SQL | Snelgids & Samenvatting

R. Brinkman – https://www.RudyBrinkman.nl

SQL | Snelgids & Samenvatting

Verzamelde notifies, syntax en voorbeelden – Rudy Brinkman

Inhoud

Disclaimer, Privaat gebruik, copyrights

Deze samenvatting is gemaakt voor mijn eigen persoonlijke studie en naslag-doeleinden en is grotendeels gebaseerd op diverse boeken en vrij beschikbare informatie (zie onderstaande bronnen).

Aangezien dit mijn persoonlijke samenvatting annex uitgebreid “SQL Cheat Sheet” is, is de informatie gecomprimeerd. Alleen met toelichting die ik relevant vind of voldoende geheugensteun biedt. Wel met syntax en voorbeelden, grotendeels overgenomen van diverse websites**.**

Deze samenvatting gaat er van uit dat er reeds sprake is van een werkende opzet en toegang tot SQL. Zie voor zelf opzetten hiervan: Lokale (My)SQL server installeren

Uitsluiting aansprakelijkheid

Commercieel gebruik en verspreiden van deze samenvatting is **niet** toegestaan.

De originele samenvatting is te vinden op mijn website www.rudybrinkman.nl en daar gratis te downloaden.

Er kunnen aan deze samenvatting geen rechten worden ontleend. Gebruik van de informatie in deze samenvatting is voor eigen rekening en risico. Alle informatie is “as is” en in geen enkel geval zal (gevolg)schade door het gebruik van de informatie in deze samenvatting kunnen worden verhaald op de maker(s). Door gebruik te maken van deze samenvatting verklaart u hiermee bekend en akkoord te zijn.

Rechten

Bronnen

Bronnen voor deze samenvatting, afbeeldingen en voorbeelden. Citaten en overname op basis van het citaatrecht voor Nederland en fair use policy voor USA.

Boeken:

– SQL for Dummies, Allen G. Taylor, 9th edition, ©2019 John Wiley & Sons, Inc.
– SQL: Easy SQL Programming & Database Management For Beginners by Felix Alvaro
– SQL with practice exercises. .sLearn SQL Fast © 2016 by D Armstrong
– Fundament Programmeren PHP8 en MySQL, A.C. Gijssen, Uitgeverij Instruct

Online:

https://www.w3schools.com/sql/
https://beginnersbook.com/2018/11/introduction-to-sql/
https://www.dataquest.io/
https://learnsql.com/
https://www.w3resource.com/sql/tutorials.php
https://nl.wikipedia.org/wiki/Categorie:SQL
https://www.dofactory.com/sql
https://www.tutorialspoint.com/sql/index.htm

Inleiding

SQL (Structured Query Language) is een *vraagtaal* om uit databases gegevens op te raadplegen, updaten, verwijderen en op te slaan. Daarnaast kun je met SQL ook databases en tabellen aanmaken.

SQL (Structured Query Language) is een ANSI/ISO-*standaardtaal* voor een relationeel databasemanagementsysteem (DBMS). Het is een gestandaardiseerde taal die gebruikt kan worden voor taken zoals het bevragen en het aanpassen van gegevens in een database.

SQL varianten

Er zijn veel verschillende SQL "dialecten". Een bekend voorbeeld is MySQL, veel gebruikt in internet-omgevingen, en MS SQL (denk aan Access).

Elke variant heeft zijn eigen versie van SQL. In tegenstelling tot programmeertalen lijken ze allemaal erg op elkaar. SQL-vaardigheden in de ene technologie, zoals Microsoft SQL Server, kunnen snel en eenvoudig worden toegepast op een andere, b.v. MySQL. Door SQL te leren, kun je het vele malen gebruiken in verschillende omgevingen.

SQL versus MySQL

SQL is een *taal*, terwijl MySQL een *systeem* is. SQL geeft de mogelijkheid queries te schrijven die *relationele databases* beheren.

MySQL is een *implementatie* van SQL, een *databasesysteem* dat op een server draait. Hiermee kun je query's schrijven met behulp van de SQL-syntax om MySQL-databases te beheren.

SQL Datatypen

SQL Server data types kennen vijf categorieën:

- Nummeriek
- Datum & tijd
- String/character
- Binair
- Diversen (Miscellaneous)

SQL Statements

Belangrijke SQL statemens

- SELECT - extract data uit een database
- UPDATE - update data in een database
- DELETE - delete data in een database
- INSERT INTO – toevoegen data in een database
- CREATE DATABASE – maak een nieuwe database aan
- ALTER DATABASE – modificeer een database (aanpassen)
- CREATE TABLE – maak een nieuwe tabel in de database
- ALTER TABLE – modificeer tabel
- DROP TABLE – delete tabel
- CREATE INDEX – maak een index aan (zoeksleutel)
- DROP INDEX - delete index

!! Let op

Commando's als "drop table" en "drop index" zijn zeer destructief. De inhoud is volledig verdwenen wanneer je dit toepast.

- SQL statements zijn *English-like* database queries.
- **Keywords zijn o.a. SELECT, UPDATE, WHERE, ORDER BY, etc.**

Commentaar toevoegen

Het is een goed gebruik je 'code', dus ook SQL, te voorzien van commentaar zodat je zelf, maar meer nog anderen, later terug kunnen lezen wat de bedoeling en werking is van je code (qeury).

Maak hier vanaf dat je begint met SQL dus een (goede) gewoonte van.

Commentaar **enkele** regel: **--**

```
-- eerste commentaar

SELECT column_1, column_2, column_3      -- 2e commentaar
FROM table_name;                         -- 3e commentaar
```

Commentaar **meerdere** regels: **/* .. */**

```
/*
Dit is een commentaar
dat meerdere regels beslaat
*/
SELECT column_1, column_2, column_3
  FROM table_name;
```

Selectie: SELECT-statement

Met SELECT haal je data op uit de database.

Alle records tonen:

```
SELECT * FROM table_name;
```

Selectie alleen data uit ***bepaalde*** kolommen:

```
SELECT column1, column2, ...
FROM table_name;
```

Selecteer alle ***unieke*** records (dus geen dubbele)

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

Voorbeeld:

```
SELECT DISTINCT Country FROM Customers;
```

Tellen **unieke** records

```
SELECT COUNT(DISTINCT Country) FROM Customers;
```

In diverse ***SQL-dialecten*** werkt zoiets anders, bijvoorbeeld in Access gebruik je dan:

```
SELECT Count(*) AS DistinctCountries
FROM (SELECT DISTINCT Country FROM Customers);
```

Condities: WHERE

De opdracht zorgt er voor dat alleen die records worden geselecteerd die aan de conditie voldoen.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

WHERE wordt ***ook*** gebruikt in de UPDATE, DELETE, etc. commando's

Wanneer de voorwaarde ***tekst*** bevat gebruik je aanhalingstekens

```
SELECT * FROM Customers
WHERE Country='Nederland';
```

Nummerieke opvragingen gebruiken géén qoutes

```
SELECT * FROM Customers
WHERE CustomerID=1;
```

