BERT DINGEMANS

GRIP OP DATA

InterActory
Grip op data

Een verkenning van SELECT

Ir. Ing. Bert Dingemans

bert@data-docent.nl

INHOUDSOPGAVE

Inleiding

In dit whitepaper worden databases beschreven. Databases zijn niet weg te denken binnen data gedreven werken. Wil je een beeld krijgen van de basis van select statements binnen SQL en relationele databases bekijk dan dit whitepaper.

Dit whitepaper is een onderdeel van meerdere whitepapers over databases en business intelligence. Databases zijn onder andere gericht op relationele databases en NoSQL databases. We gaan in op het opvragen van data op basis van het select statement binnen de structured query language op basis van data opgeslagen in relationele databases. Daarnaast zijn er whitepapers over datamodelleren in deze serie van whitepapers, zie https://data-docent.nl.

SQL

SQL is een afkorting die staat voor Structured Query Language. Dit is een taal die gebruikt wordt in relationele databases zoals SQL-Server, MS-Access en Oracle. SQL is een taal die is gestandaardiseerd is. Hij is beschreven in een ISO standaard (https://www.iso.org/standard/76583.html) . Dat SQL is gestandaardiseerd is een groot voordeel voor professionals, zoals database beheerders, programmeurs, data analisten en data scientisten, die met databases werken. Door deze standaardisatie kun je werken met data die aanwezig is in relationele databases. Want door deze standaard is de opbouw van een SQL statement dezelfde.

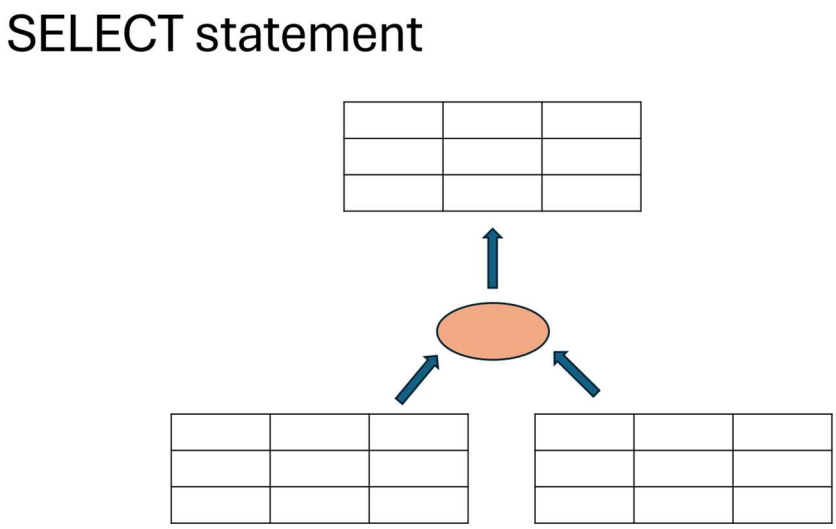
Deze standaardisatie geldt voor alle data gerelateerde commando's die je gebruikt. Voor het definiëren van tabellen en views, manipuleren van de inhoud in tabellen maar ook voor het opvragen van data uit tabellen zijn de commando's beschreven in de ISO standaard. Echter helaas heeft iedere database leverancier eigen uitbreidingen toegevoegd die veelal de werkzaamheden van de data professionals vereenvoudigen.

In dit whitepaper gaan we in op het Select statement in SQL. Het SQL statement wordt gebruikt voor het opvragen van data uit tabellen in een relationele database. Het is daarmee een belangrijk onderdeel van de taal. Het select statement is beschreven in de ISO standaard en vrijwel iedere leverancier van databases heeft eigen uitbreidingen toegevoegd. Het select statement is een belangrijk onderdeel van de SQL taal en is erg rijk aan syntax. Zeker vanuit het perspectief van data gedreven werken en business intelligence is het select statement essentieel. Vandaar om in een whitepaper dit SQL commando te verkennen.

QUERY MET SELECT

Het select statement is gericht op het opvragen van data uit één of meerdere tabellen in een relationele database. Omdat de data in een relationele in de database in de basis uit tabellen ontstaat zorgt het select statement ervoor dat de dat in tabelvorm bevraagd wordt en vervolgens ook weer als een tabel als resultaat getoond als het select statement de bevraging heeft uitgevoerd.

Een select statement bestaat uit een aantal stappen die uitgevoerd worden bij het opvragen van data. Bij die bevraging binnen het select statement worden een aantal tijdelijke transformaties uitgevoerd op de data en vertaald naar een resultaat. Dat wordt ook vaak een resultaat set of dataset genoemd en is meestal tabelvorming.



In bovenstaande afbeelding zie je de basis van de stappen en (tijdelijke) bewerkingen op de data in het select statement. Je ziet dat er een of meerdere tabellen zijn in de database waarin de data is opgeslagen die we nodig hebben om onze vraag te beantwoorden. Daarbij is in het select statement feitelijk de eerste stap in het select statement de vraag in welke tabellen zit de relevante data. Hiervoor gebruik je het **from** sub commando.

De tweede stap is als er meerdere tabellen zijn, hoe worden deze tabellen aan elkaar verbonden zodat alleen de data in de tabellen die een relatie met elkaar hebben getoond worden. Je maakt hierbij gebruik van **joins** en geeft op welke regels hiervoor gelden.

