Data Retrieval met SQL Select

Hoe je met het Select-statement van SQL informatie op maat uit een database kunt ophalen.

Jeroen Vuurens
Druk I: 19 juli 2002
Druk II: 30 juni 2003
Druk III: 23 juni 2004

Inhoudsopgave

1 INLEIDING

1.1 Gebruik databases

Informatica houdt zich bezig met het ondersteunen of uitvoeren van taken mbv ICT technologie. Taken worden op elkaar afgestemd door communicatie. Een inkoper kan zijn taak beter uitvoeren als hij weet wat er verkocht is, welke producten er in de aanbieding komen, welke producten met elkaar concurreren en welke doelen het management stelt tav de levertijd en voorraadgrootte. Als de afdeling inkoop de orders van de afdeling verkoop doorkrijgt, maar een order wordt geannuleerd dan moet die annulering worden doorgegeven aan de afdeling inkoop, anders kopen ze in voor niks. Voorbeelden als deze geven aan dat een goede samenwerking valt en staat met goede afspraken, en dat dat meer communicatie (annuleringen ook doorgeven) en meer risico inhoudt (er kan meer fout gaan).

Stabiele factor

Gegevens worden gezien als de meest stabiele factor in een organisatie. Wat we met automatisering doen is dan ook niet zo hoogstaand; we kiezen gewoon een ander medium en andere hulpmiddelen om te kunnen communiceren. Met automatisering krijgen we echter ook nieuwe risico's (digitale gegevens gaan gemakkelijker verloren, mensen kunnen opeens bij gegevens waar ze niet bij mogen) en nieuwe kansen (bv. informatie precies op tijd leveren).

Voordelen databases

Door communicatie via databases te laten verlopen kan men veel automatiseringsproblemen de baas. De focus verschuift van "Hoe kom ik aan de informatie?" naar "Wat wil ik weten?". Op bijna alle vragen over een bepaalde set van gegeven antwoord kan worden gegeven, mits die in een goed ontworpen database staan. Een goed DBMS biedt mogelijkheden om :

- gegevens te bewaren op een centrale plaats
- redundancy te beperken (wat ook een probleem kan zijn in een papieren organisatie)
- kwaliteit van gegevens tot op zekere hoogte te garanderen
- beheer makkelijker te maken (back-ups)
- rechten van gebruikers beperken
- gegevens te combineren op overzichten

N.B. In dit stuk gaan we alleen over het opvragen van gegevens verder.

1.2 Opvragen gegevens

Betere kwaliteit informatie

Als alle gegevens digitaal bij elkaar staan krijgen we daardoor nieuwe mogelijkheden die de papieren organisatie nog niet kende; we kunnen heel snel op een nieuwe manier de gegevens met elkaar combineren. Op die manier kan men bij de uitvoer van een taak beschikken over informatie op maat en het management kan goed overzicht houden en de organisatie op een goede manier aansturen.

Herontwerp informatiebehoefte

Ook vandaag de dag zijn mensen nog niet gewend aan de mogelijkheden van digitale technieken. Men houdt vast aan het bestaande beeld van een papieren organisatie, alleen voer je de orders op een scherm in en komen ze dan uit de printer rollen. Als je dan mensen gaat vragen waar ze echt behoefte aan hebben, komen ze vaak niet verder dan de formulieren die in de oude situatie ook al bestonden. Voor de leek lijkt het samenstellen van andere overzichten een tijdrovende klus, weten zij veel dat de computer dat in een handomdraai kan maken. Een informatieanalist moet daarom kritisch kijken naar de taken, en voorstellen doen met welke overzichten makkelijker hun werk kunnen uitvoeren. Vaak is een herontwerp van overzichten en formulieren nodig, maar kom je ook in de nieuwe situatie zul je zien dat men vaak een beperkt aantal overzichten nodig heeft.

1.3 SQL als taal

Informatiebehoefte

Om een DBMS opdracht te geven antwoord te geven op een bepaalde informatiebehoefte heb je een taal nodig om uit te leggen wat je wilt. Voor relationele databases is SQL de meest gangbare taal. We kunnen met een SQL statement in weinig woorden en veel detail aangeven wat we precies willen zien. De taal is uitermate logisch en efficiënt, wat de leesbaarheid niet echt ten goede komt. Maar als je eenmaal goeie SQL queries hebt opgesteld om antwoord te geven op de informatiebehoefte die er bestaat, dan hoeven ze gelukkig zelden meer te worden aangepast (omdat gegevens en informatiebehoeften over het algemeen stabiel zijn).

Vaste volgorde uitwerken query

Het schrijven van goede SQL queries blijkt een hele kunst te zijn. Alhoewel de taal in potentie krachtig genoeg is om met een query de meeste moeilijke vraag te beantwoorden, is het opstellen van zo'n query lastig. De insteek van dit stuk is om dat te leren door volledig inzicht te krijgen in de manier waarop een DBMS queries uitwerkt. Door te begrijpen in welke volgorde een DBMS de stappen neemt leer je zien wat de weg naar succes is.

2 DE METHODE

2.1 Inleiding

Gezond verstand kan tot denkfouten leiden

Veel mensen zijn geneigd om met gezond verstand de uitkomst van een query te bepalen. Ze maken daarbij denkfouten. De computer verwerkt een query niet met gezond verstand, maar leest gewoon exact wat daar staat en voert het uit. We gaan zowel bij het lezen als schrijven van queries steeds uit van de vaste volgorde waarin de stappen door de computer worden uitgevoerd. Door steeds de vaste stappenreeks te volgen kunnen we exact bepalen wat de uitkomst van een query is. Wees niet bang als er termen in voorkomen die nu nog niet duidelijk zijn, alles wordt nog uitgelegd.

2.2 De Stappenreeks

1. **subselect**
2. **from**
3. **where (+gecorreleerde subselect)**
4. **group by**
5. **kolomfuncties**
6. **having**
7. **select**
8. **distinct**
9. **union**
10. **order by**

2.3 Hoe te gebruiken

Op papier maken

Het probleem van SQL in de praktijk is dat een query bijna altijd een antwoord teruggeeft. En als de query niet past past bij de informatievraag zie je soms niet dat het resultaat niet klopt omdat het afkomstig is uit een web van vele rijen en tabellen. Het is daarom **absolute noodzaak** dat je feilloos in staat bent de correctheid van queries vast te stellen. Want SQL is krachtig maar juist daarom kan er ook veel misgaan. Oefen daarom op papier en stel vragen over de dingen die onzeker zijn. Gebruik geen trial-and-error leermethode achter een PC en probeer niet teveel naar goed antwoorden te kijken.

Instinkers

Ik heb mijn verhaal dat ik gebruik om SQL programmeren uit te leggen op papier gezet zodat jullie achteraf stukken na kunnen lezen. SQL leren programmeren is pittig, vooral omdat bij het kijken naar goede antwoorden het er relatief makkelijk uitziet maar als je zelf opgaven moet maken valt dat vies tegen. Wat betreft het lezen van queries vragen we op een toets nagenoeg nooit queries die je met gevoel kunt bepalen, het zijn vaak instinkers. Het gaat erom dat je exact begrijpt wat elk stukje van een query doet. Pas als je dat begrijpt kun je in de praktijk vaststellen waar het bij een query aan schort, en ben je in staat foutloze informatie uit een database te halen. Voorbeelden van opgaven vind je dan ook in de tekst.

3 QUERIES OP 1 TABEL LEZEN

3.1 Inleiding

stappenreeks

Om queries te leren lezen gaan we de exacte betekenis van elke begrip uit de stappenreeks uitleggen. Na het uitleggen van elk begrip gaan we steeds de uitkomst van queries bepalen door in volgorde de stappen na te lopen. Let erop dat nauwkeurigheid voorop staat, grote gedachtesprongen leiden tot fouten!

REGEL: De stappenreeks wordt altijd in de vaste volgorde doorlopen

We beginnen eenvoudig met *From*, *Select* en *Where*. We gaan dan gaandeweg steeds meer termen uit de stappenreeks erbij nemen.

REGEL: Termen uit de stappenreeks die niet in de query voorkomen mag je gewoon overslaan.

3.2 De basis

3.2.1 From

Welke tabellen

In stap 2 het *From*-gedeelte geef je aan uit welke tabellen je informatie nodig hebt. Het eenvoudigst is een query op 1 tabel. In een query geeft **From medewerker** aan dat je de **hele table medewerker** gaat gebruiken. **Alle** rijen en kolommen die daar dus inzitten.

3.2.2 Select

Welke kolommen

Uiteindelijk moet het antwoord op je vraag op het scherm worden getoond. Je bent wellicht niet geïnteresseerd in alle kolommen uit een tabel. Bij sommige bedrijven kan de tabel *medewerker* misschien wel 50 attributen bevatten, en daar wil je dus een selectie uit maken. Bij de eenvoudigste vorm geef je achter *Select* de kolommen op die je wilt zien. **Select naam, woonplaats** toont dan uiteindelijk alleen de naam en woonplaats. We kunnen onze eerste query maken :

Query 1

Toon de naam en woonplaats van alle medewerkers

```
Select naam, woonplaats
From medewerker
```

naam	woonplaats
Piet	Leiden
Elske	Leiden
Jan	Lisse
Peter	Berkel
Chris	Leiden
Kees	

*Select-list **

We noemen hetgeen achter het woordje *Select* staat ook wel de *Select-list*. Als je wel alle kolommen uit de tabel wilt zien dan kun je als *Select-list* een * neerzetten, wat staat voor de hele rij (alle kolommen). **Select * From medeweker** toont dus de hele tabel medewerker die staat in *bijlage A*.

Expressies

In feite zijn de kolommen die je noemt achter *Select* allemaal expressies, gescheiden door komma's. Elke expressie stelt een kolom voor. Voor elke rij die moet worden getoond wordt de waarde van de expressie uitgerekend. Dus **From medewerker** betekent dat alle rijen uit medewerker worden gebruikt. Van de eerste rij in die tabel worden dan de expressies uit de *Select-list* **naam** en **woonplaats** bepaald. Vervolgens wordt dat voor de tweede rij gedaan, de derde rij etc. Als expressie kun je ook wat ingewikkelders neerzetten, bijvoorbeeld een tabel waar orderregels in staan kan een kolom *aantal* en een kolom *artikelprijs* hebben. **aantal * artikelprijs** is dan een geldige expressie, voor elke rij uit de tabel wordt dan de uitkomst berekend en in de desbetreffende kolom gezet.

AS

In de *Select-list* kun je aan expressies headers (kolomkopjes) meegeven mbv het woord AS. **Select aantal * artikelprijs AS totaalprijs From ….** Als je de query aan het DBMS geeft zal hij totaalprijs als naam van de kolomkop gebruiken. In praktijk gebruiken eindgebruikers zelden rechtstreeks SQL, maar staan SQL statements embedded in een andere programmeertaal. De naam van het kopje is dan vaak de naam van een lokale variabele waar de waarde in wordt gestopt.

Opgave 1

Select naam, baas – nr From medewerker

Opgave 2

Select naam, 'Leiden' From medewerker

3.2.3 Where

Schrappen rijen

Achter *Where* kunnen we een bewering plaatsen. Elke rij die we selecteren uit tabellen mbv *From* wordt aan die bewering getoetst. Is de uitkomst *TRUE* dan blijft de rij geselecteerd, is de uitkomst *FALSE* dan wordt die rij uit het tussenresultaat geschrapt.

Query 2

Toon alle medewerkers uit Leiden

```
Select *
From medewerker
Where woonplaats = 'Leiden'
```

nr	naam	woonplaats	baas	functie
11	Piet	Leiden	15	PR
12	Elske	Leiden	15	PR
15	Chris	Leiden	14	PL

In stapjes:

2. From medewerker : alle rijen in de tabel medewerker doen mee
3. Where woonplaats = ‘Leiden’ : jan, peter en kees worden uitgefilterd

nr	naam	woonplaats	baas	functie	woonplaats = ‘Leiden’
11	Piet	Leiden	15	PR	TRUE
12	Elske	Leiden	15	PR	TRUE
~~13~~	~~Jan~~	~~Lisse~~	~~15~~	~~SA~~	
~~14~~	~~Peter~~	~~Berkel~~		~~PL~~	
15	Chris	Leiden	14	PL	TRUE
~~16~~	~~Kees~~		~~15~~		

7. Select * : laat alle kolommen zien

Filter

Beweringen in het *Where*-gedeelte bevat doorgaans kolommen uit de tabel(len) van het *From* gedeelte. Op basis van de waarden kan besloten worden of we wel of niet geïnteresseerd zijn in die rij. Elke rij uit de tabel medewerkers wordt getoetst. De bewering **woonplaats = ‘Leiden’** wordt voor Piet dan **‘Leiden’ = ‘Leiden’**. De uitkomst daarvan is **TRUE** en dus zien we Piet in het resultaat. De bewering woonplaats = ‘Leiden’ wordt bij Peter dan **‘Berkel’ = ‘Leiden’**. De uitkomst is **FALSE** dus wordt Peter geschrapt uit het resultaat. We kunnen zeggen dat het *Where*-gedeelte van de query een filter is dat bepaalt in welke rijen we wel en niet geïnteresseerd zijn. De voorwaarde die in de informatievraag *Toon alle medewerkers uit Leiden* staat, wordt daartoe vertaald naar woonplaats = ‘Leiden’.

Regelgewijs

SQL kan op twee manieren werken: **regelgewijs** en **groepsgewijs**. Het groepsgewijs werken komt straks terug als we gaan kijken naar kolomfuncties, alle andere queries werken regelgewijs. Attributen die op een regel naast elkaar staan zijn onlosmakelijk met elkaar verbonden. De attributen *11, Piet, Leiden, 15, PR* zitten als het ware aan elkaar vast. We kunnen in het *Where* gedeelte alleen hele rijen schrappen.

Samengestelde beweringen AND/OR

Het *Where* gedeelte kan ook een ingewikkeldere bewering bevatten. We mogen beweringen samenstellen met AND en OR :

Query 3

Toon alle medewerkers uit Leiden die functie PE hebben

```
Select *
From medewerker
Where woonplaats = ‘Leiden’
  And functie = ‘PR’
```

nr	naam	woonplaats	baas	functie
11	Piet	Leiden	15	PR
12	Elske	Leiden	15	PR

Query 4 **Toon alle medewerkers uit Leiden en Lisse**

```
Select naam
From medewerker
Where woonplaats = ‘Leiden’
    OR woonplaats = ‘Lisse’
```

naam
Piet
Elske
Jan
Chris

OR In het Nederlands zeg je nogal makkelijk uit Leiden **en** Lisse, maar er wordt duidelijk **of** bedoeld. Let op! Als je AND en OR met elkaar combineert moet je () haakjes zetten om het deel dat bij elkaar hoort, AND en OR kennen in SQL geen vaste prioriteitsregels.

