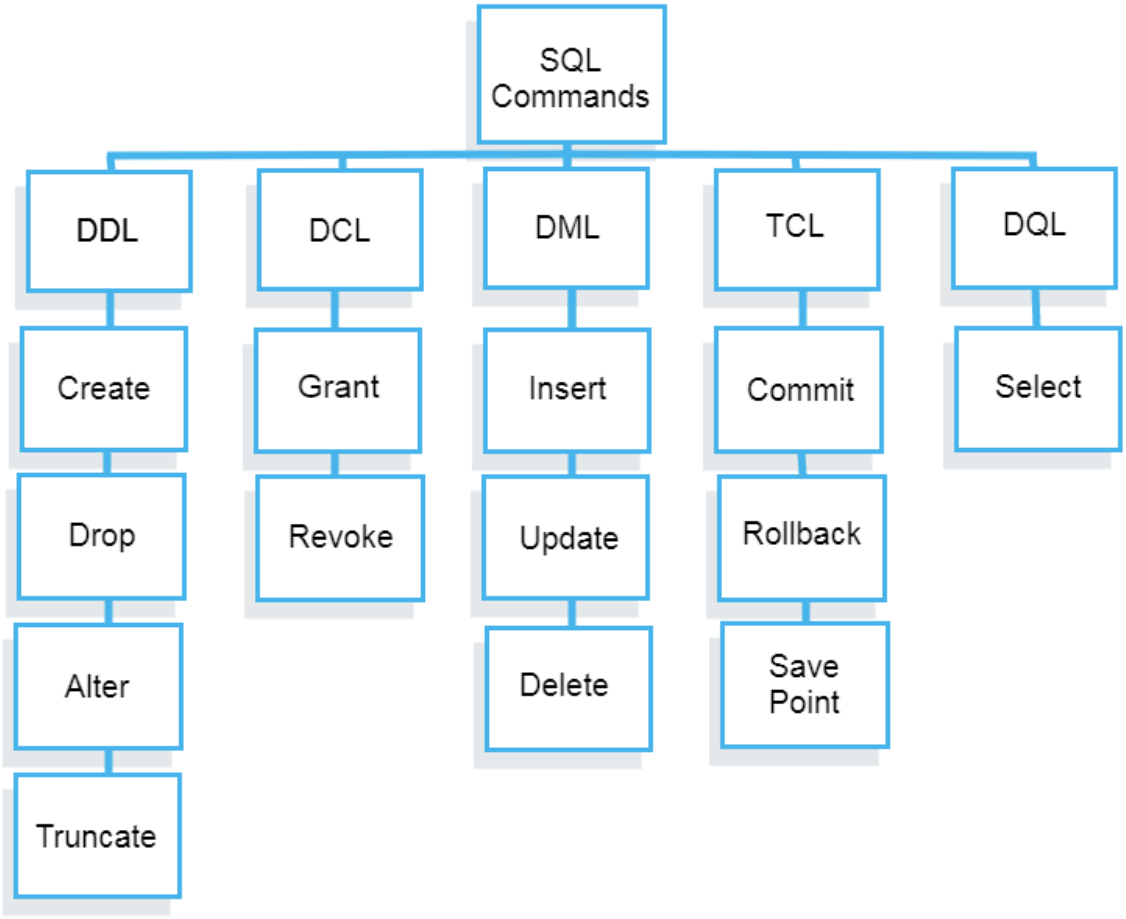
SQL

Wat is SQL?

SQL is een databasetaal die is ontworpen voor het ophalen en beheren van gegevens in een relationele database. SQL is de standaardtaal voor databasebeheer. Alle ***Relational Database Management Systems*** (RDBMS) -zoals MySQL, MS Access, Oracle, Sybase, Postgres en SQL Server gebruiken SQL als hun standaard databasetaal.

Soorten SQL

In de afbeelding hieronder staan vijf soorten veelgebruikte SQL-query's met hun instructies.



DDL

Data Definition Language helpt je bij het definiëren van de databasestructuur of het schema.

Er zijn vijf typen DDL-opdrachten in SQL zijn:

SQL

CREATE-instructies

Deze worden gebruikt om het databasestructuurschema te definiëren.