Ten derde wordt gefilterd in de tabellen welke data relevant is om getoond te worden in de resultaat set. Dit wordt gedaan met het **where** sub commando. Filteren kan een complexe bewerking zijn op de data omdat er een combinatie van vragen aan de data gesteld kunnen worden in de vorm van filters.

Ten vierde ga je de gegevens in de resultaat et sorteren op volgorde zetten. Dit kan natuurlijk zowel oplopend en aflopend zijn. In het select statement wordt daarvoor **order by** gebruikt. Daarnaast kunnen er een aantal extra bewerkingen worden gedaan. Bijvoorbeeld groeperen van data als je berekeningen wilt opnemen in het resultaat zoals optellen, gemiddelde etc. Daarvoor gebruiken we het **group by** sub commando.

Laatste stap is dat de resultaat set bepaald wordt. Welke kolommen gaan we tonen aan de gebruiker die het select statement heeft uitgevoerd. Alle elementen hieraan voorafgaand worden meegenomen in het tonen van de uiteindelijke resultaat set. Deze stap bestaat uit het **select** sub commando. De volgorde van deze sub commando's in het **select** statement staat vast en dit is

1. SELECT
2. FROM JOIN
3. WHERE
4. GROUP BY
5. ORDER BY

Hiermee hebben we een basis geïntroduceerd van het **select** commando en welke onderdelen of sub commando's in het select statement kunnen zitten en wat de vastgestelde volgorde is in het **select** statement. In de volgende hoofdstukken gaan we al deze sub commando's afzonderlijk bespreken.

COMMANDO ONDERDELEN

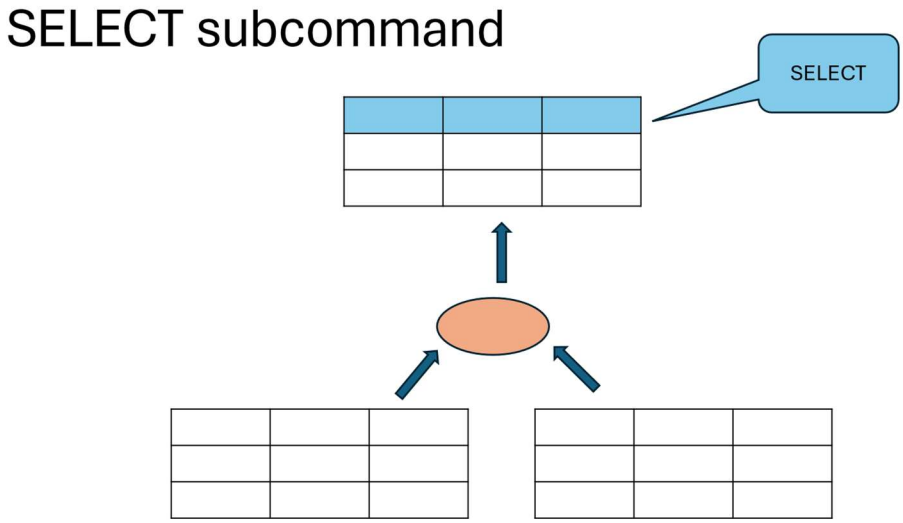
In dit hoofdstuk gaan we de sub commando's behandelen die uiteindelijk zorgen dat data zoals aanwezig in een database getoond wordt. De resultaten worden getoond in een opbouw of structuur waarmee de gebruiker inzicht kan krijgen uit de data. Dat stelt hoge eisen aan het select commando omdat er meerdere soorten transformaties nodig zijn om de data te transformeren naar het bovengenoemde inzicht.

Opbouw select

Select is het eerste sub commando binnen het select commando. Select bepaalt welke data getoond wordt aan de gebruiker. Echter dan wel met een focus op welke kolommen getoond worden. De rijen van de data krijgen daarmee de structuur zoals gedefinieerd met de kolommen in het select sub commando. Welke rijen getoond worden wordt bepaald met het **where** sub commando.

In de afbeelding hieronder zie je dat het select gericht is op de kolommen in de resultaat set. Hierin worden de data getoond zoals die zijn opgebouwd op basis van de andere sub commando's in het select statement. Zoals **From, Where** en **Order by**.

Select



Het select statement bouwt dus een lijst van weer te geven kolommen op Het meest eenvoudige is een select met een *. Deze * zorgt ervoor dat alle kolommen getoond worden. Als er meerdere tabellen met elkaar getoond worden dan wordt een combinatie van alle kolommen van alle te koppelen tabellen getoond. Hieronder een voorbeeld van select waarbij op basis van dit commando alle kolommen en alle rijen getoond worden als resultaat set.

```
SELECT *
FROM VerkoopDashBoard
```

Vervolgens kun je als je niet alle kolommen uit een tabel of tabellen wilt zien een lijst opgeven welke kolommen dat zijn. Je volgt hiermee de namen van de kolommen zoals die in de onderliggende tabel(len) gedefinieerd zijn in het datamodel. Je kunt hierbij zelf de volgorde bepalen van de kolommen. Die worden in de resultaat set in die volgorde getoond.

```
SELECT verkoopdatum
, neerslag
, temperatuur
FROM VerkoopDashBoard
```

