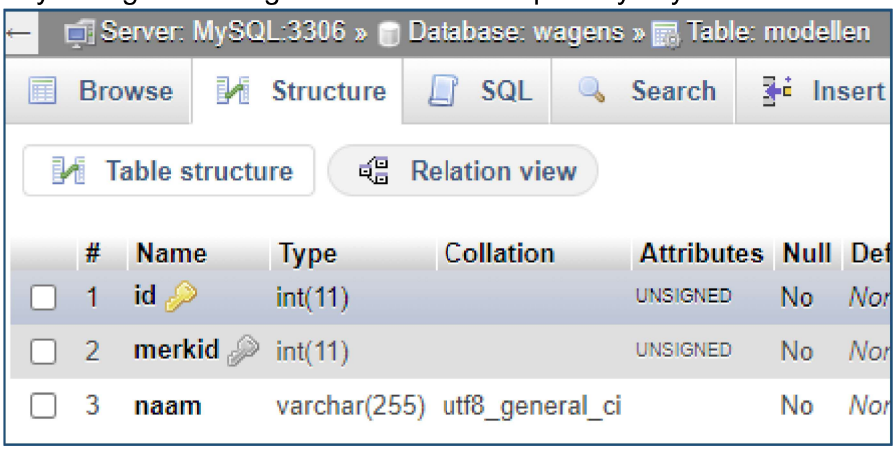
Zoals je kan zien hebben beide tabellen een unieke primary key. Het merkid echter is een sleutel die vreemd is aan de tabel modellen en die werd geërfd van de tabel merken. Deze sleutel noemen we een FOREIGN KEY.

Om nu de relatie te bewerkstelligen in PHPmyadmin dienen we de TWEEDE tabel modellen te selecteren en het tabblad STRUCTURE aan te klikken. Klik vervolgens op RELATION VIEW.



De opties die mogelijk zijn bij "on delete en on update" zijn:
RESTRICT
Wanneer je data die verbonden is in 2 tabellen zou willen wissen, dan zal dit niet lukken. Wanneer je bijvoorbeeld audi in de tabel merken zou willen zal de database een restriction error geven omdat je modellen van audi in de tabel modellen zitten hebt.
CASCADE
Cascade zal wel wissen. Wanneer je audi zou wissen in de tabel merken dan zullen al zijn modellen in de tabel modellen ook gewist worden.
SET NULL
Wanneer je audi zou wissen in de tabel merken dan zullen de rijen in de tabel modellen niet gewist worden maar zal de inhoud van de rij op NULL worden geplaatst.

Wanneer we nu kijken naar de tabel modellen dan zie je een grijze sleutel staan die de foreign key weergeeft. Een gouden sleutel is de primary key van de tabel zelf.



Op het database niveau kan je terug naar het designer tabblad gaan en zie je het uiteindelijke resultaat van een één-op-veel relatie.

Code: FOREIGN KEY CONSTRAINTS

Wanneer je alle bovenstaande acties in code zou willen uitvoeren dan kan dit natuurlijk ook.
WIS de tabel modellen uit de database.
Open nu het SQL tabblad op database niveau en voeg onderstaande code toe:

```
CREATE TABLE modellen (
 id INT(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
 merkid INT(11) UNSIGNED NOT NULL,
 naam VARCHAR(255) NOT NULL,
 FOREIGN KEY (merkid) REFERENCES merken(id)
     ON UPDATE RESTRICT ON DELETE RESTRICT
 )
```

Bovenstaande geeft hetzelfde resultaat terug en maakt de tabel modellen aan met de foreign key en zijn restriction.

CRUD

CRUD = CREATE READ UPDATE DELETE
Wanneer we over CRUD spreken, spreken we over alle mogelijk manipulaties die we op één of meerdere tabellen van een database kunnen uitvoeren.
De SQL syntax is de taal die we nodig zullen hebben om deze handelingen/manipulaties uit te voeren.

SELECT STATEMENT

We starten eerst met het READ gedeelte van de C**R**UD. Deze wordt in SQL uitgevoerd door het SELECT STATEMENT

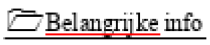
Hieronder zie je de voorstelling van het volledige select statement die gebruikt KAN worden.

```
SELECT field_references
    [ALL | DISTINCT | DISTINCTROW ]
    [HIGH_PRIORITY]
    [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr] ...
    [into_option]
    [FROM table_references
      [PARTITION partition_list]]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

We hebben de SAKILA database samen geïnstalleerd. Deze gaan we nu gebruiken om het select statement in detail aan te leren.

OEFENINGEN

BASIS SELECT EN/OF ORDER BY



ORDER BY: zorgt voor het alfabetisch sorteren van velden. Opties zijn ASC en DESC
DISTINCT: zorgt dat een VOLLEDIG record (rij) UNIEK is in het resultaat.

- Selecteer alle velden uit de tabel actor

```
SELECT *
FROM actor
```

- Selecteer enkel familienaam en voornaam uit de tabel actor

```
SELECT actor.last_name, actor.first_name
FROM actor
```

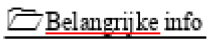
- Selecteer enkel familienaam en voornaam uit de tabel actor en sorteer deze eerst volgens familienaam (A-Z) en vervolgens volgens voornaam (A-Z)

```
SELECT actor.last_name, actor.first_name
FROM actor
ORDER BY actor.last_name ASC, actor.first_name DESC
```

- Selecteer enkel de unieke familienaam en voornaam uit de tabel actor en sorteer Z-A

```
SELECT DISTINCT actor.last_name, actor.first_name
FROM actor
ORDER BY actor.last_name DESC, actor.first_name DESC
```

WHERE CLAUSE



LOGISCHE OPERATOREN: and, or, IN, BETWEEN

- Selecteer alle velden uit de tabel actor waar de familienaam gelijk is aan ‘ZELLWEGER’

```
SELECT *
FROM actor
WHERE actor.last_name = 'ZELLWEGER'
```

- Selecteer alle velden uit de tabel actor waar de familienaam gelijk is aan ‘ZELLWEGER’ of ‘GUINESS’

  oplossing 1:

```
SELECT *
FROM actor
WHERE actor.last_name = 'ZELLWEGER' or actor.last_name ='GUINESS'
```

  oplossing 2:

```
SELECT *
FROM actor
WHERE actor.last_name IN ('ZELLWEGER','GUINESS')
```

- Selecteer alle velden uit de tabel actor waar de familienaam begint met Z of met D

```
SELECT *
FROM actor
WHERE actor.last_name LIKE ('Z%') OR actor.last_name LIKE ('D%')
```

- Selecteer alle velden uit de tabel actor waar de familienaam DAVIS, DUKAKIS of ZELLWEGER is.

```
SELECT *
FROM actor
WHERE actor.last_name IN ('DAVIS','DUKAKIS','ZELLWEGER')
```

- Selecteer alle acteurs waar het id groter of gelijk is dan 5 en kleiner of gelijk is aan 10.

oplossing 1:

```
SELECT *
FROM actor
WHERE actor.actor_id >= 5 and actor.actor_id <= 10
```

oplossing 2:

```
SELECT *
FROM actor
WHERE actor.actor_id BETWEEN 5 and 10
```