Syntaxis:

```
CREATE DATABASE DATABASE_NAME;
CREATE DATABASE IF NOT EXISTS DATABASE_NAME;

CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);
CREATE TABLE IF NOT EXISTS TABLE_NAME (COLUMN_NAME DATATYPES[,....]);
```

Bijvoorbeeld:

```
Create database university;
```

DROP-instructies

Drop-opdrachten verwijderen tabellen en databases uit RDBMS.

Syntaxis:

```
Drop object_type object_name;
Drop object_type IF EXISTS object_name;
```

Bijvoorbeeld:

```
DROP DATABASE university;
DROP TABLE students;

DROP DATABASE IF EXISTS university;
DROP TABLE IF EXISTS students;
```

ALTER

Met de opdracht Alter kunt je de structuur van de database wijzigen.

Syntaxis:

Om een nieuwe kolom in de tabel toe te voegen

```
ALTER TABLE table_name ADD column_name COLUMN-definition;
```

SQL

Een bestaande kolom in de tabel wijzigen:

```
ALTER TABLE MODIFY(COLUMN DEFINITION....);
```

Bijvoorbeeld:

```
ALTER TABLE cijfer ADD gewicht integer;
```

TRUNCATE

Deze opdracht werd gebruikt om alle rijen uit de tabel te verwijderen en de ruimte van de tabel vrij te maken.

Syntaxis:

```
TRUNCATE TABLE table_name;
```

Bijvoorbeeld:

```
TRUNCATE table students;
```

DML

Met Data Manipulation Language (DML) kunt je de database-instantie wijzigen door gegevens in te voegen, aan te passen en te verwijderen. Het is verantwoordelijk voor het uitvoeren van alle soorten gegevenswijzigingen in een database.

Er zijn drie basis instructies waarmee het databaseprogramma en de gebruiker gegevens en informatie kunnen verwerken:

- INSERT,
- UPDATE,
- DELETE

INSERT:

Dit is een statement is een SQL-query. Deze opdracht wordt gebruikt om gegevens in de rij van een tabel in te voegen.

SQL

Syntaxis:

```
INSERT INTO TABLE_NAME  (col1, col2, col3,.... col N)
VALUES (value1, value2, value3, .... valueN);

Or

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

Bijvoorbeeld:

```
INSERT INTO students (RollNo, FIrstName, LastName) VALUES ('60', 'Tom', 'Sawyer');
```

UPDATE:

Deze opdracht wordt gebruikt om de waarde van een kolom in de tabel bij te werken of te wijzigen.

Syntaxis:

```
UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]
```

Bijvoorbeeld:

```
UPDATE students
SET FirstName = 'Jhon', LastName= 'Wick'
WHERE StudID = 3:
```

DELETE:

Deze opdracht wordt gebruikt om een of meer rijen uit een tabel te verwijderen.

Syntaxis:

```
DELETE FROM table_name [WHERE condition];
```

Bijvoorbeeld:

```
DELETE FROM students
WHERE FirstName = 'Jhon';
```

SQL

USE

Om een database te manipuleren moet deze eerst geselecteerd (actief gemaakt) worden. Dit wordt gedaan met de instructie USE.

Syntaxis:

```
USE database_namen;
```

Bijvoorbeeld:

```
USE games;
```

DQL

Data Query Language (DQL) wordt gebruikt om de gegevens uit de database op te halen. Het gebruikt slechts één opdracht:

SELECT

Syntaxis:

```
SELECT [DISINCT] {{kolom-expressie} ... | *}
FROM {tabelnaam [alias]} ...
[WHERE conditie]
[AND/OR conditie]
[GROUP BY {kolomspecificatie} ...
[HAVING conditie]]
[ORDER BY {kolomspecificatie sorteervolgorde] ...]
```