Opgave 3 Select * From medewerker Where (woonplaats = ‘Leiden’ OR woonplaats = ‘Lisse’) AND baas = 15;

Opgave 4 Select * From medewerker Where woonplaats = ‘Leiden’ OR (woonplaats = ‘Lisse’ AND baas = 15);

Opgave 5 Select baas From medewerker Where woonplaats = ‘Leiden’;

Opgave 6 Select * From medewerker Where nr = baas;

3.3 Eenvoudige uitbreidingen

3.3.1 Order by

Niet gesorteerd Per definitie zijn gegevens in databases **niet** gesorteerd. Bij een hoop queries en DBMS-en zul je de resultaten zien in de volgorde waarin ze zijn ingevoerd, maar je hebt daar geen enkele garantie voor. Als je de gegevens in een bepaalde volgorde wilt zien moet je concreet aangeven op welke expressies wordt gesorteerd met *Order by*. Ook hier zal een expressie vaak uit een enkel veld bestaan, maar je kunt best sorteren op iets als *prijs * aantal* als je dat leuk vindt.

Query 5 **Toon de naam en woonplaats van alle medewerkers gesorteerd op woonplaats**

```
Select naam, woonplaats
From medewerker
Order by woonplaats
```

naam	woonplaats
Peter	Berkel
Elske	Leiden
Piet	Leiden
Chris	Leiden
Jan	Lisse
Kees	

```
Select naam, woonplaats
From medewerker
Order by woonplaats,
         naam
```

naam	woonplaats
Peter	Berkel
Chris	Leiden
Elske	Leiden
Piet	Leiden
Jan	Lisse
Kees	

```
Select naam, woonplaats
From medewerker
Order by naam,
         woonplaats
```

naam	woonplaats
Chris	Leiden
Elske	Leiden
Jan	Lisse
Kees	
Peter	Berkel
Piet	Leiden

2 expressies Merk op dat bij het linker voorbeeld vast staat dat eerst Berkel komt, dan 3x Leiden en dan Lisse. Maar de volgorde van de drie medewerkers uit Leiden is willekeurig. Als je per woonplaats ook nog eens op naam wilt sorteren dan kun je dat aangeven door deze daarachter te zetten zoals bij het middelste voorbeeld. De volgorde van de velden bij de *Group by* maakt uit. Eerst wordt op woonplaats gesorteerd, en als de woonplaats hetzelfde is dan op de naam. Bij het rechter voorbeeld zie je de uitkomst als je dat omdraait.

Je kunt door ASC of DESC achter Order by aangeven of er oplopend of aflopend moet worden gesorteerd.

Opgave 7 Select * From medewerker Order by woonplaats ASC, naam DESC;

3.3.2 Distinct

Dubbele verwijderen In sommige gevallen ontstaan er op natuurlijke wijze dubbele gegevens in het resultaat. Stel dat je wilt weten in welke woonplaatsen medewerkers wonen. Dan wil je niet 100x Leiden, dan 200x Rotterdam etc. zien. De dubbele gegevens komen wel vaker in tabellen voor, we kunnen achter *Select* het woord **Distinct** gebruiken om dubbele rijen eruit te halen. De *Distinct* wordt uitgevoerd in stap 8 en werkt rij-gewijs. Als in de uitvoertabel twee volledig identieke rijen staan dan worden de dubbelen eruit gehaald.

Query 6 **Toon van elke medewerker de woonplaats**

```
Select woonplaats
From medewerker
```

woonplaats
Leiden
Leiden
Lisse
Berkel
Leiden

Toon alle woonplaatsen waar medewerkers wonen

```
Select Distinct woonplaats
From medewerker
```

woonplaats
Leiden
Lisse
Berkel

Merk op dat sorteren niet nodig is om met *Distinct* dubbelen te verwijderen, maar *Distinct* het resultaat ook niet sorteert.

3.3.3 Operatoren

Stringfuncties We kunnen in SQL gebruik maken van vele operatoren. Naast het bekende rekentuig als +, - enz. en de booleaanse AND en OR bestaan er ook operatoren om Strings te bewerken. De expressie String + String plakt bijvoorbeeld twee Strings achter elkaar. De meeste DBMS-en ondersteunen ook een soort substr() functie waarmee je een deel van een String kunt pakken, maar helaas kan die functie bij 2 verschillende DBMS-en een andere naam hebben.

LIKE Een van de meest gebruikte stringoperatoren in oefenopgaven over SQL is de LIKE operator. De LIKE vergelijkt een String met een patroon waarin het %-teken staat voor 0 of meer willekeurige tekens. ‘a%’ komt overeen met elke String die begint met een a. ‘%a’ komt overeen met elke String die eindigt op een a. ‘%a%’ met elke String waarin tenminste 1 a voorkomt. In onze voorbeelddatabase zou ‘%e%e%’ overeenkomen met Leiden en Berkel maar niet met Lisse. De LIKE operator geeft *TRUE* terug als de String overeenkomt met het patroon. Let er wel op dat Strings in SQL case-sensitive zijn.

Query 7 **Geef alle functienamen die met een P beginnen**

```
Select naam
From functie
Where naam like ‘P%’
```

naam
Projectleider
Programmeur

Opgave 8 Select naam From functie Where code like ‘S%’;

NULL In een database kunnen velden ook niet ingevuld zijn, dat noemen we een **NULL**-waarde. Merk op dat een NULL-waarde niet hetzelfde is als het getal 0 of een lege String. Bij *Foreign Keys* wordt meestal een NULL-waarde vertaald als dat iets er niet is (zoals medewerker Peter geen baas heeft), bij andere attributen is een meer correcte vertaling van een NULL-waarde dat de waarde onbekend is. Als het telefoonnummer van een klant niet is ingevuld betekent dat immers niet dat hij er geen heeft.

IS (NOT) Vergelijkingen met een NULL moeten altijd gebeuren met de **IS** en de **IS NOT** operator. Vergelijkingen van NULL met = en andere operatoren zijn gedoemd te mislukken omdat het antwoord hoe dan ook **altijd** *FALSE* zal zijn.

Query 8 **Geef alle medewerkers die geen baas hebben**

```
Select *
From medewerker
Where baas IS NULL
```

nr	naam	woonplaats	baas	functie
14	Peter	Berkel		PL

Opgave 9 Select woonplaats From medewerker Where baas IS NOT NULL

3.4 Kolomfuncties

3.4.1 Over de hele tabel

Afgeleide gegevens Soms ben je niet in de gegevens zelf geïnteresseerd maar in afgeleiden daarvan. Als we bijvoorbeeld het aantal medewerkers willen weten, dan zijn de gegevens van die medewerkers niet relevant. We willen als het ware het aantal rijen in de tabel medewerker tellen. SQL werkt normaal gesproken altijd rij-gewijs, maar we kunnen wel statistieken over hele tabellen en kolommen uitrekenen. We hebben daarvoor de beschikking over kolomfuncties :

- Count(bewering) : aantal rijen in een tabel tellen (NULL telt niet mee)
- Sum(bewering): de uitkomst van de bewering wordt voor alle rijen opgeteld.
- Avg(bewering): het gemiddelde van de uitkomsten van de bewering bepalen.
- Min(bewering): de laagste waarde in de uitkomsten van de bewering bepalen.
- Max(bewering) : de hoogste waarde

uitkomst 1 rij Als je een kolomfunctie gebruikt dan wordt de uitkomst van de tabel gereduceerd tot 1 rij. Over de hele tabel worden de uitkomsten van de kolomfuncties bepaald. De bewering tussen de haakjes mag van alles zijn. Laten we naar een paar voorbeeldjes gaan kijken :

Query 9 **Wat is het totale inzetpercentage van alle medewerkers**

```
Select Sum(inzet_perc) as totale_inzet
From werkt
```

totale_inzet
390

In stapjes :

2. From werkt : alle rijen in de tabel werkt doen mee
5. kolomfunctie Sum(inzet_perc) : van alle rijen de waarde van inzet_perc sommeren dus

medewerker	project	inzet perc
11	C	20
11	B	70
14	A	100
13	A	50
13	C	50
15	C	100

Is inderdaad 390

7. Select Sum(inzet_perc) : alleen dit antwoord tonen

Query 10 **Wat is het totale inzetpercentage van de medewerker met nr 11**

```
Select Sum(inzet_perc)
From werkt
Where nr = 11
```

Sum(inzet_perc)
90

In stapjes :

2. From werkt : alle rijen in de tabel werkt doen mee
3. Where medewerker = 11 : alleen de eerste twee rijen blijven over
5. kolomfunctie Sum(inzet_perc) : van alle rijen de waarde van inzet_perc sommeren dus

medewerker	project	inzet perc
11	C	20
11	B	70

Is 90

7. Select Sum(inzet_perc) : alleen dit antwoord

Count() *Count()* telt simpelweg het aantal rijen. Er wordt geen rekening gehouden met dubbele rijen, of wat voor waarden er in de rij staan, het aantal rijen wordt gewoon klakkeloos geteld. We kunnen als bewering een * neerzetten om aan te geven dat het totaal aantal rijen moeten worden geteld. We kunnen een andere bewering neerzetten als we willen dat bepaalde rijen niet mee worden geteld. We moeten er dan voor zorgen dat voor die rijen die niet mee moeten worden geteld de bewering tussen de haakjes een NULL-waarde oplevert :

Query 11 **Het aantal medewerkers**

```
Select Count(*)
From medewerker
```

Count(*)
6

met een nr ingevuld

```
Select Count(nr)
From medewerker
```

Count(nr)
6

met een baas

```
Select Count(baas)
From medewerker
```

Count(baas)
5

NULL-waarde Je ziet bij Count(baas) dat er 5 uitkomt omdat Peter geen baas heeft.

Query 12

Het hoogste nr

```
Select Max(nr)
From medewerker
```

Max(nr)
16

het laagste nr

```
Select Min(nr)
From medewerker
```

Min(nr)
11

het gemiddelde nr

```
Select Avg(nr)
From medewerker
```

Avg(nr)
13.5

Functies als Min(), Max() en Avg() beschouwen overigens alleen de waarden die zijn ingevuld. Als op een rij NULL staat ingevuld telt die dus niet mee voor het gemiddelde.

Opgave 10 Select Count(inzet_perc) From werkt Where medewerker > 11

Opgave 11 Select Count(baas) From medewerker Where nr > 11

Opgave 12 Select Count('Leiden') From medewerker

3.4.2 Distinct in kolomfunctie

Distinct haalt dubbele rijen eruit. Je kunt de dubbele rijen eruit laten halen voordat kolomfuncties worden toegepast of erna.

Query 13

Wat is het aantal medewerkers werkende medewerkers

```
Select Distinct Count(medewerker)
From werkt
```

Count(medewerker)
6

```
Select Count(Distinct medewerker)
From werkt
```

Count(medewerker)
4

Nutteloze Distinct De linker query geeft als antwoord 6. Het *From* gedeelte levert immers 6 rijen op, dan wordt de kolomfunctie behandeld en *Count()* levert 6 op (6 rijen waarvoor medewerker geen NULL-waarde heeft, dubbelen worden gewoon meegeteld). Dan hebben we een tussentabel met 1 rij uitvoer (6) en daar wordt dan *Distinct* over uitgevoerd. In de linker query is *Distinct* dus totaal niet nuttig, de query levert maar 1 rij op en zal dus geen dubbele rijen op kunnen leveren.

Van binnen naar buiten De rechter query is beter. De functies worden altijd van binnen naar buiten uitgewerkt. Bij de som 2 x (1+3) reken je ook eerst dat uit dat tussen de haakjes staat. Dat doet het DBMS ook. Dus eerst worden de dubbele medewerkernummers eruit gehaald, en dan worden het aantal rijen pas geteld.

Opgave 13 Select Count(Distinct woonplaats) From medewerker;

3.4.3 Group by

per groep

We hebben gezien dat we met kolomfuncties over alle rijen van een tabel gegevens kunnen afleiden. Er blijft dan slechts één rij met uitkomst(en) over. Een uitbreiding daarvan is het werken met groepen. De tabel wordt denkbeeldig in groepen opgedeeld en per groep wordt de uitkomst van de kolomfuncties bepaald. Elke groep levert dan een rij op in de uitkomsttabel.

Zelfde uitkomst= zelfde groep

We maken daarvoor gebruik van *Group by* gevolgd door een of meer expressies. Als alle *Group by* expressies van twee rijen dezelfde uitkomst geven dan horen die rijen tot dezelfde groep :

Query 14

Wat is het aantal medewerkers per woonplaats

```
Select woonplaats, Count(*)
From medewerker
Group by woonplaats
```

woonplaats	Count(*)
Leiden	3
Lisse	1
Berkel	1
	1

In stapjes :

2. From medewerker : alle 5 rijen in de tabel medewerkers doen mee

4. Group by woonplaats : de tabel wordt in 4 denkbeeldige groepen verdeeld

nr	naam	woonplaats	baas	functie
11	Piet	Leiden	15	PR
12	Elske	Leiden	15	PR
15	Chris	Leiden	14	PL
13	Jan	Lisse	15	SA
14	Peter	Berkel		PL
16	Kees		15	

5. kolomfunctie Count(*) : per groep wordt de uitkomst bepaald

nr	naam	woonplaats	baas	functie	Count(*)
11	Piet	Leiden	15	PR	3
12	Elske	Leiden	15	PR	
15	Chris	Leiden	14	PL	
13	Jan	Lisse	15	SA	1
14	Peter	Berkel		PL	1
16	Kees		15		1

7. Select Count(*) : van elke groep wordt 1 rij getoond. In elke rij staat de uitkomst van Count(*)

NULL

Een groep met een NULL-waarde is dus ook een groep!

Groepering is alleen maar zinnig als je onderscheid in verschillende groepen kunt maken. **Group by nr** is dus niet zinvol omdat elke medewerker een uniek nummer heeft en dus een eigen groep vormt. Je mag wel attributen met elkaar combineren. **Group by woonplaats, functie** zorgt ervoor dat de combinatie *Leiden, PR* een andere groep is dan *Leiden, PL* en levert het volgende resultaat :

Query 15 **Wat is het aantal medewerkers met dezelfde functie per woonplaats**

```
Select woonplaats, functie, Count(*)
From medewerker
Group by woonplaats, functie
```

woonplaats	**functie**	**Count(*)**
Leiden	PR	2
Leiden	PL	1
Lisse	SA	1
Berkel	PL	1
		1

REGEL : Als je kolomfuncties gebruikt ben je verplicht om alle attributen die je niet in kolomfuncties gebruikt maar wel in de Select-list, op te nemen in de Group by!!!

