

© 2025 AUTEUR/ TOM VANHOUTTE

Cursus MySQL

Intro

*In deze cursus leer je hoe je gegevens in een database kunt aanmaken, bewerken, opvragen en verwijderen. Dit doen we met behulp van **MySQL**, één van de populairste relationele databasesystemen ter wereld.*

Wat is SQL?

SQL staat voor *Structured Query Language*. Het is de standaardtaal om relationele databases te beheren. Met SQL kun je tabellen aanmaken, relaties leggen en data manipuleren.

Wat is MySQL?

MySQL is een implementatie van SQL. Het ondersteunt de standaard SQL-commando's, maar voegt ook eigen uitbreidingen toe (zoals specifieke functies of optimalisaties). Andere implementaties van SQL zijn bijvoorbeeld Oracle, PostgreSQL en Microsoft SQL Server.

Hoe leer je in deze cursus?

- Belangrijke begrippen en commando's worden stap voor stap uitgelegd.
- Bij elk onderdeel vind je voorbeelden en oefeningen.
- Aan het einde van elke module krijg je opdrachten die je individueel oplost om je kennis te toetsen.

Database

Een **database** is een georganiseerde verzameling gegevens die op een gestructureerde manier wordt opgeslagen en beheerd door een computersysteem. Dankzij een database kun je grote hoeveelheden data efficiënt bewaren, terugvinden en bewerken.

Relationeel versus niet-relationeel

Er bestaan verschillende soorten databases:

- **Relationele databases (RDBMS)**

Hierbij worden gegevens opgeslagen in **tabellen** (vergelijkbaar met Excel). Tabellen kunnen onderling met elkaar verbonden worden via sleutels (primary en foreign keys). Voorbeelden: **MySQL**, **PostgreSQL**, **Oracle Database**, **Microsoft SQL Server**, **MariaDB**.

- **Niet-relationele databases (NoSQL)**

Deze zijn flexibeler qua structuur en worden vaak gebruikt voor grote hoeveelheden ongestructureerde data (zoals JSON-documenten of key-value stores).

Voorbeelden: **MongoDB**, **CouchDB**, **Redis**.

In deze cursus focussen we op **relationele databases**, met MySQL als praktische omgeving.

Waarom relationele databases?

Relationele databases hebben verschillende voordelen:

- ⚡ **Snel terugvinden van informatie** dankzij indexing en query-optimalisatie.
- ⇛ **Relaties tussen tabellen** via sleutels (bijv. een klant met meerdere bestellingen).
- 🔍 **Efficiënt opslaan van data** zonder onnodige duplicatie.
- 🔒 **Beveiliging en rechtenstructuur** voor verschillende gebruikers.
- 🌐 **Gestandaardiseerde taal (SQL)**: dezelfde basiscommando's werken in vrijwel alle relationele databasesystemen.

Installatie

*E*en MySQL database wordt gebruikt in combinatie met webtoepassingen. Om lokaal te kunnen oefenen, hebben we een webserveromgeving nodig waarin zowel MySQL als PHPmyadmin beschikbaar zijn.

Installatie van MySQL en phpMyAdmin

Een MySQL-database wordt vaak gebruikt in combinatie met webtoepassingen. Om lokaal te kunnen oefenen, hebben we een **webserveromgeving** nodig waarin zowel MySQL als phpMyAdmin beschikbaar zijn.

Stap 1: Kies een serverpakket

De eenvoudigste manier om dit lokaal op je computer te installeren, is via een kant-en-klaar pakket dat Apache (webserver), MySQL (database) en PHP combineert. Enkele populaire opties zijn:

- **WampServer** (Windows) → <https://www.wampserver.com>
- **XAMPP** (Windows, macOS, Linux) → <https://www.apachefriends.org>
- **MAMP** (macOS, Windows) → <https://www.mamp.info>

In deze cursus gebruiken we **WampServer** als voorbeeld.

Stap 2: Installatie

1. Download de juiste versie (meestal de **64-bit versie**) van WampServer.
2. Voer het installatiebestand uit en volg de instructies.
3. Na installatie verschijnt een icoontje in de taakbalk rechtsonder.

● = server is uitgeschakeld ● = server is gedeeltelijk actief (Apache of MySQL niet gestart) ● = alles is correct gestart

Stap 3: phpMyAdmin openen

- Klik met de linkermuisknop op het groene WampServer-icoon.
- Kies **phpMyAdmin** in het menu.
- Er opent een browservenster waar je kunt inloggen.

Standaard log je in met:

- **Gebruikersnaam:** root
- **Wachtwoord:** leeg (druk gewoon op Enter)

Belangrijk: dit is enkel veilig in een **lokale ontwikkelomgeving**. Zodra je met een echte online server werkt, moet je altijd een sterk wachtwoord instellen.

Stap 4: Eerste kennismaking

Na het inloggen kom je in de **phpMyAdmin-interface** terecht. Hier kun je:

- nieuwe databases aanmaken,
- tabellen en velden definiëren,
- queries uitvoeren,
- gebruikersrechten beheren.

Onderaan het startscherm zie je informatie zoals:

- de **webserver** die draait (Apache),
- de **databaseserver** (MySQL),
- de **huidige beheeromgeving** (phpMyAdmin).

Belangrijk over storage engines

MySQL ondersteunt verschillende **storage engines**. De belangrijkste zijn:

- **InnoDB** → de standaard sinds MySQL 5.5, ondersteunt relaties (foreign keys), transacties en is de aanbevolen keuze.
- **MyISAM** → oude engine zonder transacties of foreign keys. Wordt vandaag **niet meer gebruikt** voor nieuwe projecten.

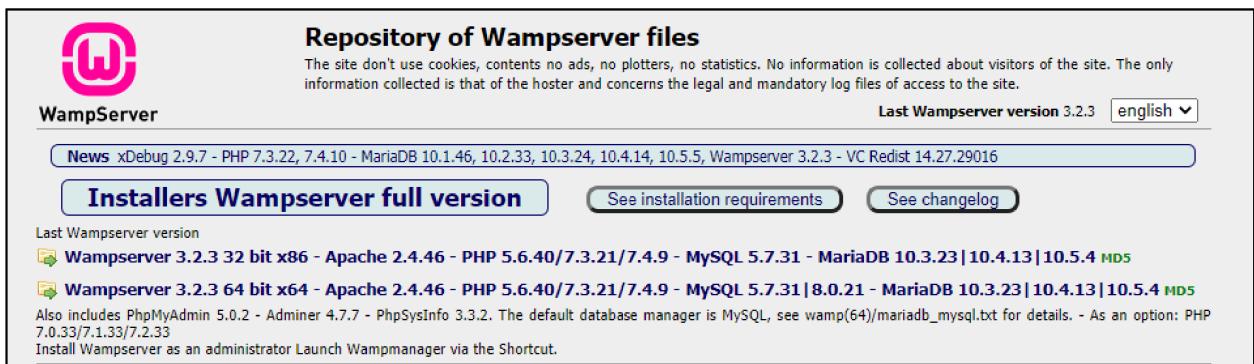
Zorg dat je tabellen altijd met **InnoDB** aanmaakt. Je hoeft dit meestal niet meer manueel in te stellen, want InnoDB is standaard actief.

Windows - wampserver

Belangrijke info

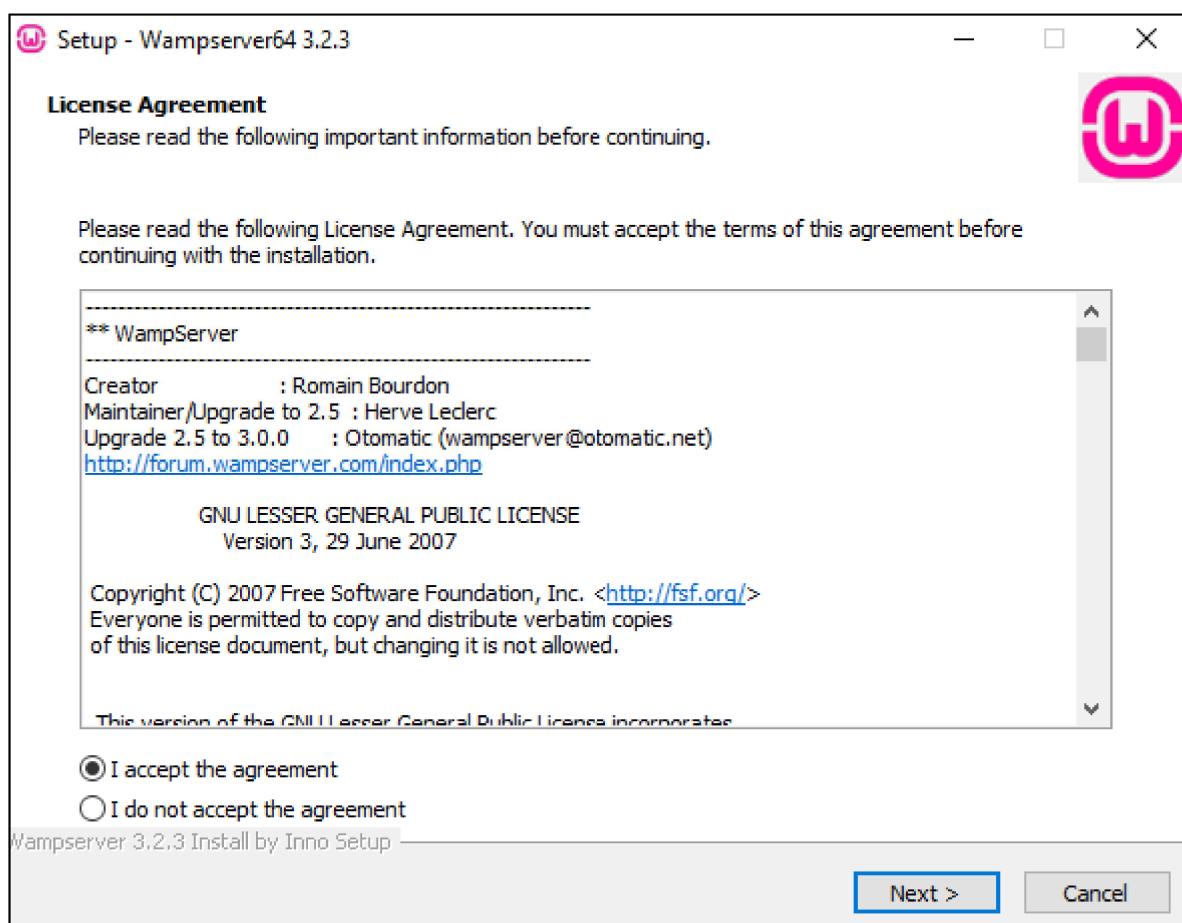
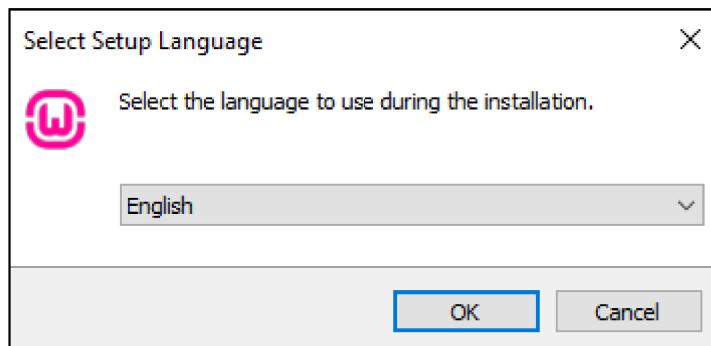
Om de apache webserver en de MySQL omgeving te installeren surf je naar de volgende link:
<http://wampserver.aviatechno.net>

