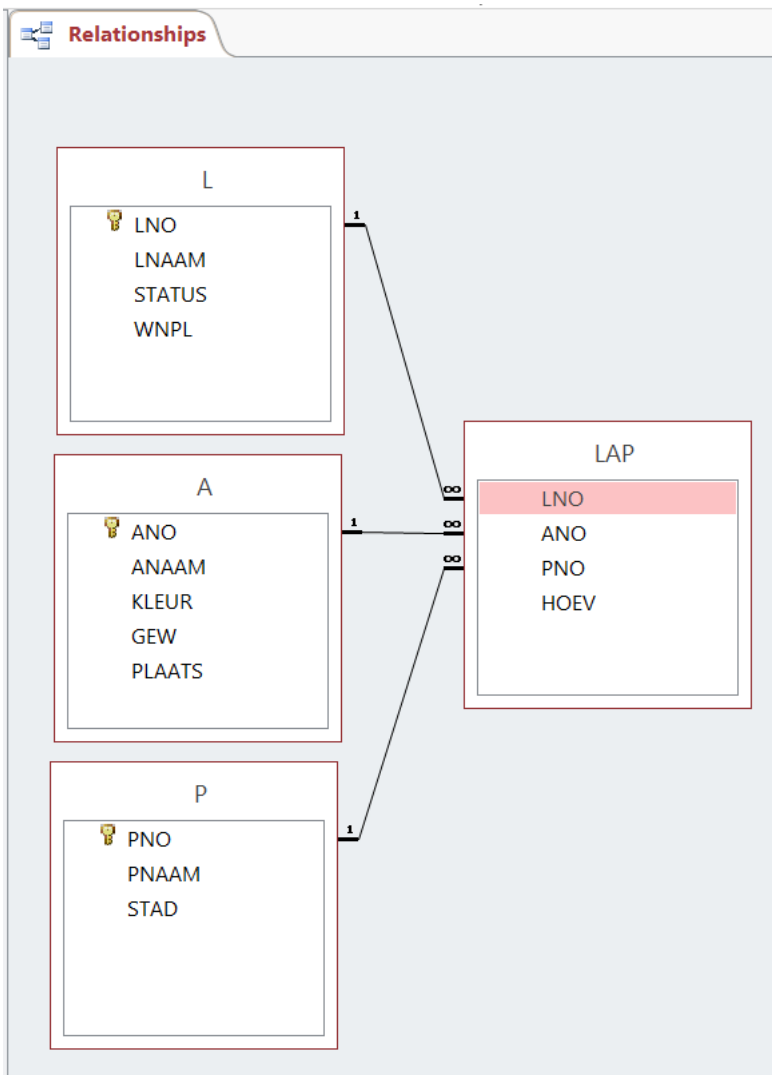
Data Manipulation Language

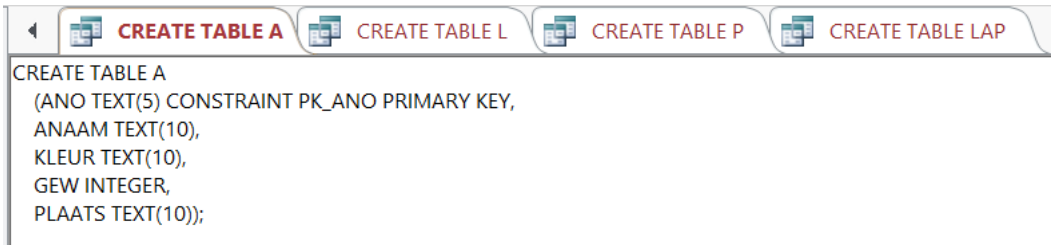
Data Manipulation Language (DML)

In de vorige les hebben we een database structuur gemaakt van 4 tabellen.