```
Kolom-expressie:
{ tabelnaam.* | expressive | functie }
```

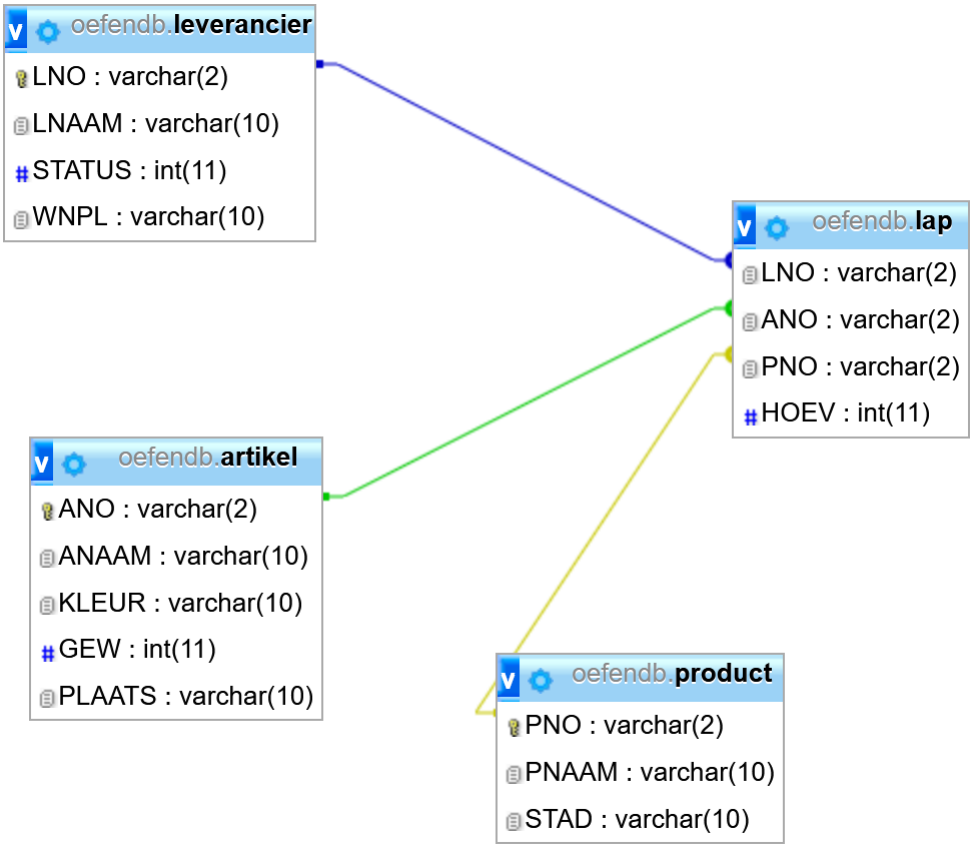
```
Expressie:
{ kolomspecificatie | constant }
```

Queries

Queries zijn vragen die je stelt aan de database. Er zijn verschillende soorten queries mogelijk. Wanneer je bezig bent met queries ben je bezig met informatie te vragen uit één of meerdere verzamelingen van data. Voor

SQL

de voorbeelden die hier onder volgen, wordt van de volgende database-structuur gebruik gemaakt:



Opdracht:
Probeer deze structuur zelf aan te maken in een ***.sql*** bestand. Gebruik hiervoor Visualstudio Code, Notepad++ of een andere teksteditor die schone tekstbestanden kan aanmaken. Sla het bestand op als ***oefenDB.sql***.

Importeer dit tekstbestand in MySQL in en kijk of het klopt. Zorg dat de database geselecteerd en kies de Designer (misschien onder menu van More) op. Importeer de tabellen in de Designer en kijk of de relaties kloppen.

Data vullen in de database

De databases bevat de informatie van leveranciers die artikelen leveren voor de productie van een bepaald product. De inhoudt van de tabellen is als volgt:

SQL

Tabel Artikel:

ANO	ANAAM	KLEUR	GEW	PLAATS
A1	Moer	Rood	12	Amsterdam
A2	Bout	Groen	17	Oranjestad
A3	Schroef	Blauw	17	Bogota
A4	Schroef	Rood	14	Amsterdam
A5	Nok	Blauw	12	Oranjestad
A6	Kamrad	Rood	19	Amsterdam

Tabel Leverancier:

LNO	LNAAM	STATUS	WNPL
L1	Ellis	20	Amsterdam
L2	Tromp	10	Oranjestad
L3	Arends	30	Oranjestad
L4	Fernandez	20	Amsterdam
L5	Kock	30	Willemstad

Tabel Product:

PNO	PNAAM	STAD
P1	Scherm	Oranjestad
P2	Ponser	Bogota
P3	Lezer	Willemstad
P4	Console	Willemstad
P5	Schijf	Amsterdam
P6	Terminal	Caracas
P7	Band	Amsterdam

SQL

Tabel LAP (Leverancier-Artikel-Product):

LNO	ANO	PNO	HOEV
L1	A1	P1	200
L1	A1	P4	700
L2	A3	P1	400
L2	A3	P2	200
L2	A3	P3	200
L2	A3	P4	500
L2	A3	P5	600
L2	A3	P6	400
L2	A3	P7	800
L2	A5	P2	100
L3	A3	P1	200
L3	A4	P2	500
L4	A6	P3	300
L4	A6	P7	300
L5	A1	P4	100
L5	A2	P2	200
L5	A2	P4	100
L5	A3	P4	200
L5	A4	P4	800
L5	A5	P4	400
L5	A5	P5	500
L5	A5	P7	100
L5	A6	P2	200
L5	A6	P4	500

Opdracht:
Voeg de code aan het eerder gemaakte ***.sql*** bestand toe om de tabellen te vullen met de data die hier getoond is per tabel.

Simpele query

De basis instructie voor de SELECT is:

SELECT <Veldnaam 1>, <Veldnaam 2> of <*>
FROM <Tabelnaam 1>, <Tabelnaam 2>;

Bij het SELECT gedeelte kunnen één of meerdere velden van de verschillende tabellen geschreven worden. Deze worden dan in het overzicht getoond die de RDMS maakt.

SQL

Een * wordt gebruikt om alle velden te tonen van een tabel.

Bijvoorbeeld, als gevraagd wordt:

Geef een overzicht van alle productnummers van producten waar door leveranciers artikelen aan geleverd wordt.

Dan zou de query als volgt zijn:

```
1 SELECT pno from lap;
2
```

Je zou in eerste instantie misschien kiezen voor de de tabel producten om daar de productnummers van te tonen. Maar het kan zijn dat niet voor alle producten artikelen worden geleverd door een leverancier. De tabel LAP bevat de informatie waar door verschillende leveranciers artikelen geleverd worden. Het resultaat van deze query is:

pno
P1
P1
P1
P2
P2
P2
P2
P2
P3
P3
P4
P4
P4
P4
P4
P4
P4
P4
P5
P5
P6
P7
P7
P7

Merk op dat bij de uitvoer van de query productnummers meerdere keren voorkomen. Waarom zou dit zijn?

Stel we passen de vraag aan en willen alleen de verschillende productnummers zien dan moet de query aangepast worden.

```
1 SELECT distinct pno from lap;
2
```

De instructie **distinct** is nu toegevoegd aan de query om ervoor te zorgen dat alleen de verschillende productnummers door MySQL getoond wordt.

Het resultaat van deze query is dan:

pno
P1
P2
P3
P4
P5
P6
P7

Query met conditie

Stel dat het volgende gevraagd wordt:

Geef een overzicht van alle artikelen die een rode kleur hebben.

SQL

De query is dan als volgt:

```
SELECT * FROM artikel
where KLEUR = "Rood";
```

Het resultaat van deze query zal dan zijn:

ANO	ANAAM	KLEUR	GEW	PLAATS
A1	Moer	Rood	12	Amsterdam
A4	Schroef	Rood	14	Amsterdam
A6	Kamrad	Rood	19	Amsterdam

Bij deze query is een conditie toegevoegd. Dit gebeurd met behulp van de **where** instructie. Daar moet wordt aangegeven wat de restrictie is die MySQL moet toepassen voor de uitvoer van de query.

Wanneer er meerdere restricties zijn is kan er gebruik gemaakt worden van de **AND** en **OR** instructies bij de **WHERE** instructie. Bekijk de volgende vraag:

Geef de nummers van de Leveranciers in Oranjestad met een status groter dan 20.

De query voor deze vraag is dan als volgt:

```
SELECT lno
FROM leverancier
WHERE WNPL = "Oranjestad"
AND STATUS > 20;
```

Het resultaat van deze query is:

lno
L3

Een query met de **OR**-instructie wordt op dezelfde manier geschreven.

Query met LIKE-instructie

Met behulp van de **LIKE** operator kun je informatie uit een database filtreren die aan een bepaald patroon voldoen. Stel dat het volgende gevraagd wordt:

Geef een overzicht van alle artikelen waar bij de artikel naam met "Sch" begint.

De query ziet er dan als volgt uit:

```
SELECT *
FROM artikel
WHERE ANAAM LIKE "Sch%"
```

SQL

Het resutaat dat getoond wordt door MySQL van de bovengenoemde query is:

ANO	ANAAM	KLEUR	GEW	PLAATS
A3	Schroef	Blauw	17	Bogota
A4	Schroef	Rood	14	Amsterdam

Alleen artikelen waarvan de naam begint met 'Sch' worden getoond.

Het '**%**' teken is een ***wildcard*** of ook wel ***joker***-teken dat gebruikt kan worden in MySQL om verschillende filter-patronen te maken

Er zijn twee jokertekens die vaak worden gebruikt in combinatie met de **LIKE**-operator:

1. Het procentteken (**%**) staat voor nul, één of meerdere tekens.
2. Het onderstrepingsteken (**_**) staat voor één enkel teken.

Het procentteken en het onderstrepingsteken kunnen ook in combinaties worden gebruikt.

Join Queries

De kracht van een **RDBMS** ligt in het feit dat twee of meer tabellen met elkaar verbonden kunnen worden. Dit is de reden waarom er eerst genormaliseerd is en er relaties gemaakt zijn tussen de tabellen.
Een **JOIN** is een vraag waarmee gegevens uit meer dan één tabel worden gehaald.

Er kunnen verschillende joins gemaakt worden. Het is ook mogelijk om de join op verschillende manieren te schrijven.

Bekijk de volgende vraag die gesteld wordt aan de database:

Geef alle combinaties van leveranciersnummer/artikelnummer zodanig dat vestigingsplaats en opslagplaats gelijk zijn.

De query voor deze vraag zou als volgt geschreven kunnen worden:

SQL

```
1 SELECT leverancier.lno, artikel.ano
2 FROM leverancier, artikel
3 WHERE leverancier.WNPL = artikel.PLAATS;
4
```

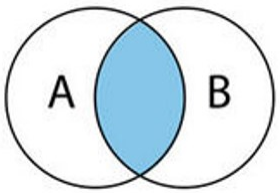
Het resultaat van deze query is dan:

lno	ano
L1	A1
L4	A1
L2	A2
L3	A2
L1	A4
L4	A4
L2	A5
L3	A5
L1	A6
L4	A6

De query zou echter ook met een andere instructie geschreven kunnen worden, de INNER JOIN ON- instructie. De query zou er dan als volgt uit zien:

```
1 SELECT leverancier.lno
2 FROM leverancier
3 inner join artikel
4 on leverancier.WNPL = artikel.PLAATS;
5
```

Het resultaat van deze tweede manier is exact hetzelfde als de eerste. Een INNER JOIN bepaald de doorsnede van twee verzamelingen (tabellen)



Bij een join tussen meer dan twee tabellen moet goed gekeken worden waar de relaties tussen de tabellen en waar welke informatie vandaan gehaald moet worden.

Bekijk de volgende vraag en probeer eerst zelf de juiste query te schrijven voor de vraag.

Geef alle <WNPL, PLAATS> paren, voor leveranciers en artikelen die in dezelfde projecten worden gebruikt.

Bijvoorbeeld, leverancier L1 levert artikel A1 aan project P1. L1 is gevestigd in Amsterdam en artikel A1 ook. Dus <Amsterdam, Amsterdam> is een paar van het resultaat.

Het resultaat is:

WNPL	PLAATS
Amsterdam	Amsterdam
Willemstad	Amsterdam
Willemstad	Oranjestad
Oranjestad	Bogota
Willemstad	Bogota
Oranjestad	Amsterdam
Oranjestad	Oranjestad

SQL

De query zou als volgt geschreven kunnen worden:

```
SELECT DISTINCT leverancier.WNPL, artikel.PLAATS
FROM leverancier, artikel, lap
WHERE leverancier.LNO = lap.LNO
AND lap.ANO = artikel.ANO;
```

Herschrijf de query met de INNER-JOIN instructie. Voer de query uit, als het goed geschreven is hoort hetzelfde resultaat getoond te worden door MySQL.

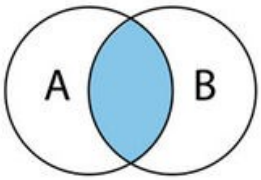
De query met INNER-JOIN ziet er dan als volgt uit.:

```
SELECT DISTINCT leverancier.WNPL, artikel.PLAATS
FROM leverancier
inner join lap
on leverancier.LNO = lap.LNO
inner join artikel
on  lap.ANO = artikel.ANO;
```

In de afbeelding hieronder zie is een overzicht te zien van de verschillende JOIN’s en de SQL-syntax die nodig is om deze te maken.

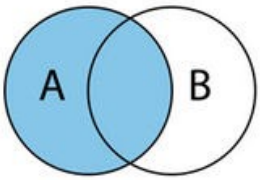
SQL

INNER JOIN



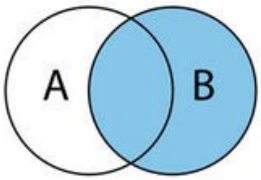

```
SELECT <select_list>
FROM Table1 A
INNER JOIN Table2 B
ON A.Key=B.Key
```

LEFT JOIN



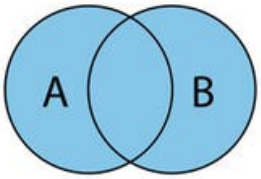

```
SELECT <select_list>
FROM Table1 A
LEFT JOIN Table2 B
ON A.Key=B.Key
```

RIGHT JOIN



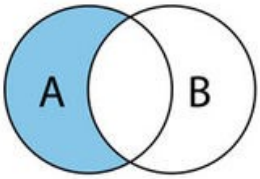

```
SELECT <select_list>
FROM Table1 A
RIGHT JOIN Table2 B
ON A.Key=B.Key
```

FULL JOIN



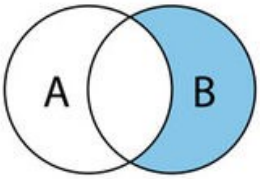

```
SELECT <select_list>
FROM Table1 A
FULL JOIN Table2 B
ON A.Key=B.Key
```

LEFT JOIN / IS NULL



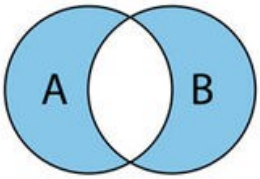

```
SELECT <select_list>
FROM Table1 A
LEFT JOIN Table2 B
ON A.Key=B.Key
WHERE B.Key IS NULL
```

RIGHT JOIN / IS NULL



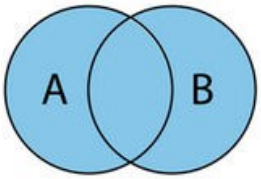

```
SELECT <select_list>
FROM Table1 A
RIGHT JOIN Table2 B
ON A.Key=B.Key
WHERE B.Key IS NULL
```

FULL JOIN / IS NULL




```
SELECT <select_list>
FROM Table1 A
FULL JOIN Table2 B
ON A.Key=B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

UNION




```
SELECT <select_list>
FROM Table1 A
LEFT JOIN Table2 B
ON join_condition
UNION
SELECT <select_list>
FROM Table1 A
RIGHT JOIN Table2 B
ON join_condition
```

SQL
JOINS

SQL

Ingebouwde functies

In een SQL-opdracht kunnen ingebouwde functies gebruikt worden. Het resultaat van een functie is een enkelvoudige waarde. Hieronder staat een lijst van deze functies:

- **COUNT**
  - Aantal waarden van een kolom
- **SUM**
  - Som van de waarden van een kolom
- **AVG**
  - Gemiddelde van de waarden van een kolom
- **MAX**
  - Grootste waarde van een kolom
- **MIN**
  - Kleinste waarde van een kolom

Voor de functies **SUM** en **AVG** moet de kolom een numerieke waarde bevatten.

Voor **COUNT** moet **DISTINCT** worden gebruikt, behalve voor de speciale functie **COUNT(*)**, daar mag **DISTINCT** niet gebruikt worden.

Functies in het SELECT deel van een SQL opdracht

Geef het aantal leveranciers weer.

De query is dan:

```
1 select count(*)
2 from leverancier;
```

De uitvoer is:

count(*)
5

Functies in het SELECT met een niet-lege WHERE deel van een SQL

Geef het aantal leveranciers dat artikel A2 kan leveren.