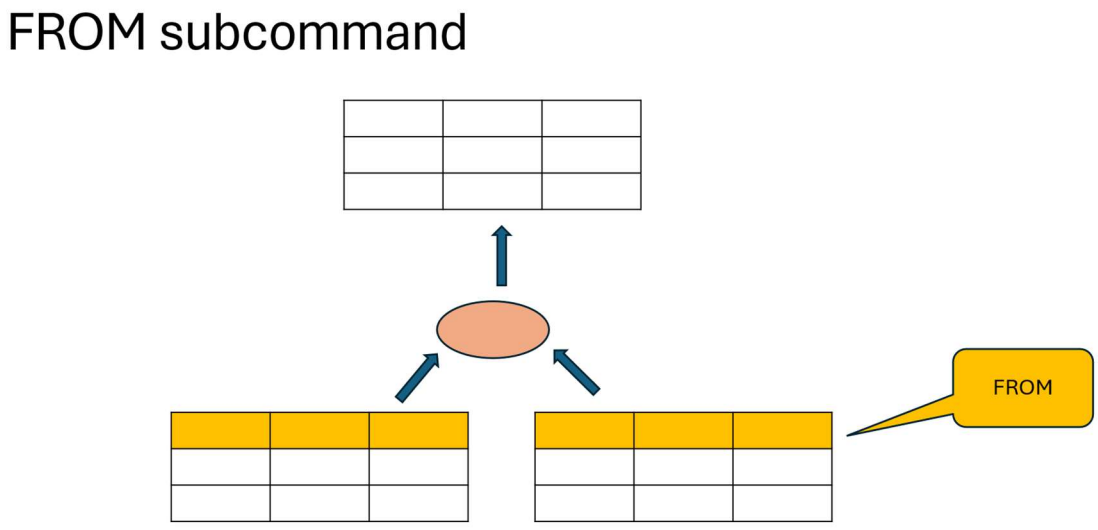
Naast dat je een rechtstreekse verwijzing naar de kolommen uit de tabel kunt opnemen kun je ook extra bewerkingen doen op de inhoud van de kolommen en de wijze waarop ze gepresenteerd worden. Hieronder een paar kenmerkende opties:

- Werken met constanten is dat je een vaste waarde kunt tonen, in dit geval als een tekst 'soort '. Misschien hier niet zo relevant maar bij andersoortige commando een krachtige mogelijkheid.
- Ook kun je een kolom desgewenst hernoemen. Hier wordt gewerkt met as [soort] om de inhoud van de kolom te hernoemen. Dat kan met alle kolommen.
- Vervolgens kun je kolommen met bewerkingen met elkaar combineren. In het voorbeeld hieronder gaan we berekeningen doen op basis van de inhoud van de verschillende kolommen. Ook hierbij krijgt de kolom een logische naam met as [productprijs].

```
SELECT verkoopdatum
, verkoopbedrag
, verkoopaantal
, 'verkoop ' as [soort]
, (verkoopbedrag / verkoopaantal) as [productprijs]
FROM VerkoopDashBoard
```

From

Met **From** gaan we bepalen waar de data vandaan moet komen uit de relationele database. Dat zijn veelal data die afkomstig zijn uit één of meerdere tabellen binnen de relationele database. De meest eenvoudige opzet is dat je de data uit een tabel haalt. Want in dat geval hoef je niet aan te geven op welke wijze de tabellen aan elkaar gekoppeld worden. Hieronder zie je hoe een dergelijke sub commando ervoor zorgt dat de tabellen waaruit de data komen opgenomen worden in de resultaat set.




```
SELECT verkoopdatum
, neerslag
, temperatuur
FROM VerkoopDashBoard
```

Joins

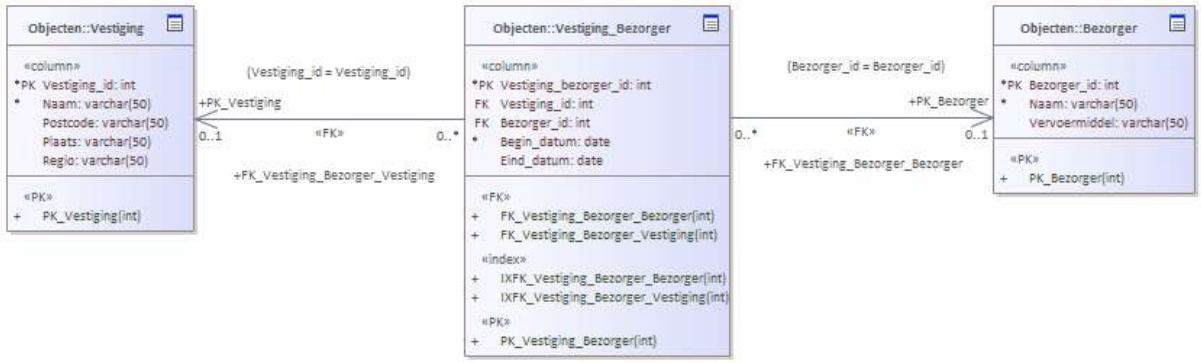
Wil je data uit meer dan één tabel halen dan kun je een lijst opgeven van tabellen die je wilt combineren. Echter als je dat doet krijg je een verrassend resultaat. Hij laat namelijk alle data uit de tabellen zien (dus alle rijen en kolommen) maar ook in alle mogelijke combinaties. Met als gevolg dat je enorm veel rijen terugkrijgt. Bijvoorbeeld als de ene tabel 50 rijen heeft en de andere 100 en je onderneemt geen verdere actie dan krijg je 50 * 100 rijen terug. Dat wordt een cartesisch product genoemd.