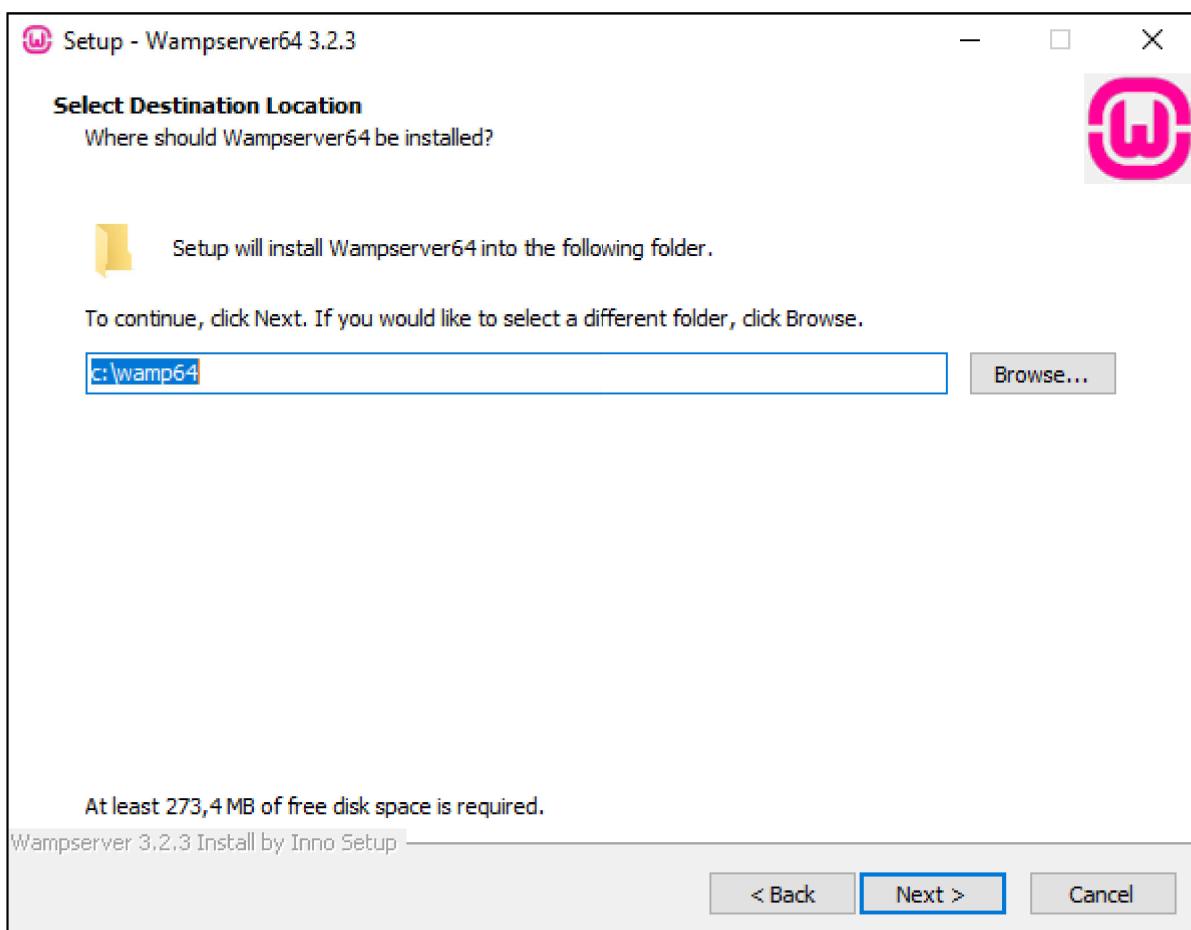
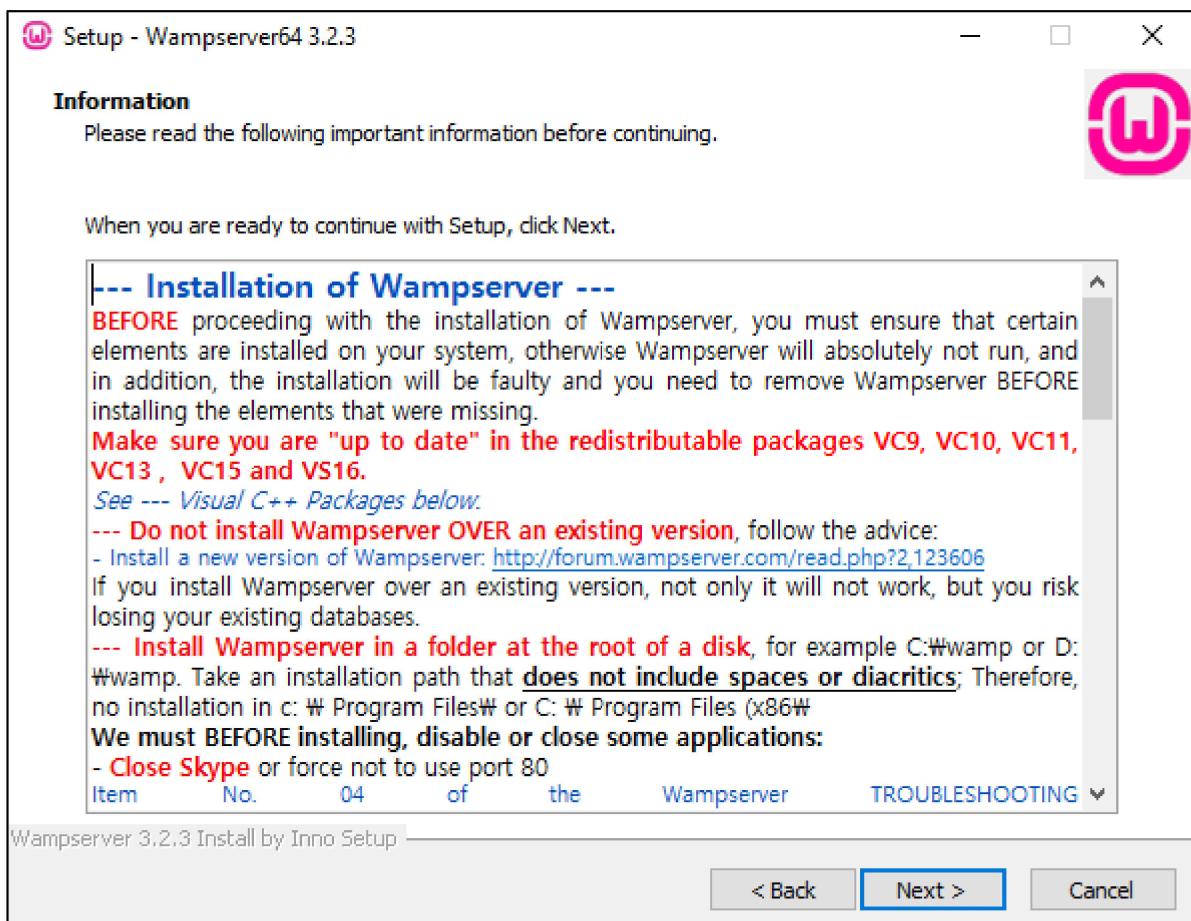
Download de gepaste versie (voor de meeste pc's zal dit de x64 versie zijn).