Query 16 **Wat is het aantal medewerkers met dezelfde functie per woonplaats**

```
Select woonplaats, functie, Count(*)
From medewerker
Group by woonplaats
```

ERROR omdat alle velden uit de Select-list in de Group by moeten voorkomen

Opgave 14 Select baas, Count(*) From medewerker Group by baas;

3.4.4 Having

Where vs Having Op exact dezelfde manier als *Where* gedeelte rijen uitfiltert, kun je met **Having** groepen wegfilteren. Het *Where* gedeelte van een query wordt (stap 3) uitgevoerd voordat de tabel in groepen wordt gedeeld (stap4) en voordat de kolomfuncties worden uitgerekend. *Where* kan dus alleen uitfilteren op basis van de gegevens van die ene rij en niet op basis van kolomfuncties. Als je de woonplaatsen wilt zien waar minimaal 2 medewerkers wonen, dan kun je op basis van de gegevens van 1 rij die beslissing niet nemen, er kan nl. wel of geen andere rij zijn met dezelfde woonplaats. Een bewering met kolomfuncties als *Count(*) > 2* is in het *Where* gedeelte dan ook onmogelijk.

kolomfuncties In het *Having*-gedeelte kun je dezelfde beweringen maken als in het *Where* gedeelte maar dan uitgebreid met het gebruik van kolomfuncties. Alle beweringen die niks met kolomfuncties te maken hebben komen normaliter in het *Where* gedeelte te staan, dat is qua performance van de query ook beter.

Query 17 **Geef alle woonplaatsen waar minstens twee medewerker wonen**

```
Select woonplaats
From medewerker
Group by woonplaats
Having Count(*) > 1
```

woonplaats
Leiden

Kolomfunctie niet in Select-list

De *Count(*)* mag in de *Select-list* voorkomen, maar het hoeft niet. Nog steeds geld de eis dat alle velden die je los gebruikt van een kolomfunctie (dus los in de *Select-list* in dit geval, maar ook velden die los in beweringen van *Having* of in de *Order by* voorkomen) in de *Group by* moeten worden opgenomen. Want als je niet groepeert op woonplaats dan zou een groep uit verschillende woonplaatsen kunnen bestaan en dat mag niet omdat elke groep maar één rij mag opleveren in de uitkomsttabel.

Opgave 15

```
Select woonplaats, Count(*)
From medewerker
Group by woonplaats, baas
Having Count(*) > 1;
```

4 QUERIES OP MEER TABELLEN LEZEN

4.1 Inner Join

4.1.1 Gegevens uit twee tabellen combineren

Relaties tussen tabellen

We hebben tot nu toe de basisprincipes geleerd op gegevens uit 1 tabel om het eenvoudig te houden. Dezelfde principes werken namelijk ook op queries uit 2 tabellen. Als je gegevens uit twee of meer tabellen wilt gebruiken dan moet je kiezen op wat voor manier die tabellen met elkaar verbonden moeten worden. In 99% van de gevallen ligt er een relatie tussen twee tabellen in de vorm van een **Foreign Key (FK)** in de ene tabel, die overeenkomt met een **Primary Key (PK)** in de andere tabel. Maar er zijn vaak verschillende combinaties mogelijk. Een persoon kan in een voetbalteam spelen, er trainer van zijn of aanvoerder. Dat zijn drie verschillende relaties die op verschillende manieren met elkaar gelegd zijn. Als je zou willen kijken welke speler het oudste is van zijn team dan moet je geboortedatums met elkaar gaan vergelijken. Omdat er zoveel mogelijkheden zijn kan het DBMS niet weten in welke relatie tussen de gegevens je bent geïnteresseerd. Je moet daarom altijd expliciet aangeven hoe de gegevens met elkaar verbonden zijn.

Inner Join

De meest basale vorm om twee tabellen met elkaar te combineren is de *Inner Join*. Die bestaat **altijd** uit twee handelingen:

- In het *From* gedeelte aangeven welke twee tabellen het betreft
- In het *Where* gedeelte aangeven hoe de gegevens zijn verbonden
  (meestal **Foreign Key = Primary Key of kortweg FK = PK**)

Foreign Key= Primary Key

Stel dat we van elke medewerker de naam en de naam van hun functie willen zien. Hun naam staat in de tabel medewerker, de naam van de functie in de tabel functie. Maar bij een bepaalde medewerker, zeg Jan, hoort maar 1 functie, in dit geval Systeem Analist. Hoe weet het DBMS welke functie bij jan hoort? Omdat de *Foreign Key* **functie (=SA)** overeenkomt met de *Primary Key* **SA** in de tabel functie. Dat verband moeten we beschrijven in het *Where* gedeelte met **FK = PK**. De Foreign Key heet in dit geval functie, en de Primary Key code. Dus **Where functie = code**.

Query 18

Geef van elke medewerker de naam en de naam van hun functie

```
Select medewerker.naam, functie.naam
From medewerker, functie
Where functie = code
```

medewerker.naam	functie.naam
Piet	Programmeur
Elske	Programmeur
Jan	Systeem Analist
Peter	Projectleider
Chris	Projectleider

Correlatienaam Omdat de kolom naam in beide tabellen voorkomt moet je steeds aangeven uit welke tabel je de naam wilt hebben. In plaats van **Select naam** moet je dan **Select medewerker.naam** neerzetten. We noemen **medewerker** in deze context officieel een correlatienaam, maar je mag het ook onthouden als de naam van de tabel.

Aantal rijen Inner Join Je ziet in de uitkomsttabel 5 rijen omdat er 5 medewerkers met een functie zijn. In de tabel medewerker staat een Foreign Key naar functie. Als die Foreign Key een NULL-waarde heeft dan wordt de medewerker niet getoond. Er kan immers geen combinatie met een functie worden gemaakt waarvoor de bewering functie = code *TRUE* is. Omgekeerd komen niet alle functies voor. Er is geen medewerker met de Foreign Key functie = 'SO' dus komt de functie Systeem Ontwerper niet voor in de uitkomsttabel. We noemen het een *Inner Join* omdat niet alle functies en niet alle medewerkers getoond hoeven te worden.

Cartesisch product Als je de verbinding in het *Where* gedeelte zou vergeten, dan kan de database niet bepalen welke medewerker en functie bij elkaar horen. Het resultaat is dan het Cartesisch Product dwz alle mogelijke combinaties van een medewerker met een functie. In dit geval zijn er 6 medewerkers en 4 functies en dus 24 combinaties mogelijk. Zonder *Where* gedeelte krijgen we dus 24 rijen te zien, al kan dat nooit de bedoeling zijn. Een veelgebruikte uitleg van een Join is dat het *Where* gedeelte, uit die 24 rijen met alle mogelijke combinaties degenen waarvan de bewering functie=code niet waar is, wegfiltert.

Regelgewijs SQL werkt per definitie **regelgewijs** (als er geen kolomfuncties worden gebruikt). Attributen die op een regel naast elkaar staan zijn onlosmakelijk met elkaar verbonden. De attributen *11, Piet, Leiden, 15, PR* zitten als het ware aan elkaar vast. Het From gedeelte van een query is de enige plaats waar we rijen uit verschillende tabellen kunnen combineren tot 1 rij. Staan de gewenste gegevens na het From-gedeelte niet naast elkaar kunnen we dat nooit meer voor elkaar krijgen.

Opgave 16 Select M.naam, woonplaats, F.naam From medewerker M, functie F Where nr = 11;

Opgave 17 Select F.naam, M.naam From functie F, medewerker M Where functie = code and F.naam like 'S%'

Opgave 18 Select naam, inzet_perc From medewerker, werkt Where medewerker = nr;

4.1.2 Twee keer dezelfde tabel

Alias Je kunt ook dezelfde tabel vaker gebruiken. Als je van alle medewerkers de naam en de naam van zijn baas wilt zien, dan kun je die alleen maar naast elkaar op een regel krijgen door twee keer de tabel medewerker te gebruiken. Dus **From medewerker, medewerker**. Omdat in beide tabellen de kolom naam voorkomt en de tabelnamen gelijk zijn moeten we aliassen gebruiken om de twee tabellen uit elkaar te kunnen houden. Achter de tabelnaam zet je een spatie en dan de naam zoals je hem in het vervolg in de query wilt noemen. Een goed gebruik is om voor aliassen hoofdletters te gebruiken. Dus bijvoorbeeld de M van medewerker en de B van zijn baas. Dan krijg je:
From medewerker M, medewerker B.

Query 19 **Geef van elke medewerker de naam en de naam van hun baas**

```
Select M.naam, B.naam
From medewerker M, medewerker B
Where M.baas = B.nr
```

M.naam	B.naam
Piet	Chris
Elske	Chris
Jan	Chris
Chris	Peter
Kees	Chris

NULL Je ziet bij deze query dat niet alle medewerkers worden getoond. Peter heeft geen baas dus komt zijn naam niet in de linkerkolom voor. Bij een *Inner Join* zien we rijen met een NULL-waarde voor de Foreign Key M.baas niet.

4.1.3 3 of meer tabellen

zelfde techniek Met 3 of meer tabellen werkt de *Inner Join* exact hetzelfde als met een query op 2 tabellen. Je moet er wel op letten dat je elke extra tabel die je in het *From* gedeelte opneemt, moet verbinden met een van de andere tabellen in het *Where*-gedeelte. Bij een query op 3 tabellen zul je dus 2x *Foreign Key=Primary Key* in het *Where* gedeelte tegenkomen. Bij een query op 4 tabellen 3x FK=PK. Bij een query op 100 tabellen 99x.

Verbindende tabel Stel je wilt de naam van een medewerker en de naam van het project waaraan zij werken. De naam van de medewerker staat in de tabel medewerker, de naam van het project staat in de tabel project. Maar wat moet er in het *Where*-gedeelte komen te staan? Er bestaat geen directe *Foreign Key* van de ene tabel naar de andere. Maar in dit geval bestaat er wel een verbindende tabel werkt, waarin staat welke medewerker bij welk project hoort. Zo'n verbindende tabel moet je ook opnemen in het *From* gedeelte om in het *Where*-gedeelte als schakel te functioneren tussen medewerker en project.

Query 20 **Geef van elke medewerker hun naam en de naam van hun projecten**

```
Select M.naam, P.naam
From medewerker M, Werkt, project P
Where medewerker = nr
  And  project = code
```

M.naam	P.naam
Piet	Oliecontrole
Piet	Waterafvoer
Peter	Netwerk
Jan	Netwerk
Jan	Waterafvoer
Chris	Waterafvoer

Aantal rijen bepalen We zien hier alles bij elkaar 6 rijen, omdat er 6 rijen in de tabel werkt staan waar beide Foreign Keys in zitten. Bij elke Foreign Key worden als het ware de gegevens uit de andere tabel erbij gezocht. Bij tupel W1 in werkt hoort medewerker M1 en project P3. alle kolommen van deze drie tabellen staan dan op 1 rij naast elkaar. Uiteindelijk wordt alleen de medewerkernaam en projectnaam getoond. Als je begint bij de tabel met de Foreign Keys kun je gemakkelijk bepalen hoeveel rijen eruit komen.

Query 21 **Geef van elke medewerker hun naam, functienaam en de naam van hun projecten**

```
Select M.naam, P.naam, F.naam
From medewerker M, werkt, project P, functie F
Where medewerker = nr
  And  project = P.code
  And  functie = F.code
```

M.naam	P.naam	F.naam
Piet	Oliecontrole	Programmeur
Piet	Waterafvoer	Programmeur
Peter	Netwerk	Projectleider
Jan	Netwerk	Systeem analist
Jan	Waterafvoer	Systeem analist
Chris	Waterafvoer	Projectleider

Extra tabel toevoegen En als we daarbij ook nog de functie van de werknemer willen tonen, dan kunnen we dat eenvoudigweg bereiken door functie toe te voegen in het *From* gedeelte, en de tabel functie in het *Where* gedeelte te verbinden met de tabel medewerker.

Opgave 19 Select Bedrijf, nr, F.naam From project P, medewerker M, functie F Where functie = F.code and functie = P.code

4.1.4 Inner Join met Group by

We kunnen alle eerder besproken technieken hierop toepassen. De volgende query geeft per medewerker de naam en het totale inzet_perc.

Query 22 **Geef van elke medewerker hun naam en het totale inzet_perc**

```
Select naam, Sum( inzet_perc )
From medewerker, werkt
Where medewerker = nr
Group by naam
```

M.naam	Sum(inzet_perc)
Piet	90
Peter	100
Jan	100
Chris	100

Opgave 20 Select Count(*) From functie, medewerker Where functie = code;

Opgave 21 Select Count(functie) From medewerker, functie Where code = functie;

Opgave 22 Select bedrijf, Sum(inzet_perc) From project, werkt Where code = project Group by bedrijf;

4.2 Left Join

4.2.1 Op 2 tabellen

NULL We zagen bij de *Inner Join* van medewerker en functie dat alleen medewerkers met een functie en functies met een medewerker worden getoond. Als de Foreign Key functie van een medewerker NULL is, zien we die medewerker niet (zoals bij Kees), en als de Primary Key van een functie niet als Foreign Key bij een medewerker voorkomt (zoals bij Systeem Ontwikkelaar) dan zien we die ook niet.

medewerker.naam	**functie.naam**
Piet	Programmeur
Elske	Programmeur
Jan	Systeem Analist
Peter	Projectleider
Chris	Projectleider

Alle = vaak Left Join In veel gevallen zijn we in gegevens uit een bepaalde tabel geïnteresseerd en willen we indien aanwezig gegevens die er bij horen uit andere tabellen, erbij zien. In dit geval een overzicht van alle medewerkers met evt. hun functienaam. Om zulk soort queries mogelijk te maken bestaat de *Left Join*. Bij zulke opgaven moet je getriggerd worden door woordjes als **alle** en **elke**.

Query 23 **Geef van alle medewerker de naam en zo mogelijk de naam van hun functie**

```
Select medewerker.naam, functie.naam
From medewerker LEFT JOIN functie
                              ON functie = code
```

medewerker.naam	**functie.naam**
Piet	Programmeur
Elske	Programmeur
Jan	Systeem Analist
Peter	Projectleider
Chris	Projectleider
Kees	

ON De syntax van een *Left Join* wijkt iets af van die van een *Inner Join*. Het wegfilteren van rijen in het *Where*-gedeelte in onverbiddelijk, daarom kunnen we de tabellen niet in het *Where*-gedeelte met elkaar verbinden. Elke *Left Join* wordt daarom gevolgd door een zgn. ON-clausule. In die on-clausule kunnen we net als in de *Where*-clausule aangeven hoe de tabellen verbonden zijn. Als voor een rij in de linker tabel (in dit geval medewerker) geen matchende rij in de rechter tabel wordt gevonden, dan wordt die medewerker toch getoond en hebben alle kolommen van de rechter tabel een NULL-waarde.