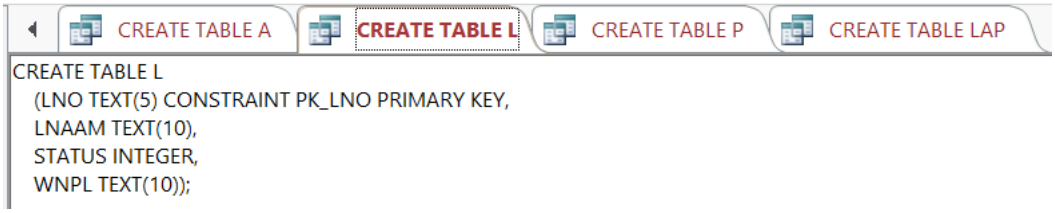
Hiervoor worden de volgende scripts gebruikt voor de verschillende tabellen:

Tabel A

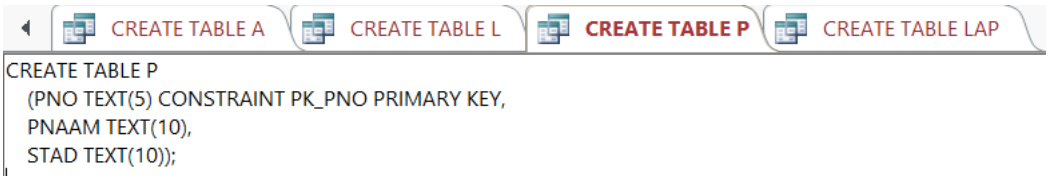


Data Manipulation Language

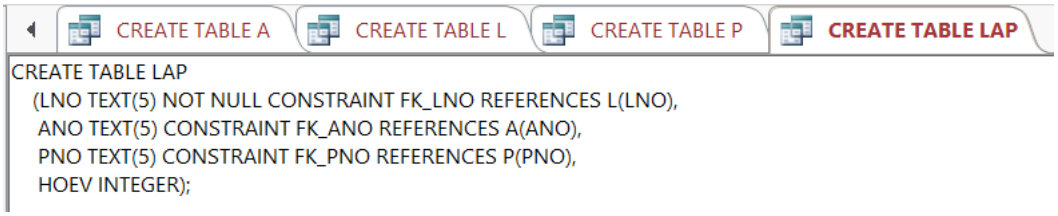
Tabel L



Tabel P



Tabel LAP



De tabellen zijn echter helemaal leeg. Voor het volgend onderdeel dienen ze gevuld te worden met data. In de eerdere methode opende je een tabel en moest je regel voor regel informatie in elke tabel stoppen.

We gaan nu gebruik maken van de DML opdracht INSERT INTO om informatie in de tabellen te stoppen.

De syntax van deze instructie ziet er zo uit:

```
INSERT INTO Tablename [(ColumnNames, ...)]
VALUES (values, ...);
```

Als je één record wilt toevoegen aan tabel L dan zou dat er zo uit zien:

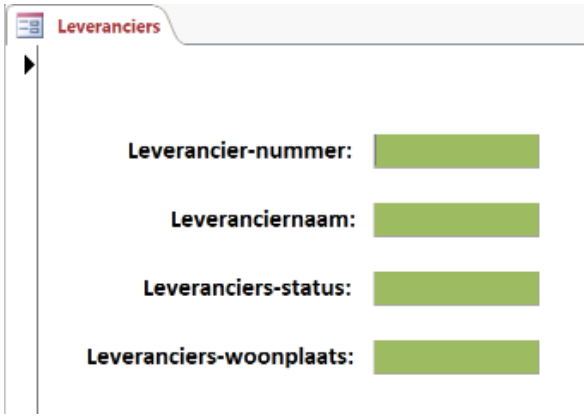
```
INSERT INTO L (LNO, LNAAM, STATUS, WNPL)
VALUES ("L1","Ellis", 20, "Amsterdam");
```

Data Manipulation Language

Formulieren

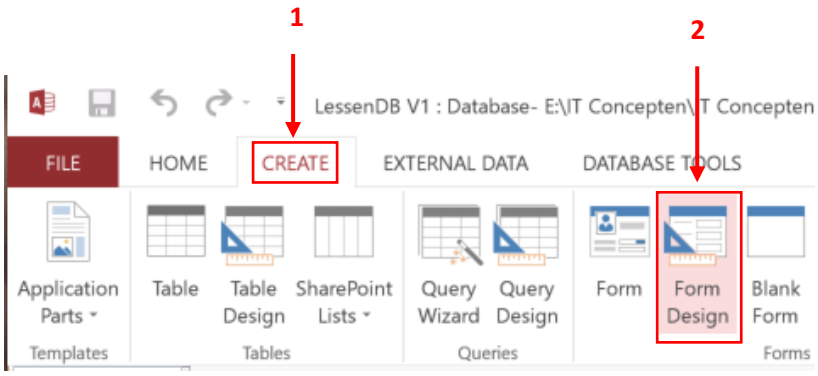
Om de informatie in de tabellen in te voeren gaan we vier simpele formulieren maken.

