Creatief

met

ANSI SQL

INHOUDSOPGAVE CREATIEF MET ANSI SQL

1 Inleiding

1.1 Definitie

SQL (Structured Query Language) is een ANSI/ISO-standaardtaal voor een relationeel database management systeem (DBMS). Het is een gestandaardiseerde taal die gebruikt kan worden voor taken zoals het bevragen en het aanpassen van gegevens in een relationele databank. SQL kan met vrijwel alle moderne relationele databankproducten worden gebruikt. SQL is een vierde-generatie-taal omdat ze niet imperatief maar declaratief is, zoals Prolog. Een taal is declaratief als de basiselementen geen opdrachten zijn, maar beschrijvingen die tot ingewikkelder beschrijvingen worden samengesteld.

In de informatica worden computertalen ingedeeld in declaratieve talen en imperatieve talen. Heel kort gezegd is het verschil dat men in een declaratieve taal opschrijft wat er aan de hand is, en in een imperatieve taal hoe iets moet gebeuren. Een volkomen helder onderscheid is niet te maken.

1.2 Beschrijving

SQL is gebaseerd op de relationele algebra en werd in de loop van de jaren zeventig ontwikkeld door IBM (San Josй). Sinds het ontstaan van SQL hebben reeds vele verschillende SQL-versies het levenslicht gezien. Pas in de loop van de jaren 80 werd SQL gestandaardiseerd. Tegenwoordig gebruiken de meeste RDBMSen SQL-92.
Bij het beschouwen van de verschillende SQL-implementaties moeten we vaststellen dat bijna elk DBMS zijn eigen extra functies heeft toegevoegd aan SQL-92. Dit maakt dat computerprogramma's waarbij de databank-interface werd geschreven met behulp van SQL niet noodzakelijk zonder problemen kunnen worden gemigreerd van de ene naar de andere SQL-compatibele databank. In vele gevallen werd door de ontwikkelaar van de software wel een of andere SQL-functie gebruikt die alleen maar bestaat in de SQL-implementatie van ййn specifiek DBMS.
In eerste instantie werd SQL ontwikkeld als een vraagtaal voor de eindgebruiker. Het idee was dat businessmanagers SQL zouden gaan gebruiken om bedrijfgegevens te analyseren. Achteraf is gebleken dat SQL te complex is om door eindgebruikers toegepast te worden. Het gebruik van SQL impliceert immers een volledige kennis van de structuur van de te ondervragen databank.

Tegenwoordig wordt SQL vrijwel uitsluitend door tussenkomst van een applicatie gebruikt. De programmeur van de applicatie benadert de database met SQL via een Application Programming Interface (API), zoals ODBC of ADO (MS Windows), JDBC (Java) of een productspecifieke API. SQL is dus in essentie omgevormd van een taal voor eindgebruikers tot een brug tussen applicaties en databanken.

SQL is niets anders dan een verzameling van commandos/instructies die zijn te categoriseren in de volgende vijf groepen

Afk.	beschrijving	Instructies
DQL	Data Query Language	SELECT
DML	Data Manipulation Language	DELETE, INSERT, UPDATE
DDL	Data Definition Language	CREATE, DROP, TRUNCATE, ALTER
TCL	Transaction Control Language	COMMIT, ROLLBACK, SAVEPOINT
DCL	Data Control Language	GRANT, REVOKE

2 Databasenormalisatie

2.1 Inleiding

Databasenormalisatie is het belangrijkste hulpmiddel bij het ontwerpen van gegevensbanken. Oorspronkelijk begonnen als noodzaak om de data zo compact mogelijk in een database te kunnen opslaan, gezien de beperkte opslagcapaciteit van de toenmalige computers, heeft databasenormalisatie zijn eigen rechtvaardiging gekregen.

Door herhaalde gegevens in een tabel apart op te slaan in een gerelateerde tabel is het niet alleen mogelijk het dubbel opslaan van gegevens te vermijden, maar zijn vooral ook foute dubbelingen te voorkomen. Zo kunnen plaatsnamen bij adresgegevens in een aparte tabel opgenomen worden waardoor we verkeerde spelling kunnen vermijden. Ook kunnen we naam-, adres- en woonplaatsgegevens (NAW) van order- of abonnementsgegevens apart opslaan. Hierdoor kunnen we vermijden dat een klant of abonnee twee keer dezelfde rekening krijgt voor een eenmalig gekocht product.

2.2 Hiλrarchie van normaalvormen

Er zijn verschillende stappen om overbodige (redundante) gegevens te beperken. Deze stappen noemen we normaalvormen. Een normaalvorm is een stap in het ontwerp van een relationele database waarbij we in fasen naar een verantwoorde wijze van gegevensopslag toewerken. Belangrijke punten hierbij zijn het voorkomen van dubbele opslag van gegevens en dat elke regel in elke tabel met behulp van een unieke identificatie opgevraagd kan worden. Elke normaalvorm stelt bepaalde eisen aan de manier waarop de gegevens zijn opgeslagen.

Er zijn meerdere normaalvormen bekend, maar de meest gebruikte zijn de zogenaamde eerste, tweede, derde en vierde normaalvorm. De gegevens staan in een bepaalde normaalvorm wanneer we aan een aantal voorgeschreven voorwaarden voldaan hebben. Gegevens staan bijvoorbeeld in de tweede normaalvorm als en slechts als ze voldoen aan de eerste normaalvorm en aan een aantal extra regels.

A. Niet-genormaliseerde gegevens: nulde normaalvorm
Ieder niet gestructureerd gegevensbestand is in de *nulde normaalvorm* (0NF) of niet-genormaliseerd. Gegevens van verschillende soorten kunnen op elke regel voorkomen, bijgevolg kunnen deze niet in kolommen worden opgedeeld.

Om tot de Nulde Normaalvorm te komen moeten we de informatiebehoefte gaan inventariseren. Voor het gemak hebben we even een afleverbon gemaakt:

AFLEVERBON

ORDERNR.: 3405 ORDERDATUM: 15-06-2004
KLANTNR.: 100658

C1000 Oud-Beijerland
Graaf van Egmondstraat 67B
3261 AK Oud-Beijerland

Artnr.	Art.omschrijving	Aantal	Prijs	Totaal
15201	Biologisch ei 6st per 20	1	12,50	12,50
21987	Biologische kaas Campina 10*0.5 kg Aanbiedingpr.	6	8,99	53,94
12365	Rolcontainer	1	125,00	125,00
12366	CBL Krat Zwart 24, emballage	7	3,60	25,20

Totaal € 216,64

Nu gaan we inventariseren. Dit betekent dat we alle gegevens op de afleverbon netjes onder elkaar gaan zetten. We krijgen dan het volgende:

Nulde normaalvorm

ORDERS
ordernr

orderdatum
klantnr
klantnaam
adres
postcode
plaats
artnr
artomschrijving
aantal
prijs
regeltotaal
eindtotaal

Zoals we zien hebben we de inventarisatie ook een naam gegeven: **ORDERS**.

B. Eerste normaalvorm (1NV)
De eerste normaalvorm is een set basale ontwerpregels die op elke database van toepassing zijn. Een tabel is de representatie van een ding of object uit het systeem dat we maken. Bijvoorbeeld een bestelling, een klant, een contactmoment, een product, etc. Elke rij in de tabel is een uniek exemplaar van dat ding. Een rij vertegenwoordigt bijvoorbeeld 1 bestelling, 1 klant, 1 contactmoment, etc.

- Elke tabel heeft een primaire sleutel: een zo klein mogelijk aantal velden dat een rij (record) uniek identificeert.
- Atomiciteit of ondeelbaarheid: elk veld bevat maar ййn ondeelbaar gegeven. Een adres bijvoorbeeld horen we op te slaan in aparte velden voor de straatnaam, het huisnummer en de huisnummerextensie.
- De volgorde van de rijen in de tabel is onbelangrijk (als we willen weten in welke volgorde bestellingen zijn binnengekomen moeten we hiervoor een datum/ tijd veld maken. Hiervoor de rijvolgorde in de database gebruiken is niet de bedoeling)

Na het inventariseren is het de bedoeling dat we naar de Eerste Normaalvorm gaan. Dit doen we altijd met de volgende stappen:

- Maak de gegevens ondeelbaar
- Verwijder alle procesgegevens.
- Geef de sleutel van de groep aan.
- Geef de deelverzameling aan die een herhaald aantal keren voorkomt t.o.v. de primaire sleutel.
- Herhaal de sleutelgegevens van de oorspronkelijke groep samen met de gegevens van de zich herhalende deelverzameling als een nieuwe groep.
- Verwijder de zich herhalende deelverzameling uit de oorspronkelijke groep.

Okй, dan nu verder met ons voorbeeld:

AFLEVERBON

ORDERNR.: 3405 ORDERDATUM: 15-06-2004
KLANTNR.: 100658

C1000 Oud-Beijerland
Graaf van Egmondstraat 67B
3261 AK Oud-Beijerland

Artnr.	Art.omschrijving	Aantal	Prijs	Totaal
15201	Biologisch ei 6st per 20	1	12,50	12,50
21987	Biologische kaas Campina 10*0.5 kg Aanbiedingpr.	6	8,99	53,94
12365	Rolcontainer	1	125,00	125,00
12366	CBL Krat Zwart 24, emballage	7	3,60	25,20

Totaal € 216,64

Maak de gegevens ondeelbaar
In een kolom mogen alleen gelijksoortige gegevens staan. Adres moeten we dus splitsen in straat, huisnummer en eventuele toevoeging.

Verwijder alle proces- of afleidbare gegevens.

In ons voorbeeld is het natuurlijk overduidelijk wat de procesgegevens zijn. Het regeltotaal wordt berekend uit aantal * prijs en het eindtotaal wordt berekend als som van alle regeltotalen. Deze twee strepen we dus weg.

Geef de sleutel van de groep aan.
Een sleutel is altijd uniek en dient voor het uniek identificeren van een tupel of een rij. In ons voorbeeld is de sleutel **ordernr**, immers een klant kan vaker een order plaatsen, maar de orders kunnen nooit hetzelfde nummer hebben. De sleutel geven we aan door deze te onderstrepen.

Geef de deelverzameling aan die een herhaald aantal keren voorkomt t.o.v. de primaire sleutel.
Deze deelverzameling noemt men ook wel eens de Repeterende Groep (RG). Dit zijn de gegevens die vaker voorkomen. In ons voorbeeld is dit het tabelletje met de bestelde goederen. Geef deze gegevens aan in de inventarisatie. We krijgen dan dus:

Nulde normaalvorm

ORDERS
ordernr
orderdatum
klantnr
klantnaam
straat
huisnummer
huisnummer toevoeging
postcode
plaats
RG artnr
RG artomschrijving
RG aantal
RG prijs
X regeltotaal (procesgegeven)
X eindtotaal (procesgegeven)

Herhaal de sleutelgegevens van de oorspronkelijke groep samen met de gegevens van de zich herhalende deelverzameling als een nieuwe groep.

Betere beschrijving nodig? Die kunnen we niet verzinnen. Kijk maar naar het voorbeeld

Nulde normaalvorm

ORDERS
ordernr
orderdatum
klantnr
klantnaam
straat
huisnummer
huisnummer toevoeging
postcode
plaats
RG artnr
RG artomschrijving
RG aantal
RG prijs

BESTELDE_ARTIKELEN
ordernr
artnr
artomschijving
aantal
prijs

Zoals we zien heeft de tweede groep nog geen sleutel. De sleutel moeten we zodanig kiezen dat er zo min mogelijk herhaalde groepen voorkomen t.o.v. deze sleutel. Het liefst natuurlijk geen herhaalde groepen meer, anders moeten we de vorige stappen nog een keer herhalen.

Meestal kunnen we een combinatie nemen van de sleutel van de oorspronkelijke groep en het gegeven dat in de Repeterende Groep de sleutelrol vervult. De sleutel wordt in dit geval een combinatie van ordernr en artnr.

Verwijder de zich herhalende deelverzameling uit de oorspronkelijke groep.
Hier moeten we de RG dus weer opruimen en dan hebben we de Eerste Normaalvorm:

Eerste normaalvorm

ORDERS
ordernr
orderdatum
klantnr
klantnaam
straat
huisnummer
huisnummer toevoeging
postcode
plaats

BESTELDE_ARTIKELEN
ordernr
artnr
artomschrijving
aantal
prijs

C. Tweede normaalvorm (2NV)*:* het verwijderen van redundante gegevens

- De database voldoet aan alle regels van de eerste normaalvorm.
- Zo min mogelijk gegevens worden dubbel opgeslagen in de database.
- De velden die geen primaire sleutel zijn, zijn afhankelijk van de primaire sleutel.

Nu is het tijd om naar de Tweede Normaalvorm te gaan. Dit doen we ook weer met een aantal vaste stappen:
- Geef de attributen aan die niet functieeel afhankelijk zijn van de volledige sleutel.
- Formeer een aparte groep voor ieder deel van de sleutel waarvan de attributen functieeel afhankelijk zijn.
- Neem in iedere groep de attributen met het bijbehorende sleuteldeel op en wijs de primaire sleutel aan.
- Verwijder deze attributen uit de oorspronkelijke groep.

Het voorbeeld:

AFLEVERBON

ORDERNR.: 3405 ORDERDATUM: 15-06-2004
KLANTNR.: 100658

C1000 Oud-Beijerland
Graaf van Egmondstraat 67B
3261 AK Oud-Beijerland

Artnr.	Art.omschrijving	Aantal	Prijs	Totaal
15201	Biologisch ei 6st per 20	1	12,50	12,50
21987	Biologische kaas Campina 10*0.5 kg Aanbiedingpr.	6	8,99	53,94
12365	Rolcontainer	1	125,00	125,00
12366	CBL Krat Zwart 24, emballage	7	3,60	25,20

Totaal € 216,64

Geef de attributen aan die niet functieeel afhankelijk zijn van de volledige sleutel.
We herhalen nog even de Eerste Normaalvorm:

Eerste normaalvorm

ORDERS
ordernr

orderdatum
klantnr
klantnaam
straat
huisnummer
huisnummer toevoeging
postcode
plaats

BESTELDE_ARTIKELEN
ordernr
artnr
artomschrijving
aantal
prijs

De eerste groep komt niet in aanmerking voor deze stap omdat het niet beschikt over een samengestelde sleutel. Binnen de andere groep zijn er wel gegevens die functieeel afhankelijk zijn van een deel van de sleutel.

Kijk maar eens naar artomschrijving en prijs. Blijkbaar zijn deze afhankelijk van artnr en niet van ordernr. We kunnen dit nagaan door te kijken wat er veranderd als het artnr gewijzigd wordt.

We geven dit zo aan in het voorbeeld:

Eerste normaalvorm

ORDERS
ordernr
orderdatum
klantnr
klantnaam
straat
huisnummer
huisnummer toevoeging
postcode
plaats

BESTELDE_ARTIKELEN
ordernr
> artnr
> artomschrijving
aantal
> prijs

Let erop dat aantal niet alleen afhankelijk is van artnr. Het aantal per artikel kan verschillen per order.

Formeer een aparte groep voor ieder deel van de sleutel waarvan de attributen functieeel afhankelijk zijn.

Het kan gebeuren dat een samengestelde sleutel in meerdere delen gesplitst kan worden en dat van ieder deel afzonderlijk attributen functieeel afhankelijk zijn. Er moeten dan meerdere groepen gevormd worden. In het voorbeeld ontstaat slechts ййn nieuwe groep: ARTIKELEN.

Neem in iedere groep de attributen met het bijbehorende sleuteldeel op en wijs de primaire sleutel aan.

Eerste normaalvorm

ORDERS
ordernr
orderdatum
klantnr
klantnaam
straat
huisnummer
huisnummer toevoeging
postcode

plaats

BESTELDE_ARTIKELEN
ordernr
> artnr
> artomschrijving
aantal
> prijs

ARTIKELEN
artnr
artomschrijving
prijs

Verwijder deze attributen uit de oorspronkelijke groep.

Na deze stap hebben we onze Tweede Normaalvorm en begint het al lekker op te schieten:

Tweede normaalvorm

ORDERS
ordernr
orderdatum
klantnr
klantnaam
straat
huisnummer
huisnummer toevoeging
postcode
plaats

BESTELDE_ARTIKELEN
ordernr
artnr
aantal

ARTIKELEN
artnr
artomschrijving
prijs

D. Derde normaalvorm (3NV)
De derde normaalvorm: transitieve (overdraagbare) afhankelijkheden, een moeilijke term voor iets vrij eenvoudigs.

- De database voldoet aan alle regels van de tweede normaalvorm.
- Van een tabel die voldoet aan de derde normaalvorm zijn alle velden die geen primaire sleutel zijn uitsluitend afhankelijk van de primaire sleutel.