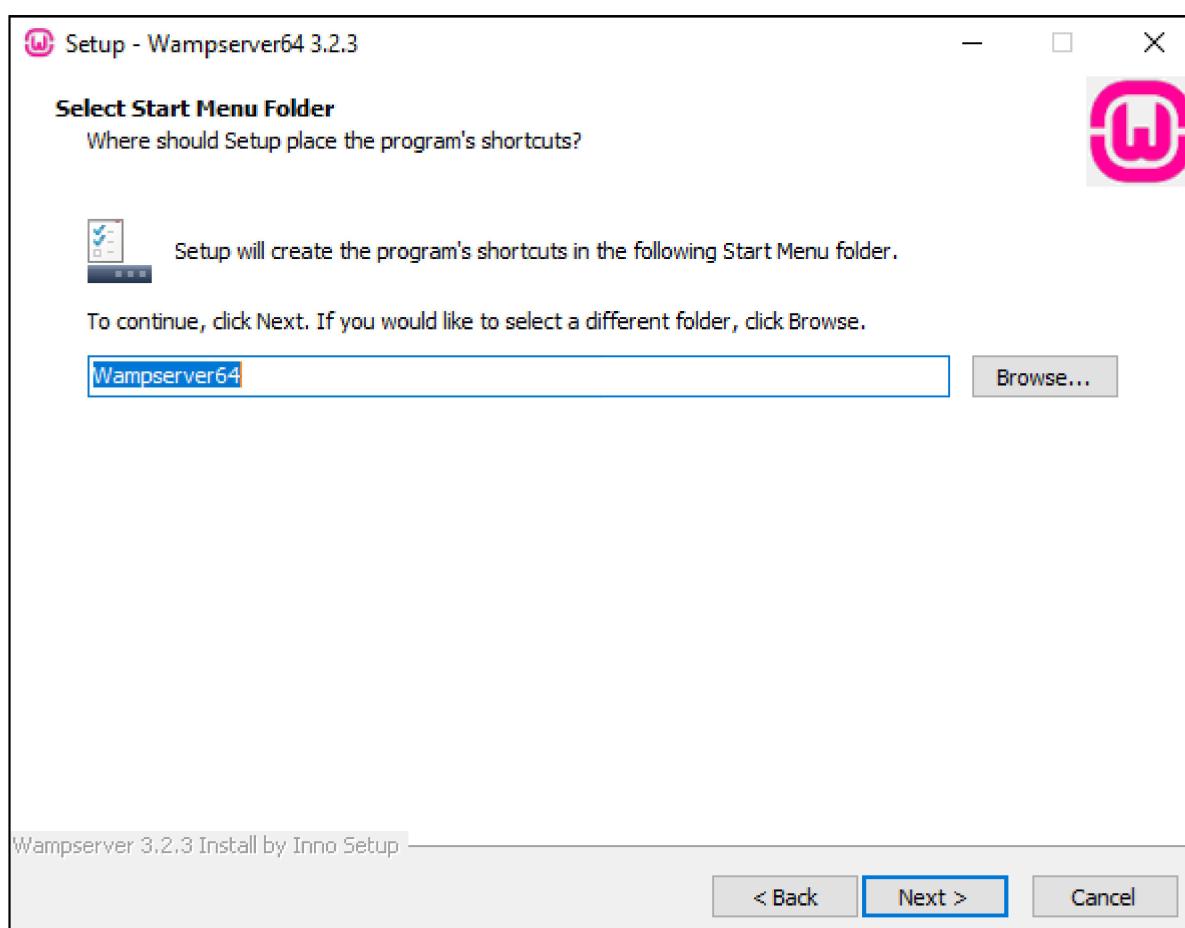
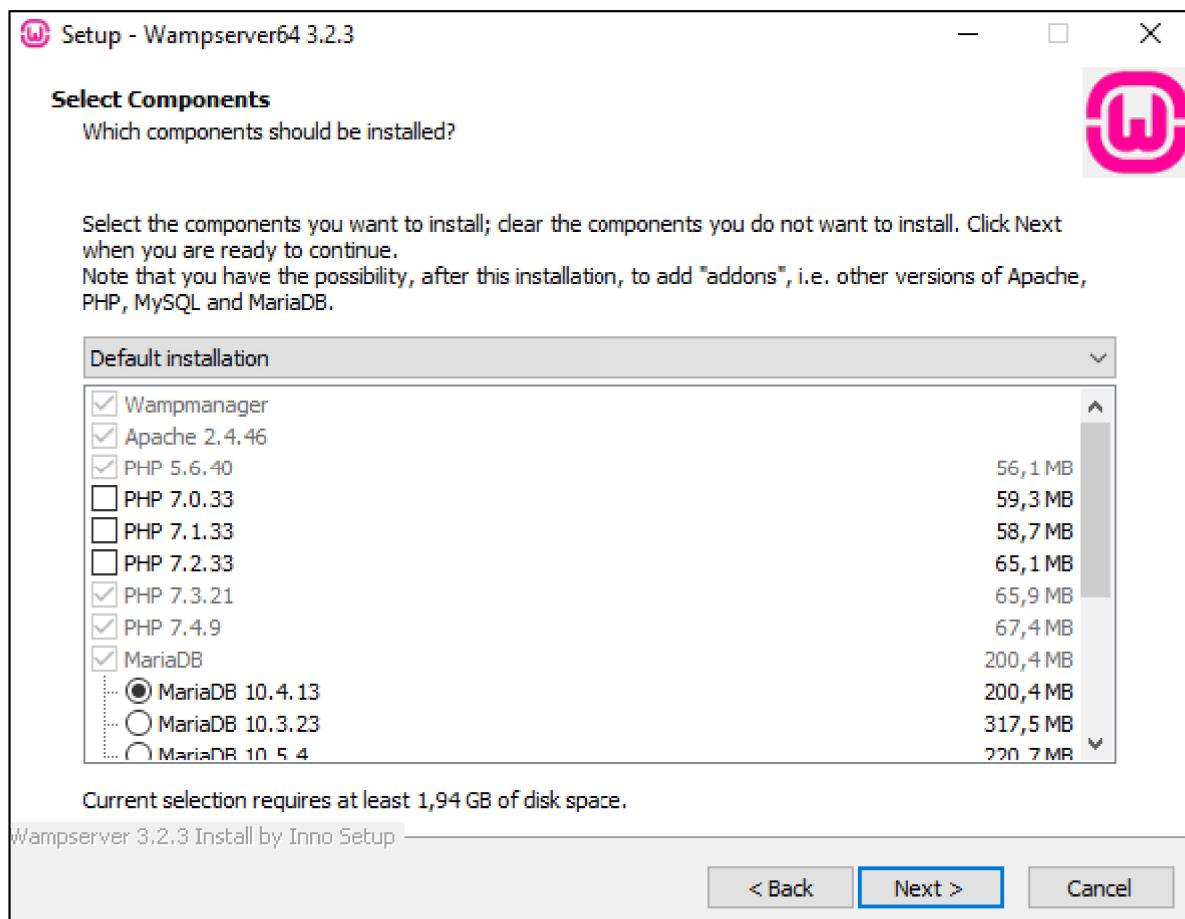


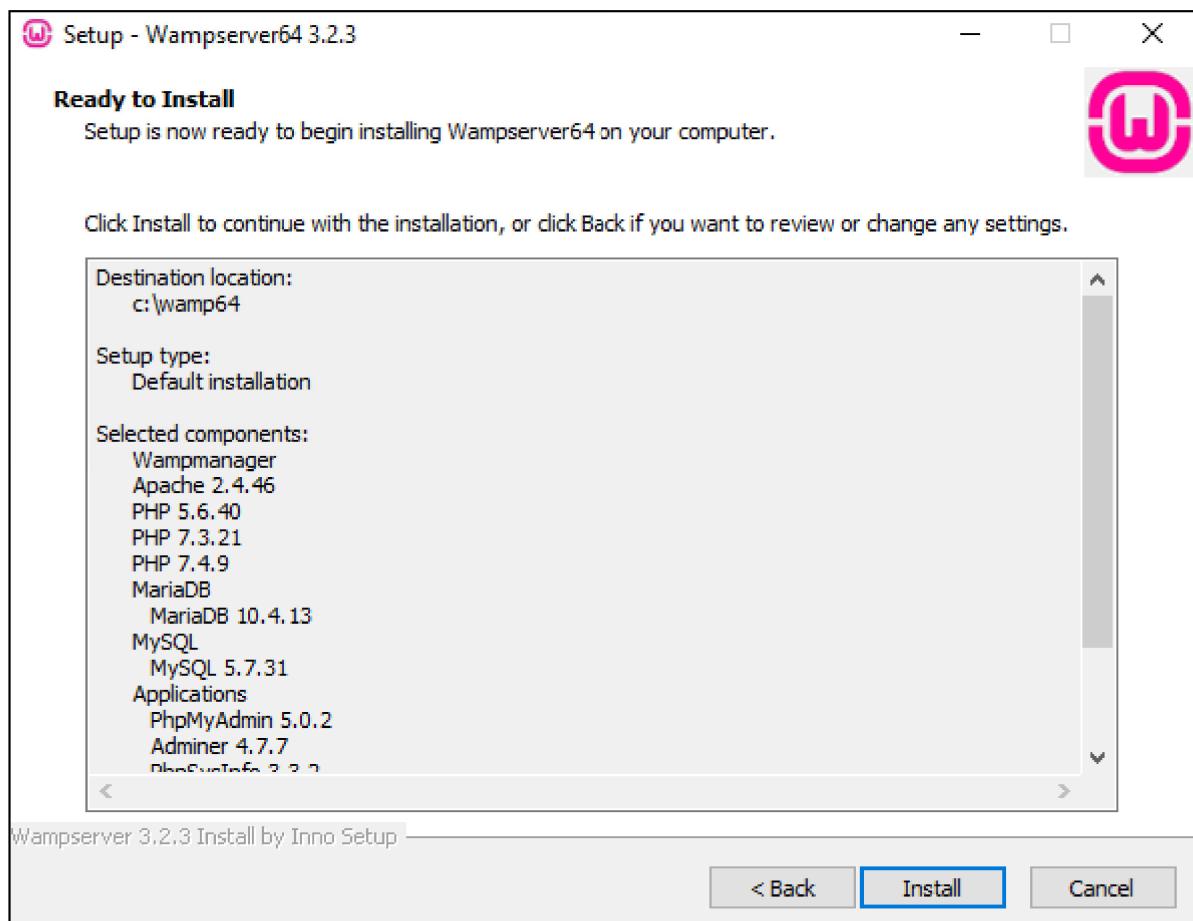
The screenshot shows the 'Repository of Wampserver files' page. It features the WampServer logo, a brief privacy notice, and a link to the latest version (3.2.3). Below this are two main download links: 'Wampserver 3.2.3 32 bit x86 - Apache 2.4.46 - PHP 5.6.40/7.3.21/7.4.9 - MySQL 5.7.31 - MariaDB 10.3.23 | 10.4.13 | 10.5.4' and 'Wampserver 3.2.3 64 bit x64 - Apache 2.4.46 - PHP 5.6.40/7.3.21/7.4.9 - MySQL 5.7.31 | 8.0.21 - MariaDB 10.3.23 | 10.4.13 | 10.5.4'. A note at the bottom indicates the download includes PHPMyAdmin 5.0.2 and Adminer 4.7.7.

In de downloadmap dubbelklik je op het uitvoeringsbestand en start je de installatie: volgt de schermafbeeldingen hieronder.









PHPmyadmin

De database omgeving waarin wij zullen werken is PHPmyadmin. We zullen dus de MySQL taal rechtstreeks op verschillende databases uitvoeren. Bij manipulatie van een database kan je dus met deze taal acties uitvoeren die dikwijls niet omkeerbaar zijn. Bijvoorbeeld het verwijderen van een tabel in een database of het wijzigen van gebruikers, Heden ten dage draait alles om data die zeer waardevol is voor verschillende mogelijke partijen, waaronder personen met minder goede bedoelingen zoals hackers. Wanneer je bijvoorbeeld later een carrière als ethisch hacker wil uitoefenen, dan dien je dus de SQL-taal al zeer goed te kennen en de omgevingen waarin deze taal wordt uitgevoerd.

Openen



Na de installatie van de wampserver zal je dit icoontje terugvinden op je bureaublad.

Dubbelklik op dit icoontje op zowel de apache webserver te starten alsook MySQL. Je pc fungeert nu eigenlijk als een lokale webserver die NIET geconnecteerd is met het web. Een reëele webserver is WEL geconnecteerd met het web.

Webservers voorzien dus enerzijds hosting om je website of webapplicatie op te laden alsook de PHPmyadmin databaseomgeving anderzijds.

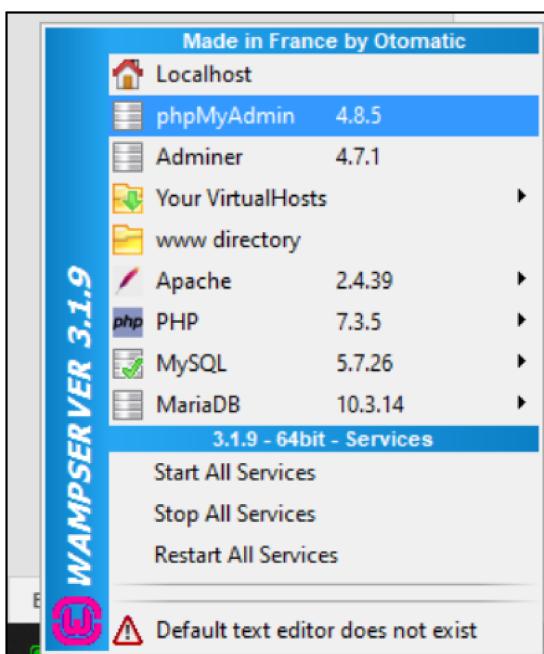
Als developer zullen we dus onze webapplicaties eerst lokaal maken alvorens deze op te laden naar een actieve hosting.