Voor de leverancier moet het formulier er ongeveer zo uit komen tet zien:



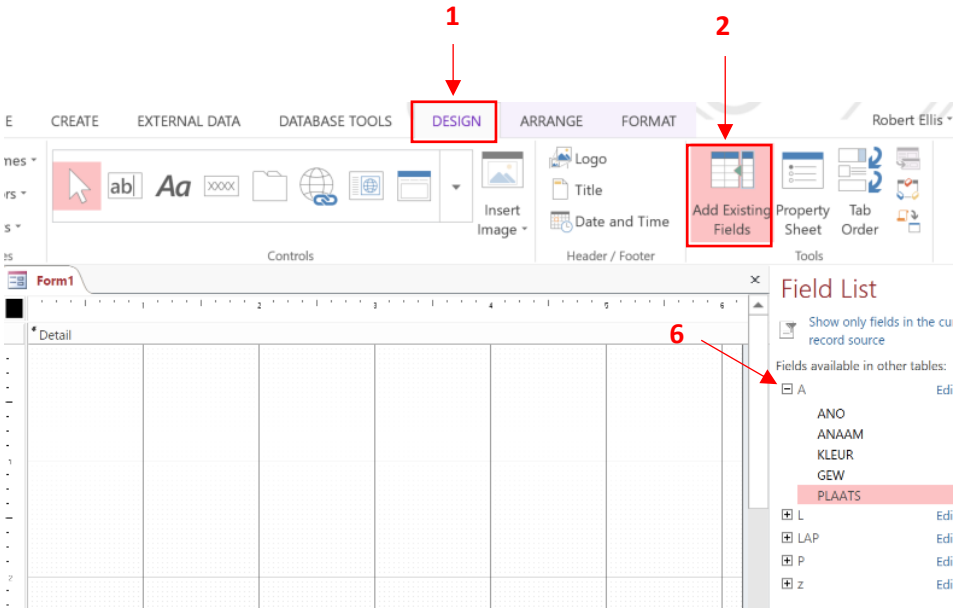
Om een formulier te maken moet je de volgende stappen uitvoeren:

1. In het Lint (*ribbon*) kies Create (*Maken*)
2. Klik op de optie Form Design (*Formulier ontwerpen*)



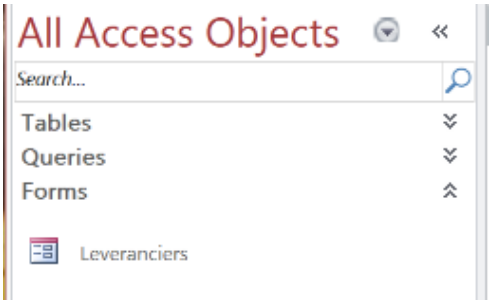
3. Het Formulieren ontwerp scherm wordt dan getoond.
4. Klik op de Tab Design van het Lint
5. Klik op de knop Add Existing Fields
6. Klik op de + naast de naam van de tabel

Data Manipulation Language



7. Sleep de velden die je wil gebruiken één voor één op het formulier.
8. Pas de opmaak aan zodat je formulier er zoals de eerder getoonde afbeelding uit komt te zien.

Door te dubbel-klikken op het formulier in het Objecten paneel links in het venster van Access, start je het formulier op en kun je de informatie gaan invullen. Deze wordt nu automatisch in de tabel toegevoegd.



Maak ook de formulieren voor de andere drie tabellen (A, P en LAP). Gebruik daarna elk formulier om de informatie ***uit hoofdstuk 3*** in te voeren in de tabellen.

Data Manipulation Language

Hier onder wordt de data die in elk formulier moet worden ingevoerd weer gegeven.

L

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

A

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

P

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

LAP

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

Data Manipulation Language

Het formulier voor de tabel LAP dient als laatste te worden ingevoerd. Dit omdat in deze tabel de koppelingen worden gemaakt met de andere drie tabellen. De informatie in de andere drie tabellen moet eerst aanwezig zijn, om de koppelingen te kunnen maken.

Opmerking:
Helaas is het niet mogelijk om in Query deel van Access meerdere insert-instructies in één query te zetten.

Queries

Een query is een vraag. Bij een database is het de vraag naar informatie uit de DB. In Access is het mogelijk om een wizard te gebruiken die je helpt een query te maken.

Echter Access heeft ook de mogelijkheid om zelf een query te schrijven met de vraagtaal **Structured Query Language**, kortweg **SQL**.