Hiervoor hebben we de volgende stappen:

- Geef de niet-sleutel attributen aan die functieeel afhankelijk zijn van andere niet-sleutel attributen.
- Formeer een aparte groep voor ieder attribuut of combinatie van attributen, waar andere attributen functieeel van afhankelijk zijn.
- Neem in iedere groep de attributen met bijbehorende sleutel op en wijs de primaire sleutel aan.
- Verwijder de attributen van de nieuwe groep(en) uit de oorspronkelijke groep.

We beginnen nog maar eens met de uitkomst van de Tweede Normaalvorm:

Tweede normaalvorm

ORDERS

ordernr
orderdatum
klantnr
klantnaam
straat
huisnummer
huisnummer toevoeging
postcode
plaats

BESTELDE_ARTIKELEN
ordernr
artnr
aantal

ARTIKELEN
artnr
artomschrijving
prijs

Geef de niet-sleutel attributen aan die functieeel afhankelijk zijn van andere niet-sleutel attributen.

Als we kijken in de tabel ORDERS zien we dat er een aantal gegevens zijn die niet afhankelijk zijn van ordernr. Deze zijn klantnaam, adres, postcode en plaats. Deze zijn afhankelijk van het klantnr. Klantnr is in deze tabel een niet-sleutelattribuut, dus deze stap is makkelijk uit te voeren: (de afhankelijkheid is aangegeven met een A)

Tweede normaalvorm

ORDERS
ordernr
orderdatum
A klantnr
A klantnaam
A straat
A huisnummer
A huisnummer toevoeging
A postcode
A plaats

BESTELDE_ARTIKELEN
ordernr
artnr
aantal

ARTIKELEN
artnr
artomschrijving
prijs

Formeer een aparte groep voor ieder attribuut of combinatie van attributen, waar andere attributen functioneel van afhankelijk zijn.

Vrij vertaald moeten we voor de groep die we net aangegeven hebben een nieuwe groep maken:

Tweede normaalvorm

ORDERS
ordernr
orderdatum
A klantnr
A klantnaam
A straat
A huisnummer
A huisnummer toevoeging
A postcode
A plaats

KLANTEN
klantnaam
adres
postcode
plaats

BESTELDE_ARTIKELEN
ordernr
artnr
aantal

ARTIKELEN
artnr
artomschrijving
prijs

Neem in iedere groep de attributen met bijbehorende sleutel op en wijs de primaire sleutel aan.

We moeten dus de sleutel waarvan de nieuwe groep afhankelijk is in de nieuwe groep plaatsen, daarna moeten we de sleutel aangeven. De nieuwe groep ziet er dan zo uit:

KLANTEN
klantnr
klantnaam
straat
huisnummer
huisnummer toevoeging
postcode
plaats

Let er wel op dat klantnr niet zomaar uit ORDERS gehaald kan worden, dan zouden de orders namelijk nooit aan een klant gekoppeld kunnen worden.

Verwijder de attributen van de nieuwe groep(en) uit de oorspronkelijke groep.

Als we deze stap hebben gedaan zijn we klaar met het normaliseren. Dit ziet er dan zo uit:

Derde normaalvorm

ORDERS
ordernr
klantnr
orderdatum

KLANTEN
klantnr
klantnaam
straat
huisnummer
huisnummer toevoeging
postcode
plaats

BESTELDE_ARTIKELEN
ordernr
artnr
aantal

ARTIKELEN
artnr
artomschrijving
prijs

E. Boyce-Codd Normaalvorm (BCNF)
De Boyce-Codd normaalvorm is gelijk aan de derde normaalvorm maar kent een striktere definitie voor transitieve (overdraagbare) afhankelijkheid: Ook niet voor de hand liggende transitieve afhankelijkheden moeten we elimineren. Goed voorbeeld hiervan is het netnummer van een stad: is de stad bekend, dan is ook het netnummer bekend. Ander voorbeeld: in Nederland verwijst de combinatie postcode + huisnummer uniek naar adres en plaats.

BCNF richt zich op het elimineren van functieele afhankelijkheden die binnen het kader van een informatiesysteem niet bestaan, maar in de dagelijkse praktijk wel.

F. Vierde Normaalvorm (4NF)
Een sleutel mag nooit, maar dan ook nooit eigenschappen bevatten die meerdere mogelijkheden of instanties kunnen opleveren. De kans dat dit voorkomt is minimaal klein, en kan enkel gebeuren bij meerdere primary keys. Wat natuurlijk wel slim kan zijn, is zo vaak mogelijk het gebruik van meerdere primaire sleutels in een tabel, te vermijden.

Voorbeeld:

Werknnr	Cursusnr	Afdelingnr
15	10	102
15	10	104
15	20	102
15	20	104
45	10	102
45	30	102

Werknnr	Cursusnr
15	10
15	20
45	10
45	30

Werknnr	Afdelingnr
15	102
15	104
45	102

G. Vijfde Normaalvorm (5NV)
Een tabel voldoet aan de vijfde normaalvorm als hij aan de vierde normaalvorm voldoet en als het onmogelijk is hem te vervangen door twee of meer van zijn projecties (lees: set van kolommen) zonder dat hierbij informatie verloren gaat. Kortom als een tabel niet langer op te splitsen is.

2.3 Databasenormalisatie: functioneel belicht

Tijdens de stap van de tweede normaalvorm kwamen we tot de tabel **Artikelen**:

ARTIKELEN
artnr
artomschrijving
prijs

De uitgevoerde normalisatie is technisch correct. De vraag is evenwel of het resultaat geschikt is voor elke situatie. Stel dat we een artikel een nieuwe prijs willen geven. We kunnen dit nu alleen doen door de oude prijs te vervangen. We raken dan onze historie kwijt. Willen we die historie kunnen vasthouden, dan moeten we deze tabel opsplitsen.

ARTIKELEN
artnr
artomschrijving

en, er van uitgaande dat de prijs niet twee keer op dezelfde dag veranderd kan worden:

ARTIKELEN_PRIJS_HISTORIE
artnr
prijs
datumstart
datumeind

Hetzelfde geldt min of meer voor de tabel Klanten

KLANTEN
klantnr
klantnaam
straat
huisnummer
huisnummer toevoeging
postcode
plaats

Deze zouden we kunnen opsplitsen in:

KLANTEN
klantnr
datumstart
datumeind
klantnaam

en

KLANTEN
klantnr
datumstart
datumeind
straat
huisnummer
huisnummer toevoeging
postcode
plaats

3 Data Query Language

3.1 Inleiding

Een **SELECT** instructie kan worden samengesteld uit een aantal componenten. Elk van deze componenten gaan we apart bespreken.

Syntax:

```
SELECT *
FROM table_name
WHERE column_name operator value1
GROUP BY column_name
HAVING column_name operator value1
ORDER BY column_name
```

3.2 SELECT

We kunnen SELECT zelfs helemaal los gebruiken:

```
SELECT 'Ola'
```

3.3 SELECT FROM

De **SELECT** instructie wordt gebruikt om data te selecteren uit een database. Het resultaat wordt opgeslagen in wat we een ***resultset*** noemen**.**

Syntax:

```
SELECT *
FROM table_name
```

Let op: de SQL taal is niet hoofdlettergevoelig (case sensitive). SELECT is het zelfde als select. Wel is het afspraak om uitdrukkingen uit SQL taal met hoofdletters te schrijven, maar alleen voor de helderheid.

Voorbeeld van een tabel:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

We willen de inhoud van de kolommen **LastName** en **FirstName** selecteren. We gebruiken de volgende **SELECT** instructie:

```
SELECT LastName,FirstName
FROM Persons
```

Het resultaat ziet er zo uit:

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Nu willen we alle kolommen van de tabel **Persons** selecteren. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
```

Tip: **de asterisk (*) is een snelle manier om alle kolommen te selecteren! Het maakt een query evenwel niet sneller als we alle kolommen kiezen, zie paragraaf 16.3.**

Het resultaat ziet er zo uit:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Let op: **als er spaties of min-tekens staan in veldnamen moeten we deze namen markeren met bijvooorbeeld aanhalingstekens (of accent grave) , werkt in MySQL en MS Access; in databases als MS Access of SQL Server kunnen we ook vierkante haken ([en]) gebruiken**

3.4 TOP

De **TOP** component wordt gebruikt om het maximum aantal op te vragen records aan te even. Voor de duidelijkheid: **TOP** geeft de bovenste niet de hoogste.

Deze **TOP** component kan erg bruikbaar zijn bij grote tabellen met duizenden records omdat grote aantallen impact kan hebben op de performance.

Let op: **niet alle database systemen ondersteunen de TOP component.**

Syntax:

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

SQL SELECT TOP equivalent in MySQL en Oracle

MySQL Syntax:

```
SELECT column_name(s)
FROM table_name
LIMIT number
```

Voorbeeld:

```
SELECT *
FROM Persons
LIMIT 5
```

Oracle Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE ROWNUM <= number
```

Voorbeeld

```
SELECT *
FROM Persons
WHERE ROWNUM <=5
```

TOP PERCENT wordt niet ondersteund door Oracle.

SQL **TOP** Voorbeeld

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger

Nu willen we alleen de eerste twee records selecteren. We gebruiken de volgende **SELECT** instructie:

```
SELECT TOP 2 *
FROM Persons
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

SQL **TOP PERCENT** Voorbeeld

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger

Nu willen we alleen 50% van de records in bovenstaande tabel. We gebruiken de volgende **SELECT** instructie:

```
SELECT TOP 50 PERCENT *
FROM Persons
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Let op: het opgegeven aantal bij TOP kan in SQL Server ook via een geneste subquery (zie paragraaf 8.7) worden opgegeven.

```
SELECT TOP
  (
  SELECT COUNT(*)/10
  FROM persons
  ) *
FROM Persons
```

3.5 ORDER BY

ORDER BY wordt gebruikt om de result-set te sorteren op een specifieke kolom of expressie. Standaard sorteert **ORDER BY** oplopend (**ASC**). Als we aflopend willen sorteren, dan gebruiken we **DESC**.

Let op: diverse databases staan niet toe dat er op expressies gesorteerd wordt. MS Access en SQL Server wel, SOLID bijvoorbeeld niet.

Syntax:

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC
```

Voorbeeld

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger

Nu willen we alle personen uit bovenstaande tabel selecteren, gesorteerd op achternaam. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
ORDER BY LastName
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
4	Nilsen	Tom	Vingvn 23	Stavanger
3	Pettersen	Kari	Storgt 20	Stavanger
2	Svendson	Tove	Borgvn 23	Sandnes

ORDER BY DESC Voorbeeld

Dezelfde selectie maar dan aflopend. We gebruiken de volgende SELECT instructie:

```
SELECT *
FROM Persons
ORDER BY LastName DESC
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger
1	Hansen	Ola	Timoteivn 10	Sandnes

Let op: de kolom waarop gesorteerd wordt hoeft geen deel uit te maken van de SELECT

Verkorte verwijzing bij ORDER BY
We kunnen ook de verkorte verwijzing gebruiken: sorteren op het rangnummer van de kolom.

Syntax

```
SELECT column_name(s)
FROM table_name
ORDER BY column_number(s) ASC|DESC
```

Voorbeeld

```
SELECT LastName, FirstName
FROM table_name
ORDER BY 1 ASC , 2 DESC
```

We kunnen bij **ORDER BY** ook kolomnamen en cijfers combineren.

Let op: het rangnummer waarop gesorteerd wordt, kan alleen verwijzen naar kolommen die deel uit maken van de selectie

3.6 Alias

We kunnen een tabel of kolom een andere naam geven door een alias te gebruiken. Dit kan nuttig zijn bij lange, complexe tabel- of kolomnamen. Een alias mag alles zijn maar is meestal kort.

Syntax voor tabellen:

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

Syntax voor kolommen:

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

We mogen ook de naam van een kolom als alias voor een andere kolom gebruiken mits deze kolomnaam niet in de **SELECT** gebruikt worden.

Voorbeeld

Stel dat we een tabel **Persons** hebben en een andere die **Product_Orders** heet. We geven de tabellen de aliassen van **p** respectievelijk **po**.

We willen nu alle orders waar Ola Hansen verantwoordelijk voor is. We gebruiken de volgende **SELECT** instructie:

```
SELECT po.OrderID, p.LastName, p.FirstName
FROM Persons AS p, Product_Orders AS po
WHERE p.LastName='Hansen'
AND p.FirstName='Ola'
```

Dezelfde **SELECT** instructie zonder aliassen:

```
SELECT Product_Orders.OrderID, Persons.LastName, Persons.FirstName
FROM Persons, Product_Orders
WHERE Persons.LastName='Hansen'
AND Persons.FirstName='Ola'
```

Zoals we zien in deze twee **SELECT** instructies, aliassen maken queries gemakkelijker te schrijven en te lezen.

Let op: het woord AS voor de alias mag bijvoorbeeld in Access en SQL Server ook weggelaten worden.

3.7 DISTINCT

In een tabel kunnen sommige kolommen dubbele waarden bevatten. Dat is geen probleem al zullen we soms alleen de verschillende waarden willen zien. **DISTINCT** kan hiervoor gebruikt worden. Het tegenovergestelde van **DISTINCT** is **ALL** maar dat kan weggelaten worden.

Syntax:

```
SELECT DISTINCT column_name(s)
FROM table_name
```

Voorbeeld:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

We willen alleen de unieke waarden uit de kolom **City** zien. We gebruiken de volgende **SELECT** instructie:

```
SELECT DISTINCT City
FROM Persons
```

Het resultaat zal er zo uit zien:

City
Sandnes
Stavanger

Let op: gebruik van DISTINCT zal SQL queries trager maken; zie paragraaf 16.4.

3.8 Opgaven

1. Stel een lijst samen van alle werknemers van Noordenwind. De lijst hoeft alleen de voornamen en de toestelnummers te bevatten. Zorg er voor dat de voornamen op alfabet gerangschikt staan.
2. Maak een lijst van alle werknemers met voor- en achternaam en telefoonnummer thuis. Sorteer de lijst op achternaam.
3. Maak een lijst van alle plaatsen waar klanten zitten. Elle plaats mag maar ййn keer voorkomen. Zet de plaatsen op volgorde. (**DISTINCT** doet dat automatisch)
4. Welke klanten kent Noordenwind? Maakt een lijst met bedrijfsnaam en klantennummers.

4 Concateneren en casten

4.1 Concateneren

Met concateneren bedoelen we het samenvoegen van verschillende gegevens uit meer kolommen in ййn kolom.

Voorbeeld

```
SELECT Achternaam + ','  + Voornaam + ', '  + tussenvoegsel
FROM namen
```

Verschillende databases bieden daar verschillende functionaliteit voor.

MS Access	+ of &
SQL Server	+
ORACLE	CONCAT()
MySQL	CONCAT()

Let op: concateneren kan zowel bij SELECT als bij WHERE, GROUP BY, HAVING en ORDER BY worden toegepast

4.2 CASTING

De **CAST**() functie wordt gebruikt om gegevens van het ene datatype in het andere om te zetten. Voor de betekenis en een overzicht van datatypes in verschillende SQL versies, zie hoofdstuk 18 Datatypes.

Syntax

```
SELECT CAST(column_name AS operand)
FROM table_name
```

Voorbeeld

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen

We willen **Customer** en **OrderDate** samenvoegen. We gebruiken de volgende **SELECT** instructie:

```
SELECT Customer + ' heeft besteld op ' +CAST(OrderDate AS CHAR) AS besteld
FROM Orders
```

Het resultaat zal er zo uit zien:

Besteld
Hansen heeft besteld op 2008/11/12
Nilsen heeft besteld op 2008/10/23

Let op: casten kan zowel bij SELECT als bij GROUP BY en ORDER BY worden toegepast

Let op: MS Access kent de CAST functie niet maar wel vergelijkbare functies als CDBL, CDATE en dergelijke; naast CAST worden ook wel CONVERT of afleidingen daarvan gebruikt.