Opgave 23 Select M.naam, F.naam From functie LEFT JOIN medewerker ON functie = code

Where na Left Join In de stappenreeks komt na de *From* het *Where*-gedeelte. Alle Joins, dus ook de *Left Join* behoren tot het *From*-gedeelte. Nadat de tabellen zijn gejoind wordt het *Where* gedeelte uitgevoerd. In dat *Where*-gedeelte kunnen nog steeds alle rijen worden weggegooid zodat uiteindelijk niet alle medewerkers over blijven.

Query 24 **Geef van alle medewerkers met als baas 15 de naam en evt. functienaam**

```
Select medewerker.naam, functie.naam
  From medewerker LEFT JOIN functie
                                ON functie = code
Where baas = 15
```

medewerker.naam	**functie.naam**
Piet	Programmeur
Elske	Programmeur
Jan	Systeem Analist
Kees	

Opgave 24 Select naam, inzet_perc From medewerker Left Join werkt Where medewerker = nr;

Opgave 25 Select F.naam, M.naam From functie F Left Join medewerker M ON functie = code Where woonplaats = 'Leiden';

4.2.2 Op 3 tabellen

Op dezelfde manier als met de *Inner Join* kun je ook meer dan 1 *Left Join* gebruiken. Elke *Left Join* krijgt dan zijn eigen ON-clausule :

Query 25 **Geef van alle medewerkers hun naam, functienaam en de naam van hun baas**

```
Select M.naam, F.naam, B.naam
From medewerker M LEFT JOIN functie F ON functie = code
                        LEFT JOIN medewerker B ON M.baas = B.nr
```

M.naam	**F.naam**	**B.naam**
Piet	Programmeur	Chris
Elske	Programmeur	Chris
Jan	Systeem Analist	Chris
Peter	Projectleider	
Chris	Projectleider	Peter
Kees		Chris

Door de *Left Joins* te gebruiken komen Peter en Kees in de uitkomsttabel voor. We kunnen *Left Joins* en *Inner Joins* ook met elkaar combineren. Hieronder twee varianten van dezelfde opgave :

Query 26 **Geef van alle medewerkers die een functie hebben: naam, functie, naam baas**

```
Select M.naam, F.naam, B.naam
From medewerker M
 JOIN functie F ON M.functie = code
      LEFT JOIN medewerker B
            ON M.baas=B.nr
```

```
Select M.naam, F.naam, B.naam
From medewerker M, functie F
LEFT JOIN medewerker B ON M.baas=B.nr
Where M.functie = code
```

M.naam	F.naam	B.naam
Piet	Programmeur	Chris
Elske	Programmeur	Chris
Jan	Systeem Analist	Chris
Peter	Projectleider	
Chris	Projectleider	Peter

Verschil Join en , Combinatie Inner en Left Join

De linker en rechter query zijn identiek, met dien verstande dat sommige DBMS-en slechts een van de twee toestaan. De Join tussen medewerker en functie is nu een *Inner Join*. Dus Kees wordt geschrapt. De Join met de medewerker die hun baas is, is een *Left Join*, dus een medewerker die geen baas heeft (Peter) blijft staan. Op zo'n manier mogen combinaties tussen *Inner* en *Left Joins* worden gemaakt.

4.2.3 Join vs Left Join

Voor gevorderden: pas op voor de situatie waarin je met de tabel rechts van je *Left Join* weer verder gaat met gegevens zoeken… Dat mag in de meeste gevallen namelijk niet zomaar een *Inner Join* zijn. Aanschouw de volgende query:

Query 27 **Geef van alle medewerkers hun naam, baas en de functienaam van hun baas**

```
Select M.naam, F.naam, B.naam
From medewerker M  LEFT JOIN medewerker B ON M.baas = B.nr
                              JOIN functie F ON B.functie = F.code
```

M.naam	B.naam	F.naam
Piet	Chris	Programmeur
Elske	Chris	Programmeur
Jan	Chris	Systeem Analist
Peter		Projectleider
Kees	Chris	Projectleider

Er lijkt ogenschijnlijk niets met deze query mis te zijn, maar er is een probleem als iemands baas geen functie blijkt te hebben. Laten we Joins nader gaan bekijken:

2. From medewerker M Left Join medewerker B ON M.baas = B.nr :
alle 6 rijen in de tabel medewerker doen mee en zo mogelijk wordt de gegevens van hun baas ernaast gezet

nr	naam	baas	functie	B.nr	B.naam	B.functie	etc.
11	Piet	15	PR	15	Chris	PL	
12	Elske	15	PR	15	Chris	PL	
13	Jan	15	SA	15	Chris	PL	
14	Peter		PL				
15	Chris	14	PL	14	Peter	PL	
16	Kees	15		15	Chris	PL	

Maar als nu deze tabel met een *Inner Join* wordt verbonden met functie
JOIN functie ON B.functie = F.code

nr	naam	baas	functie	B.nr	B.naam	B.functie	code	F.naam
11	Piet	15	PR	15	Chris	PL	PL	Projectleider
12	Elske	15	PR	15	Chris	PL	PL	Projectleider
13	Jan	15	SA	15	Chris	PL	PL	Projectleider
~~14~~	~~Peter~~		~~PL~~					
15	Chris	14	PL	14	Peter	PL	PL	Projectleider
16	Kees	15		15	Chris	PL	PL	Projectleider

2x Left Join

Dan wordt Peter geschrapt! Want Peter heeft geen baas, B.functie heeft een NULL-waarde en kan geen passende functie worden gevonden. De ON-clausule is hier hetzelfde als een *Where*-clausule, en schrapt onverbiddelijk alle rijen waarvan de bewering onwaar is. Als je ALLE medewerkers incl. Peter wilt zien, dan moet je de tabel functie dus ook met een *Left Join* verbinden. Goed is dus :

```
Select M.naam, F.naam, B.naam
From medewerker M  LEFT JOIN medewerker B ON M.baas = B.nr
                   LEFT JOIN functie F ON B.functie = F.code
```

alle

Je moet worden getriggerd door woorden als **alle** en **elke** in de opgave: “Geef van **alle** medewerkers hun naam, baas en de functienaam van hun baas”. Hiermee is het duidelijk dat alle medewerkers moeten worden getoond en dus *Left Joins* moeten worden gebruikt. Indien het woord alle ontbreekt gebruik je een *Inner Join*. Zo zal het ook op toetsen worden gevraagd.

4.2.4 Left Joins met kolomfuncties

Alle technieken kunnen gecombineerd worden. Indien je alle medewerkers uit Leiden wilt zien met hun totale inzetpercentage, dan moet je een *Left Join* gebruiken om ook de medewerkers te tonen die niet in de tabel werkt voorkomen. Hun inzetpercentage zal dan 0 worden.

Query 28 **Geef van alle medewerker uit leiden hun naam en het totale inzet_perc**

```
Select naam, Sum( inzet_perc )
From medewerker LEFT JOIN werkt ON medewerker = nr
Where woonplaats = 'Leiden'
Group by naam
```

M.naam	Sum(inzet_perc)
Piet	90
Elske	0
Chris	100

Als je in plaats van een *Left Join* een *Inner Join* had gebruikt dan was Elske er niet in voorgekomen.

Opgave 26 Select F.naam, Count(nr) From functie F Left Join medewerker M ON functie = code and woonplaats = 'Leiden' group by F.naam;

4.3 Subselects

4.3.1 De IN-operator

Element van

In de verzamelingenleer kennen we de “element-van” operator (Jan ∈ KLANT). Die is *TRUE* als een bepaald element in een verzameling voorkomt. In databases kennen we die ook, we noemen dat de in-operator. Voor de in-operator komt een element te staan, en erachter een enkelvoudige verzameling. Een verzameling zet je bij SQL altijd tussen ( )-haakjes en je kunt het *Select* statement gebruiken om een verzameling te beschrijven. We noemen zo’n *Select* statement dan een **Subselect**.

Kijken in een kolom

Heb je nog onthouden dat SQL strikt regel- of groepsgewijs werkt? Welnu, er is 1 manier om daarvan af te wijken en dat is mbv *Subselects*. Je kunt daarmee beweringen opstellen om te kijken of een waarde wel of niet in een bepaalde kolom van een andere tabel voorkomt. We gebruiken *Subselects* dus in het *Where*-gedeelte van queries.

Kijk naar de volgende beweringen :

‘Shell’ IN (Select bedrijf From project)	⇔ TRUE
‘Arjan’ IN (Select naam From medewerker)	⇔ FALSE
‘Shell’ NOT IN (Select bedrijf From project)	⇔ FALSE
‘Arjan’ NOT IN (Select naam From medewerker)	⇔ TRUE

Het maakt dus niet uit hoe vaak de waarde in de *Subselect* voorkomt, en NOT IN is het tegenovergestelde van IN.

Query 29

Geef de naam van alle werkende medewerkers

```
Select naam
From medewerker
Where nr in
            (Select medewerker
              From werkt)
```

M.naam
Piet
Jan
Peter
Chris

Laten we in stapjes kijken wat er gebeurt. De *Subselect* is de eerste stap, dat betekent dat we eerst de hele *Subselect* (met zijn eigen From/Where/etc. stappen) gaan uitwerken en dan gaan we pas de hoofdselect bekijken.

1. Subselect:
2. From werkt

medewerker	**project**	**inzet perc**
11	C	20
11	B	70
14	A	100
13	A	50
13	C	50
15	C	100

7. Select medewerker

medewerker
11
11
14
13
13
15

De uitkomst van de Subselect is dus (11, 11, 14, 13, 13, 15)

2. From medewerker

nr	**naam**	**woonplaats**	**baas**	**functie**
11	Piet	Leiden	15	PR
12	Elske	Leiden	15	PR
13	Jan	Lisse	15	SA
14	Peter	Berkel		PL
15	Chris	Leiden	14	PL
16	Kees		15	

3. Where nr IN (Select….) :

nr	**naam**	**woonplaats**	**baas**	**functie**	**nr IN (Select…)**
11	Piet	Leiden	15	PR	TRUE
~~12~~	~~Elske~~	~~Leiden~~	~~15~~	~~PR~~	
13	Jan	Lisse	15	SA	TRUE
14	Peter	Berkel		PL	TRUE
15	Chris	Leiden	14	PL	TRUE
~~16~~	~~Kees~~		~~15~~		

7. Select naam :

naam
Piet
Jan
Peter
Chris

Opgave 27 Select naam From medewerker Where functie IN (Select code From functie Where naam like 'Systeem%');

Opgave 28 Select naam From functie Where code IN (Select functie From medewerker)

4.3.2 Ingewikkelde Subselects

enkelvoudig Aan *Subselects* zijn wel een paar eisen gesteld. De queries mogen maar één kolom in hun *Select-list* hebben, vandaar dat ze net enkelvoudig werden genoemd. Het gebruik van termen als *Distinct* en *Order by* zijn niet zinnig, aangezien IN alleen maar kijkt of een waarde in de verzameling voorkomt.

PK IN (Select FK …) Op dezelfde manier dat je met Join in 98% van de gevallen relaties tussen twee tabellen bekijkt aan de hand van FK = PK, doe je dat met *Subselects* ook.
Dus: PK IN (Select FK From …) is de meest voorkomende vorm.

Normale select Als *Subselect* mag je verder alles doen wat je met een normale *Select* ook mag. Je mag met *Where*, *Group by*, kolomfuncties, alles aan de slag. Maar merk wel op dat een *Subselect* dus nooit uitvoer op je scherm tovert, intern wordt gekeken of een bewering van de buitenste *Select TRUE* of *FALSE* is. Dat is de enige functie van een *Subselect*.

Query 30 **Geef de naam van alle medewerkers die 100% zijn ingezet**

```
Select naam
From medewerker
Where nr in
                (Select medewerker
                   From werkt
                Group by medewerker
                  Having Sum(inzet_perc) = 100)
```

M.naam
Jan
Peter
Chris

Andere Subselects De basisvorm van een Subselect is het gebruik ervan in de Where-clausule. Dat is ook de enige vorm die we hier gaan behandelen. Sommige DBMS-en kennen daarnaast de mogelijkheid om ook Subselects in de Selectlist en het From-gedeelte te gebruiken. Een krachtige uitbreiding die in sommige gevallen tot veel efficiëntere queries leidt.

Query 31 **Geef de naam van alle medewerkers die werken bij een bedrijf waar minstens 4 medewerkers werken**

```
Select naam                                          ← hoofdselect
From medewerker
Where nr in
            (Select medewerker                       ← Subselect 3
              From werkt
             Where project in
                        (Select code                 ← Subselect 2
                          From project
                         Where bedrijf in
                                    (Select bedrijf  ← Subselect 1
                                      From project, werkt
                                     Where project = code
                                  Group by bedrijf
                                    Having Count(*) > 3)
                        )
            )
```

Begin bij de binnenste Subselect

Wie hiervan schrikt kan zich maar beter schrapzetten. In praktijk zijn queries van een half of soms zelfs heel A4-tje geen uitzondering. Maar laat je niet te snel uit het veld slaan, als je de stapjes volgt kan er niks gebeuren. Onthoud dat we altijd beginnen bij de binnenste *Subselect* en dan langzaam naar buiten toe werken :

1. **Subselect 1**: (Select bedrijf From project, werkt ….)

2,3 From project, werkt Where project = code

medewerker	project	inzet perc	code	bedrijf
11	C	20	C	Shell
11	B	70	B	Shell
14	A	100	A	KPN
13	A	50	A	KPN
13	C	50	C	Shell
15	C	100	C	Shell

4. Group by bedrijf

medewerker	project	inzet perc	code	bedrijf
11	C	20	C	Shell
11	B	70	B	Shell
13	C	50	C	Shell
15	C	100	C	Shell
14	A	100	A	KPN
13	A	50	A	KPN

5. kolomfunctie Count(*)

medewerker	project	inzet perc	code	bedrijf	Count(*)
11	C	20	C	Shell	4
11	B	70	B	Shell	
13	C	50	C	Shell	
15	C	100	C	Shell	
14	A	100	A	KPN	2
13	A	50	A	KPN	

6. Having Count(*) > 3

medewerker	project	inzet perc	code	bedrijf	Count(*)
11	C	20	C	Shell	4
11	B	70	B	Shell	
13	C	50	C	Shell	
15	C	100	C	Shell	

7. Select bedrijf

bedrijf
Shell

Dit is dus de denkbeeldige uitkomst van Subselect 1!!!!