Meestal wil je dat niet. Dat betekent dat je de database moet vertellen in het **from** sub commando hoe de tabellen aan elkaar gekoppeld worden. Daarvoor worden joins gebruikt. Bij het werken met joins is het werken met sleutels essentieel. Bij een tabel wordt meestal een primaire sleutel gedefinieerd waarmee je ervoor zorgt dat iedere rij uniek geïdentificeerd wordt. Vaak wordt dat met een nummer gedaan.

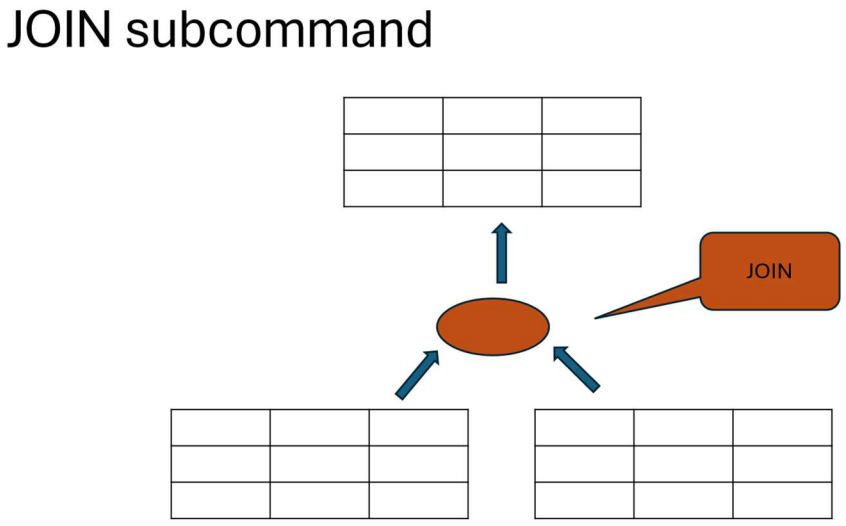
In een andere tabel kan een verwijzing opgenomen worden naar de primaire sleutel van een andere tabel. Hiermee kun je dus adequaat bepalen op welke wijze de data aan elkaar gekoppeld wordt en dit heeft dan direct invloed op de resultaat set. In ons voorbeeld van 50 en 100 rijen zal de resultaat set meestal tussen de 50 en 100 rijen bevatten.

Met joins kun je een aantal krachtige soorten joins leggen. We gaan die hieronder kort toelichten en kijken wat het effect is van het koppelen van tabellen op basis van verschillende joins. Dat doen we op basis van een eenvoudig datamodel. Hieronder zie je een voorbeeld van een datamodel. Je ziet de tabellen en daarbinnen de kolommen.

Daarnaast zie je dat tabellen op basis van kolommen en sleutels aan elkaar kunnen worden gekoppeld.



In het voorbeeld zie je drie tabellen. Een tabel met vestiging data, een tabel met bezorger data en een tabel waarin dit aan elkaar gekoppeld worden. Hierbij definieer je dat een vestiging meerdere bezorgers kan hebben maar ook dat een bezorger voor meerdere vestigingen kan werken. Hieronder gaan we de joins nader verkennen.



De from en de join sub commando's zijn nauw aan elkaar gerelateerd met from geef je op dat tabellen aan elkaar gekoppeld wordt met join geef je aan hoe en welke kolommen je daarvoor gebruikt. Er zijn vier soorten joins die we allemaal kort toelichten. We beginnen met de meest toegepaste de inner join.

INNER JOIN

Bij de inner join begin je in het from commando met het benoemen van de eerste tabel. Vervolgens moet je met join vertellen welke tabel gekoppeld gaat worden aan de eerste tabel. Dat doe je met het sub command JOIN. Er is een toevoeging, in het onderstaande voorbeeld het sub commando INNER. Wat dat inhoudt licht ik zo toe.

Na de join ga je met ON vertellen hoe de tabellen aan elkaar gekoppeld worden in het voorbeeld hieronder zie je dat de vestiging tabel met een join gekoppeld wordt aan de vestiging_bezorger tabel. Met het ON sub commando geef je op op basis van welke kolommen die verbinding gelegd wordt. In het voorbeeld hieronder dus op basis van de primaire sleutel (vestiging_id) uit de vestiging tabel combineren met de vestiging_id uit de vestiging_bezorger tabel. Dat is in deze laatste tabel de verwijzende sleutel. In ons voorbeeld hebben de kolommen dezelfde naam.

Hierbij leg je feitelijk een filter op de data. Want je definieert hier laat alleen die rijen zien uit beide tabellen waar de inhoud van beide kolommen hetzelfde is. In het voorbeeld doen we dat vervolgens nog een keer op precies dezelfde manier waarbij we de bezorger tabel met een inner join koppelen aan de vestiging_bezorger tabel. In de tweede stap echter op basis van de bezorger_id, de primaire sleutel van bezorger en de verwijzende sleutel in vestiging_bezorger. ‘

Bij de inner join laten we alle rijen zien die in beide tabellen gelijk zijn. Is een sleutel leeg (NULL) dan zie je de rijen met een NULL sleutel niet terug.

```
SELECT Bezorger.Naam
, Vestiging.Naam
FROM Bezorger INNER JOIN Vestiging_Bezorger
ON Bezorger.bezorger_id = Vestiging_Bezorger.bezorger_id
INNER JOIN Vestiging
ON Vestiging_Bezorger.vestiging_id = Vestiging.vestiging_id
```