Wanneer je alles correct geïnstalleerd hebt en alles opgestart is, zul je rechts onderaan in je taakkalk hetzelfde icoontje zien staan die GROEN dient te zijn. Bij een groen icoontje werd alles correct gestart.

Bij een ORANJE icoontje werd ofwel de apache webserver of MySQL niet goed opgestart.

Bij een ROOD icoontje werd niets opgestart.



Klik nu met je linker muisknop op dit GROENE icoontje en klik vervolgens op PHPmyadmin om de database omgeving te openen. Deze omgeving zal jullie speelplaats worden tijdens deze cursus. Naargelang de evolutie van alle programma's kunnen de versies die in de schermafbeeldingen te zien zijn verschillen van deze cursus.

CURSUS MySQL

In het volgende scherm zal je nu kunnen inloggen. In de lokale ontwikkelomgeving dienen we geen speciale username of password te gebruiken. Standaard loggen we op PHPmyadmin in met de user **root** en laten we password **blanco** staan. Klik vervolgens op **Go**.

Vanzelfsprekend wanneer we onze afgewerkte database zouden opladen naar een reëele webserver zal dit aangepast worden. Het zou bijzonder leuk zijn voor personen met slechte bedoelingen moest dit ook online op deze manier toegankelijk zijn.

phpMyAdmin

Welcome to phpMyAdmin

Language

English

Log in

Username: root

Password:

Server Choice: MySQL

Go

Na het inloggen kom je terecht in de standaard omgeving van PHPmyadmin. Hier kan je verschillende databases aanmaken en/of bevragen. Het belangrijkste in onderstaand scherm is de positionering nl. **Server:MySQL:3306**. d.w.z. dat je nu op het hoogste niveau van de MySQL Server bevindt.

phpMyAdmin

Server: MySQL:3306

Databases SQL Status User accounts Export Import

General settings

CURSUS MySQL

Er dient ook een woordje uitleg te worden gegeven over het onderstaande scherm. Hier zie je dat de **Web Server** die momenteel draaiende is dankzij wamp, de **Apache webserver** is.

De **Database Server** die we momenteel via PHPmyadmin bekijken is **MySQL**.

PHPmyadmin is de huidige omgeving waarin **MySQL** momenteel draait.

Dit kan dus evengoed een andere omgeving zijn zoals mongoDB, mariaDB die eveneens MySQL draaien, maar worden hier niet besproken.

Database server

- Server: MySQL (127.0.0.1 via TCP/IP)
- Server type: MySQL
- Server connection: SSL is not being used ⓘ
- Server version: 5.7.26 - MySQL Community Server (GPL)
- Protocol version: 10
- User: root@localhost
- Server charset: UTF-8 Unicode (utf8)

Web server

- Apache/2.4.39 (Win64) PHP/7.3.5
- Database client version: libmysql - mysqlnd 5.0.12-dev - 20150407 - \$Id: 7cc7cc96e675f6d72e5cf0f267f48e167c2abb23 \$
- PHP extension: mysqli ⓘ curl ⓘ mbstring ⓘ
- PHP version: 7.3.5

phpMyAdmin

- Version information: 4.8.5, latest stable version: 4.9.5
- Documentation
- Official Homepage
- Contribute
- Get support
- List of changes
- License

Databases

Nieuw

Een nieuwe database creëren via PHPmyadmin is eenvoudig. Aan de linkerkant zie je een menu die alle databases zal bevatten. Je kan dus oneindig veel databases op PHPmyadmin draaien voor verschillende webapplicaties. Klik op **New** om een database aan te maken.

Databases

Zoals je hierboven kan zien zie je de characterset latin_swedish_ci. Dit is de standaard. Deze characterset bestaat uit 8 bits in maximale grootte. Standaard is dit meestal meer dan genoeg om tekens (letters, cijfers, speciale tekens) te bewaren.

In deze characterset zitten natuurlijk alle tekens die ook in de ASCII tabel zitten en meer. Waarom wordt dan de ASCII tabel niet gebruikt? ASCII telde initieel 127 symbolen die konden bewaard worden in een 7-bits karakterset. In een 8-bits karakterset kunnen er dus meer symbolen worden gebruikt.

Voor websites zitten we tegenwoordig aan 16-bits in grootte. De characterset die daar zal worden gebruikt is de huidige standaard UTF-8.

In de meeste gevallen heb je echter genoeg met de swedisch karakterset. Wil je zeker zijn dan kan je de characterset hier ook aanpassen naar UTF-8.

Kies nu **utf8_general_ci** als characterset en druk **create**.

De database werd nu aangemaakt. De engine die wordt gebruikt om de database te bevragen in PHPmyadmin is standaard MyISAM. Deze heeft als nadeel dat we geen relaties kunnen leggen tussen tabellen in deze omgeving. We dienen deze om te zetten naar de andere engine van PHPmyadmin, nl. InnoDB

Engine InnoDB

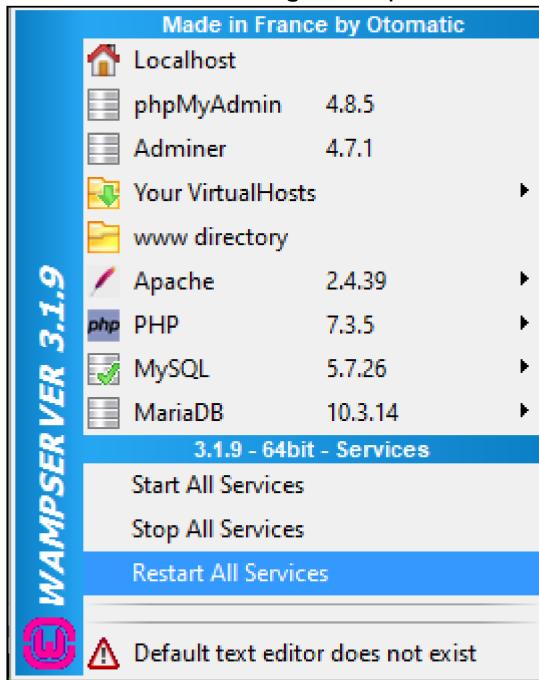
Om de engine te wijzigen dienen we het bestand my.ini te openen op onderstaande locatie (zie schermafbeelding).

This PC > Windows (C:) > wamp64 > bin > mysql > mysql5.7.26				
	Name	Date modified	Type	Size
	bin	9/28/2019 2:49 PM	File folder	
	data	9/23/2020 9:03 PM	File folder	
	docs	9/28/2019 2:49 PM	File folder	
	include	9/28/2019 2:49 PM	File folder	
	lib	9/28/2019 2:49 PM	File folder	
	share	9/28/2019 2:49 PM	File folder	
	COPYING	4/13/2019 3:32 PM	File	18 KB
	my.ini	9/23/2020 9:03 PM	Configuration sett...	7 KB
	README	4/13/2019 3:32 PM	File	3 KB
	wampserver.conf	12/11/2015 11:55 AM	CONF File	1 KB

In my.ini zoek je achter de myISAM engine en wijzig je deze naar InnoDB.

```
; The default storage engine that will be used when create new tables
default-storage-engine=InnoDB
; New for MySQL 5.6 default_tmp_storage_engine if skip-innodb enable
; default_tmp_storage_engine=MYISAM
```

Herstart nu de volledige wampserver als volgt om de engine in te laden:



Verwijderen

Databases kunnen we ook verwijderen uit PHPmyadmin.

- klik bovenaan op **Server:MySQL:3306**
 - klik vervolgens op het tabblad **Databases**
- 
- Selecteer de te verwijderen database
 - Klik op **Drop** en druk **OK**

Sample Databases

Voor MySQL werden er enkel databases aangemaakt speciaal voor studenten die de taal wensen onder de knie te krijgen.

Link:<https://dev.MySQL.com/doc/index-other.html>

De databases die o.a. worden gebruikt zijn employee, world, sakila en menagerie.

Wij zullen de **sakila** database gebruiken, maar eerst installeren. **Download** hier het **Zip** bestand van de database. We pakken deze uit op de c:\ schijf.

Example Databases			
Title	Download DB	HTML Setup Guide	PDF Setup Guide
employee data (large dataset, includes data and test/verification suite)	GitHub	View	US Ltr A4
world database	Gzip Zip	View	US Ltr A4
world_x database	TGZ Zip	View	US Ltr A4
sakila database	TGZ Zip	View	US Ltr A4
menagerie database	TGZ Zip		

Installatie SAKILA database

Hiervoor zullen we de **HTML Setup Guide** gebruiken van de sakila database. Druk dan op **Installation**.

Sakila Sample Database

Table of Contents

- [1 Preface and Legal Notices](#)
- [2 Introduction](#)
- [3 History](#)
- [4 Installation](#)
- [5 Structure](#)
- [6 Usage Examples](#)
- [7 Known Issues](#)
- [8 Acknowledgments](#)
- [9 License for the Sakila Sample Database](#)
- [10 Note for Authors](#)
- [11 Sakila Change History](#)

CURSUS MySQL

In de volgende pagina zie je nu enkele commando's die we in een console dienen in te geven:

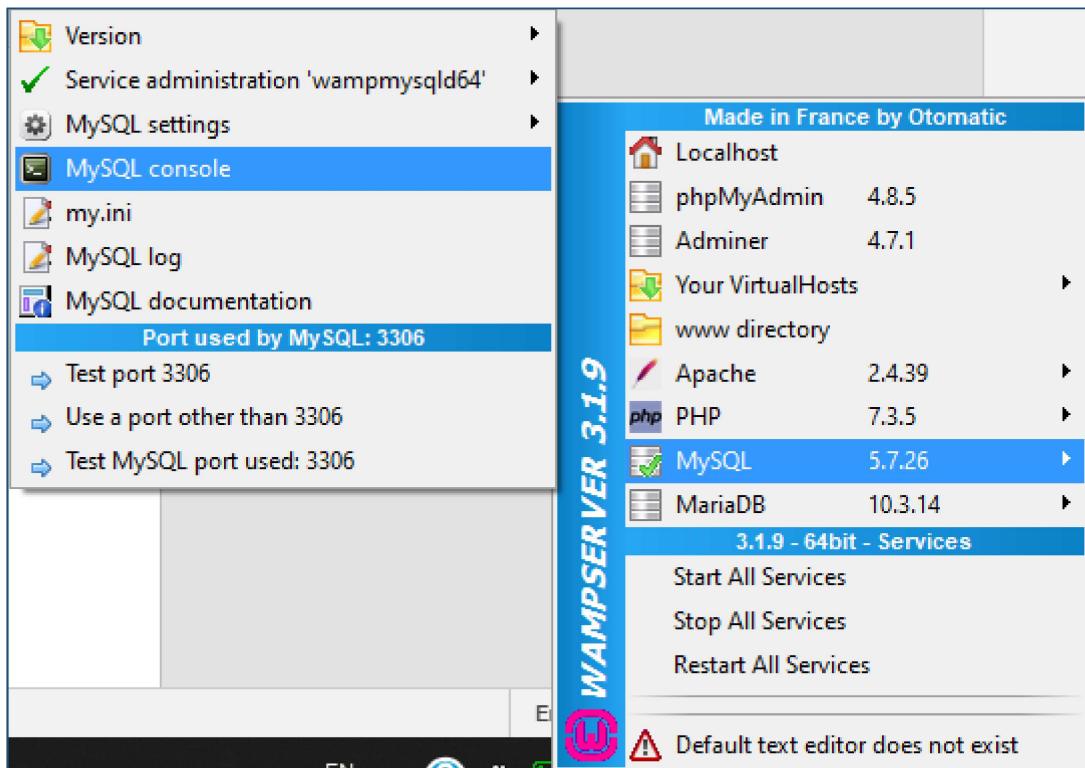
To install the Sakila sample database, follow these steps:

1. Extract the installation archive to a temporary location such as `c:\temp\` or `/tmp/`. When you unpack the `sakila-schema.sql` and `sakila-data.sql` files.

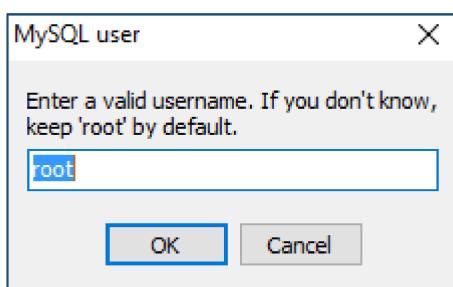
2. Connect to the MySQL server using the `mysql` command-line client with the following command:

```
1 | shell> mysql -u root -p
```

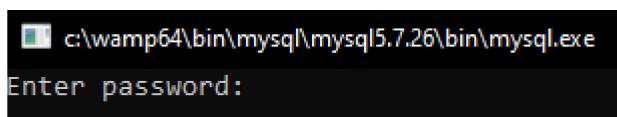
De console die we hiervoor gaan gebruiken is de MySQL console die je in de schermafbeelding hieronder ziet staan.



De standaard username van PHPmyadmin zal worden gevraagd. Druk hier op **OK**.



Het paswoord die je dient in te vullen is **blanco**. Druk dus gewoon op **ENTER**.



CURSUS MySQL

Wanneer alles correct werd uitgevoerd zit je in de console van MySQL.

```
c:\wamp64\bin\mysql\mysql5.7.26\bin\mysql.exe
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 58
Server version: 5.7.26 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Nu dienen we het de sakila database die we uitgepakt hebben op de c schijf te installeren.
Eerst wordt de structuur (tabellen) geïnstalleerd met het volgende commando. Let op dat de locatie juist is.

```
mysql> SOURCE C:/sakila-db/sakila-schema.sql
```

Vervolgens gaan we de data ook in de tabellen injecteren met het volgende commando.

```
mysql> SOURCE C:/sakila-db/sakila-data.sql
```

Opdracht

Probeer ook de world database te installeren.

CURSUS MySQL

Sluit nu de consoletoepassing af en open terug PHPmyadmin.

Resultaat:

The screenshot shows the PHPMyAdmin interface for the Sakila database. The top navigation bar includes tabs for Structure, SQL, Search, Query, Export, Import, Operations, Privileges, Routines, Events, and Triggers. A 'Filters' section with a search input field is present. The main content area displays a table of tables with the following columns: Table, Action, Rows, Type, Collation, Size, and Overhead. The table lists 21 tables from the Sakila database, such as actor, actor_info, address, category, city, country, customer, customer_list, film, film_actor, film_category, film_list, film_text, inventory, language, nicer_but_slower_film_list, payment, rental, sales_by_film_category, sales_by_store, staff, staff_list, and store. The 'country' table is currently selected, indicated by a blue background.

Table	Action	Rows	Type	Collation	Size	Overhead
actor		200	InnoDB	utf8mb4_general_ci	32 Kib	-
actor_info		~0	View	---	-	-
address		603	InnoDB	utf8mb4_general_ci	112 Kib	-
category		16	InnoDB	utf8mb4_general_ci	16 Kib	-
city		600	InnoDB	utf8mb4_general_ci	64 Kib	-
country		109	InnoDB	utf8mb4_general_ci	16 Kib	-
customer		599	InnoDB	utf8mb4_general_ci	128 Kib	-
customer_list		~0	View	---	-	-
film		1,000	InnoDB	utf8mb4_general_ci	272 Kib	-
film_actor		5,462	InnoDB	utf8mb4_general_ci	272 Kib	-
film_category		1,000	InnoDB	utf8mb4_general_ci	80 Kib	-
film_list		~0	View	---	-	-
film_text		1,000	InnoDB	utf8mb4_general_ci	192 Kib	-
inventory		4,581	InnoDB	utf8mb4_general_ci	368 Kib	-
language		6	InnoDB	utf8mb4_general_ci	16 Kib	-
nicer_but_slower_film_list		~0	View	---	-	-
payment		16,049	InnoDB	utf8mb4_general_ci	2.1 MiB	-
rental		16,044	InnoDB	utf8mb4_general_ci	80 Kib	-
sales_by_film_category		~0	View	---	-	-
sales_by_store		~0	View	---	-	-
staff		2	InnoDB	utf8mb4_general_ci	48 Kib	-
staff_list		~0	View	---	-	-
store		2	InnoDB	utf8mb4_general_ci	48 Kib	-

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 tijdsdatatype

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.

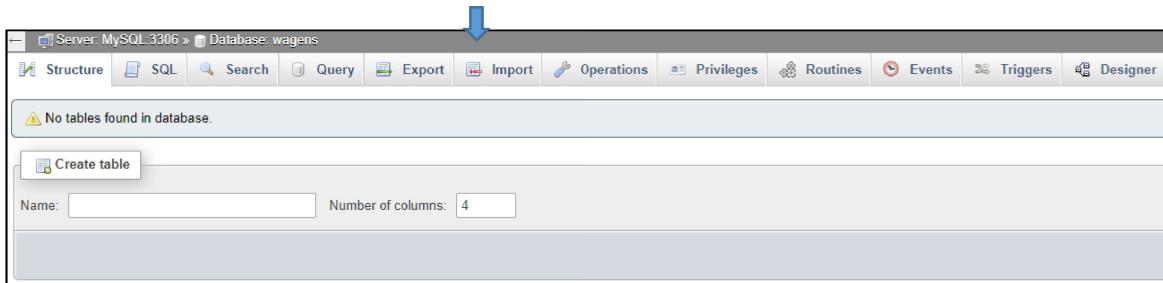
5. Praktische keuzes

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

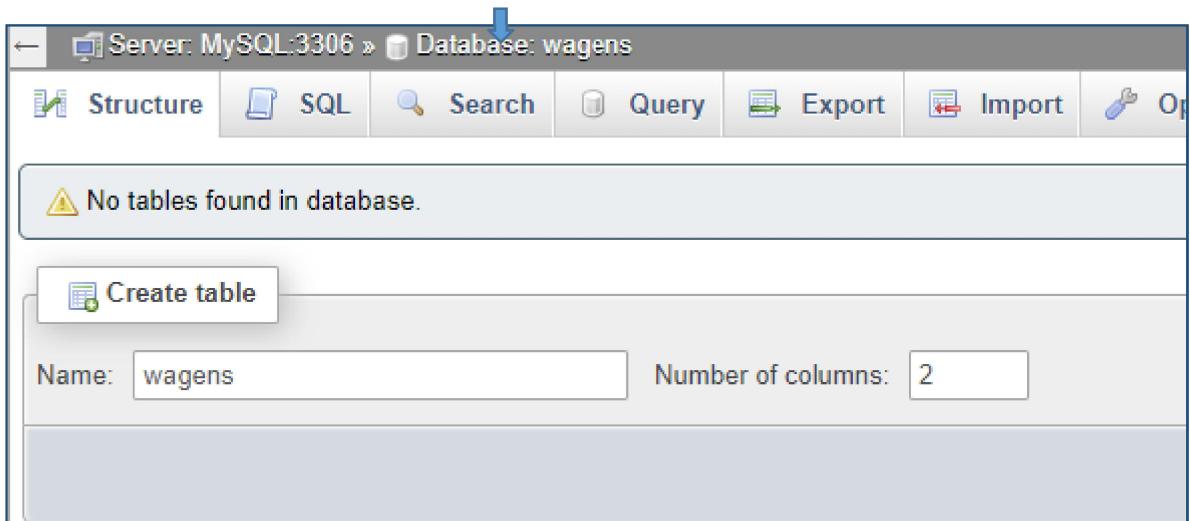
Een nieuwe tabel aanmaken is eenvoudig. Je dient eerst je database te selecteren. Je kan steeds controleren of je in de correcte database zit bovenaan PHPmyadmin. Net zoals in windows verkenner is dit een folder structuur die vertrekt van het server niveau.



In ons voorbeeld hebben we het over wagens. Wagens bestaan uit merken en modellen. Hiervoor creëren 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.

Name	Type	Length/Values	Default	Collation	Attributes	Null	Index	A_I	Comments	Virtuality	Move column
id	INT	11		None	UNSIGNED	<input checked="" type="checkbox"/>	PRIMARY	PRIMARY	<input checked="" type="checkbox"/>		
merk	VARCHAR	255		None							

Table comments: Storage Engine: InnoDB

PARTITION definition:

Partition by: (Expression or column list)

Partitions:

Preview SQL Save

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 biljoen 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
)
```

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	merkid	int(11)		UNSIGNED	No	None			Change Drop More
3	naam	varchar(255)	utf8_general_ci		No	None			Change Drop More

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,...