In deze les zullen we alleen gebruik maken van de mogelijkheid om een eigen query te schrijven met SQL.

SQL

De vraagtaal SQL heeft een eenvoudige syntax structuur, bestaande uit één opdracht met enkele uitbreidingen die mogelijk zijn.

SQL-opdracht:

Hieronder zie je de volledige structuur van de SELECT opdracht waarmee informatie uit de database gevraagd kan worden.:

```
 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 }

Data Manipulation Language

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.

Een * wordt gebruikt om alle velden te tonen van een tabel. Als het gaat om één tabel of om alle velden van alle tabellen tin het FROM genoemde deel is alleen één * noodzakelijk. Je kunt echter ook combinaties hebben zoals:

SELECT **L.***, **A.ANO**
FROM **L**, **A**;

Hier wordt alle velden van tabel L (Leverancier) getoond en allen het veld ANO van de tabel A (Artikel).

Op het eerste gezicht lijkt dit allemaal ingewikkeld, maar doormiddel van enkele voorbeelden zullen de belangrijkste eigenschappen van de SELECT opdracht aan de hand van voorbeelden uitgelegd worden.

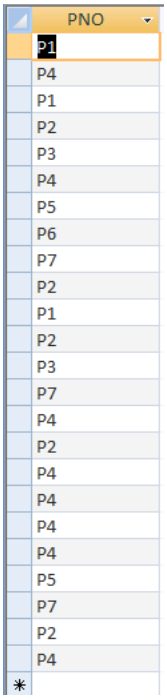
In de voorbeelden wordt de vraag aan de database getoond samen met de query die aan deze vraag beantwoord en het resultaat als die query uitgevoerd wordt.

Data Manipulation Language

Opzoeken

Stel er wordt het volgende gevraagd van de database:
Geef de projectnummers van de projecten waar door leveranciers artikelen aan geleverd worden.

Men wil dus een overzicht van projectnummers hebben. In principe zou dit bij de Tabel P (projecten) kunnen, maar daar er gevraagd wordt op projecten waar artikelen aan geleverd zijn moet er gekeken worden naar de tabel LAP (Leverancier – Artikel – Projecten) die aangeeft hoeveel van een bepaald artikel door een leverancier geleverd wordt. Het zou best kunnen dat er in de tabel Projecten (p) een project staat waar nog niets aan geleverd wordt.



SELECT **PNO**
FROM **LAP;** ← Vergeet de ; niet aan het eind van de query!

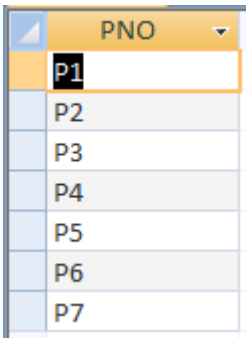
Het resultaat is te zien in de afbeelding hiernaast:

Merk op dat er in het resultaat duplicaten voorkomen.

Om duplicaten te elimineren moet het sleutelwoord DISTINCT gebruikt worden in de SELECT instructie.