In de tabel hieronder zie je een (deel van) de resultaat set van deze inner join. Je ziet hier dat een bezorger op meerdere vestigingen kan werken, zoals bijvoorbeeld Carla maar ook dat een vestiging meerdere bezorgers kan hebben zie bijvoorbeeld Oude gracht.

bezorger	vestiging
Carla	Oude gracht
Conny	Oude gracht
Karin	Oude gracht
Mille	Oude gracht
Jeroen	Oude gracht
Piet	Oude gracht
Gijsbert	Vianen
Jaap	Vianen
Carla	Vianen
Conny	Vianen
Karin	Vianen
Mille	Vianen
Jeroen	Vianen
Piet	Vianen
Gijsbert	Leerdam
Jaap	Leerdam
Carla	Leerdam
Conny	Leerdam
Karin	Leerdam
Mille	Leerdam
Jeroen	Leerdam
Piet	Leerdam

LEFT JOIN

Bij de left join is de opbouw van het From, Join en On subcommando hetzelfde als bij de inner join. Echter in het select statement vervangen we het Inner subcommando voor Left. Wat is hiervan het effect? In het from statement lees je de nnamen van de tabellen van links naar rechts. In ons voorbeeld is dus bezorger links en vestiging_bezorger rechts.

Het effect nu is dat alle rijen waarbij de inhoud van de sleutels hetzelfde is. Net als bij de Inner join. Echter bij de left join worden ook alle rijen getoond van de bezorger tabel die niet voorkomen in de vestiging_bezorger tabel.

```
SELECT Bezorger.Naam
, Vestiging.Naam
FROM Bezorger FULL JOIN Vestiging_Bezorger
ON Bezorger.bezorger_id = Vestiging_Bezorger.bezorger_id
LEFT JOIN Vestiging
ON Vestiging_Bezorger.vestiging_id = Vestiging.vestiging_id
```

In de resultaat set zie je dit effect. We zien alle rijen waarbij de sleutels zijn ingevuld. Maar je ziet het effect er zijn drie bezorgers die blijkbaar bij geen vestiging werken. Dat levert een bijzonder inzicht. Er zijn blijkbaar bezorgers die nooit bezorgen.

bezorger	vestiging
Gijsbert	Overvecht
Gijsbert	Vreeswijk
Gijsbert	Vianen
Gijsbert	Leerdam
Conny	Oude gracht
Conny	Overvecht
Conny	Vreeswijk
Conny	Vianen
Conny	Leerdam
Jeroen	Oude gracht
Jeroen	Overvecht
Jeroen	Vreeswijk
Jeroen	Vianen
Jeroen	Leerdam
Piet	Oude gracht
Piet	Overvecht
Piet	Vreeswijk
Piet	Vianen
Piet	Leerdam
Peter	NULL
Anneke	NULL
Kees	NULL

RIGHT JOIN

Het right join sub commando werkt precies de andere kant op. Ook hierbij wordt de from opgebouwd met een lijst van tabellen van links naar rechts. Maar nu gaat de rechts voor. In ons voorbeeld dus de vestiging tabel.

```
SELECT Bezorger.Naam
, Vestiging.Naam
FROM Bezorger FULL JOIN Vestiging_Bezorger
ON Bezorger.bezorger_id = Vestiging_Bezorger.bezorger_id
FULL JOIN Vestiging
ON Vestiging_Bezorger.vestiging_id = Vestiging.vestiging_id
```

In de tabel met de resultaat set zie je nu het effect namelijk een aantal vestigingen waar geen bezorgers werken. Ook nu weer een bijzonder inzicht. Kunnen er vestigingen zijn waar geen bezorgers werken, wie doet de bezorgingen dan?

bezorger	vestiging
Carla	Oude gracht
Conny	Oude gracht
Karin	Oude gracht
Mille	Oude gracht
Jeroen	Oude gracht
Piet	Oude gracht
Gijsbert	Overvecht
Jaap	Overvecht
Carla	Overvecht
Conny	Overvecht
Karin	Overvecht
Mille	Overvecht
Jeroen	Overvecht
Piet	Overvecht
Gijsbert	Leerdam
Jaap	Leerdam
Carla	Leerdam
Conny	Leerdam
Karin	Leerdam
Mille	Leerdam
Jeroen	Leerdam
Piet	Leerdam
NULL	Geldermalsen
NULL	Culemborg

OUTER JOIN

Outer join is het laatste type join wat er is. En dit is feitelijk een combinatie van Left en Right join. Je ziet nu dus alle combinaties die mogelijk zijn.

```
SELECT Bezorger.Naam
, Vestiging.Naam
FROM Bezorger FULL JOIN Vestiging_Bezorger
ON Bezorger.bezorger_id = Vestiging_Bezorger.bezorger_id
FULL JOIN Vestiging
ON Vestiging_Bezorger.vestiging_id = Vestiging.vestiging_id
```

In de tabel met de resultaat set wordt dit getoond. Er zijn nu twee vestigingen en drie bezorgers die geen relatie hebben met de andere tabel. Hiermee zijn de vier combinaties gedemonstreerd. Let op niet alle database platformen bieden de full outer join opzet. De inner, left en right komen wel in alle database platformen voor.