5 Voorwaarden

5.1 WHERE

De **WHERE** component wordt gebruikt om waarden uit de database te filteren die voldoen aan een specifiek criterium.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator value
```

Voorbeeld:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

We willen de personen selecteren die in de stad Sandnes wonen. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
WHERE City='Sandnes'
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

SQL gebruikt enkelvoudige aanhalingstekens rond tekstwaarden (de meeste database systemen zullen ook dubbele accepteren).

Numerieke waarden moeten niet tussen aanhalingstekens gezet worden. In MS Access moeten datums tussen # worden gezet; bij andere databases worden datums als strings gezien.

Voor tekstwaarden

Dit is correct:

```
SELECT *
FROM Persons
WHERE FirstName='Tove'
```

Dit is fout:

```
SELECT *
FROM Persons
WHERE FirstName=Tove
```

Voor numerieke waarden:

Dit is correct:

```
SELECT *
FROM Persons
WHERE Year = 1965
```

Dit is fout:

```
SELECT *
FROM Persons
WHERE Year = '1965'
```

5.2 OPERATOREN

Operatoren die zijn toegestaan in de **WHERE** Component

Operator	Beschrijving
=	Gelijk aan
<>	Ongelijk aan
>	Groter dan
<	Kleiner dan
>=	Groter dan of gelijk aan
<=	Kleiner dan of gelijk aan
BETWEEN	Tussen twee voorwaarden
LIKE	Zoeken op een patroon
IN	Als we de exacte waarde weten die we terug willen hebben

Let op: in sommige versies van SQL kan de <> operator worden geschreven als !=; MS Access accepteert dat bijvoorbeeld niet; SQL Server wel.

5.3 NULL

Wanneer een veld in een kolom niet is ingevuld, is de inhoud onbepaald, in SQL termen: NULL. NULL is dus niet hetzelfde als nul of 0.

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IS NULL / NOT NULL
```

Voorbeeld

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson		Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

We willen de personen selecteren zonder voornaam. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
WHERE FirstName IS NULL
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
2	Svendson		Borgvn 23	Sandnes

5.4 Opgaven

5. Welke klanten zitten in Londen? Maak een lijst met bedrijfsnamen en sorteer deze oplopend

6. Welke werknemers zijn na 1 januari 1993 in dienst gekomen? Maak een lijst met achternaam, functie en datum in dienst
7. Welke orders hebben meer dan € 300,- aan vrachtkosten? Maak een lijst met ordernummer, orderdatum, vrachtkosten en klantnummer. Zet het hoogste bedrag bovenaan
8. Maak een lijst van alle bedrijven en de telefoonnummers van klanten in Rio de Janeiro
9. Van welke producten heeft Noordenwind er meer dan 100 op voorraad?
10. Maak een overzicht van alle leveranciers uit Frankrijk
11. Zoek de gegevens van de klant die hoort bij ordernummer 10261; we hebben dan 2 queries nodig

5.5 AND en OR

De **AND** operator selecteert een record als zowel de eerste als de tweede voorwaarde waar is.

De **OR** operator selecteert een record als of de eerste of de tweede voorwaarde waar is.

SQL kent geen **XOR** of exclusive **OR**. Wel gebruikt SQL Server daar voor het ^ teken. In SQL Server 2012 kan de **XOR** wel gebruikt worden. **XOR** of ^ geeft True als een van beide waarden True en de andere False is.

Voorbeeld van ^:

```
WHERE
   (CASE WHEN ColumnA IS NOT NULL) THEN 1 ELSE 0 END) ^
   (CASE WHEN CoulmnB IS NOT NULL) THEN 1 ELSE 0 END) = 1
```

We zouden de **XOR** ook anders kunnen construeren:±

```
WHERE (ColumnA = 0 AND columnB = 1) OR (ColumnA = 1 AND ColumnB = 0)
```

Voorbeeld AND operator

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

We willen iedereen selecteren met de voornaam Tove **AND** de achternaam Svendson. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
WHERE FirstName='Tove'
AND LastName='Svendson'
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes

Voorbeeld **OR** Operator

Nu willen we iedereen selecteren met de voornaam Tove **OR** de voornaam Ola. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
WHERE FirstName='Tove'
OR FirstName='Ola'
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

AND & OR gecombineerd
Je kunt **AND** en **OR** ook combineren (gebruik haakjes voor complexe constructies; haakjes kunnen ook de evaluatievolgorde beпnvloeden).

We willen nu de personen selecteren met de achternaam Svendson **AND** de voornaam gelijk aan Tove **OR** Ola. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
WHERE LastName='Svendson'
AND (FirstName='Tove' OR FirstName='Ola')
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes

Let op: voor de performance kunnen we beter IN dan een serie OR's gebruiken; zie 16.1.

5.6 Opgaven

12. Maak een overzicht van alle orders uit de tweede helft van februari 1998; dus van na 15 februari
13. Maak een lijst van alle werknemers die tussen 1960 en 1965 geboren zijn
14. Maak een lijst van klanten uit de kleinere landen Belgiл, Denemarken en Oostenrijk
15. Maak een lijst van alle producten met een voorraad tussen de 50 en de 100 stuks
16. Maak een lijst van alle dranken en fruit met een prijs tussen de 10 en 20 euro; maak gebruik van de tabel Categorieлn om de categorienummers te bepalen
17. Maak een overzicht van alle producten die uit het assortiment gehaald zijn met een prijs onder de 40 euro

5.7 IN

De **IN** operator geeft ons de mogelijkheid meer waarden te specificeren in een **WHERE** component. **IN** kan ook gebruikt worden als schakel naar een zogenaamde subquery, zie paragraaf 8.1. **IN** is in feite een vervanger van **OR**.

Syntax:

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

Voorbeeld

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

We willen nu de personen selecteren met de achternaam Hansen of Pettersen.

We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
WHERE LastName IN (Hansen,Pettersen)
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

5.8 NOT

We kunnen **NOT** voor een operand of **IN** plaatsen.

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name NOT IN (value1,value2,...)
```

of

```
SELECT column_name(s)
FROM table_name
WHERE column_name NOT OPERAND value
```

Voorbeelden

```
SELECT *
FROM Persons
WHERE LastName NOT IN ('Hansen','Pettersen')
```

of

```
SELECT *
FROM Persons
WHERE LastName
NOT BETWEEN 'Hansen' AND 'Pettersen'
```

5.9 BETWEEN

De **BETWEEN** operator wordt gebruikt in een **WHERE** component om een bereik van gegevens te selecteren die tussen twee waarden liggen. De waarden kunnen numeriek, tekst of datums zijn.

Syntax:

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

Voorbeeld:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Nu willen we de mensen selecteren met achternamen tussen Hansen en Pettersen. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
WHERE LastName
BETWEEN 'Hansen' AND 'Pettersen'
```

Het resultaat ziet er zo uit:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes

Let op: **de verschillende databases gaan verschillend om met de BETWEEN operator. In sommige databases zullen de mensen met de achternamen Hansen or Pettersen weggelaten worden omdat de BETWEEN operator alleen de tussenliggende velden selecteert en de grenzen weglaat. Andere databases zullen deze grenzen wel meenemen. En er zijn tot slot ook databases die de eerste wel meenemen maar de laatste weglaten. Dus: check hoe de database omgaat met de BETWEEN operator!**

5.10 LIKE

De **LIKE** operator wordt gebruikt om in een kolom naar een specifiek patroon te zoeken.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern
```

Voorbeeld

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

We willen nu iedereen selecteren die woont in een stad die begint met een s. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
WHERE City LIKE 's%'
```

Het % teken kunnen we gebruiken om jokers te definiλren (ontbrekende letters in het patroon) zowel voor als na het patroon.

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Nu willen we iedereen selecteren die woont in een stad die eindigt op een s.

We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
WHERE City LIKE '%s'
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Ten slotte willen we de mensen selecteren die in een stad wonen die tav. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
WHERE City LIKE '%tav%'
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
3	Pettersen	Kari	Storgt 20	Stavanger

Het is ook mogelijk mensen te selecteren die **NOT** in een stad wonen die tav bevat, door **NOT** te gebruiken. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
WHERE City NOT LIKE '%tav%'
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

5.11 Jokers

SQL jokers kunnen ййn of meer karakters vervangen bij het zoeken in een database. Jokers moeten in combinatie met de **LIKE** operator gebruikt worden.

In SQL kunnen de volgende jokers gebruikt worden:

Joker	Beschrijving
%	Als vervanging voor 0 of meer karakters
_	Als vervanging voor precies ййn karakter
[charlist]	Elk karakter in een charlist
[charlist-charlist]	Van elk karakter tot en met elk karakter in een charlist
[^charlist] of [!charlist]	Elk karakter niet in een charlist

Let op: in MS Access gebruikt de asterisk (*) in plaats van het procenteken (%) en het vraagteken (?) in plaats van het liggend streepje (_).

Voorbeeld

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

We willen alle personen selecteren die in een stad wonen die begint met sa. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
WHERE City LIKE 'sa%'
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Vervolgens willen de personen selecteren die in een stad wonen die het patroon nes. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
WHERE City LIKE '%nes%'
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

We willen nu alle personen selecteren die een voornaam hebben die met een willekeurige letter begint gevolgd door la. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
WHERE FirstName LIKE '_la'
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes

We willen alle personen selecteren van wie de achternaam begint met een S, gevolgd door een willekeurige letter en dan gevolgd door end, gevolgd door een willekeurige letter, gevolgd door on. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
WHERE LastName LIKE 'S_end_on'
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes

[charlist]
Nu willen we de mensen selecteren van wie de achternaam begint met een b, s of p. We gebruiken de volgende SELECT instructie:

```
SELECT *
FROM Persons
WHERE LastName LIKE '[bsp]%'
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Nu de personen van wie de achternaam niet begint met een b, s of p. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
FROM Persons
WHERE LastName LIKE '[!bsp]%'
```

Het resultaat zal er zo uit zien:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes

Let op: **je mag ook meer Charlists achterelkaar gebruiken: [abc][abc] etc. ; alle combinaties van abc met abc zijn dan mogelijk. Ook [a-d] is mogelijk: we krijgen dan alles tussen a en d.**

Let op: **wat betreft de strings kunnen databases wel hoofdlettergevoelig zijn. Zo maken MySQL en Solid verschil tussen LIKE 'S%' en LIKE 's%'.**

5.12 CASE

De operator **CASE** is eigenlijk een volledig expressie waarmee in alle componenten van de SQL opdracht, behalve **FROM**, velden voorwaardelijk kunnen worden benoemd.

Syntax

```
SELECT column_name(s),
CASE
    WHEN when_expression THEN result_expression
    [ELSE else_result_expression]
END AS column_name
FROM table_name
WHERE
CASE
    WHEN when_expression THEN result_expression
    [ELSE else_result_expression]
END AS colum_name operator value
```

Voorbeeld

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

We willen de **OrderPrice** indelen in categorieʌn van 1000. We gebruiken de volgende **SELECT** instructie:

```
SELECT Customer,
CASE
    WHEN OrderPrice BETWEEN 0 AND 1000 THEN 'A'
    WHEN OrderPrice BETWEEN 1001 AND 2000 THEN 'B'
    ELSE 'C'
END AS category
FROM Orders
```

Het resultaat ziet er zo uit:

Customer	Category
Hansen	A
Nilsen	B
Hansen	A
Hansen	A
Jensen	B
Nilsen	A

Bij de CASE constructie kan de kolomnaam ook direct achter CASE staan. Bij WHEN mogen dan alleen enkele vergelijkingen staan.

```
SELECT Customer,
CASE OrderPrice
    WHEN 10 THEN 'A'
    WHEN 20 THEN 'B'
    ELSE 'C'
END AS category
FROM Orders
```

Let op: **bij WHEN hoeven we niet steeds naar hetzelfde veld of dezelfde expressie te verwijzen; bij overlappende voorwaarden geldt de eerste.**

Let op: **MS Access kent de CASE operator niet; hier moeten we de immediate if functie IIF() of de SWITCH functie gebruiken.**

De **SELECT** instructie komt er in MS Access met **SWITCH**() dan zo uit te zien:

```
SELECT Customer,
SWITCH
(
  OrderPrice >=0 AND OrderPrice<=1000, 'A',
  OrderPrice >1000 AND OrderPrice<=2000, 'B',
  OrderPrice >2000, 'C'
)
FROM Orders
```

5.13 Opgaven

18. Welke producten zijn in flessen verpakt?
19. Maak een overzicht van de klanten in grote landen als Canada, Brazilia en de Ver. Staten.
20. Welke klanten hebben in de periode januari – maart 1997 orders geplaatst; geef een lijst met unieke klantnummers?
21. Maak een lijst van de werknemers met voor- en achternaam samengevoegd, en de functie van der werknemers.
22. Welke producten zijn in dozen verpakt en kosten tussen de 10 en 20 euro?
23. Geef de producten tussen de 0 en de 10 euro het label D, tussen 10 en 20 het label C, tussen 20 en 30 het label B en alle andere producten het label A (gebruik CASE WHEN in SQL Server).

6 Functies

6.1 Rekenkundige functies

Zowel in **SELECT** als bij **WHERE** kunnen we gebruik maken van te berekenen waarden. We gebruiken daarvoor:

*	Vermenigvuldigen
/	Delen
+	Optellen
-	Aftrekken
%	Modulo delen

Let op: voor modulo delen gebruikt MS Access mod; bovendien kent MS Access \ voor Integer delen

Bij de modulo deling blijft de rest over: 12 % 5 geeft dus 2. Bij de integer deling is er geen rest: 13 \ 5 geeft dus 2. Bijvoorbeeld SQL Server kent de \ niet daar zouden we het dus op moeten lossen met alleen de modulo: (12 - 12 % 5) / 5

6.2 Statistische functies

SUM()
De **SUM**() functie retourneert de totale som van een kolom met numerieke waarden.

Syntax:

```
SELECT SUM(column_name)
FROM table_name
```

Voorbeeld

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

We willen de som berekenen van het **OrderPrice** veld. We gebruiken de volgende **SELECT** instructie:

```
SELECT SUM(OrderPrice) AS OrderTotal
FROM Orders
```

Het resultaat zal er zo uit zien:

OrderTotal
5700

AVG()
De **AVG**() functie retourneert het gemiddelde van een kolom met numerieke waarden; de velden met een NULL waarde tellen daarbij niet mee.

Syntax

```
SELECT AVG(column_name)
FROM table_name
```

Voorbeeld

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

We willen het gemiddelde berekenen van het veld **OrderPrice**. We gebruiken de volgende **SELECT** instructie:

```
SELECT AVG(OrderPrice) AS OrderAverage
FROM Orders
```

Het resultaat zal er zo uit zien:

OrderAverage
950

Nu willen we de klanten selecteren die een **OrderPrice** hebben die hoger is dan het gemiddelde. We gebruiken de volgende **SELECT** instructie met een zogenaamde subquery (zie paragraaf 8.1 en verder):

```
SELECT Customer
FROM Orders
WHERE OrderPrice >
(
SELECT AVG(OrderPrice) FROM Orders
)
```

Het resultaat zal er zo uit zien:

Customer
Hansen
Nilsen
Jensen

COUNT()
De **COUNT**(*) functie retourneert het aantal rijen dat voldoet aan specifieke criteria.

Syntax

De **COUNT**(column_name) functie retourneert het aantal waarden (**NULL** waarden worden niet meegeteld) van de gespecificeerde kolom:

```
SELECT COUNT(column_name)
FROM table_name
```

Syntax **COUNT**(*):

```
SELECT COUNT(*)
FROM table_name
```

Syntax SQL **COUNT**(**DISTINCT** column_name):
Deze functie retourneert het aantal verschillende waarden in de gespecificeerde kolom.

```
SELECT COUNT(DISTINCT column_name)
FROM table_name
```

Let op: **COUNT(DISTINCT) werkt in ORACLE en Microsoft SQL Server, maar niet in Microsoft Access. Daar moeten we een alternatief gebruiken met een subquery**

```
SELECT COUNT(*)
(
SELECT DISTINCT column_name
FROM table_name
)
```

Voorbeeld **COUNT**(column_name)

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Nu willen we het aantal orders tellen van **Customer** Nilsen. We gebruiken de volgende **SELECT** instructie:

```
SELECT COUNT(Customer) AS CustomerNilsen
FROM Orders
WHERE Customer= 'Nilsen'
```

Het resultaat zal 2 zijn:

CustomerNilsen
2

Voorbeeld SQL **COUNT**(*)

Als we de **WHERE** component weglaten:

```
SELECT COUNT(*) AS NumberOfOrders
FROM Orders
```

Het resultaat zal er zo uit zien:

NumberOfOrders
6

Gelijk aan het totale aantal rijen in de tabel.

Voorbeeld SQL **COUNT**(**DISTINCT** column_name)

We willen het aantal unieke klanten in de **Orders** tabel. We gebruiken de volgende **SELECT** instructie:

```
SELECT COUNT(DISTINCT Customer) AS NumberOfCustomers
FROM Orders
```

Het resultaat zal er zo uit zien:

NumberOfCustomers
3

Gelijk aan het aantal unieke klanten (Hansen, Nilsen, and Jensen) in de **Orders** tabel.

We kunnen de uitkomst van twee COUNT's ook op elkaar delen:

```
SELECT COUNT(provincie) / CAST(COUNT(*) AS FLOAT)*100
AS deling
FROM klanten
```

De uitkomst van een COUNT is een integer of een geheel getal. We moeten daarom hier de tweede uitkomst CASTEN naar een FLOAT.