Operators

Voor de WHERE opdracht kunnen de volgende operators worden gebruikt

Operator	Beschrijving
=	gelijk aan
>	Groter dan
<	Kleiner dan
>=	Groter of gelijk
<=	Kleiner of gelijk
<>	Ongelijk aan
!=	Ongelijk aan (in sommige varianten van SQL)
BETWEEN	tussen een zekere range
LIKE	Patroon (pattern) zoeken
IN	Meerdere waarden voor een kolom

AND, OR en NOT Operators

AND	Als aan alle voorwaarden wordt voldaan
OR	Als aan één van de voorwaarden wordt voldaan
NOT	Als NIET aan de voorwaarde wordt voldaan (exclusie)

Voorbeelden:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Combinatie AND, OR en NOT

```
SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München');
```

```
SELECT * FROM Customers
WHERE NOT Country='Germany' AND NOT Country='USA';
```

!! Geneste combinaties zijn mogelijk hierbij

Zie voor meer operators het gedeelte “Operators, rekenen, vergelijken”

Sorteren: ORDER BY

Voor het sorteren van de resultaten kan gebruik worden gemaakt van **ORDER BY**.

Sortering kan oplopend (Ascending) en aflopend (Descending). Default is: Ascending. Er wordt dan begonnen met de kleinste waarde.

(oplopend)

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC;
```

(aflopend)

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... DESC;
```

Aangezien default ASC is, kan een volgende opdracht gebruikt worden. Deze maakt een lijst van alle records gesorteerd op land (oplopend, alfabetisch)

```
SELECT * FROM Customers
ORDER BY Country;
```

Ook hier zijn diverse variaties mogelijk:

(meerdere kolommen)

```
SELECT * FROM Customers
ORDER BY Country, CustomerName;
```

(verschillende sorteringen!)

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

Toevoegen: INSERT INTO

"Insert into" is letterlijk "Voeg in". Dat is wat het statement dan ook doet. Records toevoegen kan op twee verschillende manieren.

1. Specificeer de kolommen en de waarden:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

2. Wanneer je waarden toevoegt voor alle kolommen van de tabel, hoef je de kolomnamen niet op te geven in de SQL-query. Zorg er echter voor dat de volgorde van de waarden in dezelfde volgorde staat als de kolommen in de tabel.

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

Voorbeeld:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City,
PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen
21', 'Stavanger', '4006', 'Norway');
```

De unieke sleutels die auto-incrementeel zijn voeg je nooit in, die worden automatisch aangemaakt.

Voeg alleen gegevens in specifieke kolommen in:

```
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

Lege velden: Null-waarden

Een veld met een NULL-waarde is een veld ***zonder*** waarde.

Als een veld in een tabel optioneel is, is het mogelijk om een nieuw record in te voegen of een record bij te werken zonder een waarde aan dit veld toe te voegen. Vervolgens wordt het veld opgeslagen met een NULL-waarde.

Controleren of er null-waarden voorkomen in een kolom:

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

Controleren (tegengestelde) of er géén null-waarden voorkomen, met andere woorden of de data in alle kolommen gevuld is:

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

Met een dergelijke opdracht kun je bijvoorbeeld makkelijk zien of er gegevens ontbreken in een tabel.

Voorbeeld:

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

Gegevens bijwerken: UPDATE

Met de update opdracht kan bestaande data worden bijgewerkt.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

!! Let op

De update opdracht gaat **altijd** vergezelfd van het WHERE statement. Als je dat niet gebruikt wordt **alle data bijgewerkt** naar de nieuwe waarde(n).

Voorbeeld:

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

Als je hier de ‘where’ opdracht vergeet wordt dus bij álle klanten de naam en de plaats overschreven met de gegevens bovenstaand.

Meerdere records op correcte wijze in één keer updaten, voorbeeld:

```
UPDATE Customers
SET ContactName='Jan'
WHERE Country='Netherlands';
```

.. en zo heten alle Nederlanders in de database opeens ‘Jan’...

Een nuttig voorbeeld: stel je wilt de aanduiding ‘Holland’ vervangen door ‘Netherlands’ (een ‘zoek en vervang’ opdracht):

```
UPDATE Customers
SET Country='Deutschland'
WHERE Country='Germany';
```

Gegevens verwijderen: DELETE

Met het delete-statement verwijder je records in een tabel.

Record verwijderen

```
DELETE FROM table_name WHERE condition;
```

!! Let op

De WHERE-clausule geeft aan **welke record(s)** moeten worden verwijderd. De records die aan de WHERE voorwaarde voldoen, worden verwijderd. Als je de WHERE-clausule **weglaat**, worden **alle records** in de tabel **verwijderd**!

Voorbeeld:

```
DELETE FROM Customers WHERE CustomerName='Alfred';
```

Alles verwijderen

Om **alle** records te verwijderen waarbij de tabelstructuur in stand blijft kun je (dus) een opdracht als onderstaand gebruiken:

```
DELETE FROM table_name;
```

Limiteren: TOP, LIMIT, FETCH FIRST of ROWNUM

Bij grote databases is het nuttig om het aantal records dat je ophaalt met het select statement te limiteren.

SELECT TOP = specificeren aantal records

De diverse 'dialecten' hebben verschillende manieren om dat te doen:

SQL Server / MS Access

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE condition;
```

MySQL

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

Oracle 12

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s)
FETCH FIRST number ROWS ONLY;
```

Voorbeelden:

```
SELECT TOP 3 * FROM Customers;
```

—

```
SELECT * FROM Customers
LIMIT 3;
```

—

```
SELECT * FROM Customers
FETCH FIRST 3 ROWS ONLY;
```

—

```
SELECT TOP 50 PERCENT * FROM Customers;
```

(in Oracle:)

```
SELECT * FROM Customers
FETCH FIRST 50 PERCENT ROWS ONLY;
```

WHERE statement gebruiken

Naast limitering op basis van aantal of percentage kan er ook een **WHERE** statement worden toegevoegd (zie syntax eerder):

```
SELECT TOP 3 * FROM Customers
WHERE Country='Germany';
```

MySQL

```
SELECT * FROM Customers
WHERE Country='Germany'
LIMIT 3;
```

Oracle

```
SELECT * FROM Customers
WHERE Country='Germany'
FETCH FIRST 3 ROWS ONLY;
```

MIN() and MAX() Functie

Met min() en max() kun je de kleinste of grootste waarde selecteren.

MIN()

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

MAX()

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

Voorbeeld:

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;

—
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

Het **AS** statement definieert een *alias* bijv. zoals hierboven 'SmallestPrice' en geeft dat als resultaat.

Het verschil is met name te zien in de output, zie verder **SQL Alias Syntax**.