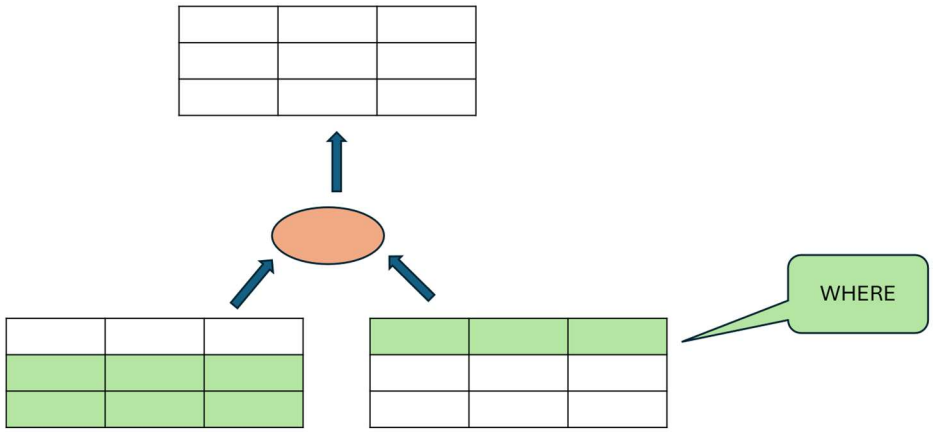
bezorger	vestiging
Gijsbert	Overvecht
Gijsbert	Vreeswijk
Gijsbert	Vianen
Gijsbert	Leerdam
Jaap	Overvecht
Jaap	Vreeswijk
Jaap	Vianen
Jaap	Leerdam
Jeroen	Oude gracht
Jeroen	Overvecht
Jeroen	Vreeswijk
Jeroen	Vianen
Jeroen	Leerdam
Piet	Oude gracht
Piet	Overvecht
Piet	Vreeswijk
Piet	Vianen
Piet	Leerdam
Peter	NULL
Anneke	NULL
Kees	NULL
NULL	Geldermalsen
NULL	Culemborg

Where

In de voorgaande paragrafen lag de focus eerst op de kolommen. Vervolgens keken we naar welke rijen getoond worden in een resultaat set op basis van het koppelen van tabellen op basis van joins. In dit hoofdstuk gaan we filteren in de rijen en dat doen we met het **Where** sub commando.

In onderstaande afbeelding zie je getoond dat door filtering bepaalde rijen in de resultaat set wel terugkomen. Feitelijk een soort van filteren maar wel een bijzonder krachtige filtering. Want je kunt filters met elkaar combineren om zo een filtering toepast die uiteindelijk die rijen toont die voor het gewenste inzicht van de gebruiker nodig zijn. Je wilt daarmee voorkomen dat gebruikers van het select commando zelf moeten uitzoeken welke rijen relevant zijn en welke niet. Als select ontwikkelaar ligt die verantwoordelijkheid natuurlijk bij jou.

WHERE subcommand



Het **where** sub commando bestaat uit een statement dat in de kleinste vorm bestaat uit drie delen. Je geeft een kolomnaam of een combinatie van kolomnamen op, vervolgens een operator bijvoorbeeld = of > en vervolgens een waarde. Wat doet een database met deze combinatie: Hij gaat per rij controleren of de data in die rij voldoet aan het statement. Dus per rij wordt bepaald of het statement waar of niet waar is. Als het statement waar is voor deze rij dan wordt deze rij getoond in de resultaat set en anders niet.

Dat is dus een eenvoudige basis maar kunnen we gaan combineren met andere statements die ook waar of niet waar als resultaat geven. Daarmee ontstaat een fascinerend en uitdagend werkveld voor de data specialist. Want hoe ga ik vragen uit de organisatie zo efficiënt mogelijk omzetten naar één of een combinatie van statements.

```
SELECT verkoopdatum
, neerslag
, temperatuur
FROM VerkoopDashBoard
WHERE maand = 'January'
```

Hieronder zie je een voorbeeld van een commando met in het **where** sub commando een statement op basis van de maand January. Nu kun je per rij in de tabel gaan bepalen of het statement waar of onwaar als resultaat geeft. Let op dat bij alfanumerieke kolommen de waarde tussen `'--'` dient te staan.

```
SELECT neerslag
, productcategorie
FROM VerkoopDashboard
WHERE neerslag IS NULL
OR productcategorie LIKE '%IJs%'
```

In het bovenstaande commando zie je dat je meerdere statements gaat combineren. Daar gebruiken we drie soorten voor **Or, And** en **Not**. Die kunnen we eindeloos gaan combineren. In het bovenstaande voorbeeld is het een combinatie maken van twee statements maar het kunnen er ook heel veel zijn. Dit afhankelijk van de complexiteit van de vraag die je moet beantwoorden.

AND

Met AND moet een commando aan beide statements voldoen. Je noemt dit vaak dat de resultaat set verkleind wordt. Want er moet aan beiden voldaan worden. Hieronder zie je het statement uitgewerkt in een tabelvorm van mogelijke combinaties

AND		Statement 1	
Statement 2		Waar	Niet waar
	Waar	X	
	Niet waar		

```
SELECT VerkoopAantal
, productcategorie
FROM VerkoopDashboard
WHERE productcategorie IN('IJstaart', 'Softijs')
AND verkoopaantal BETWEEN 3 AND 7
```

In bovenstaande statement zie je naast dat je gebruik kunt maken van rekenkundige operatoren zoals = > >= etc. ook de beschikking hebt om het statement met functies te combineren zoals IN() waarbij je een lijstje van waarden op kunt geven of met BETWEEN voor een bereik tussen hoog en laag. Daarmee kun je veelal je statement vereenvoudigen.