1. **Subselect 2**: (Select code From project Where bedrijf IN (Subselect 1))

2. From project

code	bedrijf	naam	projectleider
A	KPN	Netwerk	14
B	Shell	Oliecontrole	13
C	Shell	Waterafvoer	13
D	Unilever	Zeepfabriek	

3. Where bedrijf IN (Subselect 1) ← Subselect 1 was (Shell)

code	bedrijf	naam	projectleider
B	Shell	Oliecontrole	13
C	Shell	Waterafvoer	13

7. Select code

Code
B
C

Dit is dus de denkbeeldige uitkomst van Subselect 2!!!

1. **Subselect 3**: (Select medewerker From werkt Where code IN (Subselect 2))

2. From werkt

medewerker	project	inzet perc
11	C	20
11	B	70
14	A	100
13	A	50
13	C	50
15	C	100

3. Where code IN (Subselect 2) ← Subselect 2 was (B, C)

medewerker	project	inzet perc
11	C	20
11	B	70
13	C	50
15	C	100

7. Select medewerker

medewerker
11
11
13
15

Dit is het resultaat van Subselect 3!!!

Hoofdquery: From medewerker Where nr IN (Subselect 3)

2. From medewerker

nr	naam	woonplaats	baas	functie
11	Piet	Leiden	15	PR
12	Elske	Leiden	15	PR
13	Jan	Lisse	15	SA
14	Peter	Berkel		PL
15	Chris	Leiden	14	PL
16	Kees		15	

3. Where nr IN (Subselect 3) ← Subselect 3 was (11, 11, 13, 15)

nr	naam	woonplaats	baas	functie
11	Piet	Leiden	15	PR
~~12~~	~~Elske~~	~~Leiden~~	~~15~~	~~PR~~
13	Jan	Lisse	15	SA
~~14~~	~~Peter~~	~~Berkel~~		~~PL~~
15	Chris	Leiden	14	PL
~~16~~	~~Kees~~		~~15~~	

7. Select naam

naam
Piet
Jan
Chris

Dit is dus de uitkomst van de totale query!

Bij sommige van zulke queries kan de uitkomst erg verrassend zijn. Door de stappen secuur uit te voeren en er geen over te slaan kom je er altijd uit!

4.4 Ontkenningen met Subselects

4.4.1 NOT IN

Ontkenning van een relatie

Voor sommige vragen heb je ontkenningen nodig met een Subselect. Geef alle medewerkers die niet zijn ingezet. Je moet dan als het ware gaan kijken welke medewerkers niet voorkomen in de tabel werkt. Omdat de *Subselect* de enige manier is om de regelgewijze werking van SQL te doorbreken, kun je deze vraag alleen met een *Subselect* oplossen.

Query 32 **Geef de namen van alle medewerkers die niet zijn ingezet**

```
Select naam
From medewerker
Where nr NOT in
            (Select medewerker
              From werkt)
```

naam
Elske
Kees

In stapjes :

1. **Subselect**: Select medewerker From werkt

medewerker
11
11
14
13
13
15

2. **hoofdselect**: From medewerker Where nr NOT IN (Subselect)

nr	naam	woonplaats	baas	functie	nr NOT IN (Subselect)
~~11~~	~~Piet~~	~~Leiden~~	~~15~~	~~PR~~	
12	Elske	Leiden	15	PR	TRUE
~~13~~	~~Jan~~	~~Lisse~~	~~15~~	~~SA~~	
~~14~~	~~Peter~~	~~Berkel~~		~~PL~~	
~~15~~	~~Chris~~	~~Leiden~~	~~14~~	~~PL~~	
16	Kees		15		TRUE

7. Select naam

naam
Elske
Kees

4.4.2 Vergelijkingen met NULL-waarden

Unknown

Een bewering in SQL is geen boolean. Elke bewering kan de uitkomst *TRUE*, *FALSE* of **Unknown** opleveren. De uitkomst Unknown komt voor als je vergelijkingen met NULL-waarden maakt, want een NULL-waarde betekent zoiets als onbekend.

'Jan' = 'Jan'	TRUE	'Jan' <> 'Jan'	FALSE
'Jan' = 'Piet'	FALSE	'Jan' <> 'Piet'	TRUE
'Jan' = NULL	Unknown	'Jan' <> NULL	Unknown

Unknown => FALSE

Maar uiteindelijk wordt een rij wel of niet op het scherm getoond. Unknown wordt derhalve in het laatste stadium naar *FALSE* vertaald. Dit leidt tot veel verwarring.

Query 33

Geef alle medewerkers die wel/niet in Leiden wonen

```
Select naam
From medewerker
Where woonplaats = 'Leiden'
```

naam
Piet
Elske
Chris

```
Select naam
From medewerker
Where woonplaats <> 'Leiden'
```

naam
Jan
Peter

NULL waarden geschrapt

Maar waar is Kees gebleven? Kees zijn woonplaats is NULL, dus als je vraagt woont hij in Leiden is het antwoord Unknown. Als je vraagt woont hij niet in Leiden is het antwoord ook Unknown (als zijn woonplaats onbekend is zou hij nl. best in Leiden kunnen wonen). Beide vergelijkingen leiden tot Unknown en uiteindelijk tot *FALSE*, dus vandaar dat alle NULL-waarden eruit vallen. Deze twee queries lijken elkaars complement (tegengestelde) maar zijn het dus niet.

Query 34 **Geef alle medewerkers die wel/niet in Leiden wonen**

```
Select naam
From medewerker
Where woonplaats = NULL
```

naam

```
Select naam
From medewerker
Where woonplaats <> NULL
```

naam

IS (NOT) Beide queries leiden tot een lege tabel, omdat de *Where* clausule voor alle rijen Unknown oplevert wordt er niks getoond. Als je vergelijkingen wilt doen met NULL-waarden moet je dat doen met de IS en IS NOT operatoren. Zoals je ziet is vergelijken van NULL-waarden met = of <> niet zinnig.

Query 35 **Geef alle medewerkers die wel/niet in Leiden wonen**

```
Select naam
From medewerker
Where woonplaats IS NULL
```

naam
Kees

```
Select naam
From medewerker
Where woonplaats IS NOT NULL
```

naam
Piet
Elske
Jan
Peter
Chris

4.4.3 NULL-waarden in Subselects

Bij normale attributen is de betekenis van een NULL-waarde Unknown. Relaties zijn in de praktijk vaak zwart-wit; je bent officieel eigenaar van een auto of niet, je bent officieel getrouwd of niet. Bij Foreign Keys heeft een NULL-waarde dus eigenlijk een andere betekenis, namelijk dat er geen relatie met de andere tabel bestaat. Bij Foreign Keys willen we eigenlijk dat een vergelijking met een NULL-waarde *FALSE* oplevert ipv Unknown. SQL ziet het verschil niet, dus moeten we met NULL-waarden rekening houden als we ontkennende Subselects opstellen.

Query 36 **Toon alle medewerkers die wel/geen programmeur zijn**

```
Select naam
From medewerker
Where functie IN
  (Select code From functie
    Where naam = 'Programmeur')
```

naam
Piet
Elske

```
Select naam
From medewerker
Where functie NOT IN
  (Select code From functie
    Where naam = 'Programmeur')
```

naam
Jan
Peter
Chris

Waar is Kees? Kees is geen programmeur (want dan zou dat wel als functie zijn ingevuld), dus willen we hem bij de rechter query tonen. Omdat zijn functiecode een NULL waarde bevat wordt de volgende vergelijking gemaakt :
NULL NOT IN ('PL') ⇔ Unknown

NULL-waarden toevoegen

We kunnen dit ongemak van SQL corrigeren door aan de query toe te voegen: OR functie IS NULL. Dus omdat functie de waarde NULL aan kan nemen, breiden we het Where-gedeelte uit met **OR functie IS NULL**.

Query 37

Toon alle medewerkers die geen programmeur zijn

```
Select naam
From medewerker
Where functie NOT IN
  (Select code From functie
   Where naam = 'Programmeur')
      OR functie IS NULL
```

naam
Jan
Peter
Chris
Kees

NULL-waarden in Subselect

Op een soortgelijke manier kunnen er ook NULL waarden in de Subselect voorkomen met hetzelfde probleem.

Query 38

Geef alle functies waar geen medewerker bij hoort

```
Select naam
From functie
Where code NOT IN
  (Select functie From medewerker)
```

naam

Deze query geeft een lege tabel als uitvoer. Kijk maar wat er gebeurt :
PL NOT IN (PR, PR, SA, PL, PL, NULL) ⇔ FALSE
PR NOT IN (PR, PR, SA, PL, PL, NULL) ⇔ FALSE
SA NOT IN (PR, PR, SA, PL, PL, NULL) ⇔ FALSE
SO NOT IN (PR, PR, SA, PL, PL, NULL) ⇔ Unknown

NULL-waarden uit de Subselect verwijderen

We willen functie SO echter wel zien, omdat er geen medewerker met die functie is. Dat kunnen we bereiken door de NULL-waarden uit de Subselect te verwijderen met: **Where functie IS NOT NULL**. De volgende query laat dat zien.

Query 39 **Geef alle functies waar geen medewerker bij hoort**

```
Select naam
From functie
Where code NOT IN
  (Select functie From medewerker
  Where functie IS NOT NULL)
```

naam
Systeem ontwerper

Opgave 29 Select naam From medewerker Where nr NOT IN (Select baas From medewerker)

Opgave 30 Select naam From medewerker Where nr NOT IN (Select medewerker From werkt Where project = 'C')

Opgave 31 Select baas From medewerker Where baas NOT IN (Select medewerker From werkt Where project = 'A')

4.5 Gecorreleerde Subselects

4.5.1 Extra criterium bij het verbinden van 2 tabellen

Als je twee tabellen met elkaar Joint, dan heb je wel eens meer dan één criterium.

Query 40 **Toon alle medewerkers die meer verdienen dan hun baas**

```
Select naam
  From medewerker M, medewerker Baas
Where M.baas = Baas.nr
    And M.uurloon > Baas.uurloon
```

naam
Chris

Er zijn verschillende oplossingen mogelijk. Bij de oplossing met een Inner Join zie je dat we de tabel op basis van 2 criteria (nl. baas en uurloon) verbinden. Bij Subselects kun je met de IN operator maar 1 criterium gebruiken. Maar in het Where gedeelte van de Subselect kun je net als bij de Inner Join de tweede verbinding leggen.

Query 41 **Toon alle medewerkers die meer verdienen dan hun baas**

```
Select naam
  From medewerker M
Where uurloon >
    (Select uurloon
      From medewerker
    Where nr = M.baas)
```

```
Select naam
  From medewerker M
Where baas IN
    (Select nr
      From medewerker
    Where M.uurloon > uurloon)
```

naam
Chris

Gecorreleerde Subselect

Beide uitwerkingen zijn goed. In de linker query gebruiken we een > in plaats van IN. De =, <, > tekens in combinatie met een Subselect zijn toegestaan als de Subselect precies 1 waarde oplevert. Dat is hier altijd het geval omdat In de Subselect staat **Where nr = M.baas**, dus alleen de baas van medewerker M wordt in de Subselect geselecteerd. Zo'n verbinding van de Subselect met de hoofdselect is toegestaan. In dit geval is bij de hoofdselect een alias nodig omdat beiden tabellen dezelfde attributen bevatten. Met die alias kun je aangeven dat je het veld baas uit de hoofdselect bedoelt. Met zo'n extra verbinding in het Where-gedeelte naar een veld buiten de Subselect noemen we dit een gecorreleerde Subselect.

In de rechter query staan in de Subselect steeds alle medewerkers die minder verdienen dan medewerker M. in het Where gedeelte wordt gekeken of het baasnr van medewerker M daarin voorkomt. Het geeft dezelfde uitkomst.

4.5.2 Performance nadeel

performance

Er bestaat een belangrijk performance verschil tussen een gecorreleerde Subselect en de andere Subselects die we eerder zagen. Bij de normale (ongecorreleerde) Subselects kan het DBMS eenmalig bepalen wat de uitkomst van de Subselect is, en die uitkomst steeds vergelijken met een veld in de hoofdselect. Aanschouw de volgende 2 queries.

Query 42

Toon alle medewerkers met dezelfde stad als een projectleider (die hun baas is)

```
Select *
  From medewerker
Where woonplaats in
   (Select woonplaats
      From medewerker
    Where code = 'PL')
```

```
Select *
  From medewerker M
Where woonplaats IN
    (Select woonplaats
       From medewerker
     Where code = 'PL'
         and nr = M.baas)
```

Gecorreleerde Subselect valt onder stap 3

Stel je hebt 1000 medewerkers. De linker query bestaat in feite uit 2 Selects. Één Select om de (statische) uitkomst van de Subselect te bepalen en één Select voor de hoofdselect waar de uitkomst van de Subselect tegenaan wordt gehouden. De rechter query heeft echter een variabele M.baas in de Subselect zitten. Je kunt de Subselect pas bepalen als je weet wat de waarde van M.baas is. Het DBMS kan dus weinig anders dan eerst de hoofdselect uitwerken, en voor elke rij in de hoofdselect (waarvan M.baas bekend is) een nieuwe Subselect te maken. Het resultaat is 1001 Selects ipv 2. Om die reden vallen gecorreleerde Subselects onder stap 3 in de stappenreeks in plaats van stap 1! Je kunt immers de Subselect pas opstellen als je weet welke rijen er in de hoofdselect staan.

Indien mogelijk vermijden

Je kunt met gecorreleerde Subselects resultaten bereiken die zonder niet mogelijk zijn. De query optimiser van een DBMS kan complexe Joins optimaliseren zodat er geen Cartesische producten hoeven te worden gemaakt, maar bij gecorreleerde Subselects is dat nauwelijks mogelijk. De performance van gecorreleerde Subselects kan een probleem opleveren, dus indien mogelijk moet je gecorreleerde Subselects vermijden.

4.5.3 Exists

Rijen in de Subselect

Een variant op een gecorreleerde Subselect is de Exists. Exists levert TRUE op als er 1 of meer rijen in een Subselect voorkomen. Exists is FALSE als er geen rijen voorkomen. Omdat er ander dan bij IN geen waarden tussen Selects worden vergeleken is een Exists eigenlijk alleen maar zinnig in combinatie met een gecorreleerde Subselect.