De MAX() Functie
De **MAX**() functie retourneert de hoogste waarde uit de geselecteerde kolom.

Syntax:

```
SELECT MAX(column_name)
FROM table_name
```

Voorbeeld

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

We willen nu de hoogste waarde uit de kolom OrderPrice. We gebruiken de volgende SELECT instructie:

```
SELECT MAX(OrderPrice) AS LargestOrderPrice
FROM Orders
```

Het resultaat zal er zo uit zien:

LargestOrderPrice
2000

De MIN() Functie
De **MIN**() functie retourneert de laagste waarde uit de geselecteerde kolom.

Syntax:

```
SELECT MIN(column_name)
FROM table_name
```

Voorbeeld

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

We willen de laagste waarde uit de kolom **OrderPrice**. We gebruiken de volgende **SELECT** instructie:

```
SELECT MIN(OrderPrice) AS SmallestOrderPrice
FROM Orders
```

Het resultaat zal er zo uit zien:

SmallestOrderPrice
100

Let op: **naast genoemde statische functies komen ook STDEV en VARIANCE voor al ondersteunen niet alle databases deze functies; in MS Access gebruikt VAR in plaats van VARIANCE en het kent daarnaast ook nog FIRST en LAST**

6.3 Opgaven

24. Hoeveel orders zijn er uit de eerste helft van 1997?
25. Wat is het totale bedrag aan orders van de ordernummers tussen 10248 en 10290? We moeten dan eerst hoeveelheid en prijs per eenheid uit de tabel **Orderinformatie** berekenen en dat totaliseren.
26. Wat is het gemiddelde orderregelbedrag van de orders met ordernummers tussen 10248 en 10290?
27. In hoeveel verschillende steden heeft Noordenwind klanten? In MS Acces moet deze met een subquery worden opgelost.
28. Wat is het grootste orderregelbedrag? En het kleinste?

6.4 Scalaire functies

SQL kent een groot aantal functies waarmee we gegevens kunnen wijzigen, omzetten of beperken. Vervelend genoeg hebben deze functies in de verschillende databasesystemen vaak andere benamingen. We geven een voorbeeld. Om de huidige datum op te roepen worden in de diverse dialecten sterk verschillende benamingen gebruikt:

sql functie	access	sql server	mysql	oracle	solid	db2
currentdate	date	getdate / current_timestamp	current_date	sysdate	curdate	curdate

Let op: **voor een overzicht van een groot aantal scalaire functies en hun benamingen in diverse SQL dialecten, zie hoofdstuk 17.**

Syntax

```
SELECT YEAR(column_name), CURRENTDATE()
FROM table_name
```

Voorbeeld

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

We willen uit de **Orderdate** het jaartal halen. We doen dan met de volgende **SELECT** instructie:

```
SELECT YEAR(Orderdate) AS Jaar
FROM orders
```

Het resultaat ziet er dan zo uit:

Jaar
2008
2008
2008
2008
2008
2008

6.5 Opgaven

29. Bereken het verschil in jaren tussen de datum in dienst en het huidige jaar.
30. Maak een lijst van de dag en de maand waarop de werknemers jarig zijn. Sorteer de lijst op maand en daarbinnen op dag.
31. Maak een lijst van de klanten waarbij de bedrijfsnamen en plaatsnamen in hoofdletters worden weergegeven.
32. Maak een lijst van de ordernummers en de weeknummers waarin ze besteld zijn. Gebruik voor Access en SQL Server de datepart functie (met deze functie kunnen we het gewenste interval uit een datum filteren, bijv **datepart**(ww,datum)).

7 Groeperen

7.1 GROUP BY

De **GROUP BY** instructie wordt gebruikt samen met de aggregatie functies (**SUM**, **AVG**, **MAX**, **COUNT** etc.) om de result-set op ййn of meer kolommen of expressies te groeperen.

Let op: **diverse databases staan niet toe dat er op expressies gegroepeerd wordt. MS Access en SQL Server wel, SOLID bijvoorbeeld niet.**

Syntax:

```
SELECT column_name, aggregate_functie(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
```

Voorbeeld

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

We willen nu de totale ordersom per customer. We moeten dan **GROUP BY** gebruiken om de customers te groeperen. We gebruiken de volgende **SELECT** instructie:

```
SELECT Customer,SUM(OrderPrice)
FROM Orders
GROUP BY Customer
```

Het resultaat zal er zo uit zien:

Customer	SUM(OrderPrice)
Hansen	2000
Nilsen	1700
Jensen	2000

Laten we eens zien wat er gebeurt als we **GROUP BY** weglaten:

```
SELECT Customer,SUM(OrderPrice)
FROM Orders
```

Het resultaat zal er zo uit zien:

Customer	SUM(OrderPrice)
Hansen	5700
Nilsen	5700
Hansen	5700
Hansen	5700
Jensen	5700
Nilsen	5700

Deze result-set hierboven is niet wat we willen. MS Access zou hier trouwens een foutmelding geven.

Verklaring: in de **SELECT** instructie zijn twee kolommen opgegeven (**Customer** en **SUM**(**OrderPrice**). De **SUM**(**OrderPrice**) retourneert een enkele waarde (de totale som van de kolom **OrderPrice**), terwijl de kolom Customer 6 waarden retourneert 6 (een waarde voor elke rij in de tabel **Orders**). Daarom krijgen we een incorrect resultaat.

GROUP BY op meer kolommen
We kunnen ook groeperen op meer kolommen, zoals hier:

```
SELECT Customer,OrderDate,SUM(OrderPrice)
FROM Orders
GROUP BY Customer,OrderDate
```

7.2 HAVING

De **HAVING** component is toegevoegd aan SQL omdat **WHERE** niet gebruikt kan worden bij aggregatie functies als **SUM**, **AVG** etc.. Het verschil is dat de **WHERE** component betrekking heeft op het tussenresultaat van de **FROM** component en **HAVING** op het gegroepeerde tussenresultaat van de **GROUP BY** component. Het effect is wel hetzelfde, ook in de **HAVING** component worden rijen met een conditie geselecteerd.

Syntax:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_functie(column_name) operator value
```

Voorbeeld

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

We willen alle **customers** met een ordertotaal van minder dan 2000. We gebruiken de volgende **SELECT** instructie:

```
SELECT Customer,SUM(OrderPrice)
FROM Orders
GROUP BY Customer
HAVING SUM(OrderPrice) < 2000
```

Het resultaat zal er zo uit zien:

Customer	SUM(OrderPrice)
Nilsen	1700

Nu willen we weten of de customers Hansen of Jensen een ordertotaal hebben van meer dan 1500. We voegen een **WHERE** component toe aan de **SELECT** instructie:

```
SELECT Customer,SUM(OrderPrice)
FROM Orders
WHERE Customer='Hansen'
OR Customer='Jensen'
GROUP BY Customer
HAVING SUM(OrderPrice) > 1500
```

Het resultaat zal er zo uit zien:

Customer	SUM(OrderPrice)
Hansen	2000
Jensen	2000

HAVING kan ook zonder **GROUP BY** gebruikt worden en heeft dan betrekking op het generaal totaal bij **SELECT**. We geven een voorbeeld met **EXIST** (zie paragraaf 8.4) uit de **NOORDENWIND** database

```
SELECT *
FROM producten AS p
WHERE EXISTS
(
  SELECT *
  FROM producten AS pr
  HAVING MAX(pr.[prijs per eenheid]) = p.[prijs per eenheid]
)
```

Dit voorbeeld geeft het product met de hoogste prijs uit de tabel producten.

7.3 Opgaven

33. We willen een overzicht van het aantal orders dat in 1997 per klantnummer geplaatst is.
34. We willen een lijst met het aantal klanten per land.
35. Maak een lijst van de verkochte hoeveelheid per productnummer.
36. Geef een overzicht van de productnummers waarbij er voor meer dan 50.000 euro verkocht is.
37. Geef bij de vorige lijst ook het totale bedrag minus de korting voor deze producten ([hoeveelheid]*[prijs per eenheid])*(1-[korting]).

8 Meer tabellen

8.1 Subqueries: regels

Een **subquery** is een tabel-expressie die voorkomt binnen een andere tabel-expressie. Alternatieve namen zijn **subselect** en **innerselect**. De definitie van een subquery wijkt wat af van een normale **SELECT** instructie. Er zijn drie verschillen:

- Als de subquery zonder de **EXISTS** operator wordt gebruikt, dan mag in de **SELECT** component maar ййn kolom of expressie gespecificeerd worden, om dat deze anders een tabel oplevert met meer kolommen. Bij de voorwaarden met een subquery wordt het resultaat van de subquery vergeleken met ййn enkele waarde. De asterisk (*) mag dus alleen bij **EXISTS** gebruikt worden.
- De **SELECT** component mag geen **DISTINCT** bevatten. **DISTINCT** is ook onnodig want de betekenis van de waarden van een subquery verandert niet als de dubbele worden weggelaten
- **ORDER BY** is ook niet toegestaan om dezelfde reden als **DISTINCT**

8.2 Subqueries met IN

Met een subquery kunnen we gegevens uit ййn tabel tonen maar wel zo dat gegevens uit andere tabellen in de voorwaarden worden gebruikt. Hier komen we zowel gecorreleerde als niet gecorreleerde subqueries tegen

Syntax

```
SELECT column_name(s)
FROM table_name1
WHERE column_name IN
(
  SELECT column_name
  FROM table_name2
  [WHERE table_name1.column_name = table_name2.column_name]
)
```

Voorbeeld van een niet gecorreleerde subquery met **IN**:

De tabel **Persons**

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

De tabel **Orders**

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

We willen nu de P_Id zien uit de tabel **Orders** die niet voorkomen in de tabel **Persons**. We gebruiken de volgende **SELECT** instructie:

```
SELECT P_Id
FROM Orders
WHERE P_Id NOT IN
(
  SELECT P_Id
  FROM Persons
)
```

Het resultaat zal er zo uit zien:

P_Id
15

Let op: **de SELECT in de subquery mag maar ййn kolom bevatten; de verbindende kolom hoeft niet dezelfde te zijn, als deze maar wel gelijksoortige gegevens bevat zoals datums**

Voorbeeld van een gecorreleerde **IN** subquery:

```
SELECT P_Id
FROM Orders AS O
WHERE P_Id NOT IN
(
  SELECT P_Id
  FROM Persons
  WHERE city <> O.city
)
```

8.3 Subqueries als expressie

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name [= / > /< / <> />= / <=]
(
  SELECT column_name
  FROM table_name
)
```

8.4 Subqueries met EXISTS

We krijgen bij **EXISTS** te maken met een zogenaamde gecorreleerde subquery: De **SELECT** in de subquery refereert aan de hoofdquery.

Syntax

```
SELECT column_name(s)
FROM table_name1
WHERE EXISTS
(
  SELECT *
  FROM table_name2
  WHERE table_name1.column_name = table_name2.column_name
)
```

Voorbeeld

De tabel **Persons**

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

De tabel **Orders**

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

We willen nu de P_Id zien uit de tabel **Orders** die niet voorkomen in de tabel **Persons**. We gebruiken de volgende **SELECT** instructie:

```
SELECT P_Id
FROM Orders AS o
WHERE NOT EXISTS
(
  SELECT *
  FROM Persons AS p
  WHERE o.P_Id = p.P_Id
)
```

Het resultaat zal er zo uit zien:

P_Id
15

Dit betekent dus dat er voor elke P_Id in de tabel **Orders** wordt gekeken of dit in de subquery geen rijen oplevert.

Nog een voorbeeld uit de Noordenwind database om de hoogst prijs per categorie te berekenen:

```
SELECT p1.Categorienummer, p1.[Prijs per eenheid]
FROM Producten AS p1
WHERE EXISTS
(
  SELECT *
  FROM producten AS p2
  WHERE p1.Categorienummer = p2.Categorienummer
  HAVING MAX(p2.[prijs per eenheid]) = p1.[Prijs per eenheid]
)
ORDER BY 1,2
```

Let op: de EXIST operator wordt gebruikt om snelheidswinst te boeken, zie paragraaf 16.11.

8.5 Subqueries met ALL en ANY

Hier komen we zowel gecorreleerde als niet gecorreleerde subqueries tegen.

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name expressie ANY / ALL
(
  SELECT column_name
  FROM table_name
  [WHERE table_name1.column_name = table_name2.column_name]
)
```

Voorbeeld niet gecorreleerde **ANY**

De tabel **Orders**

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

We willen alle orders die een hoger **OrderPrice** bedrag hebben dan een bedrag van alle orders. De **SELECT** instructie is als volgt

```
SELECT *
FROM Orders
WHERE OrderPrice  >  ANY
(
  SELECT OrderPrice
  FROM Orders
)
```

De laagste valt er dan uit, want die is niet groter dan ййп van de andere bedragen. Het resultaat ziet er zo uit:

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen

Voorbeeld gecorreleerde **ANY**

We willen alle orders die een hoger **OrderPrice** bedrag hebben dan een bedrag van alle orders voor dezelfde klant. De **SELECT** instructie is als volgt

```
SELECT *
FROM Orders AS O
WHERE OrderPrice  >  ANY
(
  SELECT OrderPrice
  FROM Orders
  WHERE O.CUST_ID = CUST_ID
)
```

Voorbeeld **ALL**

De tabel **Orders**

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

We willen alle orders die een hoger of gelijk **OrderPrice** bedrag hebben als een bedrag van alle orders. De **SELECT** instructie is als volgt

```
SELECT *
FROM Orders
WHERE OrderPrice  >=  ALL
 (
  SELECT OrderPrice
  FROM Orders
  WHERE O.CUST_ID = CUST_ID
 )
```

Het resultaat ziet er zo uit:

O_Id	OrderDate	OrderPrice	Customer
5	2008/08/30	2000	Jensen

Voorbeeld gecorreleerde **ALL**:

We willen alle orders die een hoger of gelijk **OrderPrice** bedrag hebben als een bedrag van alle orders van de klant. De **SELECT** instructie is als volgt

```
SELECT *
  FROM Orders AS O
  WHERE OrderPrice  >=  ALL
   (
    SELECT OrderPrice
    FROM Orders
   )
```

8.6 Opgaven

38. Geef een lijst van klanten die nooit iets besteld hebben en dus niet in de ordertabel voorkomen.
39. Maak een lijst van alle producten die op ordernummer 10248 zijn besteld.
40. Maak een lijst van klanten die in 1997 orders hebben geplaatst met een totaal bedrag van meer dan 5000 euro.
41. Maak een lijst van alle klanten waarvan de orders zijn genoteerd door Davolio.
42. Maak een lijst van categorieʌn waarvan er producten uit het assortiment zijn genomen.
43. Wat is het goedkoopste product dat Noordenwind verkoopt?
44. Welke werknemer is het langst in dienst?
45. Wat is de grootste order qua totaal bedrag die ooit bij Noordenwind is geplaatst?
46. Welke werknemer heeft er in de eerste maand van 1997 geen enkele order genoteerd?
47. Geeft een lijst van producten die niet zijn verkocht in 1998 maar wel in een ander jaar.
48. Geef een lijst van producten waarvan er in 1997 minder dan 1000 zijn verkocht.
49. Selecteer alle gegevens van klanten die wel eens een product uit de categorie dranken hebben besteld (hoofdquery met 4 subqueries).
50. Wat is het duurste product per categorie (met **ALL** en een gecorreleerde subquery)?

8.7 Geneste subqueries in de SELECT

Subqueries kunnen ook in de **SELECT** staan op voorwaarde dat deze slechts één getal voor elke rij opleveren. Dat kan bijvoorbeeld door een statistische functie te gebruiken of door een gecorreleerde subquery.

Voorbeeld

De tabel **Orders**

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

We willen naast de **OrderPrice** ook het gemiddelde zien. De **SELECT** instructie is als volgt:

```
SELECT OrderPrice,
(
  SELECT AVG(OrderPrice)
  FROM orders
) AS GemOrderPrice
```

Het resultaat ziet er zo uit:

OrderPrice	GemOrderPrice
1000	950
1600	950
700	950
300	950
2000	950
100	950

Het volgende is een voorbeeld van een gecorreleerde subquery in de **SELECT**.