Rekenkundige functies: COUNT(), AVG(), SUM()

Met Count(), AVG() en SUM() kun je het volgende doen:

COUNT()	tellen records
AVG()	gemiddelde waarde kolom
SUM()	optellen waarde (som) kolom

Ook hier geldt dat middels WHERE er condities toegepast kunnen worden. Voorbeelden/syntax:

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;                    ← optioneel
```

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;                    ← optioneel
```

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;                    ← optioneel
```

Voor AVG() en SUM() is het vereist dat de kolommen numeriek zij. Voor COUNT() is dat *niet* het geval, die telt namelijk de records zelf.

Voorbeeld: AVG()

```
SELECT AVG(Price)
FROM Products;
```

Zoeken (wildcards): LIKE / NOT LIKE

LIKE wordt in combinatie met WHERE gebruikt om te zoeken naar een patroon in een kolom. Er kunnen twee wildcards worden gebruikt

%	nul, een of meerdere karakters
_	één karakter

*(nb. Access gebruikt een * en ? hiervoor)*

Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

Voorbeelden

Alles beginnend met een 'a':

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

—

Alles eindigend op 'a':

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

Alles met een 'mst' in de records:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%mst%';
```

Begint met 'a' en eindigt op 'o´

```
SELECT * FROM Customers
WHERE ContactName LIKE 'a%o';
```

(enz)

NOT LIKE
Met LIKE kun je ook uitsluiten door NOT LIKE te gebruiken:

```
SELECT * FROM Customers
WHERE CustomerName NOT LIKE 'a%';
```

Alles wat met een 'a' begint wordt hierdoor uitgesloten.

Wildcard karakters

Een wildcard wordt gebruikt om een of meer tekens in een tekenreeks te vervangen. Wildcards worden gebruikt met de LIKE-operator.

De LIKE-operator wordt in een WHERE-component gebruikt om naar een opgegeven patroon in een kolom te zoeken.

SQL	ACCESS	
%	*	0 of meer tekens
_	?	één teken
[ ]	[ ]	één van de tekens tussen de teksthaken. Bijvoorbeeld: h[oa]t vindt 'hat' en 'hot' maar 'hit' niet
^	!	uitsluiten wat tussen de teksthaken staat: h[^oa]t vindt hit, maar niet hot en hat
-	-	enkel teken binnen het opgegeven bereik: k[a-b]t vindt kat en kbt
	#	*elk numeriek karakter. 2#5 vind bijv. 205, 215, 225, 235*

De **[ ]** wildcard is een ‘charlist’, dus een lijst van karakters. Voorbeeld:

```
SELECT * FROM Customers
WHERE City LIKE '[bsp]%';
```

Dit selecteert alle records die beginnen met een b, s of p in de kolom Customers. Wat hier feitelijk staat is

```
SELECT * FROM Customers
WHERE City LIKE 'b%' AND
WHERE City LIKE 's%' AND
WHERE City LIKE 'p%';
```

Er kan ook worden geschreven

```
SELECT * FROM Customers
WHERE City LIKE '[a-k]%';
```

In dit geval duidt de ‘–’ een range aan (van a t/m k). Gebruik je bijvoorbeeld de ‘!’ dan sluit je het juist uit:

```
SELECT * FROM Customers
WHERE City LIKE '[!bsp]%';
```

Alles wat begint met b, s, p wordt niet geselecteerd. Maar dit kan ook worden bereikt door de **NOT LIKE** te gebruiken.

Voorbeeld:

```
SELECT * FROM Customers
WHERE City NOT LIKE '[bsp]%';
```

Meerdere waarden: WHERE IN, NOT IN

De IN operator is een verkorte notatie/mogelijkheid voor meerder ‘OR’ statements.

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

En kan tevens genest worden gebruikt:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

Voorbeeld:

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

Selecteert iedereen in Germany, France, UK. Is hetzelfde resultaat als:

```
SELECT * FROM Customers
WHERE Country = 'Germany' or 'France' or 'UK';
```

NOT IN

Voorbeeld:

```
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

Selectie bereik: BETWEEN

De BETWEEN-operator selecteert waarden **binnen** een bepaald bereik. De waarden kunnen getallen, tekst of datums zijn. De operator BETWEEN is **inclusief: begin- en eindwaarden zijn inbegrepen**.

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Voorbeeld:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

Exclusie: NOT BETWEEN

Waarden uitsluiten kan via **NOT BETWEEN**.

Voorbeeld:

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

BETWEEN met IN of **NOT IN** (range) die in/uitgesloten kan worden is eveneens een mogelijkheid.

Voorbeeld:
Prijs 10 tot/met 20, uitsluiten categorie 1,2,3 en sorteren op prijs

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3)
ORDER BY  Price;
```

BETWEEN met DATUMS

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;
```

of

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```

SQL Alias (AS) Syntax

Aliassen worden gebruikt om een **tabel**, of een **kolom** in een tabel, een tijdelijke naam te geven. Een alias bestaat **alleen** voor de duur van die zoekopdracht.

Er wordt een alias gemaakt met het AS-sleutelwoord.

Alias maken voor een **kolom**:

```
SELECT column_name AS alias_name
FROM table_name;
```

Alias voor een **tabel**:

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

Of:

```
SELECT column_name1, column_name2, ...
FROM table_name AS alias_name;
```

Een handige optie om bijvoorbeeld een meer leesbare layout te krijgen in je uitvoer.

Voorbeeld *(afb. beginnersbook.com)*:

Table: STUDENT

```
STUDENT_ID    STUDENT_NAME   STUDENT_AGE     STUDENT_ADDRESS
----------    ------------   -----------     ---------------
1001           Negan           29             Noida
1002           Sirius          28             Delhi
1003           Ron             28             Delhi
1004           Luna            30             Agra
```

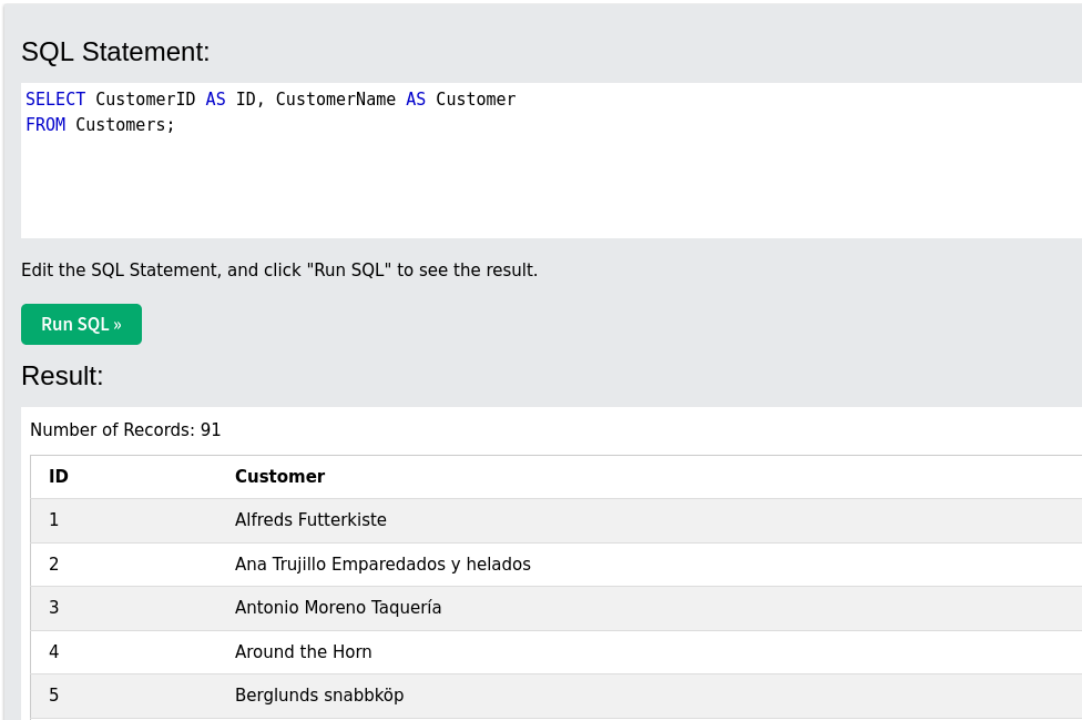
The following SQL statement creates three aliases, alias ID for STUDENT_ID column, alias NAME for STUDENT_NAME column and alias ADDRESS for STUDENT_ADDRESS column.