SELECT **DISTINCT** PNO
FROM LAP;

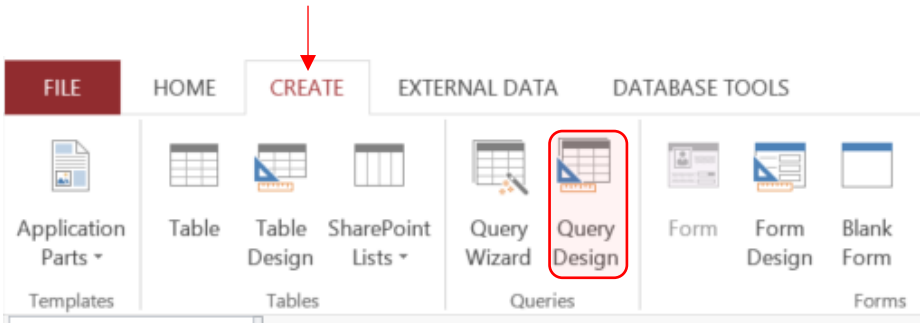
Het resultaat is zoals in de afbeelding hiernaast te zien:



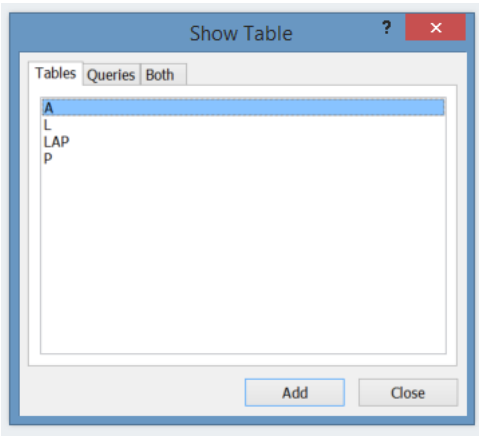
Data Manipulation Language

SQL in Access

Om SQL opdrachten in te voeren in Access kies je in het lint voor de Tab **Create** (*Maken* in de Nederlandse versie). Daar klik je op de optie Query Design.



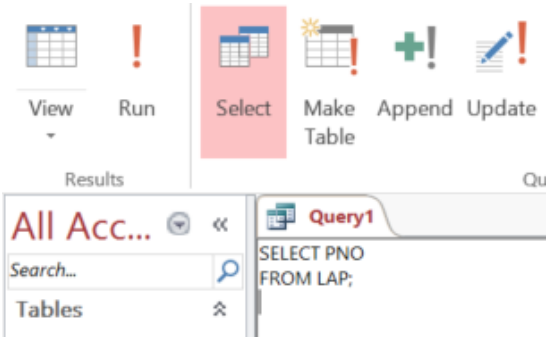
Het programma vraagt je dan welke tabellen je wilt gebruiken. Je kunt deze selecteren of afsluiten zonder een tabel te selecteren.



In het helemaal rechts onder in het Query Design venster staat de optie om je eigen SQL opdracht in te voeren.

Wanneer je op SQL klikt geeft het programma je de mogelijkheid om je eigen SQL opdracht in te voeren.

Met de optie **Run** (*Uitvoeren*) wordt de opdracht uitgevoerd en het resultaat van de opdracht getoond.



Data Manipulation Language

Nu we weten hoe we SQL opdrachten kunnen invoeren en het resultaat van de opdrachten kunnen bekijken, gaan we verder met het bestuderen van verschillende varianten van SQL opdrachten.

Sorteren en uitdrukkingen *(= expressie)*

Stel je wil de volgende vraag stellen aan het DBMS:

Geef voor alle artikelen het artikelnummer en het gewicht van het artikel in grammen.

(in tabel A zijn de gewichten in Engelse ponden gegeven. Voor het omrekenen, moet het gewicht in de tabel met het getal 454 vermenigvuldigd worden.)

De SQL opdracht ziet er dan als volgt uit:

Uitdrukkingen kunnen veldnamen en/of constanten bevatten die door rekenkundige operatoren (+, -, *, /) worden samengevoegd en waarbij optionele haakjes de evaluatievolgorde aangeven.

De ORDER BY opdracht zorgt ervoor dat het resultaat gesorteerd wordt. Waarbij "ASC" (ascending, opklimmend) en "DESC" (descending, dalend) de sorteerrichting bepaald.

Het resultaat is:

Query1

ANO	Expr1001	Expr1002
A1	gewicht in grammen =	5448
A2	gewicht in grammen =	7718
A3	gewicht in grammen =	7718
A4	gewicht in grammen =	6356
A5	gewicht in grammen =	5448
A6	gewicht in grammen =	8626

Data Manipulation Language

Opzoeken van volledige tabellen

Neem de volgende vraag:

Geef de volledige details van alle leveranciers.