```
SELECT Categorienummer, [Prijs per eenheid],
(
  SELECT AVG([Prijs per eenheid])
  FROM Producten AS p2
  WHERE p2.Categorienummer = p1.Categorienummer
) AS Gemiddelde
FROM Producten AS p1
```

We krijgen dan de categorieën en prijzen en bijbehorende gemiddelde prijs per categorie uit de tabel producten uit de database Noordenwind.

8.8 Opgaven

51. Geef een overzicht van de producten met hun prijzen en totaal verkochte aantal.
52. Welke orderregels bevatten totaalbedragen die minder dan 10 procent van het gemiddelde bedragen? Toon de totaalbedragen en de gemiddelde bedragen.

8.9 Geneste subquery in de FROM

We kunnen een subquery ook in de **FROM** plaatsen. Daar zal deze zich verder net zoals een gewone tabel gedragen.

Syntax

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

Voorbeeld

De tabel **Persons**

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

We willen de unieke plaatsen uit de tabel **Persons** tellen. De **SELECT** instructie is als volgt

```
SELECT COUNT(*)
FROM
(
  SELECT DISTINCT City
  FROM persons
) AS Plaatsen
```

Let op: bovenstaande instructie moeten we in MS Access gebruiken. In andere databasesystemen kunnen we dit ook anders oplossen, zie paragraaf 6.2.

8.10 Joins

SQL joins worden gebruikt om data uit meer dan twee tabellen te selecteren, gebaseerd op de relatie tussen bepaalde kolommen in deze tabellen.

Tabellen in database zijn vaak aan elkaar gerelateerd met sleutels. Een primaire sleutel is een kolom of combinatie van kolommen met een unieke waarde voor elke rij. Elke primaire sleutel moet uniek zijn binnen de tabel. Het doel is om data uit verschillende tabellen te verbinden zonder dat de data in elke tabel herhaald hoeven te worden.

Tabel Persons:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Merk op dat de kolom P_Id de primaire sleutel is in de tabel **Persons**. Dit betekent dat geen twee rijen dezelfde P_Id kunnen hebben. De P_Id onderscheidt de personen van elkaar zelfs als ze dezelfde naam hebben.

Tabel **Orders**:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Merk op dat de kolom O_Id de primaire sleutel is in de tabel **Orders** en dat de kolom P_Id verwijst naar de persoon in de tabel **Persons** zonder hun namen te gebruiken. De relatie tussen de twee tabellen is de kolom P_Id.

Verschillende JOIN's
Voor we voorbeelden gaan geven, geven we eerst een overzicht van de soorten **JOIN's** en de verschillen ertussen.

(INNER) JOIN:	Retourneert rijen als er minstens een match is in beide tabellen
LEFT (OUTER) JOIN:	Retourneert alle rijen uit de linker tabel, ook als er geen matches zijn de rechter tabel
RIGHT (OUTER) JOIN:	Retourneert alle rijen uit de rechter tabel, ook als er geen matches zijn de linker tabel
FULL (OUTER) JOIN:	Retourneert rijen als er een match is in ййn van de tabellen

INNER JOIN
INNER JOIN retourneert rijen als er minstens een match is in beide tabellen.

Syntax:

```
SELECT column_name(s)
FROM table_name1
INNER JOIN table_name2
ON table_name1.column_name = table_name2.column_name
```

Let op: in veel databases kan ook USING worden gebruikt maar bijvoorbeeld niet in MS Access, SQL Server en Sybase. USING zorgt er ook voor dat de dubbele kolom automatisch verwijderd wordt als van de gekoppelde tabellen alle kolommen gekozen worden.

Let op: In plaats van het = teken mogen we ook <>, > of < gebruiken; uiteraard is het dan geen INNER JOIN meer.

Syntax:

```
SELECT column_name(s)
FROM table_name1
INNER JOIN table_name2
USING(column_name)
```

Let op: INNER JOIN is het zelfde als JOIN.

In de ouderwetse aanpak werd een dergelijke link met **WHERE** tot stand gebracht.

```
SELECT column_name(s)
FROM table_name1
WHERE table_name1.column_name = table_name2.column_name
```

Voorbeeld

De tabel Persons

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

De tabel Orders

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

We willen nu alle personen zien met orders. We gebruiken de volgende **SELECT** instructie:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
INNER JOIN Orders
ON Persons.P_Id = Orders.P_Id
ORDER BY Persons.LastName
```

Het resultaat zal er zo uit zien:

LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678

Als er rijen in **Persons** die geen matches hebben in **Orders**, dan zullen die niet getoond worden.

LEFT JOIN
LEFT JOIN retourneert alle rijen uit de linker tabel, ook als er geen matches zijn in de rechter tabel

Syntax

```
SELECT column_name(s)
FROM table_name1
LEFT JOIN table_name2
ON table_name1.column_name = table_name2.column_name
```

Let op: in sommige databases wordt LEFT JOIN, LEFT OUTER JOIN genoemd.

Voorbeeld

De tabel **Persons**:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

De tabel Orders:

O_Id	OrderNo	P_Id
1	77895	3

2	44678	3
3	22456	1
4	24562	1
5	34764	15

We willen nu alle personen en hun orders, als ze die al hebben. We gebruiken de volgende **SELECT** instructie:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
LEFT JOIN Orders
ON Persons.P_Id = Orders.P_Id
ORDER BY Persons.LastName
```

Het resultaat zal er zo uit zien:

LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
Svendson	Tove	

RIGHT JOIN
RIGHT JOIN retourneert alle rijen uit de rechter tabel, ook als er geen matches zijn in de linker tabel

Syntax

```
SELECT column_name(s)
FROM table_name1
RIGHT JOIN table_name2
ON table_name1.column_name = table_name2.column_name
```

Let op: in sommige databases zoals Access en SQL Server wordt RIGHT JOIN, RIGHT OUTER JOIN genoemd.

SQL **RIGHT JOIN** Voorbeeld

De tabel **Persons**:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

De tabel **Orders**:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

We willen nu alle orders met personen als die er al zijn. We gebruiken de volgende **SELECT** instructie:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
RIGHT JOIN Orders
ON Persons.P_Id = Orders.P_Id
ORDER BY Persons.LastName
```

Het resultaat zal er zo uit zien:

LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
		34764

FULL JOIN
De **FULL JOIN** retourneert rijen als er een match is in ййn van de tabellen.

Syntax

```
SELECT column_name(s)
FROM table_name1
FULL JOIN table_name2
ON table_name1.column_name = table_name2.column_name
```

Let op: in sommige databases zoals SQL Server wordt FULL JOIN, FULL OUTER JOIN genoemd.

Voorbeeld

De tabel **Persons**:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

De tabel **Orders**:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

We willen nu alle personen met hun orders en alle orders met bijbehorende personen. We gebruiken de volgende **SELECT** instructie:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
FULL JOIN Orders
ON Persons.P_Id = Orders.P_Id
ORDER BY Persons.LastName
```

Het resultaat zal er zo uit zien:

LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
Svendson	Tove	
		34764

Let op: in de FROM mogen we geen gecorreleerde subquery gebruiken.

8.11 Opgaven

53. Maak een lijst met het totale orderbedrag per werknemers.
54. Geef een overzicht van de producten waarvan er in 1997 meer dan 500 verkocht werden.
55. Geef een overzicht van de categorieʌn die meer dan 10 verschillende producten kennen.
56. Geef de naam en de jaaromzet van de oudste vertegenwoordiger van Noordenwind.
57. Welke vertegenwoordiger had in 1997 de hoogste omzet?

9 Extra opgaven bij de Data Query Language

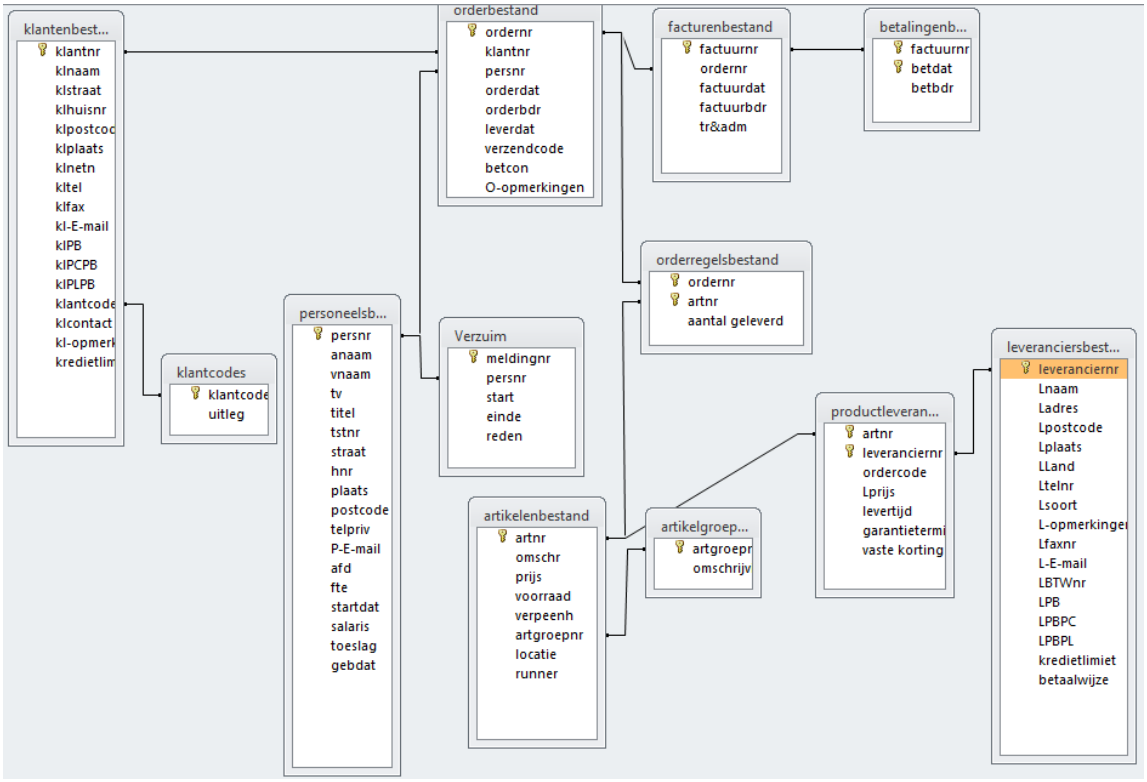
9.1 Opdrachten

Onderstaande opdrachten zijn gebaseerd op de Relate database.

58. De afdeling Promotie wil alle grote klanten een mailing sturen. Deze afdeling wil een lijst met daarop de namen en adressen van alle klanten die in 1997 een order hebben geplaatst van minimaal 8.500,-. Hoe kan deze lijst worden gemaakt?
59. Kun je een lijst maken van alle artikelen die op de grootste order van 1997 zijn besteld? (Drie subqueries)
60. De chef van de afdeling Verkoop is het niet eens met de vraag van oefening 58. Klanten die eenmalig een grote order plaatsen, zij niet altijd grote klanten. Hij zou liever een lijst hebben waarop de naam en adresgegevens zijn vermeld van alle klanten die in 1997 in totaal voor meer dan 50.000 bij Relate hebben gekocht.
61. Dick de Bruin heeft bij het skeeleren zijn pols gebroken en zal twee weken lang niet bij klanten op bezoek kunnen gaan. Martin Cohen valt voor hem in. Hij wil graag een lijst met alle klanten van Dick waarop naam, adres en telefoonnummer staan vermeld. Rangschik de klanten alfabetisch.
62. De indeling van het magazijn is niet altijd overzichtelijk. De magazijnchef zou graag weten welk artikel en hoeveel daarvan ligt opgeslagen in de vakken VC 12 tot en met VC 16.
63. De financiʌle administratie wil graag een overzicht van alle betalingen van de firma Bakker B.V. in Dordrecht in de maanden januari tot en met mei van 1998 (drie subqueries)
64. Wat is het goedkoopste product dat Relate verkoopt?
65. Welke werknemer is het langst in dienst bij Relate?
66. Wat is uitgedrukt in geld, de grootste order die ooit bij Relate geplaatst is?
67. Welke verkoper heeft in de eerste twee weken van 1997 helemaal geen orders geboekt?
68. De verkoopmanager denkt een soort regelmaat te hebben ontdekt in de ordergrootte. Vooral bij ййn klant is deze regelmaat opvallend; deze heeft klantnummer 100004. De manager zou nu graag weten hoeveel orders er zijn met een orderbedrag dat gelijk is aan een van de orderbedragen van die klant.
69. Begin 1998 heeft Relate de administratiekosten voor kleine orders afgeschaft. Hoeveel geld heeft Relate hiermee in 1998 verspeeld? Een kleine order is een order onder de 200; de kosten die in rekening werden gebracht waren 23,50.
70. De klant Expert Computerservice heeft in totaal 6 kleine orders geplaatst. De vraag is of er andere klanten zijn die nog meer kleine orders hebben geplaatst. Toon de namen van die klanten.
71. Maak een lijst waarop per medewerker het PERSNR en het totaal aantal dagen verzuimd in 1999 komt te staan. Zet de medewerker met het hoogste verzuim bovenaan.
72. Maak een lijst van personeelsnummers die in 1999 meer verzuimd hebben dan Johan Boom.
73. De chef van het magazijn wil opruiming houden. Zijn vraag is of er ook artikelen zijn die ooit wel zijn ingekocht, maar tot nu toe nooit zijn verkocht.
74. De vraag wordt nu als volgt gesteld: zijn er ook artikelen geweest waarvan er niet meer dan 1000 zijn verkocht?
75. De financiʌle administratie wil graag een overzicht van alle orders onder de 500. De lijst moet per order naast de orderdatum en het orderbedrag ook de naam, het adres en de plaats van vestiging van de klant tonen.
76. De chef van de afdeling verkoop wil graag een overzicht van de prestaties van zijn vertegenwoordigers. De lijst moet de volledige naam van de vertegenwoordiger bevatten, zijn jaarsalaris, het totaal aantal orders dat de vertegenwoordiger in 1999 heeft geboekt en de omzet die hij daarmee gehaald heeft.
77. De financiʌle administratie is op zoek naar de klant die de order met ordernummer 970019 heeft geplaatst. Maak een uitdraai waarop alle gegevens van de klant en de order inclusief de orderregels staan vermeld.
78. Breid de query van 77 zodanig uit dat ook de omschrijving van de gekochte artikelen worden getoond.
79. De verkopers zouden graag een overzicht hebben waarop per artikelgroep staat vermeld welke artikelen er zijn en wat de prijs van die artikelen is. Vermeld ook het artikelnummer op de lijst. Per artikelgroep wil men graag de prijs oplopend gerangschikt zien.
80. De afdeling promotie wil graag een lijst van de artikelen die in 1998 tijdens de decembermaand zijn verkocht op basis van de artikelgroepen. De lijst moet dus de te onderscheiden artikelgroepen weergeven die in december 1998 zijn verkocht.
81. Voor het maken van een speciale mailing wil de afdeling promotie graag een lijst van klanten die in de decembermaand van 1998 beauty-artikelen, luxe artikelen of horloges hebben gekocht. De lijst moet het volledige klantadres bevatten en gerangschikt zijn op klantnaam en het soort artikel dat door de klant is besteld.
82. Een klant belt op over een order van 9 februari 1999. De financiʌle administratie wil alle gegevens van die order op papier hebben. Maak een lijst met daarop alle ordergegevens en de daarbij horende orderregels. Er is op die dag maar ййn order geplaatst.

83. Mick Dirksen van de afdeling verkoop wil een lijst van de grote klanten. Op deze lijst wil hij de voornaamste klantgegevens zoals naam, adres en telefoonnummer gevolgd door het aantal orders en het totale orderbedrag per klant in 2000. Een grote klant is een klant die in 2000 voor meer dan 45.000 heeft ingekocht. Zet de beste klant bovenaan de lijst.
84. Maak een query die het totaal verkoopbedrag per verkopende medewerker in 1997 laat zien. Sorteer op functie en zet daarbij de beste verkoper bovenaan.
85. Runners zijn artikelen die veel en regelmatig worden verkocht. Stel dat een artikel een runner is als er meer dan 10.000 stuks per jaar worden verkocht. Welke artikelen van Relate waren in 1998 dan runners?
86. Maar een overzicht van alle kantoorartikelen waarvan er in 1997 meer dan 500 zijn verkocht.
87. Mevrouw Doornbos van de financiële administratie heeft het idee dat de klanten steeds slechter gaan betalen. Daarom wil zij graag een overzicht waarop vermeld staat wat de gemiddelde betalingstermijn per jaar is. Met betalingstermijn bedoelen we de periode tussen het versturen van de factuur en het ontvangen van het totale factuurbedrag.
88. Om een beter inzicht te krijgen in het betalingsgedrag wil Willy Doornbos bovendien nog eens per klant de maximale en minimale betalingstermijn van het jaar 1998 weten.
89. Welke facturen staan op dit moment langer dan twee weken open? Gebruik de actuele datum.
90. Liselot van der Laan houdt de verjaardagen en de jubilea bij. Zij wil graag elke morgen even handig opvragen wie er de volgende dag jarig zijn. Daarnaast wil zij graag een overzicht kunnen maken waarop per werknemer het aantal dienstjaren staat vermeld met de dag en de maand van indiensttreding. Ontwerp de juiste queries.
91. Toon de naam en de jaaromzet per jaar van de oudste vertegenwoordiger van Relate.
92. Jaarlijks krijgt de vertegenwoordiger met de hoogste omzet van de directie een prijs. Toon de naam en de klanten van de beste vertegenwoordiger van Relate van 1999.
93. In de oplossing van 85 kunnen we de regelmaat van de bestellingen niet controleren. Liever had de inkoopster een overzicht gehad van de artikelen waarvan er in 1998 meer dan 1000 per maand zijn verkocht.
94. Het overzicht van oefening 87 verschaft te weinig duidelijkheid. Misschien veroorzaken enkele uitschieters het verschil in de gemiddelden. Daarom wil Willy Doornbos nu ook een overzicht van de gemiddelde betalingstermijn per klant voor het jaar 1999 en voor het jaar 1998 (slechtste betaler vooraan).

9.2 Relatieschema database Relate



10 Set operatoren

10.1 UNION

De **UNION** operator wordt gebruikt om de resultset van twee of meer **SELECT** instructies te combineren. Merk op dat elke **SELECT** instructie binnen de **UNION** het zelfde aantal kolommen moeten hebben. De kolommen moeten ook van vergelijkbare datatypes zijn. Ten slotte moeten de kolommen van elke **SELECT** instructie in dezelfde volgorde staan.

Syntax