```
SELECT STUDENT_ID AS ID, STUDENT_NAME AS NAME, STUDENT_ADDRESS ADDRESS
FROM STUDENT;
```

Result:

```
ID             NAME          ADDRESS
-----          ------        -------
1001           Negan         Noida
1002           Sirius        Delhi
1003           Ron           Delhi
1004           Luna          Agra
```

Voorbeeld (*w3schools.com*):



SQL Statement:

```
SELECT CustomerID AS ID, CustomerName AS Customer
FROM Customers;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 91

ID	Customer
1	Alfreds Futterkiste
2	Ana Trujillo Emparedados y helados
3	Antonio Moreno Taquería
4	Around the Horn
5	Berglunds snabbköp

De **AS** kan ook gebruikt worden om tabelnamen korter te maken. In onderstaande voorbeeld data uit twee tabellen (Orders, Customers)

```
SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c.CustomerName='Around the Horn'
  AND c.CustomerID=o.CustomerID;
```

Zonder alias (**AS**)

```
SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName
FROM Customers, Orders
WHERE Customers.CustomerName='Around the Horn'
  AND Customers.CustomerID=Orders.CustomerID;
```

Gebruik alias kan handig zijn in geval er bijvoorbeeld

- meer dan één tabel betrokken is bij een query
- Functies worden gebruikt in de query
- Kolomnamen groot of niet erg leesbaar zijn
- Twee of meer kolommen worden gecombineerd

Combineren met JOIN(S)

Met JOIN kunnen rijen uit verschillende tabellen gecombineerd worden gebaseerd op een kolom met gecombineerde waarden (sleutel).

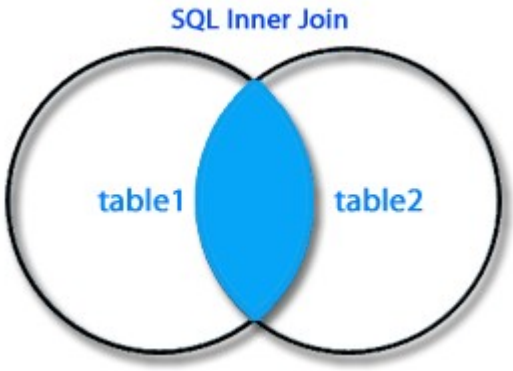
Er is een relatie noodzakelijk tussen de tabel-rijen. Eventueel via een tussentabel.

Voorbeeld:

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

INNER JOIN

Het voorbeeld bevat een INNER JOIN.
Een INNER JOIN is een join die **alle** rijen van **beide** tabellen retourneert, waarbij het sleutel-record van de ene tabel gelijk is aan het sleutel-record van een andere tabel.

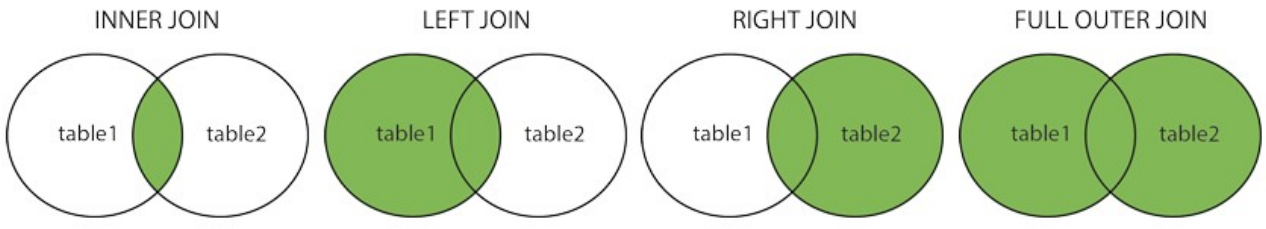


Afb. w3resource.com

Voor dit type samenvoeging is een vergelijkingsoperator nodig om rijen uit de deelnemende tabellen te matchen op basis van een gemeenschappelijk veld of kolom van beide tabellen.

Verschillende typen JOINS

- (INNER) JOIN:
  Retourneert records die overeenkomende waarden hebben in beide tabellen
- LEFT (OUTER) JOIN:
  Retourneert alle records uit de linkertabel en de overeenkomende records uit de rechtertabel
- RIGHT (OUTER) JOIN:
  Retourneert alle records uit de rechtertabel en de overeenkomende records uit de linkertabel
- FULL (OUTER) JOIN:
  Retourneert alle records wanneer er een overeenkomst is in de linker- of rechtertabel



(Afb. – screenshot, w3schools.com)

INNER JOIN

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

Voorbeeld:

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

LEFT JOIN

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

RIGHT JOIN

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

FULL OUTER JOIN

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

!! Een `FULL OUTER JOIN` kan in potentie enorm grote dataset als resultaat geven!

Voorbeeld:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Aangezien alles geretourneerd wordt is het risico groot dat er veel records geretourneerd worden die geen match hebben.

SELF JOIN

Bij een 'self join' is de join binnen de tabel zelf.

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

Een toepassing zou dan bijvoorbeeld zijn cliënten binnen dezelfde plaats te retourneren.

Voorbeeld:

```
SELECT
  A.CustomerName AS CustomerName1,
  B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
  AND A.City = B.City
ORDER BY A.City;
```

Om te voorkomen dat je dubbele records krijgt moet Customer.ID natuurlijk ongelijk zijn aan elkaar, en City wél gelijk.

Samenvoegen: UNION

Met de UNION operator worden resultaten van twee of meer SELECT statements samengevoegd.

Een UNION-opdracht in SQL **combineert gegevens van meerdere query's** van een database. De twee query's moeten hetzelfde aantal kolommen en gegevens bevatten om te worden gecombineerd. Duplicaten worden verwijderd, behalve wanneer UNION ALL wordt gebruikt.