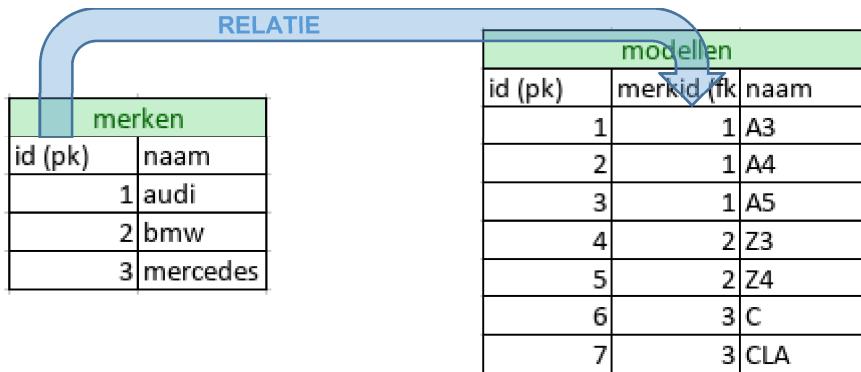
Dit noemen we een één-op-veel-relatie, want het merk audi (één) heeft meerdere modellen (veel).

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

wagens merken	wagens modellen
id : int(11) unsigned	id : int(11) unsigned
wagens : varchar(255)	merkid : int(11) unsigned

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



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**.

The screenshot shows the 'Relation view' tab selected in the 'modellen' table's structure view. It displays a 'Foreign key constraints' section where a constraint named 'model_merken' is being configured. The configuration shows 'ON DELETE RESTRICT' and 'ON UPDATE RESTRICT'. The 'Column' dropdowns show 'merkid' and 'wagens' respectively, and the 'Database' dropdown shows 'merken' with 'id' selected. The 'Table' dropdown shows 'modellen'.

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.

#	Name	Type	Collation	Attributes	Null	Def
1	id	int(11)		UNSIGNED	No	None
2	merkid	int(11)		UNSIGNED	No	None
3	naam	varchar(255)	utf8_general_ci		No	None

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 CRUD. 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
```

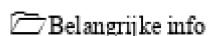
- Selecteer enkel de unieke familienaam en voornaam uit de tabel actor en sorteert Z-A

```
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 'GUINNESS'

oplossing 1:

```
SELECT *
```

```
FROM actor
```

```
WHERE actor.last_name = 'ZELLWEGER' or actor.last_name = 'GUINNESS'
```

oplossing 2:

```
SELECT *
```

```
FROM actor
```

```
WHERE actor.last_name IN ('ZELLWEGER', 'GUINNESS')
```

- 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
```

FUNCTIONS

Belangrijke info

AS: het keyword AS wordt gebruikt om een veld een anderen naam te geven in een weergave of een nieuw samengesteld veld een naam te geven. AS is dus wat noemt een **alias** van een bestaand of nieuw veld.

In MySQL kunnen we daarnaast ook functies toepassen. Functies worden toegepast op bestaande velden in. Functies worden gekenmerkt door de naamgeving en ronde haakjes. Bijvoorbeeld: de functie MIN() haalt de laagste waarde uit een kolom.

Hieronder zie je een lijst van veel gebruikte functies

COUNT	De MySQL COUNT-aggregatiefunctie wordt gebruikt om het aantal rijen in een databasetabel te tellen.
MAX	Met de MySQL MAX-aggregatiefunctie kunnen we de hoogste (maximale) waarde voor een bepaalde kolom selecteren.
MIN	Met de MySQL MIN-aggregatiefunctie kunnen we de laagste (minimum) waarde voor een bepaalde kolom selecteren.
AVG	De MySQL AVG-aggregatiefunctie selecteert de gemiddelde waarde voor een bepaalde tabelkolom.
SUM	Met de MySQL SUM-aggregatiefunctie kunt u het totaal voor een numerieke kolom selecteren.
SQRT	Dit wordt gebruikt om een vierkantswortel van een bepaald getal te genereren.
RAND	Dit wordt gebruikt om een willekeurig getal te genereren met behulp van de MySQL-opdracht.
CONCAT	Dit wordt gebruikt om elke tekenreeks binnen een MySQL-opdracht samen te voegen .
LENGTH	Geeft de lengte van een string terug
CURDATE	Retourneert de huidige datum
CURTIME	Retourneert de huidige tijd
ADDDATE	Telt dagen bij een datum op
ADDTIME	Telt tijd bij tijd op
DATEFORMAT	Hier kan je de weergave van een datum mee wijzigen
DATE_SUB	Hier kan je 2 data van elkaar aftrekken
DATE	Geeft de datum van vandaag terug
WEEKDAY	Retourneert het weekdag nummer (index)
WEEK	Retourneert het weeknummer (index)

MySQL telt dus heel wat BUILT-IN functions. Er zijn er een paar honderd. Alle functies kun je hieronder terugvinden via de volgende link:

[MySQL :: MySQL 8.4 Reference Manual :: 14.1 Built-In Function and Operator Reference](#)

- Selecteer het hoogste bedrag uit de tabel payment
`SELECT max(amount)
 from payment`
- Hoeveel rijen telt de tabel payment
`SELECT max(amount)
 from payment`

- Selecteer de gemiddelde prijs van de bedragen uit de tabel payment
- ```
SELECT avg(amount)
FROM payment
```

## KЛАSSИКАLE OEFENINGEN

### 1. Film Titels in Hoofdletters

Schrijf een query die de titels van alle films in hoofdletters retourneert.

**Query:**

```
SELECT UPPER(title) AS film_titel
FROM film;
```

### 2. Lengte van de Film Titel

Haal de titels van alle films op en geef ook de lengte van elke titel weer.

**Query:**

```
SELECT title, LENGTH(title) AS titel_lengte
FROM film;
```

### 3. Film Titels zonder Spaties aan de Randen

Geef de film titels weer waarbij eventuele leidende en afsluitende spaties worden verwijderd.

**Query:**

```
SELECT TRIM(title) AS titel_bijgesneden
FROM film;
```

### 4. Films die een Bepaalde Tekst bevatten

Haal de titels van alle films op die het woord 'love' in hun titel hebben (ongeacht hoofdletters).

**Query:**

```
SELECT title
FROM film
WHERE LOWER(title) LIKE '%love%';
```

## 5. Eerste en Laatste Teken van Acteursnamen

Toon de voornaam van elke acteur, samen met het eerste en laatste teken van hun voornaam.

**Query:**

```
SELECT first_name, LEFT(first_name, 1) AS eerste_karakter, RIGHT(first_name, 1) AS laatste_karakter
FROM actor;
```

```
SELECT count(distinct last_name) as aantal
```

```
FROM actor
```

**GROUP BY**

 Belangrijke info

Een group by groepeert rijen/velden die identiek zijn aan elkaar tot één rij.

Let op: bij een GROUP BY dien je altijd ALLE VELDEN over te nemen die in het SELECT gedeelte staan ZONDER de velden die van een functie zouden voorzien zijn.

Voorbeeld van een group by :

ADAM JONES

ADAM JONES

BRIDGET SLOAN

DENNIS HARPER



ADAM JONES

BRIDGET SLOAN

DENNIS HARPER

- Selecteer alle familienamen uit de tabel actor en voeg daarna een tweede kolom toe die het aantal per naam weergeeft. Sorteer dan volgens het aantal van Z-A.

```
SELECT last_name, count(last_name) as aantal
FROM actor
GROUP BY last_name
ORDER BY aantal DESC
```

- Selecteer alle familienamen en voornamen uit de tabel actor en voeg daarna een tweede kolom toe die het aantal per naam weergeeft. Sorteer dan volgens het aantal van Z-A.

```
SELECT last_name, first_name, count(last_name) as aantal
FROM actor
GROUP BY last_name, first_name
ORDER BY aantal DESC
```

**HAVING**

 Belangrijke info

Een having clause wordt gebruikt wanneer we uit het resultaat van een group by dienen te FILTEREN. Een having is eigenlijk de where voor een GROUP BY

- Selecteer alle familienamen uit de actor tabel. Geef dan de lijst weer met familienamen die meer dan 1 x voorkomen!

**CURSUS MySQL**

```
select last_name
from actor
group by last_name
having count(*) > 1
```

## Oefening 1

**Vraag:** Selecteer alle verschillende rental\_rate waarden uit de film tabel.

```
SELECT DISTINCT rental_rate
FROM film;
```

## Oefening 2

**Vraag:** Toon alle actors met de achternaam SMITH of JONES.

```
SELECT *
FROM actor
WHERE last_name = 'SMITH' OR last_name = 'JONES';
```

## Oefening 3

**Vraag:** Selecteer de film\_id en title van alle films met een rental\_rate van meer dan 2.99, gesorteerd op title.

```
SELECT film_id, title
FROM film
WHERE rental_rate > 2.99
ORDER BY title;
```

## Oefening 4

**Vraag:** Tel het aantal films per rating en toon enkel diegenen met meer dan 100 films.

```
SELECT rating, COUNT(*) AS film_count
FROM film
GROUP BY rating
HAVING COUNT(*) > 100;
```

## Oefening 5

**Vraag:** Selecteer alle customer\_id waar de email NULL is.

```
SELECT customer_id
FROM customer
WHERE email IS NULL;
```

## Oefening 6

**Vraag:** Selecteer alle films (title) met een rental\_duration van 5 dagen en een replacement\_cost tussen 15 en 25.

```
SELECT title
FROM film
WHERE rental_duration = 5 AND replacement_cost BETWEEN 15 AND 25;
```

## Oefening 7

**Vraag:** Toon alle inventory\_id's waarvan de store\_id niet gelijk is aan 2.

```
SELECT inventory_id
FROM inventory
WHERE NOT store_id = 2;
```

## Oefening 8

**Vraag:** Selecteer de verschillende store\_id waarden uit de staff tabel.

```
SELECT DISTINCT store_id
FROM staff;
```

## Oefening 9

**Vraag:** Selecteer het aantal klanten (customer\_id) per store\_id.

```
SELECT store_id, COUNT(customer_id) AS customer_count
FROM customer
GROUP BY store_id;
```

## Oefening 10

**Vraag:** Toon alle film titels waarvan de replacement\_cost groter is dan 20, gesorteerd op aflopende replacement\_cost.

```
SELECT title
FROM film
WHERE replacement_cost > 20
ORDER BY replacement_cost DESC;
```

## Oefening 11

**Vraag:** Selecteer alle category\_id's van de category tabel waarvan de name niet Children is.

```
SELECT category_id
FROM category
WHERE NOT name = 'Children';
```

## Oefening 12

**Vraag:** Tel het aantal inventory\_id per film\_id en toon enkel resultaten met meer dan 20 exemplaren.

```
SELECT film_id, COUNT(inventory_id) AS inventory_count
FROM inventory
GROUP BY film_id
HAVING COUNT(inventory_id) > 20;
```

## Oefening 13

**Vraag:** Selecteer de actor\_id en first\_name voor acteurs met de achternaam die begint met K.

```
SELECT actor_id, first_name
FROM actor
WHERE last_name LIKE 'K%';
```

## Oefening 14

**Vraag:** Toon het totaal aantal films per rental\_duration en toon alleen de gevallen met een rental\_duration van meer dan 3 dagen.