Query 43

Toon alle medewerkers die in dezelfde plaats wonen als een projectleider (die hun baas is)

```
Select *
  From medewerker M
Where EXISTS
   (Select *
      From medewerker
     Where code = 'PL'
        and B.woonplaats = M.woonplaats)
```

```
Select *
  From medewerker M
Where EXISTS
   (Select *
      From medewerker B
     Where code = 'PL'
        and B.woonplaats = M.woonplaats
        and B.nr = M.baas)
```

naam
Piet
Elske

naam
Piet
Elske

Uit beide queries komt toevallig hetzelfde. De projectleider die in dezelfde woonplaats als Piet en Elske woont is toevallig ook hun baas. Omdat de Subselect is gecorreleerd wordt eerst de hoofdselect uitgewerkt. Van de eerste rij (Piet) wordt dan een Subselect opgesteld (M.baas = 15, M.woonplaats = 'Leiden') en in het geval van Piet is er een medewerker die daaraan voldoet, dus EXISTS levert TRUE op, dus wordt Piet getoond. Daarna wordt er voor Elske een aparte Subselect uitgevoerd, etc.

Opgave 32

Select naam From functie Where NOT Exists (Select * From medewerker Where code = functie)

4.6 Union

Je kunt de uitkomsten van twee queries onder elkaar plakken met UNION. Stel je wilt een vergadering met alle projectleiders en alle medewerkers die nog niet zijn ingezet. Je kunt dan een query maken die alle projectleiders geeft en een aparte query die alle medewerkers geeft die niet zijn ingezet.

Toon de naam van de medewerkers die projectleider zijn / niet ingezet zijn

```
Select naam
  From medewerker, functie
Where functie = code
    and functie.naam = 'Projectleider'
```

naam
Peter
Chris

```
Select naam
  From medewerker
    Where nr NOT IN
    (Select medewerker From werkt)
```

naam
Elske
Kees

2 queries die je onder elkaar plakt

We kunnen deze twee queries eenvoudig met elkaar verbinden mbv een UNION. De uitkomsten komen dan in één tabel :

Toon de naam van de medewerkers die projectleider of niet ingezet zijn

```
Select naam
  From medewerker, functie
Where functie = code
    and functie.naam = 'Projectleider'
UNION
Select naam
  From medewerker
    Where nr NOT IN
    (Select medewerker From werkt)
```

naam
Elske
Kees
Peter
Chris

Regels union

Als je twee queries verbind met UNION zijn er een paar regels:

- Het aantal kolommen van de queries moet gelijk zijn
- Per kolom moeten beide queries een vergelijkbaar datatype hebben
- Order by kan alleen maar aan het eind van de hele query waardoor de resultaten over de gehele uitkomsttabel worden gesorteerd.
- Distinct is niet nuttig, UNION haalt sowieso dubbele rijen eruit.

Alternatief voor union

Deze query had je ook kunnen schrijven met 1 Select met een OR in het Where-gedeelte. Je kunt echter ook gegevens uit 2 verschillende tabellen mbv UNION onder elkaar zetten. In feite mag je met in achthouding van de regels verder alles met een UNION. Als jij het leuk vindt om voetbalspelers met type grasmaaiers te UNION-en, prima.

Toon van alle medewerkers hun naam en of ze bij KPN werken

```
Select naam, 'KPN' as kpn
  From medewerker
Where nr IN
   (Select medewerker
      From werkt, project
    Where project = code
       and naam = 'KPN')
UNION
Select naam, '' as kpn
  From medewerker
    Where nr NOT IN
   (Select medewerker
      From werkt, project
    Where project = code
       and naam = 'KPN')
```

naam	kpn
Jan	KPN
Peter	KPN
Piet	
Elske	
Chris	
Kees	

Opgave 33

```
Select 'Leiden', naam
From medewerker
Where woonplaats = 'Leiden'
Union
Select 'KPN', naam
From medewerker
Where nr IN (Select medewerker
                  From werkt, project
                Where project = code
                   and bedrijf = 'KPN');
```

5 QUERIES OPSTELLEN

5.1 Algemeen

5.1.1 Eisen aan antwoorden

Je kunt nu bepalen wat de uitkomst van een query zal zijn. Alle technieken die je daarbij hebt geleerd moet je ook kunnen gebruiken als je zelf queries opstelt. Steeds krijg je een vraag waarop je als antwoord een query moet schrijven.

Geef de functienaam van de medewerker die Piet heet.

Hoe je opgaven moet lezen

We maken een paar afspraken aan de hand van dit voorbeeld.

- Je moet in de query het constante gegeven 'Piet' gebruiken, je mag dus niet in de tabel het nr van Piet opzoeken en het nr van Piet in een query opschrijven.
- De uitkomst moet de naam van de functie zijn, niet de code.
- Queries moeten generiek zijn. Ze moeten dus niet alleen maar werken bij deze invulling van de database, maar ook als de database op een andere manier gevuld is.
- Het gaat niet om de uitkomst van de query, maar of de query dezelfde betekenis heeft als de opgave.
- Streef naar eenvoudige queries. Queries die complexer zijn dan nodig zijn moeilijker leesbaar.
- Streef naar een goede performance. Betrek er geen tabellen bij die niet nodig zijn, voer geen bewerkingen uit die niet nodig zijn.

5.1.2 Begin bij het begin

stappenreeks

Waar moet je mee beginnen als je een opgave krijgt? Bij het lezen van queries hebben we gebruikt gemaakt van een stappenreeks. We kunnen diezelfde reeks hanteren om te kijken welke technieken we nodig hebben, en steeds een aspect tegelijk te bekijken. Stappen die je niet nodig hebt kun je wederom gewoon overslaan. Bij elke query is in feite alleen een Select en From gedeelte verplicht.

Kijk hoe je handmatig kaartbakken zou raadplegen

Verder is een goeie manier om een Select op te stellen bij een informatievraag, eerst kijken hoe je dat handmatig zou doen. Dus als de gegevens in kaartenbakken voor je neus zouden staan. Als iets handmatig efficiënt is kun je het gemakkelijk vertalen in een efficiënte query. De volgende stappen kun je volgen :

1. handmatig de (gegevens) uitkomst van de opgave bepalen
2. Nagaan hoe je handmatig de tabellen raadpleegt, gegevens selecteert en afleidt
3. de stappenreeks volgen; per stap de benodigde techniek(en) bepalen
4. de (gegevens) uitkomst van de query bepalen (maw controleer het antwoord)

5.1.3 Tabel met mogelijke technieken

Als je met een informatievraag wordt geconfronteerd moet je per stap nagaan welke technieken je nodig hebt. Een goede oefening daarvoor is de opgaven van dit hoofdstuk bekijken om alleen als oefening te bekijken welke technieken je nodig zult hebben. Hieronder worden in een tabel

Subselect

Positieve verwijzing	**(1) Positieve verwijzing** Er worden rijen gevraagd die refereren aan gegevens uit andere tabellen (maar die gegevens uit andere tabellen hoef je niet te tonen). Kun je overigens ook met een Join oplossen.	Toon de namen van programmeurs *Where functie IN* *(Select code* *From functie* *Where naam='Programmeur')*
Ontkennende verwijzing	**(2) Ontkennende verwijzing** Als de verwijzing ontkennend is door een woord als niet of nooit dan **moet** je wel een Subselect gebruiken. Ezelsbruggetje is dat je de Subselect positief formuleert (dus de medewerkers die WEL bij project C zijn ingezet) en de ontkenning door NOT IN weergeeft.	Toon de namen van medewerkers die niet bij project C zijn ingezet *Where nr NOT IN* *(Select medewerker* *From inzet* *Where project <> 'C")*
Gecorreleerde Subselect	**(3) Gecorreleerde Subselect** Als je tussen de tabellen 2 vergelijkingen maakt, zoals in dit geval m.baas=b.nr en m.salaris > b.salaris. Je kunt met de in-operator maar een van die twee vergelijkingen maken dus de andere moet je correleren.	Toon de namen van medewerkers die wel/niet meer verdienen dan hun baas *Where baas IN* *(Select nr* *From medewerker B* *Where B.salaris > A.salaris)*
Vergelijken met kolomfunctie	**(4) Vergelijken met kolomfunctie** Ook hier heb je weer twee vergelijkingen nodig. Je moet per medewerker hun salaris vergelijken met het gemiddelde salaris uit hun woonplaats. Dat gemiddelde hoeft niet te worden getoond (dus in Subselect) en je moet het over de medewerkers uit de dezelfde woonplaats bepalen (correleren door a.woonplaats = b.woonplaats)	Toon de namen van medewerkers die meer verdienen dan het gemiddelde salaris uit hun woonplaats *Where salaris >* *(Select avg(salaris)* *From medewerker B* *Where B.woonplaats =* *A.woonplaats)*

From

Inner Join	**(1) (Inner-)Join** Alle tabellen waarvan gegevens moeten worden getoond in de Select-list moet je hier opnemen. Join toont alleen de medewerkers die een functie hebben. Inner Join gebruik je als alle medewerkers een functie moeten hebben (functie mag geen NULL waarde hebben) of als het woord alle is verbonden aan de voorwaarde dat ze een functie hebben.	Toon de naam en functienaam van alle medewerkers die een functie hebben *From medewerker, functie* *(Where functie = code)*
Left Join	**(2) Left Join** Het verschil met de Inner Join is dat alle medewerkers (of ze nou een fuctie hebben of niet) moeten worden getoond. Omdat het woord alle hier niet is beperkt (je wilt ze echt allemaal zien) moet je de tabel medewerker (want daarop heeft het woord alle betrekking) links van de Left Join zetten	Toon de naam en de functienaam van alle medewerkers *From medewerker Left Join functie ON functie=code*
Verbindende tabellen	**(3) Verbindende tabellen** Soms heb je een tabel nodig waar je weliswaar geen gegevens uit toont, maar waaruit de relatie tussen de gegevens blijkt. In dit geval hoef je weliswaar alleen iets uit medewerker en bedrijf te tonen, maar je hebt de tabel inzet nodig om bedrijven en medewerkers aan elkaar te koppelen.	Toon de naam van het bedrijf en de naam van de medewerker die bij dat bedrijf zijn ingezet. *From medewerker, werkt,* *project* *(Where nr = medewerker* *and code = project)*

Where

Selectie van rijen die je wilt tonen	**(1) Selectie van rijen die je wilt tonen** Als je niet in alle rijen die het From gedeelte oplevert geïnteresseerd bent, kun je de overbodige rijen weg selecteren door een bewering op te stellen waar relevante rijen de waarde TRUE hebben en niet relevante de waarde FALSE	Toon de naam van alle medewerkers uit Leiden met functie PR *Where woonplaats='Leiden'* *and functie = 'PR'*
Inner Join	**(2) (Inner-)Join** Als je een (Inner) Join gebruikt moet je in het Where gedeelte de relatie(s) tussen de tabellen geven	Toon de naam en functienaam van alle medewerkers die een functie hebben. *Where functie=code*

Group by en Distinct

Indelen in groepen	**(1) Indelen in groepen** Group by is de voorbereiding op het gebruik van kolomfuncties. Per groep wordt de uitkomst van kolomfuncties berekend en per groep wordt er dus maar 1 rij getoond in de uitkomsttabel.	Toon de naam en het aantal medewerkers per woonplaats *Group by woonplaats*
Distinct	**(2) Distinct** Als je geen kolomfunctie gebruikt heb je ook geen Group by nodig. Distinct haalt in het laatste stadium dubbele rijen (die geheel identiek zijn) eruit.	Toon de namen van de verschillende woonplaatsen van medewerkers *Select Distinct woonplaats*

Kolomfuncties

Count(*)	**(1) Count(*)** Telt het aantal rijen per groep	Toon per woonplaats het aantal medewerkers *Select woonplaats, Count(*)*
Count(expressie)	**(2) Count(expressie)** Telt het aantal rijen waarvoor de expressie geen NULL-waarde is. Komt vooral vaak voor in combinatie met een Left Join.	Toon van alle functies de naam en het aantal medewerkers met die functie *Select functie.naam, Count(nr)*
Statistische functies	**(3) Statistische functies sum, avg, min, max** Berekent per groep een statistische uitkomst over alle rijen in die groep	Toon het hoogste salaris per woonplaats *Select woonplaats, Max(salaris)*

Having

Selectie van groepen die je wilt tonen	**(1) Selectie van groepen die je wilt tonen** Having doet hetzelfde met groepen, als Where op rijen. Een groep waarvoor de gestelde bewering FALSE is wordt niet getoond. Vaak komen in de Having kolomfuncties voor	Toon de namen van woonplaatsen met meer dan 2 medewerkers *Having Count(*) > 2*

Select

Formules	**(1) Formules** Vaak worden er kolommen uit tabellen getoond, maar de Select-list mag ook complexe expressies of formules bevatten. Voor elke rij of groep die moet worden getoond wordt die expressie dan uitgerekend. Elke expressie komt in een aparte kolom.	Toon de naam en een schatting van het maandloon (uurloon * 160) van alle medewerkers *Select naam, uurloon * 160*

Union

Queries samenvoegen	**(1) Queries samenvoegen** Soms wil je gegevens uit 2 tabellen in 1 tabel onder elkaar krijgen bv. alle namen van studenten en docenten. Je kunt voor beide lijsten een aparte query maken en die met UNION verbinden. Let op dat het aantal kolommen en het soort data van de kolommen gelijk is en dat UNION automatisch een DISTINCT gebruikt over het eindresultaat	Toon de namen van functies en de namen van bedrijven *Select naam From functie* *UNION* *Select bedrijf From project*

Order by

Sorteren	**(1) Sorteren** Als je de rijen van de uitvoertabel wilt sorteren	Toon de namen van functies gesorteerd op naam *Order by naam*

5.2 Query op 1 tabel

5.2.1 Bepaal eerst de uitkomst handmatig

Geef de naam en uurloon van medewerkers uit Leiden

Als we naar Bijlage A kijken dan moet er uit deze opgave komen :

naam	uurloon
Piet	100
Elske	110
Chris	140

5.2.2 From

Benodigde gegevens staan concreet in opgave

In de meeste opgaven staat heel concreet genoemd in welke gegevens we geïnteresseerd zijn. In deze opgave geeft medewerker aan in welke tabel je moet kijken, Leiden is een voorwaarde die je stelt aan de rijen die je wilt zien, en naam en uurloon zijn attributen van een medewerker.