!! Let op

- Elke SELECT-instructie binnen UNION moet hetzelfde aantal kolommen hebben
- De kolommen moeten ook vergelijkbare gegevenstypen hebben
- De kolommen in elke SELECT-instructie moeten ook in dezelfde volgorde staan

UNION

Syntax:

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

Voorbeeld:

```
SELECT * FROM verkoop2017
  UNION
SELECT * FROM verkoop2018
```

UNION ALL

Syntax:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

Voorbeeld:

```
SELECT * FROM verkoop2017
  UNION ALL
SELECT * FROM verkoop2018
```

Voorbeelden met **WHERE**

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION ALL
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

Ook kan de alias (AS) gebruikt worden:

```
SELECT 'Customer' AS Type, ContactName, City, Country
FROM Customers
UNION
SELECT 'Supplier', ContactName, City, Country
FROM Suppliers;
```

Groeperen met GROUP BY

GROUP BY groepeert rijen met dezelfde waarden in overzichtsrijen, zoals "vind het aantal klanten in elk land".

De GROUP BY-instructie wordt vaak gebruikt met statistische functies (COUNT(), MAX(), MIN(), SUM(), AVG()) om de resultaten te groeperen op een of meer kolommen.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Voorbeeld:

(Tellen land met COUNT, gesorteerd op aantal per land)

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

SQL HAVING

Een HAVING-clausule in SQL geeft aan dat een SQL SELECT-instructie alleen rijen mag retourneren waarin de geaggregeerde waarden voldoen aan de opgegeven voorwaarden. HAVING en WHERE worden vaak verward maar ze dienen verschillende doelen.

- HAVING is als WHERE, maar werkt op gegroepeerde records.
- HAVING vereist dat een GROUP BY-clausule aanwezig is.
- Groepen die voldoen aan de HAVING-criteria worden geretourneerd.
- HAVING wordt gebruikt met aggregaten: COUNT, MAX, SUM, etc.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

Voorbeeld:

De volgende SQL-instructie geeft het aantal klanten in elk land. Alleen landen met meer dan 5 klanten opnemen

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

De volgende SQL-instructie vermeldt het aantal klanten in elk land, gesorteerd van hoog naar laag (alleen landen met meer dan 5 klanten opnemen):

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

Test of record bestaat: EXISTS

De **EXISTS**-operator wordt gebruikt om te testen op het bestaan van een record in een subquery. De EXISTS-operator retourneert **TRUE** (het record wordt getoond) als de subquery een of meer records retourneert.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

Vergelijking: ANY en ALL

Doe een vergelijking tussen een enkele kolomwaarde en een reeks andere waarden

ANY

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

!! De operator moet een standaard vergelijkingsoperator zijn zoals: =, <>, !=, >, >=, < of <=).

ALL

- geeft als resultaat een boolean waarde
- geeft TRUE als ALLE waarden van de subquery aan de voorwaarde voldoen
- wordt gebruikt met SELECT-, WHERE- en HAVING-statements

ALL betekent dat de voorwaarde alleen waar is als de bewerking waar is voor alle waarden in het bereik.

Syntax

(met SELECT)

```
SELECT ALL column_name(s)
FROM table_name
WHERE condition;
```

(met WHERE of HAVING)

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

Kopiëren tabel: SELECT INTO

Met **SELECT INTO** kan een kopie van data van de ene tabel naar een **nieuwe** tabel worden gedaan waarbij optioneel ook naar een andere database kan worden gekopieerd.

!! SELECT .. INTO werkt alleen wanneer de doeltabel nog niet bestaat.

Syntax:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

Alleen bepaalde kolommen kopiëren:

```
SELECT column1, column2, column3, ...
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

!! De nieuwe tabel wordt gemaakt met de kolomnamen en typen zoals gedefinieerd in de oude tabel. Nieuwe kolomnamen maken kan met **AS**.

Combinatie (voorbeeld) van tabellen naar één nieuwe tabel kan ook:

```
SELECT Customers.CustomerName, Orders.OrderID
INTO CustomersOrderBackup
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

Structuur kopieeren.

De SELECT INTO kan ook worden gebruikt om alleen de structuur te kopieeren door de WHERE zo te gebruiken dat er geen data wordt overgezet, als volgt:

```
SELECT * INTO newtable
FROM oldtable
/* voorwaarde die nooit waar is */
WHERE 1 = 0;
```

Toevoegen tabel: INSERT INTO .. SELECT

Wanneer je wilt kopiëren naar bestaande tabel kan dit met INSERT INTO SELECT
Tabel moet dus reeds bestaan en/of aangemaakt worden, waarbij data-typen in bron en doel overeen moeten komen.

Syntax:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

Alleen bepaalde kolommen kopiëren:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

Voorbeeld:

```
---- Create table (omdat table nog niet bestaat)
CREATE TABLE TmpTable (FirstName VARCHAR(100), LastName VARCHAR(100))

---- Copy data
INSERT INTO TmpTable (FirstName, LastName)
SELECT FirstName, LastName
FROM Person.Contact
WHERE EmailPromotion = 2

---- Verify that Data in TmpTable exists
SELECT * FROM TmpTable
```

(Handig voor bijvoorbeeld tussentabellen die je vervolgens weer kunt verwijderen.)

Als/Dan – CASE

Doorloopt de voorwaarden en retourneert een waarde wanneer aan de eerste voorwaarde is voldaan (zoals een if-then-else-instructie).

Zodra een voorwaarde waar is, stopt **CASE** met lezen en wordt het resultaat geretourneerd. Als er geen voorwaarden waar zijn, wordt de waarde in de **ELSE**-clausule geretourneerd.

Syntax:

```
CASE
  WHEN condition1 THEN result1
  WHEN condition2 THEN result2
  WHEN conditionN THEN resultN
  ELSE result
END;
```

Voorbeelden:

```
SELECT OrderID, Quantity,
CASE
        WHEN Quantity > 30 THEN 'The quantity is greater than 30'
        WHEN Quantity = 30 THEN 'The quantity is 30'
        ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

—

```
SELECT CustomerName, City, Country
FROM Customers
ORDER BY
(CASE
      WHEN City IS NULL THEN Country
      ELSE City
END);
```

NULL-functies

IFNULL(), ISNULL(), COALESCE(), NVL()

ISNULL() Geeft een eventuele alternatieve waarde wanneer waarde NULL is;
MS Access geeft `IsNull()` een TRUE (-1) of FALSE (0)
MySQL gebruikt MySQL IFNULL() ← is hetzelfde als ISNULL()
Oracle gebruikt NVL() functie hiervoor

Voorbeeld:

```
SELECT ProductName, UnitPrice * (InStock + ISNULL(OnOrder, 0))
FROM Products;
```

Hier stel je OnOrder op de waarde ‘0’ zodat er mee gerekend kan worden.

Stored Procedure

Een **stored procedure** is een voorbereide SQL-code die opgeslagen is/kan worden, zodat de code steeds opnieuw kan worden gebruikt. Aan een stored procedure kunnen ook parameters doorgeven worden, zodat de opgeslagen procedure kan werken op basis van de parameterwaarde(n).

Syntax:

```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;
```

Uitvoeren:

```
EXEC procedure_name;
```

Voorbeeld:

```
CREATE PROCEDURE SelectAllCustomers
AS
SELECT * FROM Customers
GO;
```

Uitvoeren:

```
EXEC SelectAllCustomers;
```

Met parameter:

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)
AS
SELECT * FROM Customers WHERE City = @City
GO;
```

Uitvoeren:

```
EXEC SelectAllCustomers @City = 'London';
```

Meerdere parameters:

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode
nvarchar(10)
AS
SELECT * FROM Customers WHERE City = @City AND PostalCode =
@PostalCode
GO;
```

Uitvoeren:

```
EXEC SelectAllCustomers @City = 'London', @PostalCode = 'WA1 1DP';
```

Operators (rekenen, vergelijken)

Een SQL-operator is een speciaal woord of teken dat wordt gebruikt om taken uit te voeren. Deze taken kunnen van alles zijn, van complexe vergelijkingen tot eenvoudige rekenkundige bewerkingen. Zie een SQL-operator als vergelijkbaar met hoe de verschillende knoppen op een rekenmachine werken.

Rekenkundige operators

+	Optellen
-	Aftrekken
*	Vermenigvuldigen
/	Gedeeld door
%	Modulus (rest). Verdeelt de linker operand door de rechter operand en geeft de rest terug.

Vergelijking

=	Gelijk aan
!=	Ongelijk aan, zoja dan “TRUE”
<>	Ongelijk aan, zoja dan “TRUE”
>	Groter dan
<	Kleiner dan
>=	Groter of gelijk aan
<=	Kleiner of gelijk aan
!<	Controleert of de waarde van de linker operand niet kleiner is dan de waarde van de rechter operand, zo ja, dan wordt de voorwaarde waar.
!>	Controleert of de waarde van de linker operand niet groter is dan de waarde van de rechter operand, zo ja, dan wordt de voorwaarde waar.

Logische operatoren

ALL	De operator ALL wordt gebruikt om een waarde te vergelijken met alle waarden in een andere waardenset.
AND	De AND-operator staat het bestaan van meerdere voorwaarden toe in de WHERE-clausule van een SQL-instructie.
ANY	De operator ANY wordt gebruikt om een waarde te vergelijken met elke toepasselijke waarde in de lijst volgens de voorwaarde.
BETWEEN	De operator BETWEEN wordt gebruikt om te zoeken naar waarden die binnen een reeks waarden vallen, gegeven de minimumwaarde en de maximumwaarde.
EXISTS	De EXISTS-operator wordt gebruikt om te zoeken naar de aanwezigheid van een rij in een opgegeven tabel die aan een bepaald criterium voldoet.
IN	De IN-operator wordt gebruikt om een waarde te vergelijken met een lijst met opgegeven letterlijke waarden.
LIKE	De LIKE-operator wordt gebruikt om een waarde te vergelijken met vergelijkbare waarden met behulp van jokertekens.
NOT	De NOT-operator draait de betekenis om van de logische operator waarmee deze wordt gebruikt. Bv: NIET BESTAAT, NIET TUSSEN, NIET IN, etc. Dit is een negatieve operator.
OR	De OR-operator wordt gebruikt om meerdere voorwaarden te combineren in de WHERE-clausule van een SQL-instructie.
IS NULL	De NULL-operator wordt gebruikt om een waarde te vergelijken met een NULL-waarde.
UNIQUE	De UNIQUE-operator doorzoekt elke rij van een opgegeven tabel op uniciteit (geen duplicaten).

APPENDIX

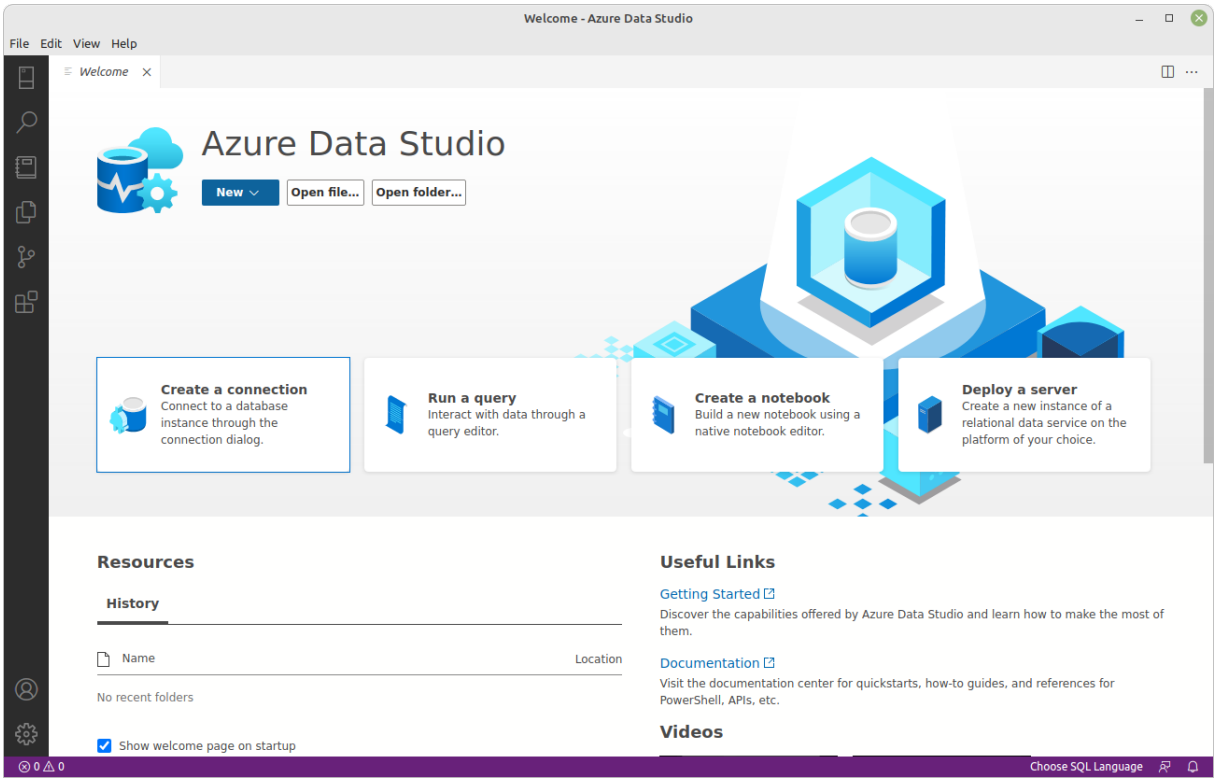
Lokale SQL server installeren

Microsoft – Microsoft biedt diverse (gratis!) tools voor Windows, Linux, Mac en cloud-toepassingen https://docs.microsoft.com/en-us/sql/. De Microsoft oplossingen gebruiken Microsoft SQL (MSQL).

– Microsoft SQL Server Management Studio (SSMS)
Windows. SSMS is een 32-bit windows applicatie en is inclusief een SQL server.

https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms

– Microsoft Azure Data Studio
Windows, Linux, Apple. Wordt standaard met SSMS mee meegleverd maar kan ook apart worden geïnstalleerd. Cross-platform, open-source. Is exclusief SQL server. Kan connecten naar diverse SQL-servers (on premise, lokale computer, cloud). Zie de QuickStart voor meer informatie.



(Azure Data Studio op Linux Mint laptop)

https://docs.microsoft.com/en-us/sql/azure-data-studio/download-azure-data-studio

Installatie SQL Server: https://docs.microsoft.com/en-us/sql/ of https://www.microsoft.com/nl-nl/sql-server/sql-server-downloads

!! Voor Linux MINT: kies de versie van Ubuntu die daar mee overeenkomt.

– Oracle Database Express (XE)
Linux, Windows, Docker, VirtualBox.

Kennismaking (video)
https://www.oracle.com/nl/database/technologies/appdev/xe.html

XE is **beperkt** tot:

- Maximaal 12 GB gebruikersgegevens
- Maximaal 2 GB RAM in database
- Maximaal 2 CPU-threads

  Ideaal voor ontwikkelaars en als test-omgeving.

Zie https://www.cyberciti.biz/faq/howto-install-linux-oracle-database-xe-server/ voor installatie op Debian-gebaseerde distro's (Ubuntu, Linux Mint)

– XAMPP (MySQL)
Linux. PHP + MySQL+ PhpMyAdmin
https://www.apachefriends.org/index.html

Ideale oplossing voor (web)ontwikkelaars, testen enzovoorts. Geen beperkingen bekend. Open Source.

!! Nooit gebruiken voor productie-omgeving of testomgeving die van buitenaf benaderbaar is want:

- **Geen** wachtwoord voor databasebeheerder;
- MySQL is toegankelijk via een **netwerk**.
- ProFTPD gebruikt een **bekend** wachtwoord;
- De lokale mailserver is **niet** beveiligd.

– WAMPSERVER (MySQL)
Windows. PHP + MySQL+ PhpMyAdmin
https://www.wampserver.com/en/

WAMP kan gebruikt worden als "productie-omgeving" mits er aan voorwaarden wordt voldaan (beveiliging) en daarnaast is bijvoorbeeld Windows10 niet geschikt voor deze taak (limieten op simultane verbindingen die het aankan). Dus alleen zeer beperkt inzetbaar.

Locale server benaderen

Het benaderen van een lokale (LAMP) server gaat vaak via de browser ("localhost") of via specieke tools.

Voor installatie, zie de handleidingen leveranciers.

Beveiliging: SQL Injectie

SQL-injectie vindt meestal plaats wanneer een gebruiker om invoer gevraagd wordt, zoals hun gebruikersnaam/gebruikers-ID, en in plaats van een naam/id geeft de gebruiker u een SQL-instructie die u onbewust op uw database laat uitvoeren daardoor.

Voorbeeld W3schools

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

UserId = is een input. Wat via "=" wordt ingevoerd is altijd geldig. Als een gebruiker nu invult:

```
105; DROP TABLE Suppliers
```

Is het resultaat:

```
SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers;
```

De opdracht wordt domweg uitgevoerd.

Een SQL injectie geeft dus min of meer rechtstreekse toegang tot de database. Wat kan een aanvaller doen? Downloaden, aanpassen of verwijderen van data in de database.

Hoe kun je een SQL injection attack voorkomen?

Eigenlijk heel simpel. Door de input te controleren. Heb je URL's op je website (zoals bol.com) waarbij bijvoorbeeld een ID-nummer van een categorie of product tot de output op de webpagina leiden? Zorg dan altijd dat je niet zomaar het veronderstelde 'ID nummer' uit de URL overneemt in de query, maar controleer of het überhaupt een nummer is en of het nummer voldoet aan de voorwaarden die je daaraan stelt. Bij URL's met tekst erin die leidt tot een SQL query wordt het wat moeilijker. Controleer dan tevens of het een string is, maar ook of de input string voldoet aan wat je ervan verwacht. (Emerce).

SQL-parameters

SQL-parameters zijn waarden die tijdens de uitvoering op een gecontroleerde manier aan een SQL-query worden toegevoegd. Klik hier voor voorbeelden.

Meer informatie:
https://www.google.com/search?q=prevent+sql+injection

Tabellen beheren

Tabellen aanmaken kan alleen wanneer je al een (lege) database hebt. Voor werken met (My)SQL is een lokale server of een server bij een (web)host nodig.

Tabel maken

```
CREATE TABLE tabelnaam (
 veldNaam1 veldtype1 [NOT NULL] [PRIMARY KEY | UNIQUE]
 ,veldNaam2 veldtype2 [NOT NULL] [PRIMARY KEY | UNIQUE]
 [, ...] );
```

Tabel verwijderen

```
DROP TABLE tabelnaam;
```

Tabel vullen

```
INSERT INTO tabelnaam [veldnaam1 [,veldnaam2 [, ....]]]
 VALUES (waarde1[,waarde2[,...]]);
```

Tabel lezen

```
SELECT veldna(a)m(en)
 FROM Tabelnaam
 [WHERE conditie]
 [GROUP BY veldnaam [, veldnaam ...]]
 [HAVING conditie2]
 [ORDER BY veldnaam [ASC | DESC] [, veldnaam [ASC | DESC] ...]];
```

Gegevens bijwerken

```
UPDATE <tabel-naam> SET veldnaam=waarde, veldnaam2=waarde2 WHERE
conditie;
```

Tabel wijzigen

Toevoegen, verwijderen en aanpassen kolommen in tabel.

ALTER TABLE *table_name*
ADD *column_name datatype*;

Tabellen: indexen

Gebruikers kunnen geen indexen zien, ze worden gebruikt om zoekopdrachten/query's te versnellen. Het updaten van een tabel met indexen kost meer tijd dan het updaten van een tabel zonder (omdat de indexen ook een update nodig hebben). Maak alleen indexen aan op kolommen waar vaak naar wordt gezocht.

CREATE INDEX

Maakt een index op een tabel. Dubbele waarden zijn toegestaan:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

Maakt een index op tabel, alleen unieke waarden zijn toegestaan:

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

Index met combinatie van kolommen, voorbeeld:

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

DROP INDEX

Index verwijderen

```
DROP INDEX table_name.index_name;
```

Index verwijderen in Access:

```
DROP INDEX index_name ON table_name;
```

Index verwijderen MySQL/MariaDB

```
ALTER TABLE table_name
DROP INDEX index_name;
```

Database management

Meer details? “Google is your friend!”

CREATE database