```
SELECT rental_duration, COUNT(*) AS film_count
FROM film
GROUP BY rental_duration
HAVING rental_duration > 3;
```

## Oefening 15

**Vraag:** Selecteer alle rental\_id waarbij de return\_date nog niet is ingevuld (NULL).

```
SELECT rental_id
FROM rental
WHERE return_date IS NULL;
```

## Oefening 16

**Vraag:** Selecteer alle verschillende ratings uit de film tabel.

```
SELECT DISTINCT rating
FROM film;
```

## Oefening 17

**Vraag:** Tel het aantal klanten per store\_id waarbij active gelijk is aan 1.

```
SELECT store_id, COUNT(customer_id) AS active_customer_count
FROM customer
WHERE active = 1
GROUP BY store_id;
```

## Oefening 18

**Vraag:** Toon alle actor\_id's waarvan de last\_name niet gelijk is aan DAVIS of JOHNSON.

```
SELECT actor_id
FROM actor
WHERE last_name NOT IN ('DAVIS', 'JOHNSON');
```

## Oefening 19

**Vraag:** Selecteer alle payment\_id's waarbij de amount groter is dan 5 en de payment\_date is in 2005.

```
SELECT payment_id
FROM payment
WHERE amount > 5 AND payment_date LIKE '2005%';
```

## Oefening 20

**Vraag:** Toon de city\_id en tel het aantal steden in elke country\_id met meer dan 5 steden.

```
SELECT country_id, COUNT(city_id) AS city_count
FROM city
GROUP BY country_id
HAVING COUNT(city_id) > 5;
```

## Oefening 21

**Vraag:** Selecteer alle film\_id waarbij de description niet NULL is.

```
SELECT film_id
FROM film
WHERE description IS NOT NULL;
```

## Oefening 22

**Vraag:** Toon de store\_id en tel hoeveel inventory\_id's elke store heeft.

```
SELECT store_id, COUNT(inventory_id) AS inventory_count
FROM inventory
GROUP BY store_id;
```

## Oefening 23

**Vraag:** Selecteer alle actor\_id en last\_name waar de first\_name gelijk is aan NICK.

```
SELECT actor_id, last_name
FROM actor
WHERE first_name = 'NICK';
```

## Oefening 24

**Vraag:** Toon alle customer\_id waarbij create\_date gelijk is aan 2006-02-14.

```
SELECT customer_id
FROM customer
WHERE create_date = '2006-02-14';
```

## Oefening 25

**Vraag:** Tel het aantal verschillende film\_id's per store\_id in de inventory tabel.

```
SELECT store_id, COUNT(DISTINCT film_id) AS unique_film_count
FROM inventory
GROUP BY store_id;
```

## Oefening 26

**Vraag:** Selecteer de rental\_id en rental\_date voor alle verhuurtransacties in februari 2005.

```
SELECT rental_id, rental_date
FROM rental
WHERE rental_date LIKE '2005-02%';
```

## Oefening 27

**Vraag:** Tel het aantal address\_id per city\_id en toon enkel steden met meer dan 3 adressen.

```
SELECT city_id, COUNT(address_id) AS address_count
FROM address
GROUP BY city_id
HAVING COUNT(address_id) > 3;
```

## Oefening 28

**Vraag:** Selecteer alle customer\_id waar de klant niet actief is (active = 0).

```
SELECT customer_id
FROM customer
WHERE active = 0;
```

## Oefening 29

**Vraag:** Selecteer de title en film\_id voor films met een length tussen 90 en 120 minuten.

```
SELECT title, film_id
FROM film
WHERE length BETWEEN 90 AND 120;
```

## Oefening 30

**Vraag:** Toon alle verschillende store\_id waarden uit de customer tabel.

```
SELECT DISTINCT store_id
FROM customer;
```

## JOINS

### **Wat is een Join?**

Een **join** in MySQL stelt ons in staat om data uit meerdere tabellen te combineren. Dit is vooral handig als de tabellen op een bepaalde manier aan elkaar gerelateerd zijn via een primaire en vreemde sleutel (foreign key). Voor de wagens-database in jouw voorbeeld hebben we twee tabellen: wagens merken en wagens modellen.

- **wagens merken:** Bevat informatie over verschillende automerken zoals Audi, BMW, enz.
- **wagens modellen:** Bevat informatie over verschillende modellen die bij elk merk horen, met een merkid die aangeeft bij welk merk het model hoort.

De sleutelrelatie tussen de tabellen stelt ons in staat gegevens uit beide tabellen te combineren, zoals welk model bij welk merk hoort. Laten we de verschillende soorten joins bekijken die in MySQL beschikbaar zijn.

### **Types van Joins**

MySQL ondersteunt vier hoofdtypen van joins: 1. **INNER JOIN** 2. **LEFT JOIN** 3. **RIGHT JOIN** 4. **CROSS JOIN**

We zullen elk type join stap voor stap behandelen, met uitleg en voorbeelden van de query's en de resulterende output.

We vullen de 2 tabellen als volgt:

merken:

|                          |  |  | <b>id</b> | <b>naam</b> |
|--------------------------|--|--|-----------|-------------|
| <input type="checkbox"/> |  |  | 1         | audi        |
| <input type="checkbox"/> |  |  | 2         | bmw         |
| <input type="checkbox"/> |  |  | 3         | volkswagen  |

modellen:

|                          |  |  | <b>id</b> | <b>merkid</b> | <b>naam</b> |
|--------------------------|--|--|-----------|---------------|-------------|
| <input type="checkbox"/> |  |  | 1         | 1             | A3          |
| <input type="checkbox"/> |  |  | 2         | 1             | A4          |

## 1. INNER JOIN

Een **INNER JOIN** geeft alleen de rijen weer die in beide tabellen een overeenkomst hebben. Dit betekent dat alleen de merken die ook modellen hebben, worden weergegeven.

### Voorbeeld:

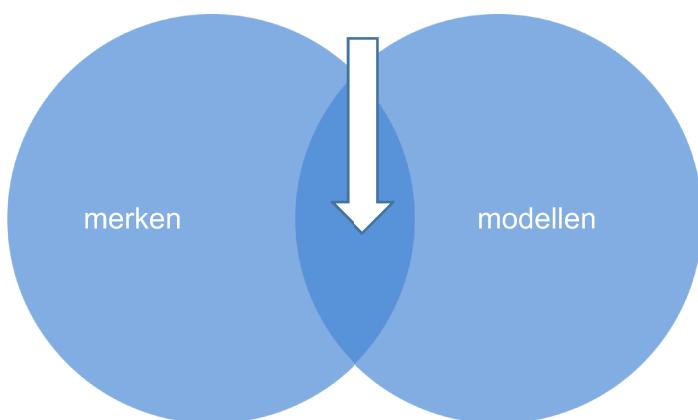
```
SELECT merken.naam, modellen.naam
FROM merken
INNER JOIN modellen ON merken.id = modellen.merkid;
```

- Hier verbinden we de tabel merken met de tabel modellen op basis van de id van merken en de merkid van modellen.
- Het resultaat laat alleen merken en hun bijbehorende modellen zien, zoals Audi met A3 en A4.

| naam | naam |
|------|------|
| audi | A3   |
| audi | A4   |

### Grafisch:

Een Venn-diagram voor een **INNER JOIN** toont alleen de overlappende gedeelten tussen merken en modellen.



## KЛАSSИКАЛЬНЫЕ ОФОРМЛЕНИЯ

### SAKILA DATABASE

1. **Toon alle films met de acteurs en de datum waarop elke film beschikbaar werd gesteld**

```
SELECT film.title, actor.first_name, actor.last_name, film.release_year
FROM film
INNER JOIN film_actor ON film.film_id = film_actor.film_id
INNER JOIN actor ON film_actor.actor_id = actor.actor_id;
```

- Hier wordt de release-informatie van de film gecombineerd met de acteurs die in de film spelen.

2. **Vind alle medewerkers met de huurtransacties die zij persoonlijk hebben afgehandeld**

```
SELECT staff.first_name, staff.last_name, rental.rental_date,
rental.return_date
FROM staff
INNER JOIN rental ON staff.staff_id = rental.staff_id;
```

- Deze query toont alle medewerkers met hun respectieve huurtransacties, wat helpt om te begrijpen wie welke klanten heeft geholpen.

## 2. LEFT JOIN

Een **LEFT JOIN** toont alle rijen uit de linkertabel (merken), ongeacht of er een overeenkomst is in de rechtertabel (modellen). Als er geen bijpassend model is, wordt er NULL weergegeven.

### Voorbeeld:

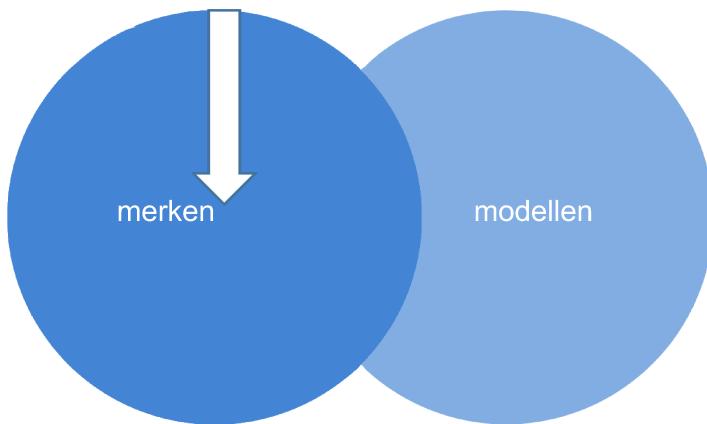
```
SELECT merken.naam, modellen.naam
FROM merken
LEFT JOIN modellen ON merken.id = modellen.merkid;
```

- Deze query selecteert alle merken, zelfs als ze geen modellen hebben. Bijvoorbeeld, als Volkswagen geen model heeft, wordt NULL weergegeven bij het model.

### Resultaat:

| naam       | naam |
|------------|------|
| audi       | A3   |
| audi       | A4   |
| bmw        | NULL |
| volkswagen | NULL |

### Grafisch:



Het Venn-diagram voor een **LEFT JOIN** laat de volledige cirkel van merken zien, plus de gedeelten die overlappen met modellen.

Voorbeeld: Stel je hebt een lijst van klanten en een lijst van bestellingen. Je wilt een overzicht van alle klanten, ook degenen die nog geen bestelling hebben geplaatst.

## KЛАSSИКАЛЬНЫЕ ОФОРМЛЕНИЯ

### SAKILA DATABASE

3. **Geef een lijst van alle klanten en hun laatst bekeken film, inclusief klanten die nog geen film hebben bekeken**