```
SELECT column_name(s)
FROM table_name1
UNION
SELECT column_name(s)
FROM table_name2
```

Let op: **de UNION operator selecteert standaard alleen unieke waarden. Om dubbele waarden toe te staan moeten we UNION ALL gebruiken**

SQL **UNION ALL** Syntax

```
SELECT column_name(s)
FROM table_name1
UNION ALL
SELECT column_name(s)
FROM table_name2
```

Let op: **de kolomnamen in de result-set van een UNION zijn altijd gelijk aan die van de eerste SELECT instructie in de UNION.**

Voorbeeld

Employees_Norway

E_ID	E_Name
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

Employees_USA

E_ID	E_Name
01	Turner, Sally
02	Kent, Clark
03	Svendson, Stephen
04	Scott, Stephen

Nu willen we **alle verschillende** employees in Norway en de USA. We gebruiken de volgende **SELECT** instructie:

```
SELECT E_Name
FROM Employees_Norway
UNION
SELECT E_Name
FROM Employees_USA
```

Het resultaat zal er zo uit zien:

E_Name
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
Scott, Stephen

Let op: **dit commando kan niet gebruikt worden om alle employees in Norway en de USA weer te geven. In het voorbeeld hiervoor hadden we twee employees met dezelfde namen en maar ййн daarvan wordt getoond.**

Voorbeeld **UNION ALL**

Nu willen we alle employees in Norway en de USA. De **SELECT** instructie is als volgt:

```
SELECT E_Name
FROM Employees_Norway
UNION ALL
SELECT E_Name
FROM Employees_USA
```

Resultaat

E_Name
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
Svendson, Stephen
Scott, Stephen

10.2 INTERSECT

De operator **INTERSECT** is eigenlijk een **AND** operator toegepast op de resultaten van de volledige zoekqueries. Dat wil zeggen dat alleen het resultaat van de queries wordt getoond voor zover dat voor beide queries gelijk is. Zoals de operator **AND** een doorsnede maakt tussen de uitkomst van de voorwaarden binnen een query, maakt de **INTERSECT** een doorsnede van de resultaten van meerdere queries.

Syntax

```
SELECT column_name(s)
FROM table_name1
INTERSECT
SELECT column_name(s)
FROM table_name2
```

Voorbeeld

Employees_Norway

E_ID	E_Name
01	Hansen, Ola

02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

Employees_USA

E_ID	E_Name
01	Turner, Sally
02	Kent, Clark
03	Svendson, Stephen
04	Scott, Stephen

We willen nu de E_Name die in beide tabellen voorkomen. De **SELECT** instructie is als volgt:

```
SELECT E_Name
FROM Employees_Norway
INTERSECT
SELECT E_Name
FROM Employees_USA
```

Het resultaat zal er zo uit zien:

E_Name
Svendson, Stephen

Let op: MS Access ondersteunt deze operator niet.

Let op: de INTERSECT operator selecteert standaard alleen unieke waarden. Om dubbele waarden toe te staan moeten we INTERSECT ALL gebruiken

SQL **INTERSECT ALL** Syntax

```
SELECT column_name(s)
FROM table_name1
INTERSECT ALL
SELECT column_name(s)
FROM table_name2
```

Maar weinig SQL varianten ondersteunen deze optie.

10.3 EXCEPT

De operator **EXCEPT** is het best te vergelijken met **NOT IN**. Door zoekqueries te koppelen met **EXCEPT** worden alleen die rijen uit het resultaat van de eerste query getoond die niet in het resultaat van de tweede voorkomen.

Syntax

```
SELECT column_name(s)
FROM table_name1
EXCEPT
SELECT column_name(s)
FROM table_name2
```

Voorbeeld

Employees_Norway

E_ID	E_Name

01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

Employees_USA

E_ID	E_Name
01	Turner, Sally
02	Kent, Clark
03	Svendson, Stephen
04	Scott, Stephen

We willen nu de E_Name die in beide tabellen voorkomen. De **SELECT** instructie is als volgt:

```
SELECT E_Name
FROM Employees_Norway
EXCEPT
SELECT E_Name
FROM Employees_USA
```

Het resultaat zal er zo uit zien:

E_ID	E_Name
01	Hansen, Ola
02	Svendson, Tove
04	Pettersen, Kari

Let op: MS Access ondersteunt deze operator niet.

Let op: de EXCEPT operator selecteert standaard alleen unieke waarden. Om dubbele waarden toe te staan moeten we EXCEPT ALL gebruiken

SQL **Except ALL** Syntax

```
SELECT column_name(s)
FROM table_name1
Except ALL
SELECT column_name(s)
FROM table_name2
```

Maar weinig SQL varianten ondersteunen deze optie.

10.4 Opgaven

95. Maak een lijst van klanten en leveranciers met bedrijfsnaam, plaats en land en een zelfgemaakte kolom waarin we een K of L zetten.
96. Maak een lijst van plaatsnamen die zowel in de klanten als de leverancierstabel voorkomen.
97. Maak een lijst van plaatsen uit de klantentabel die niet in de leverancierstabel voorkomen.
98. Maak een overzicht van plaatsen met alleen klanten en alleen leveranciers.
99. Maak een overzicht van de landen met alleen klanten, de landen met alleen leveranciers en de landen met beide op volgorde van land.

11 Data Manipulation Language

11.1 SELECT INTO

De **SELECT INTO** instructie selecteert data uit de ene tabel en voegt die toe aan een andere tabel. De instructie wordt vaak gebruikt om kopieʌn van tabellen te maken.

Syntax

We kunnen alle kolommen selecteren voor de nieuwe tabel:

```
SELECT *
INTO new_table_name [IN externaldatabase]
FROM old_tablename
```

Of we selecteren alleen de kolommen die we willen voor de nieuwe tabel:

```
SELECT column_name(s)
INTO new_table_name [IN externaldatabase]
FROM old_tablename
```

Voorbeeld

We willen een exacte kopie van onze tabel **Persons**. We gebruiken de volgende **SELECT** instructie:

```
SELECT *
INTO Persons_Backup
FROM Persons
```

We kunnen ook de **IN** component gebruiken om de tabel in een andere database te kopiʌren:

```
SELECT *
INTO Persons_Backup IN Backup.mdb
FROM Persons
```

We kunnen ook een kopie maken van maar een paar kolommen:

```
SELECT LastName,FirstName
INTO Persons_Backup
FROM Persons
```

SQL SELECT INTO met een WHERE component
De volgende **SELECT** instructie maakt een **Persons_Backup** tabel met alleen de mensen die in de plaats Sandnes wonen:

```
SELECT LastName,Firstname
INTO Persons_Backup
FROM Persons
WHERE City='Sandnes'
```

SQL SELECT INTO met gelinkte tabellen
Het volgende voorbeeld maakt **Persons_Order_Backup** tabel die data bevat uit twee tabellen **Persons** en **Orders**:

```
SELECT Persons.LastName,Orders.OrderNo
INTO Persons_Order_Backup
FROM Persons
INNER JOIN Orders
ON Persons.P_Id=Orders.P_Id
```

11.2 INSERT INTO

De **INSERT INTO** instructie wordt gebruikt om een nieuwe rij aan een tabel toe te voegen.

Syntax

We kunnen de **INSERT INTO** instructie op twee manieren schrijven. De eerste specificeert niet de kolomnamen, alleen de waarden:

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

De tweede doet beide:

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

Voorbeeld

De tabel **Persons**:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

We willen hier een nieuwe rij aan toevoegen. We gebruiken de volgende instructie:

```
INSERT INTO Persons
VALUES (4,Nilsen, Johan, Bakken 2, Stavanger)
```

De tabel **Persons** ziet er nu zo uit:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger

Voeg data alleen toe aan gespecificeerde kolommen
De volgende instructie zal een nieuwe rij toevoegen, maar alleen in de kolommen P_Id, LastName en FirstName:

```
INSERT INTO Persons (P_Id, LastName, FirstName)
VALUES (5, Tjessem, Jakob)
```

De tabel **Persons** ziet er daarna zo uit:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob		

Let op: **INSERT INTO kan ook gebruikt worden om een tabel te vullen met rijen uit een andere tabel**

Syntax

```
INSERT INTO table_name1 (column1, column2, column3,...)
SELECT column1, column2, column3,...
FROM table_name2
```

11.3 UPDATE

De **UPDATE** instructie wordt gebruikt om bestaande records bij te werken.

Syntax

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Let op: **de WHERE component specificeert welk record of records bijgewerkt moeten worden. Als we WHERE weglaten, worden alle records bijgewerkt!**

Voorbeeld

De tabel **Persons**:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob		

We willen Tjessem, Jakob bijwerken. We gebruiken de volgende instructie:

```
UPDATE Persons
SET Address='Nissestien 67', City='Sandnes'
WHERE LastName='Tjessem' AND FirstName='Jakob'
```

De tabel **Persons** ziet er nu zo uit:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	Nissestien 67	Sandnes

Let op: **als we WHERE hadden weggelaten:**

```
UPDATE Persons
SET Address='Nissestien 67', City='Sandnes'
```

Dan had de tabel **Persons** er nu zo uitgezien:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Nissestien 67	Sandnes
2	Svendson	Tove	Nissestien 67	Sandnes
3	Pettersen	Kari	Nissestien 67	Sandnes
4	Nilsen	Johan	Nissestien 67	Sandnes
5	Tjessem	Jakob	Nissestien 67	Sandnes

Let op: niet alle database staan subqueries toe in de SET component van UPDATE

11.4 DELETE

De **DELETE** instructie wordt gebruikt om rijen te wissen.

Syntax

```
DELETE FROM table_name
WHERE some_column=some_value
```

Let op: de WHERE component specificeert welk record of records gewist moeten worden. Als we WHERE weglaten, worden alle records gewist!

Voorbeeld

De tabel **Persons** ziet er nu zo uit:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	Nissestien 67	Sandnes

Nu willen we Tjessem, Jakob wissen. De instructie is als volgt:

```
DELETE FROM Persons
WHERE LastName = 'Tjessem'
AND FirstName = 'Jakob'
```

De tabel **Persons** ziet er nu zo uit:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger

11.5 DELETE alle rijen

Het is mogelijk alle rijen te wissen zonder de tabel te verwijderen. Dat betekent dat de tabelstructuur, attributen en indexen in tact blijven.

```
DELETE FROM table_name
```

of

```
DELETE * FROM table_name
```

Let op: wees voorzichtig; dit is niet terug te draaien!

11.6 Opgaven

100. Alle plaatsnamen in de klantentabel moeten voortaan met hoofdletters beschreven worden; maakt een **UPDATE** script om dit aan te passen.
101. Maak met **SELECT INTO** een kopie van de producten tabel en verhoog vervolgens alle prijzen met 10 procent.
102. Maak een kopie van de tabel orders en vervang alle lege leverdatum door de orderdatum + 30 dagen
103. Wis uit deze laatste kopie alle orders waarvan de levering korter dan 10 dagen heeft geduurd.
104. Maak een kopie van de klantentabel, leeg deze kopie vervolgens en voeg uit de tabellen leveranciers en klanten de naam van het bedrijf en de plaatsnaam toe aan deze kopie.

12 Data Definition Language

12.1 CREATE

CREATE DATABASE
Deze instructie wordt gebruikt om een database te maken.

Syntax

```
CREATE DATABASE database_name
```

Voorbeeld

We willen nu een database maken genaamd my_db.

We gebruiken de volgende instructie:

```
CREATE DATABASE my_db
```

Let op: **MS Access kent deze instructie niet.**

CREATE TABLE
De **CREATE TABLE** instructie wordt gebruikt om een tabel te maken.

Syntax

```
CREATE TABLE table_name
 (
  column_name1 data_type,
  column_name2 data_type,
  column_name3 data_type,
  ....
 )
```

Het datatype specificeert welk type data de kolom kan bevatten. Voor een compleet overzicht van datatypes in MS Access, MySQL en SQL Server, zie hoofdstuk 18.

Voorbeeld

We willen een tabel Persons maken met vijf kolommen: P_Id, LastName, FirstName, Address en City. We gebruiken de volgende instructie:

```
CREATE TABLE Persons
 (
  P_Id INT,
  LastName VARCHAR (255),
  FirstName VARCHAR (255),
  Address VARCHAR (255),
  City VARCHAR (255)
 )
```

De kolom P_Id is van het type **INT** en bevat gehele cijfers. De kolommen **LastName**, **FirstName**, **Address** en **City** zijn van het type **VARCHAR** met een maximum lengte van 255 karakters.

De lege tabel **Persons** zal er zo uit zien:

P_Id	LastName	FirstName	Address	City

CREATE INDEX

De **CREATE INDEX** instructie wordt gebruikt om indexen te maken in tabellen. Met indexen kunnen gegevens in tabellen sneller gevonden worden zonder dat de hele tabel gelezen moet worden. De gebruikers kunnen indexen niet zien. Ze worden alleen maar gebruikt om het zoekproces of de uitvoering van queries te versenellen.

Let op: het bijwerken van een tabel met indexen kost meer tijd dan wanneer er geen indexen zijn. We moeten dus alleen indexen maken op kolommen die regelmatig doorzocht moeten worden.

Syntax

Dubbele waarden zijn toegestaan:

```
CREATE INDEX index_name
ON table_name (column_name)
```

CREATE UNIQUE INDEX
Maakt een unieke index op een tabel. Dubbele waarden zijn niet toegestaan:

```
CREATE UNIQUE INDEX index_name
ON table_name (column_name)
```

Let op: de syntax om indexen te creλren loopt uiteen tussen de verschillende databases. Check dus wat de syntax is voor de betreffende database.

Voorbeeld CREATE INDEX

De SQL instructie hierna creλert een index genaamd **PIndex** op de kolom **LastName** in de tabel **Persons**:

```
CREATE INDEX PIndex
ON Persons (LastName)
```

Als we een index willen maken op een combinatie van kolommen, zet dan de lijst van kolommen tussen de haakjes, gescheiden door komma's:

```
CREATE INDEX PIndex
ON Persons (LastName, FirstName)
```

12.2 CREATE FUNCTION

Hieronder een voorbeeld van een functie gemaakt in SQL Server met Transact-SQL:

```
CREATE
FUNCTION dbo.datumomzetten(@datum int)
RETURNS
date
AS
BEGIN
DECLARE @ret date;
IF (@DATUM < 19000000)
  SET @ret = NULL;
ELSE
  SELECT @ret =
  CAST(
  SUBSTRING(CAST(@datum AS VARCHAR),5,2)+'/'+
  RIGHT(CAST(@datum AS VARCHAR),2)+'/'+
  LEFT(CAST(@datum AS VARCHAR),4)
  AS DATE);
RETURN @ret;
END;
GO
```

12.3 DROP

DROP INDEX
De **DROP INDEX** instructie wordt gebruikt om een index te wissen.

Syntax in MS Access:

```
DROP INDEX index_name
ON table_name
```

Syntax in MS SQL Server:

```
DROP INDEX table_name.index_name
```

Syntax in DB2/Oracle:

```
DROP INDEX index_name
```

Syntax in MySQL:

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

DROP TABLE
De **DROP TABLE** instructie wordt gebruikt om een tabel te wissen.

```
DROP TABLE table_name
```

DROP DATABASE
De **DROP DATABASE** instructie wordt gebruikt om de database te wissen.