```
SELECT    *
FROM      L;
```

Het resultaat is een kopie van de hele L tabel. De '*' is een verkorte notatie voor de rij van alle kolomnamen van de tabel(len) in het FORM gedeelte van de SQL opdracht. De volgorde is gelijk aan de volgorde in de tabelstructuur.

Opzoeken met condities

In een vraag kunnen condities worden gesteld. Stel je hebt de volgende vraag aan het DBMS:

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

```
SELECT    LNO
FROM      L
WHERE     WNPL = "Oranjestad"
AND       STATUS > 20;
```

Het resultaat is L3.

Het WHERE deel van de SQL opdracht mag het volgende bevatten:

- Vergelijkingsoperatoren =, <>, >, >=, <, <=
- Logische operatoren AND, OR en NOT
- Haakjes om de evaluatievolgorde *(rekenvolgorde)* aan te geven.

Data Manipulation Language

Like operator

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 A
WHERE ANAAM LIKE "Sch*";
```

Hier onder zie je een overzicht van alle mogelijke patronen en hun resultaten bij de LIKE operator in SQL.

Kind of match	Pattern	Match (returns True)	No match (returns False)
Multiple characters	a*a	aa, aBa, aBBBa	aBC
	ab	abc, AABB, Xab	aZb, bac
Special character	a[*]a	a*a	aaa
Multiple characters	ab*	abcdefg, abc	cab, aab
Single character	a?a	aaa, a3a, aBa	aBBBa
Single digit	a#a	a0a, a1a, a2a	aaa, a10a
Range of characters	[a-z]	f, p, j	2, &
Outside a range	[!a-z]	9, &, %	b, a
Not a digit	[!0-9]	A, a, &, ~	0, 1, 9
Combined	a[!b-m]#	An9, az0, a99	abc, aj0

Data Manipulation Language

JOIN Vragen

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.

Geef alle combinaties van leveranciers- en artikelinformatie zodanig dat de leverancier en het artikel in dezelfde plaats zijn gelokaliseerd.

De SQL opdracht voor deze vraag is als volgt:

```
SELECT    L.*, A.*
FROM      L, A
WHERE     L.WNPL = A.PLAATS;
```

Het resultaat van deze SQL opdracht is:

Tabel Leverancier (L) Tabel Artikel (A)

Query1

LNO	LNAAM	STATUS	WNPL	ANO	ANAAM	KLEUR	GEW	PLAATS
L1	Ellis	20	Amsterdam	A6	Kamrad	Rood	19	Amsterdam
L1	Ellis	20	Amsterdam	A4	Schroef	Rood	14	Amsterdam
L1	Ellis	20	Amsterdam	A1	Moer	Rood	12	Amsterdam
L2	Tromp	10	Oranjestad	A5	Nok	Blauw	12	Oranjestad
L2	Tromp	10	Oranjestad	A2	Bout	Groen	17	Oranjestad
L3	Arends	30	Oranjestad	A5	Nok	Blauw	12	Oranjestad
L3	Arends	30	Oranjestad	A2	Bout	Groen	17	Oranjestad
L4	Fernandez	20	Amsterdam	A6	Kamrad	Rood	19	Amsterdam
L4	Fernandez	20	Amsterdam	A4	Schroef	Rood	14	Amsterdam
L4	Fernandez	20	Amsterdam	A1	Moer	Rood	12	Amsterdam

komen, namelijk van de tabellen L en A.

In de SQL opdracht wordt daarom in het FROM gedeelte de twee tabellen aangegeven.

De verbinding tussen de twee tabellen komt in het WHERE gedeelte van de SQL opdracht tot stand. De conditie L.WNPL = A.PLAATS heet een **join-conditie**.

Als je terugkijkt naar het overzicht van de tabellen in de vorige les, dan zie je dat zowel leverancier L1 als de artikelen A1, A4 en A6 zich in Amsterdam bevinden

Wanneer de operator in de join een '**=**' teken is, dan heet het een **equijoin**. Als één van de twee kolommen wordt geëlimineerd, dan wordt

het resultaat een **natuurlijke join** genoemd. Zie het voorbeeld hieronder.

```
SELECT    LNO, LNAAM, STATUS, WNPL,
          ANO, ANAAM, KLEUR, GEW
FROM      L, A
WHERE     L.WNPL = A.PLAATS;
```

In de SELECT worden de tabelnamen weggelaten.

Groter-dan join

Geef alle combinaties van leverancier- en artikelinformatie, zodanig dat de naam van de vestigingsplaats van de leverancier alfabetisch na de naam van de opslagplaats van het artikel komt.

```
SELECT    L.*, A.*
FROM      L, A
WHERE     L.WNPL = A.PLAATS;
```

Het resultaat:

Query1

LNO	LNAAM	STATUS	WNPL	ANO	ANAAM	KLEUR	GEW	PLAATS
L2	Tromp	10	Oranjestad	A1	Moer	Rood	12	Amsterdam
L3	Arends	30	Oranjestad	A1	Moer	Rood	12	Amsterdam
L5	Kock	30	Willemstad	A1	Moer	Rood	12	Amsterdam
L5	Kock	30	Willemstad	A2	Bout	Groen	17	Oranjestad
L2	Tromp	10	Oranjestad	A3	Schroef	Blauw	17	Bogota
L3	Arends	30	Oranjestad	A3	Schroef	Blauw	17	Bogota
L5	Kock	30	Willemstad	A3	Schroef	Blauw	17	Bogota
L2	Tromp	10	Oranjestad	A4	Schroef	Rood	14	Amsterdam
L3	Arends	30	Oranjestad	A4	Schroef	Rood	14	Amsterdam
L5	Kock	30	Willemstad	A4	Schroef	Rood	14	Amsterdam
L5	Kock	30	Willemstad	A5	Nok	Blauw	12	Oranjestad
L2	Tromp	10	Oranjestad	A6	Kamrad	Rood	19	Amsterdam
L3	Arends	30	Oranjestad	A6	Kamrad	Rood	19	Amsterdam
L5	Kock	30	Willemstad	A6	Kamrad	Rood	19	Amsterdam

Join met extra conditie

```
SELECT    L.*, A.*
FROM      L, A
WHERE     L.WNPL = A.PLAATS
AND       L.STATUS <> 20;
```

Het resultaat:

Query1

LNO	LNAAM	STATUS	WNPL	ANO	ANAAM	KLEUR	GEW	PLAATS
L2	Tromp	10	Oranjestad	A5	Nok	Blauw	12	Oranjestad
L2	Tromp	10	Oranjestad	A2	Bout	Groen	17	Oranjestad
L3	Arends	30	Oranjestad	A5	Nok	Blauw	12	Oranjestad
L3	Arends	30	Oranjestad	A2	Bout	Groen	17	Oranjestad

Data Manipulation Language

De SQL opdracht van hierboven kan ook als volgt geschreven worden:

```
SELECT    L.*, A.*
FROM      L, A
WHERE     L.WNPL = A.PLAATS
AND       L.STATUS <> 20;
```

Opzoeken van gespecificeerde kolommen van een join

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

```
SELECT    L.LNO, A.ANO
FROM      L, A
WHERE     L.WNPL = A.PLAATS;
```

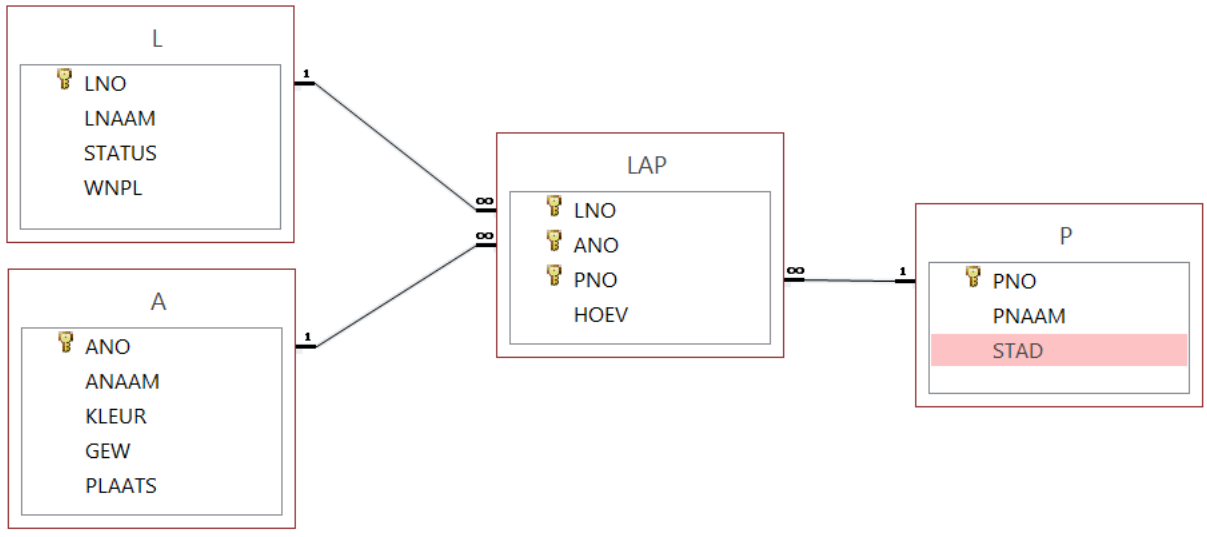
Het resultaat is:

Query1

LNO	ANO
L1	A6
L1	A4
L1	A1
L2	A5
L2	A2
L3	A5
L3	A2
L4	A6
L4	A4
L4	A1

Join van drie tabellen

Bij een join van *drie tabellen* moet je het plaatje van de relaties goed bekijken.