```
SELECT customer.first_name, customer.last_name, film.title
FROM customer
LEFT JOIN rental ON customer.customer_id = rental.customer_id
LEFT JOIN inventory ON rental.inventory_id = inventory.inventory_id
LEFT JOIN film ON inventory.film_id = film.film_id
ORDER BY rental.rental_date DESC;
```

- Deze query toont alle klanten en, indien beschikbaar, de laatst gehuurde film per klant.

4. **Toon alle categorieën van films en de hoeveelheid films die eraan zijn gekoppeld**

```
SELECT category.name, COUNT(film_category.film_id) AS aantal_films
FROM category
LEFT JOIN film_category ON category.category_id =
film_category.category_id
GROUP BY category.category_id;
```

- Deze query geeft een lijst van alle categorieën met het aantal films dat eraan gekoppeld is, inclusief categorieën zonder films.

### 3. RIGHT JOIN

Een **RIGHT JOIN** is vergelijkbaar met een LEFT JOIN, maar nu worden alle rijen uit de rechtertabel (modellen) weergegeven, ongeacht of er een overeenkomst is in de linkertabel (merken). Dit is handig als je zeker wilt weten dat je alle modellen toont, ook al is er geen bijbehorend merk.

#### Voorbeeld:

```
SELECT merken.naam, modellen.naam
FROM merken
RIGHT JOIN modellen ON merken.id = modellen.merkid;
```

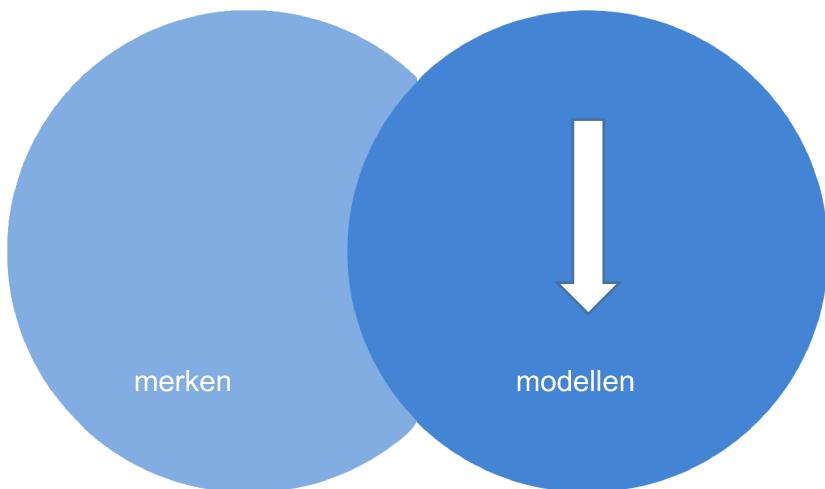
- In dit voorbeeld zouden alle modellen worden weergegeven, zelfs als er geen overeenkomstig merk is.

#### Resultaat:

Resultaat:

| naam | naam |
|------|------|
| audi | A3   |
| audi | A4   |
| NULL | A3   |
| NULL | A4   |

#### Grafisch:



Het Venn-diagram voor een **RIGHT JOIN** laat de volledige cirkel van modellen zien, plus de gedeelten die overlappen met merken.

Voorbeeld: Stel je hebt een lijst van bestellingen en een lijst van producten. Je wilt een overzicht van alle producten, ook als ze nooit zijn besteld.

## KЛАSSИКАЛЬНЫЕ ОФОРМЛЕНИЯ

### SAKILA DATABASE

5. **Toon een lijst van alle inventarisitems met de bijbehorende film, zelfs als er geen inventaris is gekoppeld aan een film**

```
SELECT inventory.inventory_id, film.title
FROM film
RIGHT JOIN inventory ON film.film_id = inventory.film_id;
```

- Dit toont alle inventarisitems en de gekoppelde film, zelfs als een film momenteel niet in de inventaris zit.

6. **Vind alle steden met winkels en de namen van die winkels, inclusief steden zonder winkels**

```
SELECT city.city, store.store_id
FROM city
RIGHT JOIN address ON city.city_id = address.city_id
RIGHT JOIN store ON address.address_id = store.address_id;
```

- Hiermee wordt een lijst gegenereerd van alle steden en, indien aanwezig, de winkels in die stad. Dit kan nuttig zijn voor inzicht in regio's zonder winkels.

## 4. CROSS JOIN

Een **CROSS JOIN** verbindt elke rij van de ene tabel met elke rij van de andere tabel. Het resultaat is een Cartesian product, wat betekent dat het aantal rijen het product is van het aantal rijen in beide tabellen. Dit type join is vaak inefficiënt voor grote datasets, maar kan nuttig zijn voor specifieke analyses.

### Voorbeeld:

```
SELECT *
FROM merken
CROSS JOIN modellen;
```

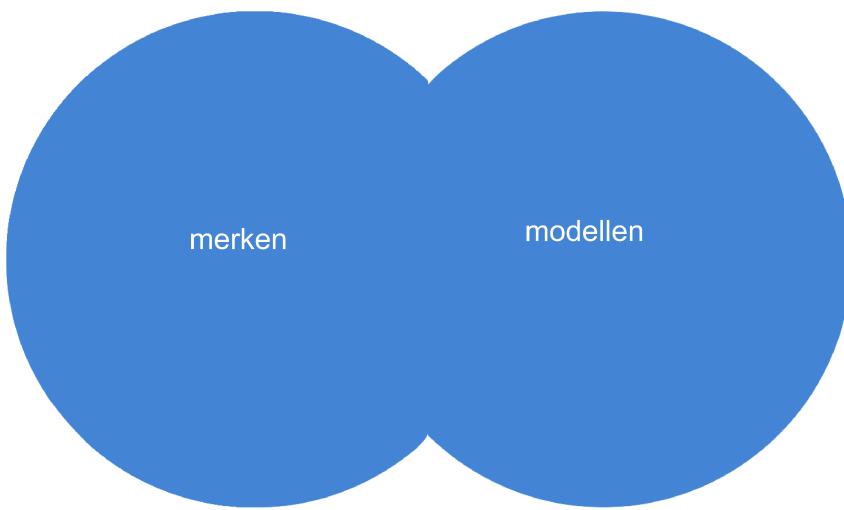
- Deze query combineert elke rij van de tabel `merken` met elke rij van de tabel `modellen`.

### Resultaat:

Resultaat:

| <b>id</b> | <b>naam</b> | <b>id</b> | <b>merkid</b> | <b>naam</b> |
|-----------|-------------|-----------|---------------|-------------|
| 1         | audi        | 1         | 1             | A3          |
| 2         | bmw         | 1         | 1             | A3          |
| 3         | volkswagen  | 1         | 1             | A3          |
| 1         | audi        | 2         | 1             | A4          |
| 2         | bmw         | 2         | 1             | A4          |
| 3         | volkswagen  | 2         | 1             | A4          |

### Grafisch:



Het Venn-diagram voor een **CROSS JOIN** toont de volledige overlapping tussen de twee tabellen, wat betekent dat elke rij in `merken` wordt gecombineerd met elke rij in `modellen`.

Voorbeeld: Stel je hebt een winkel met een lijst van klanten en een lijst van promotieproducten. Je wilt alle mogelijke combinaties van klanten en producten bekijken om te bepalen welke klanten je een promotie-aanbieding wilt sturen. In dit geval kun je een CROSS JOIN gebruiken om elk product met elke klant te combineren.

Klanten tabel:

| klant_id | naam  |
|----------|-------|
| 1        | Jan   |
| 2        | Maria |
| 3        | Peter |

producten tabel:

| product_id | naam      |
|------------|-----------|
| 1          | Koffiemok |
| 2          | T-shirt   |

Resultaat:

| klant | product   |
|-------|-----------|
| Jan   | Koffiemok |
| Jan   | T-shirt   |
| Maria | Koffiemok |
| Maria | T-shirt   |
| Peter | Koffiemok |
| Peter | T-shirt   |

## KЛАSSIKALE OEFENINGEN

### SAKILA DATABASE

#### 7. Geef een combinatie van alle klanten en alle medewerkers

```
SELECT customer.first_name AS klant, staff.first_name AS medewerker
FROM customer
CROSS JOIN staff;
```

- Deze query toont alle mogelijke klant-medewerkercombinaties, wat nuttig kan zijn voor het evalueren van de mogelijke klant-ondersteuning relaties.

#### 8. Lijst van alle films en alle acteurs om mogelijke cast-opties te onderzoeken

```
SELECT film.title, actor.first_name, actor.last_name
FROM film
CROSS JOIN actor;
```

- Deze query toont een volledige lijst van elke mogelijke combinatie van films en acteurs, wat handig kan zijn bij het simuleren van potentiële castingkeuzes.

## Samenvatting

- **INNER JOIN:** Toont alleen de rijen die overeenkomsten hebben in beide tabellen.
- **LEFT JOIN:** Toont alle rijen uit de linkertabel, zelfs als er geen overeenkomst is in de rechtertabel.
- **RIGHT JOIN:** Toont alle rijen uit de rechtertabel, zelfs als er geen overeenkomst is in de linkertabel.
- **CROSS JOIN:** Verbindt elke rij van de ene tabel met elke rij van de andere tabel.

Door deze join types goed te begrijpen, kun je effectief data combineren en relaties tussen tabellen inzichtelijk maken. Dit is een essentieel onderdeel van databasebeheer en helpt bij het opstellen van rijke datasets voor analyses en rapportages.

## 5. COMPLEXE OEFENINGEN

### KЛАССИКАЛЬНЫЕ ОУЧЕНИЯ

#### SAKILA DATABASE

9. **Geef een lijst van films die zijn gehuurd door klanten, met de naam van de medewerker die de verhuur heeft uitgevoerd en de naam van de stad waarin de klant woont**

```
SELECT film.title, customer.first_name AS klant, staff.first_name AS
medewerker, city.city AS woonplaats
FROM film
INNER JOIN inventory ON film.film_id = inventory.film_id
INNER JOIN rental ON inventory.inventory_id = rental.inventory_id
INNER JOIN customer ON rental.customer_id = customer.customer_id
INNER JOIN address ON customer.address_id = address.address_id
INNER JOIN city ON address.city_id = city.city_id
INNER JOIN staff ON rental.staff_id = staff.staff_id;
```

- Deze complexe join combineert meerdere tabellen om te tonen welke films door klanten zijn gehuurd, met de medewerkers die de transactie hebben afgehandeld en de stad waarin de klant woont.

10. **Toon het totaalbedrag dat klanten hebben betaald, gegroepeerd per stad**

```
SELECT city.city, SUM(payment.amount) AS totale_omzet
FROM payment
INNER JOIN rental ON payment.rental_id = rental.rental_id
INNER JOIN customer ON rental.customer_id = customer.customer_id
INNER JOIN address ON customer.address_id = address.address_id
INNER JOIN city ON address.city_id = city.city_id
GROUP BY city.city;
```

- Hier wordt het totale bedrag dat klanten hebben betaald per stad berekend, wat handig kan zijn voor marketinganalyse.