Bij From alleen tabellen die je nodig hebt

Als je geen Subselect nodig hebt kun je beginnen bij het opschrijven van het From-gedeelte. Welke tabellen moet je raadplegen om tot het antwoord te komen. Voor deze opgave is het duidelijk dat je alleen de tabel medewerker nodig hebt. Want uit deze tabel kun je het goede antwoord afleiden. Omdat alle benodigde gegevens hierin staan hebben we geen Joins nodig. Neem in het From gedeelte altijd alleen de tabellen op die je echt nodig hebt.

nr	naam	woonplaats	baas	functie	uurloon
11	Piet	Leiden	15	PR	100
12	Elske	Leiden	15	PR	110
13	Jan	Lisse	15	SA	120
14	Peter	Berkel		PL	130
15	Chris	Leiden	14	PL	140
16	Kees		15		100

From medewerker

5.2.3 Where – selectie van rijen die je wilt tonen

voorwaarde getoonde rijen

From medewerker betekent dat we alle medewerkers hebben geselecteerd. In de opgave staat echter dat we alleen in medewerkers uit *Leiden* zijn geïnteresseerd. Maw er wordt een *voorwaarde* genoemd waar alle getoonde rijen aan moeten voldoen. We filteren de overbodige rijen weg met een Where-clausule. Stel een bewering op die TRUE is voor de rijen die we wel willen zien en FALSE voor de rijen die we niet willen zien.

Where woonplaats = ‘Leiden’

5.2.4 Select

Attributen en afgeleiden in de opgave

Als we de Select-list opstellen zoeken we in de opgave naar attributen en afgeleiden daarvan. Naam en uurloon zijn attributen van een medewerker. Verder wordt niets gevraagd.

Select naam, uurloon

5.2.5 Uitkomst nalezen

Dus de oplossing is :

Select naam, uurloon
From medewerker
Where woonplaats = 'Leiden'

Antwoord controleren

Bij hele makkelijke queries kun je gemakkelijk zien dat er weinig fout kan gaan. Bij moeilijke queries kan er echter een adertje onder het gras zitten. Je doet er daarom verstandig aan om nog even objectief de uitkomst van de query te bepalen door de stappenreeks te volgen.

Opgave 34 Geef de namen van alle medewerkers

Opgave 35 Geef de namen van alle medewerkers uit Leiden

Opgave 36 Geef de namen van alle bedrijven

5.3 Group by en kolomfuncties

5.3.1 Bepaal eerst handmatig de uitkomst

Geef per baas het baasnr en het totale uurloon van hun medewerkers

baas	totaal uurloon
14	140
15	430

5.3.2 From

Als je handmatig de uurlonen van de medewerkers met baas=15 optelt kom je op 430. Je hebt dus geen andere tabellen nodig.

nr	naam	woonplaats	baas	functie	uurloon
11	Piet	Leiden	15	PR	**100**
12	Elske	Leiden	15	PR	**110**
13	Jan	Lisse	15	SA	**120**
14	Peter	Berkel		PL	~~130~~
15	Chris	Leiden	14	PL	~~140~~
16	Kees		15		**100**

From medewerker

5.3.3 Where – selectie van rijen die je wilt tonen

Als we de opgave goed geïnterpreteerd hebben zijn alleen de medewerkers die een baas hebben relevant. We kunnen medewerkers zonder baas uitfilteren in het Where gedeelte. Where dient niet alleen om rijen onzichtbaar te maken, maar ook voor het wegfilteren van rijen die bij vervolgstappen niet meedoen (in dit geval optellen van een kolom). In dit geval mogen rijen met baas IS NULL niet meedoen met de optelling.

Where baas IS NOT NULL

5.3.4 Group by – indelen in groepen

In de uitkomst zien we dat er 2 groepen moeten worden getoond. Die met baas 14 en baas 15. Als we Group by baas gebruiken wordt elk baasnr een eigen groep. De kolomfunctie wordt dan per groep toegepast en voor elke groep wordt dan één rij getoond, dus 2 groepen = 2 rijen. Dat is precies wat we willen.

Group by baas

5.3.5 Kolomfunctie – statistische functie SUM

We moeten per groep alle uurlonen optellen. Optellen kunnen we doen met Sum().

Sum(uurloon)

5.3.6 Select

Uiteindelijk tonen we per groep het baasnr en het afgeleide gegeven totaal_uurloon. Om het mooi te maken kun je nog as totaal_uurloon erachter zetten.

Select baas, Sum(uurloon) as totaal_uurloon

Verplicht in Group by

Merk op dat als je de Group by nog niet bedacht had, dat je bij het opstellen van deze stap baas alsnog in de Group by moet opnemen. We hebben eerder de regel gehad dat bij het gebruik van kolomfuncties alle overige velden in de Select-list in de group **moeten** worden opgenomen.

5.3.7 Query nalezen

Select baas, Sum(uurloon) as totaal_uurloon)
From medewerker
Where baas IS NOT NULL
Group by baas

Opgave 37 Geef het aantal medewerkers uit Leiden

Opgave 38 Geef het aantal medewerkers per woonplaats

Opgave 39 Geef de namen van alle medewerkers die een baas hebben

5.4 Joins

5.4.1 Bepaal eerst handmatig de uitkomst

Geef van alle volledige taken (die met inzet_perc = 100) bedrijf en medewerkernaam

naam	bedrijf
Peter	KPN
Chris	Shell

5.4.2 From – Inner Join

Bij deze query heb je een Join nodig. In de tabel werkt kun je zien welke taken er een inzet_perc van 100 hebben. Dan moet je de gegevens van het project en van de medewerker bij zoeken :

werkt			medewerker		project	
medewerker	project	inzet perc	nr	naam	code	bedrijf
11	C	20	11	Piet	C	Shell
11	B	70	11	Piet	B	Shell
14	A	100	14	Peter	A	KPN
13	A	50	13	Jan	A	KPN
13	C	50	13	Jan	C	Shell
15	C	100	15	Chris	C	Shell

Join om meer kolommen naast elkaar te krijgen

Gegevens die naast elkaar staan in de uitkomst (Peter en KPN bijvoorbeeld) moeten al naast elkaar staan na het uitwerken van het From gedeelte. Door de verder rijgewijze werking van SQL is From is de enige manier om meer rijen en meer kolommen toe te voegen aan het mogelijke resultaat. Gegevens op twee verschillende rijen kunnen alleen met een kolomfunctie worden gecombineerd tot een afgeleid gegeven, maar kunnen niet later nog naast elkaar worden gezet.

Left Join of Inner Join

We moeten dus 3 tabellen Joinen. Voor elke Join moet je kiezen tussen een Left of en Inner Join. Een left Join is nuttig als er ook rijen zijn, waar geen tupels uit een andere tabel bij hoort, maar waar toch gegevens uit moeten worden getoond. In dit geval gingen we uit van de tabel *werkt* en daar hoort altijd een *medewerker* en een *project* bij. Dus we gebruiken gewoon een Inner Join. De Inner Join bestaat altijd uit twee delen, een opsomming van tabellen achter From en het leggen van de relaties in het Where gedeelte.

From werkt, medewerker, project
Where medewerker = nr
And project = code

5.4.3 Where – Selectie van rijen die je wilt tonen

We hebben niet alle rijen uit de getoonde Join nodig. Het gaat alleen om de rijen met inzet_perc = 100. De rest kan worden geschrapt.

We hebben bij het leggen van de Join al verbindingen in het Where gedeelte gemaakt maar we moeten het volgende er straks nog aan toevoegen.

Where inzet_perc = 100

5.4.4 Select

We zijn niet in alle kolommen geïnteresseerd. We hebben ook geen afgeleide gegevens nodig. Alleen medewerker.naam en bedrijf wordt gevraagd. Het attribuut naam komt in de tabel project en de tabel medewerker voor. Dan moeten we voor naam dus ofwel de tabelnaam zetten of een alias voor de tabelnaam.

Select M.naam, bedrijf

5.4.5 Query nalezen

```
Select M.naam, bedrijf
From werkt, medewerker M, project
Where medewerker = nr
   And project = code
   And inzet_perc = 100
```

Opgave 40 Geef van alle medewerkers, hun naam, woonplaats en functienaam

Opgave 41 Geef van alle medewerkers uit Leiden, hun naam, woonplaats en functienaam

Opgave 42 Geef van alle medewerkers die voor Shell werken, hun nr, naam en de naam van het project

Opgave 43 Geef van alle onderschikten hun naam en de naam van hun baas.

Opgave 44 Geef van alle medewerkers hun naam en de naam van hun baas.

Opgave 45 Geef van alle medewerkers uit Leiden en die voor Shell werken, hun naam en het aantal projecten waaraan zij werken.

5.5 Subselects gebruiken

5.5.1 Bepaal eerst handmatig de uitkomst

Geef de naam van alle medewerkers die werken, gesorteerd op naam

naam
Piet
Jan
Peter
Chris

5.5.2 Ga na hoe je handmatig de tabellen raadpleegt

Je kijkt in de tabel werkt welke medewerkernummers daarin voorkomen. Uit de tabel medewerker toon je dan de namen van medewerkers met die nummers.

5.5.3 Join vs Subselect

From In het From gedeelte van een query hoeven feitelijk alleen de tabellen te staan waaruit je gegevens op het scherm wilt zien (of evt. kolomfuncties daarover). Je kunt onderscheidt maken in twee soorten queries: (relatie) vragen waarvoor een relatie tussen twee tabellen wordt gebruikt en (ontkennende relatie) vragen waarin een ontkennende relatie tussen twee tabellen wordt gebruikt. Vb. (relatie) Geef de naam van alle medewerkers die werken of (ontkennende relatie) Geef de naam van alle medewerkers die niet werken.

In vragen van type (relatie) kun je kiezen of je een Join of een Subselect gebruikt. Voor de performance van het DBMS maakt het niet veel uit voor welke je kiest. Sommigen vinden Subselects makkelijker leesbaar omdat je de query in stukjes opdeelt en bij grote Joins krijg je erg veel termen in het Where gedeelte. Bij queries van type (ontkennende relatie) heb je geen keus, daarvoor moet je gebruik maken van een Subselect, want om aan te tonen dat een relatie niet bestaat moet je naar alle rijen in een tabel kijken. Daarover straks een voorbeeld.

5.5.4 Subselect – positieve verwijzing

Als we kiezen voor een of meer Subselects in de query dan kun je eerst kijken naar gegevens die je wel wilt raadplegen, maar waarvan je niks rechtstreeks op het scherm toont. In dit geval moet de naam van de medewerkers worden getoond, of ze werken of niet is bepalend of ze worden getoond. Het criterium of ze werken of niet kunnen we dus in het Where gedeelte opnemen. De tabel werkt hoeft niet in het From gedeelte van de hoofdselect voor te komen en kunnen we in een Subselect verwerken.

Selectlist van een Subselect

Als we de tabel werkt in een Subselect zetten, welke kolom moet er dan in de Select-list worden gezet? We leggen een relatie tussen de tabel werkt en de tabel medewerker. Die relatie wordt gelegd door Foreign Key=Primary Key in dit geval medewerker = nr. Dus in de Subselect moeten we de kolom medewerker zetten om die te kunnen vergelijken met het nr van een medewerker.

Where nr IN (Select medewerker From werkt)

Want daaruit komt (11, 11, 14, 13, 13, 15) en in de hoofdselect willen we alleen de medewerkers die daarin voorkomen.

5.5.5 From

De rest van de query is eenvoudig. In de hoofdselect kijk je naar de tabellen waaruit je gegevens moet tonen. Dat is in dit geval de tabel medewerker.

From medewerker

5.5.6 Select

De naam moet worden getoond

Select naam

5.5.7 Order by - sorteren

De tabel moet worden gesorteerd op naam

Order by naam

5.5.8 Nalezen query

Select naam
From medewerker
Where nr IN (Select medewerker From werkt)
Order by naam

5.5.9 Uitbreidingen op makkelijke queries

Je ziet dat een gestructureerde aanpak kan helpen bij het opstellen van queries. Als de vraag ingewikkelder is dan wordt de query wellicht langer, maar niet noodzakelijkerwijs moeilijker om op te stellen. Als je bijvoorbeeld de naam van de functie erbij moet tonen dan moet je een Join leggen naar de tabel functie in de hoofdselect, voor de rest blijft alles hetzelfde.

Geef de naam en functienaam van alle medewerkers die werken

```
Select M.naam, F.naam
From medewerker M, functie F
Where functie = code
   And nr IN (Select medewerker From werkt)
```

Als er geen gegevens uit de tabel functie hoeven te worden getoond, maar we stellen de voorwaarde dat de medewerker Projectleider moet zijn. We kunnen die voorwaarde dan in een Subselect verwerken.

Geef de naam en functienaam van alle projectleiders die werken

```
Select naam
From medewerker
Where nr IN (Select medewerker From werkt)
  And functie IN (Select code From functie Where naam = 'Projectleider')
```

In sommige gevallen moet je ook wel eens verder denken dan 1 tabel. Als we alle medewerkers die bij Shell werken willen tonen dan moeten we in de Subselect van werkt alleen de rijen die bij Shell horen laten zien. Om dat te bereiken moeten we een relatie met de tabel project leggen mbv een Join of wederom een Subselect :

Geef de naam van alle projectleiders die bij Shell werken

Met Subselect in een Subselect:

```
Select naam
From medewerker
Where nr IN
 (Select medewerker From werkt
     Where project IN
      (Select code From project Where bedrijf = 'Shell')
 )
```

met Join in een Subselect:

```
Select naam
From medewerker
Where nr IN
 (Select medewerker From werkt, project
  Where project = code
     and bedrijf = 'Shell')
```

Opgave 46 Geef de namen van alle projectleiders uit leiden

Opgave 47 Geef van alle medewerkers die minder dan 100% zijn ingezet, hun naam.

5.6 Kolomfuncties met Join

5.6.1 Bepaal eerst handmatig de uitkomst

Toon alle functienamen met het aantal medewerkers met die functie uit Leiden

naam	aantal leidenaren
Projectleider	1
Programmeur	2
Systeem analist	0
Systeem ontwerper	0

5.6.2 Ga na hoe je handmatig de tabellen raadpleegt

Je kijkt in de tabel functie naar de code. In de tabel medewerkers kijk je dan naar het aantal rijen dat diezelfde code heeft en woonplaats = Leiden.

5.6.3 From – Left Join

In dit geval kunnen we alleen maar een Join gebruiken en geen Subselect. Dat komt omdat we wel degelijk gegevens uit de tabel medewerker tonen, we tellen nl. het aantal medewerkers en tonen dat.

Left Join vs Inner Join

Dan rest de keuze tussen een Inner Join en een Left Join. Stel jezelf de vraag of we ook geïnteresseerd zijn in functies waar geen medewerker bij hoort. Het antwoord is ja en we moeten dus een Left Join gebruiken. Je moet daar al op bedacht zijn als je woorden als **alle** of **elke** in de vraag tegenkomt. De tabel functie komt links te staan omdat we daarvan alle rijen willen zien. De relatie tussen functie en medewerker die we gebruiken is *functie = code*.