Data Manipulation Language

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

```
SELECT    DISTINCT L.WNPL, A.PLAATS
FROM      L, A, LAP
WHERE     L.LNO = LAP.LNO
AND       LAP.ANO = A.ANO;
```

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:

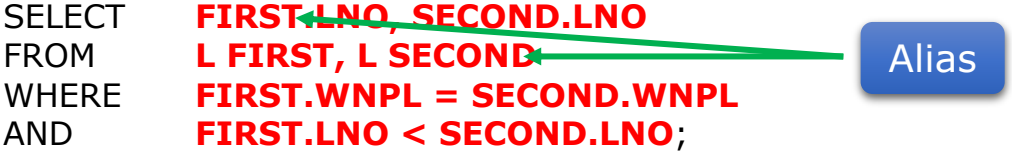
Query1

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

Join van een tabel met zichzelf

Geef alle paren van leveranciersnummers, zodanig dat de twee bijbehorende leveranciers in dezelfde plaats gevestigd zijn.

```
SELECT    FIRST.LNO, SECOND.LNO
FROM      L FIRST, L SECOND
WHERE     FIRST.WNPL = SECOND.WNPL
AND       FIRST.LNO < SECOND.LNO;
```



Het resultaat is:

Query1

FIRST.LNO	SECOND.LNO
L2	L3
L1	L4

Het gaat hier om een join van een tabel met zichzelf. Daarom staat de tabel L twee keer in het FROM deel van de SQL opdracht. Om de twee te onderscheiden, introduceren we twee aliassen “FIRST” en “SECOND”. Deze worden gebruikt om in het SELECT en het WHERE deel van de SQL opdracht de kolomnamen uniek te maken.

De conditie FIRST.LNO< SECOND.LNO wordt gebruikt om:

1. Om de paren van leveranciersnummers (x, x) te elimineren.
2. Om te voorkomen dat paren (x, y) en (y, x) samen in het resultaat voorkomen.

Dit is een voorbeeld waar het gebruik van aliassen noodzakelijk is.

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

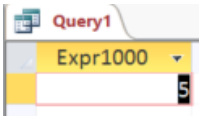
Voor de functies SUM en AVG moet de kolom een numerieke waarde bevatten.
Voor COUNT moet DISTINCT worden gebruikt, behalve voor de speciale functie COUNT(*), daar mag DISTINCT **niet** gebruikt worden.

Functie in het SELECT deel van een SQL opdracht

Geef het aantal leveranciers weer.

```
SELECT    COUNT(*)
FROM      L;
```

Het resultaat is:

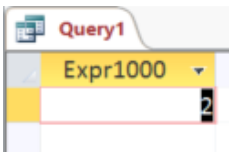


Functie in het SELECT met een niet-lege WHERE deel van een SQL opdracht

Geef het aantal leveranciers dat artikel A2 kan leveren.

```
SELECT    COUNT (*)
FROM      LAP
WHERE     ANO = 'A2';
```

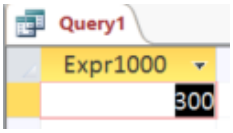
Het resultaat is:



Data Manipulation Language

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

Het resultaat is:



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.

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

Het resultaat is:

Query1

ANO	Expr1001
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.

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

Het resultaat is:

Query1

ANO
A1
A2
A3
A4
A5
A6

Data Manipulation Language

Gebruik van subqueries

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

Een subquery is een SELECT-instructie die is *genest* in een SELECT-, SELECT ... INTO, INSERT ... INTO, DELETE of UPDATE-instructie of in een andere subquery.

Syntax

Er zijn drie vormen van de syntaxis te gebruiken om een subquery te maken:

comparison [ANY | ALL | SOME] (*sqlstatement*)

expression [NOT] IN (*sqlstatement*)

[NOT] EXISTS (*sqlstatement*)

A subquery has these parts:

Part	Description
comparison	Een expressie- en een vergelijkingsoperator die de expressie vergelijkt met de resultaten van de subquery.
expression	Een expressie waarvoor de resultatenset van de subquery wordt doorzocht.
sqlstatement	Een SELECT-instructie, volgens hetzelfde formaat en dezelfde regels als elke andere SELECT-instructie. Het moet tussen haakjes staan.

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.

Data Manipulation Language

IN:

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

Wat is de vraag die bij de query hier boven 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 hier boven gesteld wordt?

EXISTS:

```
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 hier boven 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 hier boven gesteld wordt?