```
DROP DATABASE database_name
```

12.4 TRUNCATE

Hoe wissen we alleen de data en niet de tabel? Gebruik de **TRUNCATE TABLE** instructie:

```
TRUNCATE TABLE table_name
```

12.5 ALTER

De **ALTER TABLE** instructie wordt gebruikt om kolommen in een bestaande tabel toe te voegen, te wissen of te wijzigen.

Syntax:

Om een kolom toe te voegen:

```
ALTER TABLE table_name
ADD column_name datatype
```

Om te wissen:

```
ALTER TABLE table_name
DROP COLUMN column_name
```

Let op: **sommige databases staan het wissen van kolommen niet toe.**

Om het datatype van een kolom aan te passen gebruiken we de volgende syntax:

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype
```

SQL **ALTER TABLE** Voorbeeld

Kijk naar de tabel **Persons**:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

We willen een kolom toevoegen genaamd **DateOfBirth** aan de tabel **Persons**. We gebruiken de volgende instructie:

```
ALTER TABLE Persons
ADD DateOfBirth DATE
```

De nieuwe kolom **DateOfBirth** is van het type **DATE** en moet datums bevatten. Het datatype specificeert welk type gegevens een kolom kan bevatten.

Let op: **voor een compleet overzicht van alle datatype in MS Access, MySQL en SQL Server, zie hoofdstuk 18.**

De tabel **Persons** ziet er nu zo uit:

P_Id	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

Voorbeeld Change Data Type

We willen nu het datatype van de kolom **DateOfBirth** in de tabel **Persons** aanpassen. We gebruiken de volgende instructie:

```
ALTER TABLE Persons
ALTER COLUMN DateOfBirth YEAR
```

Nu is de kolom **DateOfBirth** van het type **YEAR**; deze kan nu alleen het jaar bevatten in een 2 of 4 cijferig formaat.

Voorbeeld **DROP COLUMN**

We willen de kolom DateOfBirth in de tabel Persons wissen. We gebruiken de volgende instructie:

```
ALTER TABLE Persons
DROP COLUMN DateOfBirth
```

De tabel **Persons** ziet er nu zo uit:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

13 Transaction Control Language

We definiлren een transactie als een verzameling **SELECT** instructies die door ййn gebruiker worden ingevoerd een waarbij aan het einde aangegeven moet worden of alle mutaties permanent of ongedaan gemaakt moeten worden.

Er zijn databases met een **AUTOCOMMIT** optie, dat wil zeggen dat mutaties automatisch permanent gemaakt worden. Als deze **AUTOCOMMIT** uitgezet kan worden (niet in MS Access bijvoorbeeld) kunnen de mutatataties met **COMMIT** worden bevestigd of met **ROLLBACK** worden teruggedraaaid.

Sommige database ondersteunen zogenaamde **SAVEPOINTS** zodat transacties tot een bepaald punt teruggedraaid kunnen worden.

14 Data Control Language

Deze instructies hebben betrekking op de beveiliging van gegevens. Met **GRANT** instructies kunnen bevoegdheden aan gebruikers verstrekt worden. Met **REVOKE** kunnen deze bevoegdheden weer ingetrokken worden. Met **CREATE USER** en **DROP USER** kunnen respectievelijk gebruikers gecreëerd worden en weer verwijderd.

15 Speciale commando's SQL Server

15.1 MERGE INTO

```
Merge into orders as o
using ordersnieuw as n
on (o.[order-id]=n.[order-id])
when matched then
    update set
       o.orderdatum = n.orderdatum,
       o.vrachtkosten = n.vrachtkosten
when not matched then
    insert ([order-id],klantnummer,[werknemer-
id],orderdatum,vervaldatum,leverdatum,verzendwijze,vrachtkoste
n)
    values (n.[order-id],n.klantnummer,n.[werknemer-
id],n.orderdatum,n.vervaldatum,n.leverdatum,n.verzendwijze,n.v
rachtkosten);
```

15.2 Tabel-valued functions

```
create function TopNproducten
(@t as INT)
returns table
as
return
    (select top (@t) productnummer,productnaam,[prijs per
eenheid]
    from producten
    order by [prijs per eenheid] desc)

select * from TopNproducten(10)
```

16 Performance

16.1 Zo weinig mogelijk OR gebruiken

Waarom? In dat geval gebruikt SQL meestal geen index en dat scheelt natuurlijk snelheid. Er zijn twee alternatieven: **IN** en **UNION**.

Voorbeeld OR vervangen door IN:

Met **OR**:

```
SELECT *
FROM klanten
WHERE klantcode = 1
OR klantcode = 2
OR klantcode = 3
```

Met **IN**:

```
SELECT *
FROM klanten
WHERE klantcode IN (1,2,3)
```

Voorbeeld OR vervangen door UNION:

Met **OR**:

```
SELECT *
FROM klanten
WHERE klantcode = 1
OR klplaats = 'GRONINGEN'
```

Met **UNION**:

```
SELECT *
FROM klanten
WHERE klantcode = 1
UNION
SELECT *
FROM klanten
WHERE klplaats = 'GRONINGEN'
```

16.2 De grootste tabel als laatste

De volgorde van de tabellen bij **FROM** kan effect hebben op de snelheid van verwerking. Plaats voor de zekerheid de grootste tabel achteraan.

16.3 Liever geen SELECT *

Vraag geen kolommen op die we niet nodig hebben. Bovendien brengt **SELECT** * met zich mee dat er geen indexen gebruikt worden wat ook voor een slechtere performance zorgt.

16.4 Zo weinig mogelijk DISTINCT gebruiken

Als er dubbele rijen verwijderd moeten worden, kan dat zeer tijdrovend zijn; de software moet dan elke rij met alle andere vergelijken. **DISTINCT** is zeker niet noodzakelijk in subqueries.

16.5 Pas op met UNION

Het **UNION** commando voert automatisch het equivalent van een **SELECT DISTINCT** uit, ook als er geen dubbele rijen zijn. Als we er zeker van zijn dat er geen dubbele rijen zijn, kunnen we beter **UNION ALL** gebruiken.

16.6 Snelheid van operatoren

Bij **WHERE** bepalen de verschillende operatoren hoe snel een query wordt uitgevoerd. Vaak hebben we geen keus, maar als we die wel hebben, geeft onderstaande volgorde van hoog naar laag de snelheid van uitvoering aan:

- =
- >, >=, <, <=
- LIKE
- <>

Conclusie: gebruik zo veel mogelijk = en zo weinig mogelijk <>.

16.7 Gebruik indien mogelijk BETWEEN

Aansluitend op het vorige geniet het de voorkeur **BETWEEN** te gebruiken boven een constructie met >= en <=. Bij gebruik van **BETWEEN** doet SQL namelijk wel een beroep op een eventuele index op de betreffende kolom.

16.8 BETWEEN boven IN

Als we keus hebben, gebruik dan **BETWEEN** en geen **IN**. Bijvoorbeeld:

```
SELECT customer_number, customer_name
FROM customer
WHERE customer_number in (1000, 1001, 1002, 1003, 1004)
```

Is veel minder efficiʌnt dan:

```
SELECT customer_number, customer_name
FROM customer
WHERE customer_number BETWEEN 1000 and 1004
```

Gesteld dat er een index zit op customer_number dan zal een reeks nummer veel sneller gelokaliseerd worden met **BETWEEN** dan met **IN**.

16.9 Vermijd indien mogelijk de SUBSTRING functie

De **SUBSTRING** functie kan een scan van de hele tabel tot gevolg hebben in plaats van de eventueel aanwezige index te gebruiken.

Dus in plaats van:

```
WHERE SUBSTRING(column_name,1,1) = 'b'
```

Probeer:

```
WHERE column_name LIKE 'b%'
```

16.10 Vermijd de combinatie NOT IN

Gebruik ййn van de volgende alternatieven:

- **EXISTS** of **NOT EXISTS**
- **IN**

- **LEFT OUTER JOIN** in combinatie met een **NULL** voorwaarde

16.11 IN of EXISTS

Als we de keus hebben tussen het gebruik van **IN** of **EXISTS**, **EXISTS** zal gewoonlijk sneller zijn.

17 Vergelijking scalaire functies in SQL dialecten

sql functie	access	sql server	mysql	oracle	solid	db2
abs	abs	abs	abs	abs	abs	abs
char	chr	char	char	chr	char	
concatenation	&, +	+	concat(a,b)	concat	concat	concat
currentdate	date	getdate	current_date	sysdate	curdate	curdate
currenttime	now	gettime	current_time	sysdate	curtime	curtime
date	datevalue	datevalue	date			
dateadd	dateadd, +	dateadd	date_add	+, add_months		+ type
datediff	datediff	datediff	datediff	-	-	-
datepart	datepart	datepart				
day	day	day	day		dayofweek	
dayname	weekdayname	-	dayname			
hour	hour	hour	hour		hour	hour
instr	instr	charindex		instr		
left	left	left	left			left
length	len	len	char_length	length	length	length
lower	lcase	lower	lcase / lower	lower	lcase	lcase / lower
ltrim	ltrim	ltrim	ltrim	ltrim		ltrim
minute	minute	minute	minute		minute	minute
mod	mod	mod	mod	mod	mod	mod
month	month	month	month	tomonth	month	month
monthname	monthname	datename	monthname		monthname	
quarter	datepart	datepart	quarter	toquarter	quarter	quarter
power	^	power / square	power	power	power	power
replace	replace	replace	replace	replace	replace	
right	right	right	right			
round	round	round	round	round	round	round
rtrim	rtrim	rtrim	rtrim	rtrim	rtrim	rtrim
second	second	second	second		second	second
sqrt	sqr	sqrt	sqrt	sqrt	sqrt	sqrt
substring	mid	substring	mid / substring	substr	substring	substring / substr
time	timevalue	time	time			
trim	trim	trim	trim			
upper	ucase	upper	ucase / upper	upper	ucase	ucase / upper
week	datepart	datepart	week		week	week
weekday			dayofweek			
year	year	year	year	toyear	year	year

18 Datatypes

18.1 Microsoft Access Datatypes

Data type	Description	Storage
Text	Use for text or combinations of text and numbers. 255 characters maximum	
Memo	Memo is used for larger amounts of text. Stores up to 65,536 characters. Note: You cannot sort a memo field. However, they are searchable	
Byte	Allows whole numbers from 0 to 255	1 byte
Integer	Allows whole numbers between -32,768 and 32,767	2 bytes
Long	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
Single	Single precision floating-point. Will handle most decimals	4 bytes
Double	Double precision floating-point. Will handle most decimals	8 bytes
Currency	Use for currency. Holds up to 15 digits of whole dollars, plus 4 decimal places. Tip: You can choose which countrys currency to use	8 bytes
AutoNumber	AutoNumber fields automatically give each record its own number, usually starting at 1	4 bytes
Date/Time	Use for dates and times	8 bytes
Yes/No	A logical field can be displayed as Yes/No, True/False, or On/Off. In code, use the constants True and False (equivalent to -1 and 0).Note: Null values are not allowed in Yes/No fields	1 bit
Ole Object	Can store pictures, audio, video, or other BLOBs (Binary Large OBjects)	up to 1GB
Hyperlink	Contain links to other files, including web pages	
Lookup Wizard	Let you type a list of options, which can then be chosen from a drop-down list	4 bytes

18.2 MySQL Datatypes

In MySQL zijn er drie hoofdtypes: tekst, numeriek en datum/tijd

Teksttypes:

Data type	Description
CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. Note: If you put a greater value than 255 it will be converted to a TEXT type
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 characters
BLOB	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(x,y,z,etc.)	Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. Note: The values are sorted in the order you enter them. You enter the possible values in this format: ENUM(X,Y,Z)

SET	Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice

Numerieke types:

Data type	Description
TINYINT(size)	-128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis
SMALLINT(size)	-32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis
MEDIUMINT(size)	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis
INT(size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
BIGINT(size)	-9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED*. The maximum number of digits may be specified in parenthesis
FLOAT(size,d)	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DOUBLE(size,d)	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DECIMAL(size,d)	A DOUBLE stored as a string , allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

Let op **The integer types have an extra option called UNSIGNED. Normally, the integer goes from an negative to positive value. Adding the UNSIGNED attribute will move that range up so it starts at zero instead of a negative number.**

Datum/tijd types:

Data type	Description
DATE()	A date. Format: YYYY-MM-DD Note: The supported range is from 1000-01-01 to 9999-12-31
DATETIME()	*A date and time combination. Format: YYYY-MM-DD HH:MM:SS Note: The supported range is from 1000-01-01 00:00:00 to 9999-12-31 23:59:59
TIMESTAMP()	*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch (1970-01-01 00:00:00 UTC). Format: YYYY-MM-DD HH:MM:SS Note: The supported range is from 1970-01-01 00:00:01 UTC to 2038-01-09 03:14:07 UTC
TIME()	A time. Format: HH:MM:SS Note: The supported range is from -838:59:59 to 838:59:59
YEAR()	A year in two-digit or four-digit format. Note: Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

*Even if DATETIME and TIMESTAMP return the same format, they work very differently. In an INSERT or UPDATE query, the TIMESTAMP automatically set itself to the current date and time. TIMESTAMP also accepts various formats, like YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD, or YYMMDD.

18.3 SQL Server datatypes

Character strings:

Data type	Description	Storage

char(n)	Fixed-length character string. Maximum 8,000 characters	n
varchar(n)	Variable-length character string. Maximum 8,000 characters	
varchar(max)	Variable-length character string. Maximum 1,073,741,824 characters	
text	Variable-length character string. Maximum 2GB of text data	

Unicode strings:

Data type	Description	Storage
nchar(n)	Fixed-length Unicode data. Maximum 4,000 characters	
nvarchar(n)	Variable-length Unicode data. Maximum 4,000 characters	
nvarchar(max)	Variable-length Unicode data. Maximum 536,870,912 characters	
ntext	Variable-length Unicode data. Maximum 2GB of text data	

Binary types:

Data type	Description	Storage
bit	Allows 0, 1, or NULL	
binary(n)	Fixed-length binary data. Maximum 8,000 bytes	
varbinary(n)	Variable-length binary data. Maximum 8,000 bytes	
varbinary(max)	Variable-length binary data. Maximum 2GB	
image	Variable-length binary data. Maximum 2GB	

Number types:

Data type	Description	Storage
tinyint	Allows whole numbers from 0 to 255	1 byte
smallint	Allows whole numbers between -32,768 and 32,767	2 bytes
int	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8 bytes
decimal(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
numeric(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
smallmoney	Monetary data from -214,748.3648 to 214,748.3647	4 bytes
money	Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
float(n)	Floating precision number data from -1.79E + 308 to 1.79E + 308. The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53.	4 or 8 bytes
real	Floating precision number data from -3.40E + 38 to 3.40E + 38	4 bytes

Date types:

Data type	Description	Storage
datetime	From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds	8 bytes
datetime2	From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds	6-8 bytes
smalldatetime	From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute	4 bytes
date	Store a date only. From January 1, 0001 to December 31, 9999	3 bytes
time	Store a time only to an accuracy of 100 nanoseconds	3-5 bytes
datetimeoffset	The same as datetime2 with the addition of a time zone offset	8-10 bytes
timestamp	Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable	

Andere datatypes:

Data type	Description
sql_variant	Stores up to 8,000 bytes of data of various data types, except text, ntext, and timestamp
uniqueidentifier	Stores a globally unique identifier (GUID)
xml	Stores XML formatted data. Maximum 2GB
cursor	Stores a reference to a cursor used for database operations
table	Stores a result-set for later processing