From functie Left Join medewerker ON code = functie

5.6.4 Where

Er zit bij deze opgave een addertje onder het gras. We willen dat alleen de medewerkers uit Leiden worden gebruikt. Je bent dan misschien snel geneigd om Where woonplaats = ‘Leiden’ op te nemen, maar dat is niet goed. Wat er dan zou gebeuren is het volgende :

From functie Left Join medewerker ON code = functie

code	naam	nr	naam	woonplaats	functie
PL	Projectleider	14	Peter	Berkel	PL
PL	Projectleider	15	Chris	Leiden	PL
PR	Programmeur	11	Piet	Leiden	PR
PR	Programmeur	12	Elske	Leiden	PR
SA	Systeem analist	13	Jan	Lisse	SA
SO	Systeem ontwerper				

Where woonplaats = ‘Leiden’

code	naam	nr	naam	woonplaats	functie	wpl=leiden
~~PL~~	~~Projectleider~~	~~14~~	~~Peter~~	~~Berkel~~	~~PL~~	
PL	Projectleider	15	Chris	Leiden	PL	TRUE
PR	Programmeur	11	Piet	Leiden	PR	TRUE
PR	Programmeur	12	Elske	Leiden	PR	TRUE
~~SA~~	~~Systeem analist~~	~~13~~	~~Jan~~	~~Lisse~~	~~SA~~	
~~SO~~	~~Systeem ontwerper~~					

Maar dan krijg je niet meer alle functies te zien. De extra voorwaarde betreft hier de rechter tabel van de Left Join. Je moet er dan bedacht op zijn dat je zo’n voorwaarde niet klakkeloos in het Where-gedeelte op kunt nemen. We kunnen zo’n voorwaarde toevoegen aan het on-gedeelte van de Left Join, zodat alle rijen uit de linker tabel altijd worden getoond.

From functie Left Join medewerker ON code = functie and woonplaats = Leiden

code	naam	nr	naam	woonplaats	functie
PL	Projectleider	15	Chris	Leiden	PL
PR	Programmeur	11	Piet	Leiden	PR
PR	Programmeur	12	Elske	Leiden	PR
SA	Systeem analist				
SO	Systeem ontwerper				

Uit deze Join kun je precies de informatie afleiden die je wilt hebben, vergelijk het maar met de handmatig gewenste uitvoer.

5.6.5 Group by – indelen in groepen

We willen voor elke functie een rij in de uitvoer. Dus gaan we groeperen op functie. **Group by nr** is een sterke eerste gedachte omdat dat de Primary Key van functie is. Een regel van SQL is echter dat je alle kolommen uit de Select-list ook in de Group by moet noemen. Omdat we de functienaam ook willen tonen moet die dus ook in de Group by.

Group by code, functie.naam

Group by functie.naam leidt hier tot hetzelfde resultaat. Er zou alleen een verschil optreden als er twee verschillende functies met dezelfde naam zouden zijn. Door de Primary Key op te nemen ben je zeker dat je daar geen problemen mee hebt.

5.6.6 Kolomfuncties – Count(expressie)

We willen per functie (per groep) het aantal medewerker tellen. Sommige groepen hebben echter geen medewerker. Count(*) levert niet het gewenste resultaat omdat elke rij dan voor 1 telt. Als we tussen de haakjes een expressie zetten worden rijen waarvoor die expressie NULL is niet meegeteld. De beste keus is dan de Primary Key van medewerker nr, omdat die geen NULL mag zijn, dus als daar een NULL waarde staat hoort er geen medewerker bij de functie. Andere attributen mogen vaak wel NULL zijn, dus dat is een wat minder fraaie oplossing.

Count(nr)

5.6.7 Query nalezen

```
Select functie.naam, Count(nr)
From functie Left Join medewerker ON code = functie
                                  and woonplaats = 'Leiden'
Group by code, functie.naam
```

Opgave 48 Geef het aantal projectleiders per woonplaats

Opgave 49 Geef het totale inzet percentage per medewerker

Opgave 50 Geef het aantal projecten per medewerkernr, gesorteerd op aantal projecten

Opgave 51 Geef het aantal medewerkers dat per project wordt ingezet

Opgave 52 Geef per project de projectnaam en het aantal programmeurs dat daar aan werkt

Opgave 53 Geef van alle bazen, hun naam en per functie het aantal van hun onderschikten dat die functie heeft.

Opgave 54 Geef van alle medewerkers die voor shell werken, hun nr, naam, en hun totale inzet percentage

N.B. Bij een aantal van deze opgaven zijn Subselects ook goed te gebruiken, zolang er maar geen gegevens uit de Subselect hoeven te worden getoond.

5.7 UNION

De UNION is eigenlijk heel erg gemakkelijk. Als je een vraag hebt die duidelijk in twee of meer delen uiteen valt kun je voor elk van die delen een aparte query schrijven. Met UNION worden de resultaten tot 1 query onder elkaar geplakt. In de praktijk kom je wel eens databases tegen met aparte tabellen voor klant, leverancier, medewerker, etc. Als je deze in één query wilt verenigen is UNION dé manier. In gevallen van onderdelen uit dezelfde tabel (zoals hier) kun je vaak met (Left) Joins hetzelfde resultaat bereiken als met een UNION.

Opgave 55 Geef van alle medewerkers hun naam, van de medewerkers wiens baas uit leiden komt moet je de naam van de medewerker gevolgd door de naam van de baas geven.

5.8 Kolomfuncties in de Subselect

5.8.1 Bepaal eerst handmatig de uitkomst

Toon per functie de medewerker die het meest verdiend

F.naam	M.naam	uurloon
Projectleider	Chris	140
Programmeur	Elske	110
Systeem analist	Jan	120

5.8.2 Ga na hoe je handmatig de tabellen raadpleegt

Van alle medewerkers met dezelfde functiecode pak je het hoogste uurloon. Bij een medewerker wiens loon het hoogste is van zijn functie toon je zijn naam, uurloon en de functienaam.

Beginnende SQL-ers zijn vrij snel geneigd een query op te stellen als :

```
Select F.naam, M.naam, Max(uurloon)
  From medewerker, functie
Where functie = code
Group by F.naam, M.naam
```

Maar dat gaat helaas niet. Omdat F.naam en M.naam beiden in de Select-list voorkomen moeten ze ook in de Group by staan. Maar dan komt elke medewerker in een aparte groep terecht en krijg je gewoon alle medewerkers te zien. Probleempje! Bij deze voorbeeld database zal het nooit zinnig zijn om M.naam en Max(uurloon) samen in een Select-list zetten, omdat M.naam in de Group by ervoor zorgt dat elke medewerker een eigen groep wordt. Laten we eens kijken naar de Join tussen functie en medewerker.

code	naam	nr	naam	functie	uurloon
PL	Projectleider	14	Peter	PL	130
PL	Projectleider	15	Chris	PL	140
PR	Programmeur	11	Piet	PR	100
PR	Programmeur	12	Elske	PR	110
SA	Systeem analist	13	Jan	SA	120

Eigenlijk staan alle gegevens die we willen tonen hierin, het is alleen een kwestie van rijen wegfilteren die we niet nodig hebben. Maar kun je vaststellen welke rijen moeten blijven staan? Op basis van de gegevens van de rij zelf kun je dat niet, er kan immers altijd een andere rij zijn met dezelfde functie en een hoger uurloon. De enige manier om SQL op een niet rij-gewijze manier te gebruiken is mbv een Subselect. Je moet het uurloon vergelijken met het hoogste uurloon van alle medewerkers die dezelfde functie hebben.

nr	naam	functie	uurloon
14	Peter	PL	130
15	Chris	PL	140

Stel je kijkt naar Peter, dan maak je een Subselect van alle Projectleiders, bepaalt het hoogste uurloon daarvan en kijkt of Peters uurloon daarmee overeen komt. Bij Peter is dat niet zo, bij Chris wel. Dan wordt het dus wel een gecorreleerde Subselect!

5.8.3 Subselect - Vergelijken met kolomfunctie

Where werkt rij-gewijs. Je wilt van een specifieke rij weten of je hem wel of niet moet tonen. Dan moeten we weten wat het hoogste uurloon is van alle medewerkers met dezelfde functie. Aangezien alleen de medewerkers met dezelfde functie als in de hoofdselect worden gebruikt hebben we een gecorreleerde Subselect nodig :

```
Select Max(uurloon)
  From medewerker
Where functie = M.functie
```

Stel je kijkt naar een projectleider dan is M.functie = ‘PL’. De uitkomst is dan :

Max(uurloon)
140

In de hoofdselect willen we alleen een medewerker zien als zijn uurloon het hoogste van zijn functie is. In plaats van een = mag je ook IN gebruiken. Omdat er nu maar 1 waarde uit de Subselect komt kan = ook en dat maakt de query iets leesbaarder.

```
Where uurloon =
 (Select Max(uurloon)
    From medewerker
  Where functie = M.functie)
```

5.8.4 From – (Inner-)Join

We nemen als uitgangspunt de Join tussen medewerker en functie, we hebben immers net gezien dat daar alle benodigde informatie in staat.

```
  From medewerker M, functie F
Where functie = code
```

5.8.5 Query nalezen

```
Select F.naam, M.naam, Max(uurloon)
  From medewerker M, functie F
Where functie = code
   And uurloon =
         (Select Max(uurloon)
           From medewerker
          Where functie = M.functie)
```

Opgave 56 Toon van alle medewerkers die baas zijn de naam en uurloon van degene met het laagste salaris

5.9 Ontkennende Subselect

5.9.1 Bepaal eerst handmatig de uitkomst

Toon de functies waar geen medewerker uit Leiden bij hoort

code	naam
SO	Systeem Ontwerper
PL	Systeem Analist

5.9.2 Ga na hoe je handmatig de tabellen raadpleegt

Ook bij deze opgave zit een addertje onder het gras. Veel beginners zin geneigd vrij snel een query als de volgende op te stellen:

```
Select *
  From functie
Where code IN
     (Select functie
        From medewerker
      Where woonplaats <> 'Leiden')
```

Deze query toont de functies waarbij minimaal 1 medewerker hoort die niet uit Leiden komt. Merk op dat dat niet hetzelfde is als in de opgave gevraagd wordt.

Subselect altijd positief formuleren

In deze opgave zit een ontkennende relatie. Een simpel ezelsbruggetje is dat je de benodigde **Subselect altijd positief formuleert** (dus de medewerkers die wel uit Leiden komen) en de ontkenning bij de IN operator zet. Ga er dan eerst vanuit dat je de functies wilt hebben van medewerkers die wel uit Leiden komen, en verander dan IN in NOT IN.

REGEL: formuleer voor een ontkennende relatie een Subselect ZONDER ontkenning (positief). De ontkenning wordt een NOT IN.

Per functie kijk je of de functiecode als Foreign Key in de tabel medewerker voorkomt bij iemand uit Leiden. Je hebt hiervoor een Subselect nodig met alle functiecodes van medewerkers uit Leiden. Als ie niet in dat rijtje voorkomt toon je hem.

5.9.3 Subselect – Ontkennende verwijzing

De hoofdselect hoeft alleen functienamen te tonen dus de tabel medewerker hoeft niet in het From gedeelte van de hoofdselect te staan, maar die kan in een Subselect worden weggewerkt. Omdat er een ontkennende relatie in de opgave staat **moet** daar zelfs een Subselect voor worden gemaakt. Het criterium of een medeweker wel of niet uit Leiden komt staat dus in de Subselect. De Subselect formuleren we “altijd” positief dus de medewerkers die wel uit Leiden komen.

Where code NOT IN (Select functie From medewerker Where woonplaats = ‘Leiden’)

5.9.4 From

We hoeven in de hoofdselect alleen de functie te tonen

From functie

5.9.5 Query nalezen

```
Select *
From functie
Where code NOT IN
   (Select functie From medewerker Where woonplaats = 'Leiden')
```

Helaas zijn we nog niet klaar. Als je met NOT IN werkt moet je waken voor NULL-waarden. Het komt in dit geval toevallig zo uit dat alle medewerkers uit Leiden een functiecode hebben, dus het maakt met deze invulling van de database niet uit. Maar we streven natuurlijk naar queries die het altijd doen omdat we in realistische situaties het antwoord niet met de hand kunnen controleren.

NULL in Subselect

Kijk of aan een van de twee kanten NULL waarden voor kunnen komen:

- Kan code in functie ooit een NULL-waarde zijn? Nee want het is de Primary Key. Als het wel NULL zou kunnen zijn zouden we **OR code IS NULL** toe moeten voegen.
- Kan functie in medewerker een NULL-waarde hebben? JA! Kijk maar naar Kees. Dus NULL-waarden moeten uit de Subselect worden verwijderd met **Where functie IS NOT NULL**.

```
Select *
From functie
Where code NOT IN
   (Select functie
     From medewerker
    Where woonplaats = 'Leiden'
       and functie IS NOT NULL)
```

Is dus het goede antwoord!

Opgave 57 Geef de medewerkers die geen baas uit Berkel hebben

Opgave 58 Geef de namen van projecten waaraan geen baas meewerkt

Opgave 59 Geef de medewerkers die geen Systeem-functie vervullen (like 'Systeem%')

Opgave 60 Geef de medewerkers die geen baas zijn van een andere medewerker die uit Lisse komt

Index

Bijlage A

Functie

	code	naam
F1	PL	Projectleider
F2	PR	Programmeur
F3	SA	Systeem analist
F4	SO	Systeem ontwerper

Medewerker

	nr	naam	woonplaats	baas	functie	uurloon
M1	11	Piet	Leiden	15	PR	100
M2	12	Elske	Leiden	15	PR	110
M3	13	Jan	Lisse	15	SA	120
M4	14	Peter	Berkel		PL	130
M5	15	Chris	Leiden	14	PL	140
M6	16	Kees		15		100

Project

	code	bedrijf	naam	projectleider
P1	A	KPN	Netwerk	14
P2	B	Shell	Oliecontrole	13
P3	C	Shell	Waterafvoer	13
P4	D	Unilever	Zeepfabriek	

Werkt

	medewerker	project	inzet perc
W1	11	C	20
W2	11	B	70
W3	14	A	100
W4	13	A	50
W5	13	C	50
W6	15	C	100

In het zwart staan de tabellen. De codes voor de rijen (F1, F2, .., M1, M2, .. etc.) worden gebruikt om die rijen aan te duiden.