OR

Bij een OR statement kun je ook twee of meerdere statements combineren. Hiermee kun je de resultaat set meestal uitbreiden. Want als één van de statements waar is dan wordt de rij opgenomen in de resultaat set.

het statement uitgewerkt in een tabelvorm van mogelijke combinaties

OR		Statement 1	
Statement 2		Waar	Niet waar
	Waar	X	X
	Niet waar	X	

Het commando met eeen voorbeeld zie je hieronder uitgewerkt

```
SELECT VerkoopAantal
, productcategorie
FROM VerkoopDashboard
WHERE productcategorie IN('IJstaart',
'Softijs’)
OR verkoopaantal BETWEEN 3 AND 7
```

NOT

Met **Not** kun je feitelijk Waar en niet waar omdraaien in de evaluatie van de combinatie van statements. Dat kun je vaak ook met de operatoren doen zoals != als niet is of <> als niet =.

het statement uitgewerkt in een tabelvorm van mogelijke combinaties

AND NOT		Statement 1	
Statement 2		Waar	Niet waar
	Waar		
	Niet waar	X	

Vervolgens kun je dit gaan combineren met meer dan twee combinaties maar let dan op want de resultaten kunnen dan verrassingen opleveren. Je kunt dit voorkomen door met haakjes te gaan werken zoals je dat geleerd hebt op de middelbare school. Vaak ga je dit dan uitwerken in een combinatie van letters om een en ander overzichtelijk te houden

Stel je wilt het statement hieronder vereenvoudigen in een weergave dan kan dat met A AND NOT(B OR C). Als je dit verandert naar (A AND NOT B) OR C zal dan een heel ander resultaat geven. Probeer maar eens uit op https://cursus.data-docent.nl/InteractieveQuery.aspx

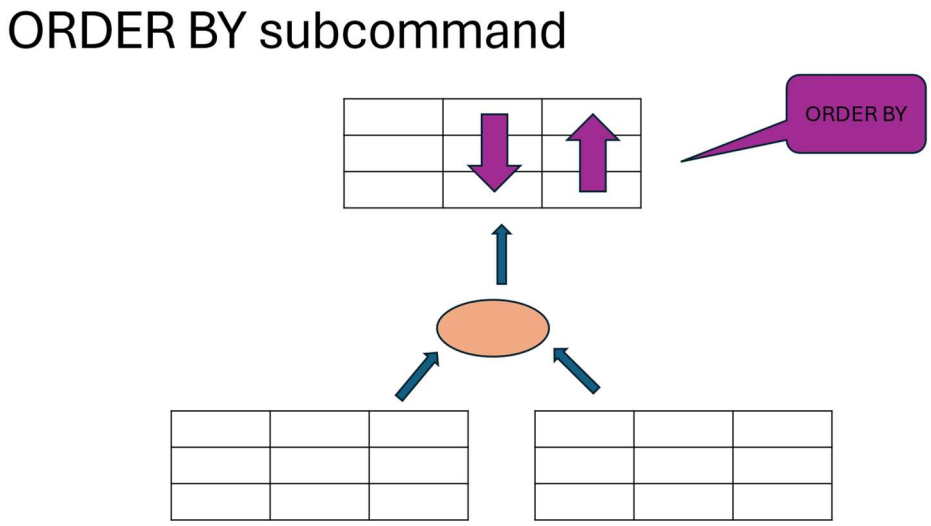
```
SELECT VerkoopAantal
, productcategorie
FROM VerkoopDashboard
WHERE productcategorie
IN('IJstaart', 'Koffie')
AND NOT(verkoopaantal BETWEEN 3 AND 7
  OR verkoopaantal > 12)
```

```
SELECT VerkoopAantal
, productcategorie
FROM VerkoopDashboard
WHERE productcategorie
IN('IJstaart', 'Koffie')
AND (verkoopaantal BETWEEN 3 AND 7
  OR NOT verkoopaantal > 12)
```

Hiermee hebben we een introductie behandeld van het filteren in de tabel of tabellen met het where statement. Let op in het voorgaande hoofdstuk zijn we ingegaan op het leggen van joins. Dit kun je zonder problemen combineren van JOINS met WHERE. Zoals al eerder gezegd met statements kun je eindeloos combineren. Er is een wiskundige tak voor aanwezig die de Booleaanse algebra genoemd wordt. Naar een brits logicus George Boole. Zie wikipedia voor meer informatie: https://nl.wikipedia.org/wiki/Booleaanse_algebra

Order by

Order by geeft je de mogelijkheid om de volgorde van de rijen in je resultaat set te bepalen. Het is daarmee een eenvoudig sub commando.




```
SELECT verkoopdatum
, neerslag
, temperatuur
FROM VerkoopDashBoard
ORDER BY verkoopdatum
```

Hierboven een voorbeeld van order voor een sortering op oplopende volgorde van de rijen in de resultaat set. Ook hier kun je combinaties maken. In dit gescheiden door een komma. Daarnaast kun je een code opgeven of de sortering oplopend of aflopend moet zijn.