```
CREATE DATABASE database_name;
```

USE database

Welke database gebruikt moet worden.

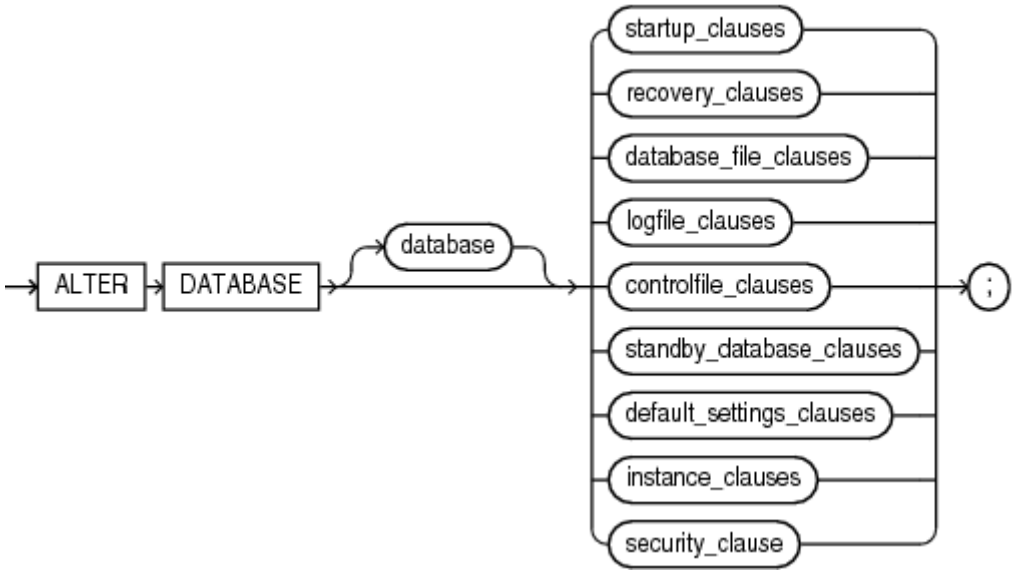
```
USE database_name;
```

BACKUP database

```
BACKUP DATABASE database_name
TO DISK = 'filepath';
```

ALTER database

Karakteristieken van de database aanpassen. Alleen mogelijk met ADMIN rechten.



(afb. Oracle)

Syntax:

```
ALTER DATABASE database_name
```

DROP database

!! Let op – DROP database vernietigt de database dus inclusief alles wat er in is is daarmee voorgoed verloren.

```
DROP DATABASE database_name;
```

W3Schools – SQL Date Data Types

MySQL comes with the following data types for storing a date or a date/time value in the database:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY

SQL Server comes with the following data types for storing a date or a date/time value in the database:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- SMALLDATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: a unique number

Note:
The date types are chosen for a column when you create a new table in your database!

W3Schools – SQL Keywords

Keyword	Description
ADD	Adds a column in an existing table
ADD CONSTRAINT	Adds a constraint after a table is already created
ALL	Returns true if all of the subquery values meet the condition
ALTER	Adds, deletes, or modifies columns in a table, or changes the data type of a column in a table
ALTER COLUMN	Changes the data type of a column in a table
ALTER TABLE	Adds, deletes, or modifies columns in a table
AND	Only includes rows where both conditions is true
ANY	Returns true if any of the subquery values meet the condition
AS	Renames a column or table with an alias
ASC	Sorts the result set in ascending order
BACKUP DATABASE	Creates a back up of an existing database
BETWEEN	Selects values within a given range
CASE	Creates different outputs based on conditions
CHECK	A constraint that limits the value that can be placed in a column
COLUMN	Changes the data type of a column or deletes a column in a table
CONSTRAINT	Adds or deletes a constraint
CREATE	Creates a database, index, view, table, or procedure
CREATE DATABASE	Creates a new SQL database
CREATE INDEX	Creates an index on a table (allows duplicate values)
CREATE OR REPLACE VIEW	Updates a view
CREATE TABLE	Creates a new table in the database
CREATE PROCEDURE	Creates a stored procedure
CREATE UNIQUE INDEX	Creates a unique index on a table (no duplicate values)
CREATE VIEW	Creates a view based on the result set of a SELECT statement
DATABASE	Creates or deletes an SQL database
DEFAULT	A constraint that provides a default value for a column
DELETE	Deletes rows from a table

DESC	Sorts the result set in descending order
DISTINCT	Selects only distinct (different) values
DROP	Deletes a column, constraint, database, index, table, or view
DROP COLUMN	Deletes a column in a table
DROP CONSTRAINT	Deletes a UNIQUE, PRIMARY KEY, FOREIGN KEY, or CHECK constraint
DROP DATABASE	Deletes an existing SQL database
DROP DEFAULT	Deletes a DEFAULT constraint
DROP INDEX	Deletes an index in a table
DROP TABLE	Deletes an existing table in the database
DROP VIEW	Deletes a view
EXEC	Executes a stored procedure
EXISTS	Tests for the existence of any record in a subquery
FOREIGN KEY	A constraint that is a key used to link two tables together
FROM	Specifies which table to select or delete data from
FULL OUTER JOIN	Returns all rows when there is a match in either left table or right table
GROUP BY	Groups the result set (used with aggregate functions: COUNT, MAX, MIN, SUM, AVG)
HAVING	Used instead of WHERE with aggregate functions
IN	Allows you to specify multiple values in a WHERE clause
INDEX	Creates or deletes an index in a table
INNER JOIN	Returns rows that have matching values in both tables
INSERT INTO	Inserts new rows in a table
INSERT INTO SELECT	Copies data from one table into another table
IS NULL	Tests for empty value
IS NOT NULL	Tests for non-empty values
JOIN	Joins tables
LEFT JOIN	Returns all rows from the left table, and the matching rows from the right table
LIKE	Searches for a specified pattern in a column
LIMIT	Specifies the number of records to return in the result set

NOT	Only includes rows where a condition is not true
NOT NULL	A constraint that enforces a column to not accept NULL values
OR	Includes rows where either condition is true
ORDER BY	Sorts the result set in ascending or descending order
OUTER JOIN	Returns all rows when there is a match in either left table or right table
PRIMARY KEY	A constraint that uniquely identifies each record in a database table
PROCEDURE	A stored procedure
RIGHT JOIN	Returns all rows from the right table, and the matching rows from the left table
ROWNUM	Specifies the number of records to return in the result set
SELECT	Selects data from a database
SELECT DISTINCT	Selects only distinct (different) values
SELECT INTO	Copies data from one table into a new table
SELECT TOP	Specifies the number of records to return in the result set
SET	Specifies which columns and values that should be updated in a table
TABLE	Creates a table, or adds, deletes, or modifies columns in a table, or deletes a table or data inside a table
TOP	Specifies the number of records to return in the result set
TRUNCATE TABLE	Deletes the data inside a table, but not the table itself
UNION	Combines the result set of two or more SELECT statements (only distinct values)
UNION ALL	Combines the result set of two or more SELECT statements (allows duplicate values)
UNIQUE	A constraint that ensures that all values in a column are unique
UPDATE	Updates existing rows in a table
VALUES	Specifies the values of an INSERT INTO statement
VIEW	Creates, updates, or deletes a view
WHERE	Filters a result set to include only records that fulfill a specified condition