De query is dan:

```
1 SELECT COUNT(*)
2 FROM LAP
3 WHERE ANO = 'A2'
```

De uitvoer is:

COUNT(*)
2

SQL

Geef de hoeveel dat van artikel met artikelnummer 'A2' is geleverd aan verschillende producten.

De query is dan:

```
SELECT SUM(HOEV)
FROM LAP
WHERE ANO = 'A2';
```

De uitvoer is:

SUM(HOEV)
300

Gebruik van GROUP BY

Het vorige voorbeeld liet zien hoe de totale hoeveelheid van artikelen in een project berekend is. Stel dat men de totale hoeveelheid van ieder artikel in de projecten wil berekenen, dat wil zeggen, voor ieder project artikel moet het artikelnummer met de totale hoeveelheid weer gegeven worden.

De query is dan:

```
SELECT ANO, SUM(HOEV)
FROM LAP
GROUP BY ANO;
```

De uitvoer is:

ANO	SUM(HOEV)
A1	1000
A2	300
A3	3500
A4	1300
A5	1100
A6	1300

Gebruik van HAVING

Geef de nummers van de artikelen die door meer dan één project gebruikt worden.

De query is dan:

```
SELECT ANO
FROM LAP
GROUP BY ANO
HAVING COUNT(*)>1;
```

De uitvoer is:

ANO
A1
A2
A3
A4
A5
A6

SQL

Gebruik van subqueries

Stel de volgende vraag wordt gesteld aan de database:

Geef de nummers van de artikelen die door meer dan één project gebruikt worden.

Voor deze vraag is een ***sub-query*** nodig. Een ***sub-query*** is een SELECT-instructie die is genest

1. in een andere SELECT-,
2. SELECT ... INTO,
3. INSERT ... INTO,
4. DELETE,
5. UPDATE-instructie of
6. in een andere subquery.

Syntax

Er zijn drie vormen van de syntaxis te gebruiken om een sub-query te maken:

1. *comparison* [ANY | ALL | SOME] (SQL-statement)
2. *expression* [NOT] IN (SQL-statement)
3. [NOT] EXISTS (SQL-statement)

Een sub-query bestaat uit de volgende delen:

Deel	Beschrijving
Comparison (vergelijking)	Een expressie- en een vergelijkingsoperator die de expressie vergelijkt met de resultaten van de sub-query.
Expression (expressie)	Een expressie waarvoor de resultaten set van de sub-query wordt doorzocht.
SQL-statement	Een SELECT-instructie, volgens hetzelfde formaat en dezelfde regels als elke andere SELECT-instructie. Het moet tussen haakjes staan.

SQL

De volgende queries zijn voorbeelden van de verschillende opties:

ANY:

```
SELECT * FROM Products
WHERE UnitPrice > ANY
(SELECT UnitPrice FROM OrderDetails
WHERE Discount >= .25);
```

Hier wordt een overzicht gegeven van alle informatie in de tabel Products waar geldt dat de UnitPrice van de tabel Products groter is dan de Unit prijzen van de in de tabel OrderDetails geselcteerde Unit prijzen waar geldt dat de korting (Discount) grote of gelijk is aan 0.25.

IN:

```
SELECT * FROM Products
WHERE ProductID IN
(SELECT ProductID FROM OrderDetails
WHERE Discount >= .25);
```

Wat is de vraag die bij de query hierboven gesteld wordt?
Expression:

```
SELECT LastName,
FirstName, Title, Salary
FROM Employees AS T1
WHERE Salary >= (SELECT Avg(Salary)
FROM Employees
WHERE T1.Title = Employees.Title) Order by Title;
```

Wat is de vraag die bij de query hierboven gesteld wordt?
EXISTS:

SQL

```
SELECT *
FROM tblOrders
WHERE EXISTS (
SELECT NULL
FROM tblCustomers
WHERE tblOrders.CustomerID = tblCustomers.CustomerID
AND tblCustomers.State = 'IL'
);
```

Wat is de vraag die bij de query hierboven gesteld wordt?
NOT EXISTS:

```
SELECT *
FROM tblOrders
WHERE NOT EXISTS (
SELECT NULL
FROM tblCustomers
WHERE tblCustomers.CustomerID = tblOrders.CustomerID
);
```

Wat is de vraag die bij de query hierboven gesteld wordt?