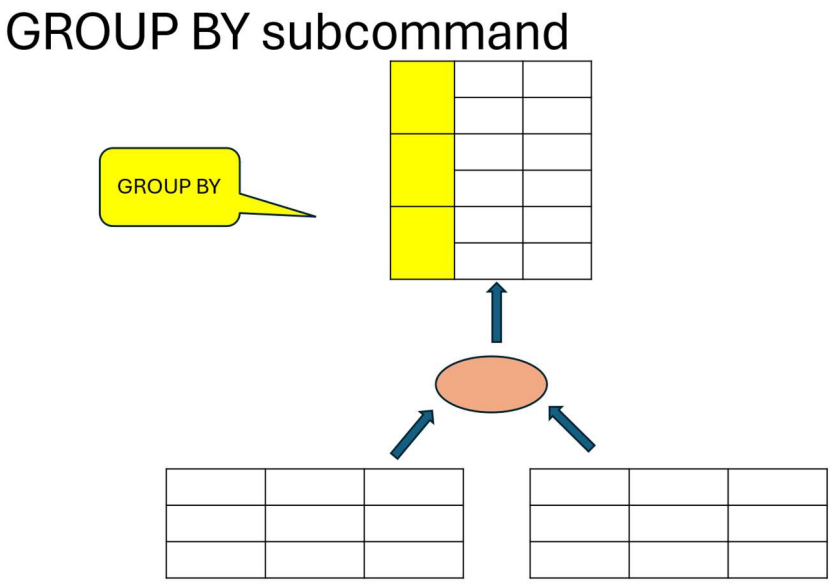
```
SELECT verkoopdatum
, neerslag
, temperatuur
FROM VerkoopDashBoard
ORDER BY verkoopdatum ASC
, temperatuur DESC
```

Group by

Group by is een wat bijzonder deel van het select statement. Want je gaat niet filteren maar de data onderverdelen in groepen. Dat doe je meestal als je wilt gaan rekenen in de data. Veel vragen afkomstig uit de organisatie gaan over het doen van berekeningen met de data. Met name voor het krijgen van inzicht zijn kenniswerkers geïnteresseerd om samenvattingen te krijgen van de data. De details bepalen wel het resultaat van de samenvatting maar deze samenvatting wordt uiteindelijk gebruikt om inzicht te krijgen en beslissingen te nemen. Iemand wil bijvoorbeeld graag weten hoeveel ijsjes er verkocht zijn in een bepaald jaar maar dan wel gegroepeerd naar vestiging en maand.

Hiervoor wordt gebruik gemaakt van set functies. Dit zijn functies die berekeningen kunnen doen op, met name numerieke, data in kolommen. Denk hierbij aan tellen, optellen, gemiddelde etc. Ga je dit combineren met **group by** dan kun je krachtige resultaatsets maken voor de kenniswerker.

Hieronder het effect van **group by** vervolgens een paar voorbeelden van de combinatie van berekeningen met **group by**.




```
SELECT Sum(verkoopbedrag) as Omzet
FROM VerkoopDashboard
```

Bovenstaand commando is een voorbeeld van een set functie. Het berekent de omzet door het verkoopbedrag te sommeren. Met **group by** zie je dat er een uitsplitsing plaatsvindt. Je kunt het statement altijd proberen op de cursus website. Voor de volledigheid geef ik ook een voorbeeld van het resultaat

```
SELECT Sum(verkoopbedrag) as Omzet
, maand
, neerslag
FROM VerkoopDashboard
GROUP BY maand
, neerslag
```

Omzet	maand	neerslag
26.47	February	NULL
1407.19	February	0
581.85	January	0
1059.66	February	1
1347.02	January	1
469.03	February	2
793.05	January	2
532.72	February	3
506.32	January	3
437.31	February	4
267.77	January	4
382.94	February	5
522.66	January	5

Hiermee zijn we klaar met een eerste verkenning van het select statement. Ik hoop dat je onder de indruk bent gemaakt van de mogelijkheden die het select statement biedt om vragen uit de organisatie om te zetten naar krachtige antwoorden op basis van data. Data die veelal gewoon aanwezig is binnen de organisatie. Daarmee is het select een enabler van data gedreven werken voor organisaties.

Meer informatie

Over het select commando is op veel internet sites informatie te vinden waarin je extra mogelijkheden kunt vinden. Vanuit mijn kant wil ik jullie graag attenderen op drie interessante bronnen met extra informatie:

- https://data-docent.nl/FrmDetail.aspx?module=webcourse&webpage=MCEBI#gsc.tab=0
- https://cursus.data-docent.nl/HomeEBI.aspx
- https://www.youtube.com/watch?v=BHrQBeKagPY&list=PLB41kCwfvJUfL9Inz4TznjbWOB_CJHzU4&index=21&pp=gAQBiAQB

Op de cursuswebsite kun je zelf werken met het maken van queries op een voorbeeld database. De voorbeelden genoemd in dit whitepaper zijn gebaseerd op deze cursus website data.

Over de auteur

Bert Dingemans is trainer op het vlak van data architectuur, data management en Big Data. Hij heeft een passie voor modelleren, modelleertools en het effectief inzetten van geautomatiseerde hulpmiddelen om modellen effectief in te zetten in de praktijk. Meer whitepapers zijn te vinden op https://data-docent.nl. Bert is per mail te bereiken via bert@data-docent.nl