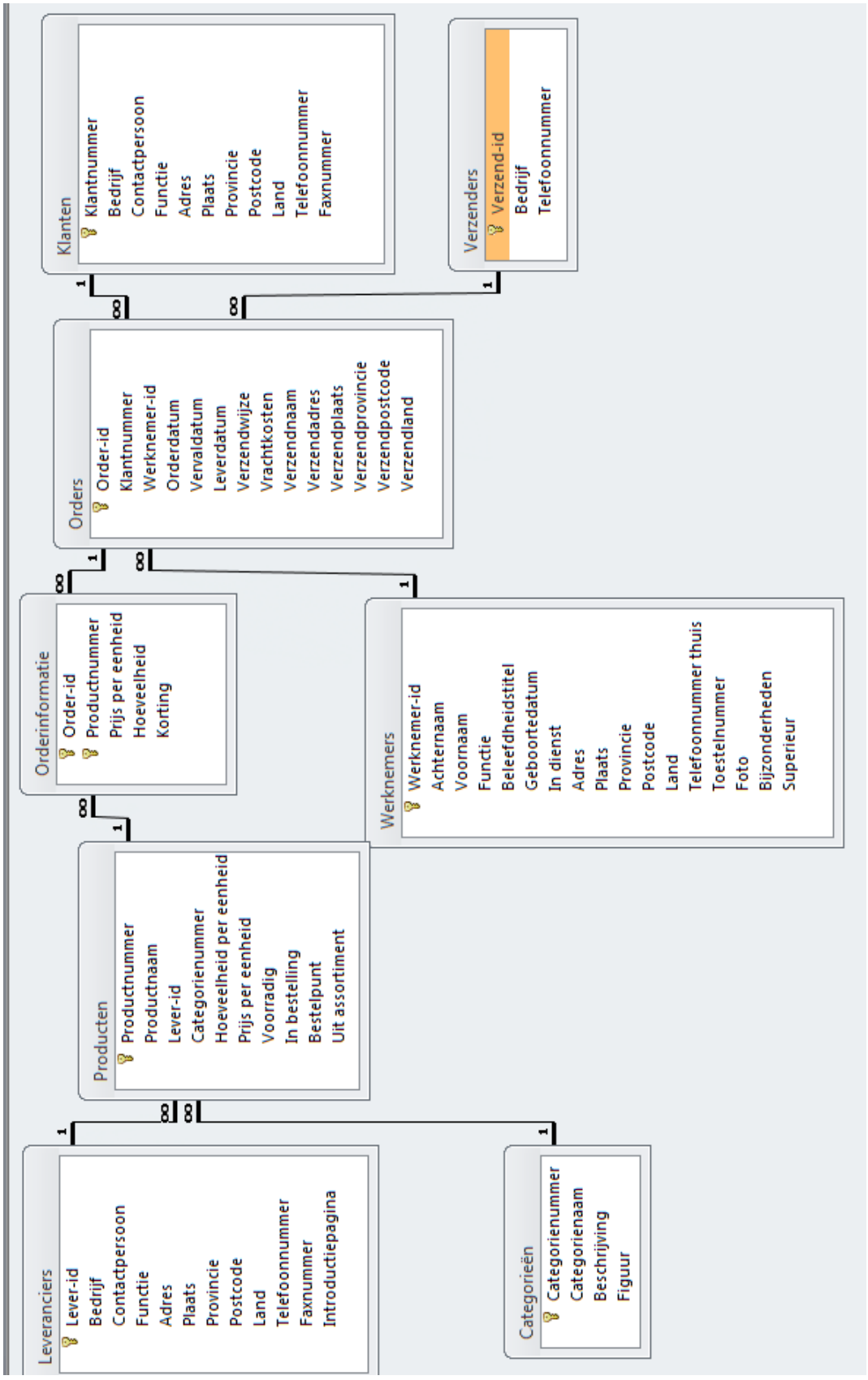
19 Quick reference

SELECT instructie	Syntax
AND / OR	SELECT column_name(s) FROM table_name WHERE condition AND|OR condition
ALTER TABLE	ALTER TABLE table_name ADD column_name datatype Of ALTER TABLE table_name DROP COLUMN column_name
AS (alias)	SELECT column_name AS column_alias FROM table_name Of SELECT column_name FROM table_name AS table_alias
BETWEEN	SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2
CREATE DATABASE	CREATE DATABASE database_name
CREATE TABLE	CREATE TABLE table_name (column_name1 data_type, column_name2 data_type, column_name2 data_type, ...)
CREATE INDEX	CREATE INDEX index_name ON table_name (column_name) or CREATE UNIQUE INDEX index_name ON table_name (column_name)
CREATE VIEW	CREATE VIEW view_name AS SELECT column_name(s) FROM table_name WHERE condition
DELETE	DELETE FROM table_name WHERE some_column=some_value of DELETE FROM table_name DELETE * FROM table_name
DROP DATABASE	DROP DATABASE database_name
DROP INDEX	DROP INDEX table_name.index_name (SQL Server) DROP INDEX index_name ON table_name (MS Access) DROP INDEX index_name (DB2/Oracle) ALTER TABLE table_name DROP INDEX index_name (MySQL)
DROP TABLE	DROP TABLE table_name

GROUP BY	SELECT column_name, aggregate_function(column_name) FROM table_name WHERE column_name operator value GROUP BY column_name
HAVING	SELECT column_name, aggregate_function(column_name) FROM table_name WHERE column_name operator value GROUP BY column_name HAVING aggregate_function(column_name) operator value
IN	SELECT column_name(s) FROM table_name WHERE column_name IN (value1,value2,..)
INSERT INTO	INSERT INTO table_name VALUES (value1, value2, value3,....) of INSERT INTO table_name (column1, column2, column3,...) VALUES (value1, value2, value3,....)
INNER JOIN	SELECT column_name(s) FROM table_name1 INNER JOIN table_name2 ON table_name1.column_name=table_name2.column_name
LEFT JOIN	SELECT column_name(s) FROM table_name1 LEFT JOIN table_name2 ON table_name1.column_name=table_name2.column_name
RIGHT JOIN	SELECT column_name(s) FROM table_name1 RIGHT JOIN table_name2 ON table_name1.column_name=table_name2.column_name
FULL JOIN	SELECT column_name(s) FROM table_name1 FULL JOIN table_name2 ON table_name1.column_name=table_name2.column_name
LIKE	SELECT column_name(s) FROM table_name WHERE column_name LIKE pattern
ORDER BY	SELECT column_name(s) FROM table_name ORDER BY column_name [ASC|DESC]
SELECT	SELECT column_name(s) FROM table_name
SELECT *	SELECT * FROM table_name
SELECT DISTINCT	SELECT DISTINCT column_name(s) FROM table_name
SELECT INTO	SELECT * INTO new_table_name [IN externaldatabase] FROM old_table_name *of* SELECT column_name(s) INTO new_table_name [IN externaldatabase] FROM old_table_name

SELECT TOP	SELECT TOP number|percent column_name(s) FROM table_name
TRUNCATE TABLE	TRUNCATE TABLE table_name
UNION	SELECT column_name(s) FROM table_name1 UNION SELECT column_name(s) FROM table_name2
UNION ALL	SELECT column_name(s) FROM table_name1 UNION ALL SELECT column_name(s) FROM table_name2
UPDATE	UPDATE table_name SET column1=value, column2=value,... WHERE some_column=some_value
WHERE	SELECT column_name(s) FROM table_name WHERE column_name operator value

20 Relatieschema Noordenwind database



21 Uitwerkingen opgaven

De onderstaande oplossingen zullen niet steeds in alle SQL dialecten werken. We hebben hier de varianten voor MS Access en SQL Server gegeven. Voor andere dialecten kunnen hier en daar kleine aanpassingen nodig zijn.

1. Stel een lijst samen van alle werknemers van **Noordenwind**. De lijst hoeft alleen de voornamen en de toestelnummers te bevatten. Zorg er voor dat de voornamen op alfabet gerangschikt staan.

```
SELECT Voornaam, Toestelnummer
FROM Werknemers
ORDER BY Voornaam;
```

2. Maak een lijst van alle werknemers met voor- en achternaam en telefoonnummer thuis. Sorteer de lijst op achternaam.

Oplossing Access

```
SELECT Voornaam, Achternaam, Telefoonnummer thuis
FROM Werknemers
ORDER BY 2;
```

Oplossing SQL Server

```
SELECT Voornaam, Achternaam, [Telefoonnummer thuis]
FROM Werknemers
ORDER BY 2;
```

3. Maak een lijst van alle plaatsen waar klanten zitten. Elle plaats mag maar ййn keer voorkomen. Zet de plaatsen op volgorde.

```
SELECT DISTINCT Plaats
FROM Klanten
ORDER BY Plaats;
```

4. Welke klanten kent Noordenwind? Maakt een lijst met bedrijfsnaam en klantennummers.

```
SELECT Bedrijf, Klantnummer
FROM Klanten;
```

5. Welke klanten zitten in Londen? Maak een lijst met bedrijfsnamen en sorteer deze oplopend

```
SELECT Bedrijf
FROM Klanten
WHERE Plaats = 'Londen'
ORDER BY Plaats;
```

6. Welke werknemers zijn na 1 januari 1993 in dienst gekomen? Maak een lijst met achternaam, functie en datum in dienst

Oplossing Access

```
SELECT Achternaam, Functie, [In dienst]
FROM Werknemers
WHERE [In dienst] > #1/1/1993#;
```

Oplossing SQL Server

```
SELECT Achternaam, Functie, [In dienst]
FROM Werknemers
WHERE [In dienst] > '1/1/1993';
```

7. Welke orders hebben meer dan € 300,- aan vrachtkosten? Maak een lijst met ordernummer, orderdatum, vrachtkosten en klantnummer. Zet het hoogste bedrag bovenaan

Oplossing Access

```
SELECT [Order-id], Orderdatum, Vrachtkosten, Klantnummer
FROM Orders
WHERE Vrachtkosten > 300
ORDER BY Vrachtkosten DESC;
```

SQL Server

```
SELECT [Order-id], Orderdatum, Vrachtkosten, Klantnummer
FROM Orders
WHERE Vrachtkosten > 300
ORDER BY Vrachtkosten DESC;
```

8. Maak een lijst van alle bedrijven en de telefoonnummers van klanten in Rio de Janeiro

```
SELECT Bedrijf, Telefoonnummer
FROM Klanten
WHERE Plaats = 'Rio de Janeiro';
```

9. Van welke producten heeft Noordenwind er meer dan 100 op voorraad?

```
SELECT Productnaam, Voorradig
FROM Producten
WHERE Voorradig > 100;
```

10. Maak een overzicht van alle leveranciers uit Frankrijk

```
SELECT Bedrijf, Land
FROM Leveranciers
WHERE Land = 'Frankrijk';
```

11. Zoek de gegevens van de klant die hoort bij ordernummer 10261; we hebben dan 2 queries nodig

Oplossing Access

```
SELECT Klantnummer
FROM Orders
WHERE [Order-id] = 10261;
```

Oplossing SQL Server

```
SELECT Klantnummer
FROM Orders
WHERE [Order-id] = 10261;
```

en

```
SELECT *
FROM Klanten
WHERE Klantnummer = 'QUEDE'
```

12. Maak een overzicht van alle orders uit de tweede helft van februari 1998; dus van na 15 februari

Oplossing Access

```
SELECT *
FROM Orders
WHERE Orderdatum >= #2/15/1998#
AND Orderdatum <= #2/28/1998#
```

Oplossing SQL Server

```
SELECT *
FROM Orders
WHERE Orderdatum >= '2/15/1998'
AND Orderdatum <= '2/28/1998'
```

13. Maak een lijst van alle werknemers die tussen 1960 en 1965 geboren zijn

Oplossing Access

```
SELECT *
FROM Werknemers
WHERE Geboortedatum >= #1/1/1960#
AND Geboortedatum <= #12/31/1965#
```

Oplossing SQL Server

```
SELECT *
FROM Werknemers
WHERE Geboortedatum >= '1/1/1960'
AND Geboortedatum <= '12/31/1965'
```

14. Maak een lijst van klanten uit de kleinere landen Belgiʌ, Denemarken en Oostenrijk

```
SELECT *
FROM Klanten
WHERE Land = 'België'
OR Land = 'Denemarken'
OR Land = 'Oostenrijk'
```

15. Maak een lijst van alle producten met een voorraad tussen de 50 en de 100 stuks

```
SELECT Productnaam
FROM Producten
WHERE voorradig >= 50
AND voorradig <=100
```

16. Maak een lijst van alle dranken en fruit met een prijs tussen de 10 en 20 euro; maak gebruik van de tabel Categorieʌn op de categorienummers te bepalen

```
SELECT Categorienummer
FROM Categorieën
WHERE Categorienaam = 'Dranken' OR Categorienaam = 'Fruit';
```

en

Oplossing Access

```
SELECT Productnaam
FROM Producten
WHERE (Categorienummer = 1 OR Categorienummer = 7)
AND ([Prijs per eenheid] >= 10 AND [Prijs per eenheid] <= 20);
```

Oplossing SQL Server

```
SELECT Productnaam
FROM Producten
WHERE (Categorienummer = 1 OR Categorienummer = 7)
AND ([Prijs per eenheid] >= 10 AND [Prijs per eenheid] <= 20);
```

17. Maak een overzicht van alle producten die uit het assortiment gehaald zijn met een prijs onder de 40 euro

Oplossing Access

```
SELECT Productnaam
FROM Producten
WHERE [Uit assortiment] = True
AND [Prijs per eenheid] < 40;
```

Oplossing SQL Server

```
SELECT Productnaam
FROM Producten
WHERE [Uit assortiment] = 'True'
AND [Prijs per eenheid] < 40;
```

18. Welke producten zijn in flessen verpakt?

Oplossing Access

```
SELECT *
FROM Producten
WHERE [Hoeveelheid per eenheid] LIKE '*fles*';
```

Oplossing SQL Server

```
SELECT *
FROM Producten
WHERE [Hoeveelheid per eenheid] LIKE '%fles%';
```

19. Maak een overzicht van de klanten in grote landen als Canada, Brazilıa en de Ver. Staten

```
SELECT *
FROM Klanten
WHERE Land IN ('Canada', 'Brazilië', 'Ver. Staten');
```

20. Welke klanten hebben in de periode januari – maart 1997 orders geplaatst; geef een lijst met unieke klantnummers?

Oplossing Access

```
SELECT DISTINCT Klantnummer
FROM Orders
WHERE Orderdatum BETWEEN #1/1/1997# AND #3/31/1997#;
```

Oplossing SQL Server

```
SELECT DISTINCT Klantnummer
FROM Orders
WHERE Orderdatum BETWEEN '1/1/1997' AND '3/31/1997';
```

21. Maak een lijst van de werknemers met voor- en achternaam samengevoegd, en de functie

Oplossing Access

```
SELECT Voornaam & ' ' & Achternaam AS Naam, Functie
FROM Werknemers;
```

Oplossing SQL Server

```
SELECT Voornaam + ' ' + Achternaam AS Naam, Functie
FROM Werknemers;
```

22. Welke producten zijn in dozen verpakt en kosten tussen de 10 en 20 euro?

Oplossing Access

```
SELECT *
FROM Producten
WHERE [Hoeveelheid per eenheid] LIKE *dozen*
AND [Prijs per eenheid] BETWEEN 10 AND 20;
```

Oplossing SQL Server

```
SELECT *
FROM Producten
WHERE [Hoeveelheid per eenheid] LIKE '%dozen%'
AND [Prijs per eenheid] BETWEEN 10 AND 20;
```

23. Geef de producten tussen de 0 en de 10 euro het label D, tussen 10 en 20 het label C, tussen 20 en 30 het label B en alle andere producten het label A (gebruik **CASE WHEN** in SQL server)

Oplossing Access

```
SELECT producten.Productnaam,
Switch
  (
  [Prijs per eenheid] >= 0 And [Prijs per eenheid] <= 10,'D',
  [Prijs per eenheid] > 10 And [Prijs per eenheid] <= 20,'C',
  [Prijs per eenheid] > 20 And [Prijs per eenheid] <= 30,'B',
  [Prijs per eenheid] > 30, 'A'
  ) AS Categorie
FROM producten
```

Oplossing SQL Server

```
SELECT Productnaam,
CASE
  WHEN [Prijs per eenheid] BETWEEN 0 AND 10 THEN 'D'
  WHEN [Prijs per eenheid] BETWEEN 10 AND 20 THEN 'C'
  WHEN [Prijs per eenheid] BETWEEN 30 AND 40 THEN 'B'
  ELSE 'A'
END AS categorie
FROM producten
```

24. Hoeveel orders zijn er uit de eerste helft van 1997?

Oplossing Access

```
SELECT COUNT(*)
FROM Orders
WHERE Orderdatum BETWEEN #1/1/1997# AND #6/30/1997#;
```

Oplossing SQL Server

```
SELECT COUNT(*)
FROM Orders
WHERE Orderdatum BETWEEN '1/1/1997' AND '6/30/1997';
```

25. Wat is het totale bedrag aan orders van de ordernummers tussen 10248 en 10290? We moeten dan eerst hoeveelheid en prijs per eenheid uit de tabel Orderinformatie berekenen en dat totaliseren.

Oplossing Access

```
SELECT SUM([Prijs per eenheid] * Hoeveelheid) AS Totaalbedrag
FROM Orderinformatie
WHERE [Order-id] BETWEEN 10248 AND 10290
```

Oplossing SQL Server

```
SELECT SUM([Prijs per eenheid] * Hoeveelheid) AS Totaalbedrag
FROM Orderinformatie
WHERE [Order-id] BETWEEN 10248 AND 10290
```

26. Wat is het gemiddelde orderregelbedrag van de orders met ordernummers tussen 10248 en 10290?

Oplossing Access

```
SELECT AVG([Prijs per eenheid] * Hoeveelheid) AS Gembedrag
FROM Orderinformatie
WHERE [Order-id] BETWEEN 10248 AND 10290
```

Oplossing SQL Server

```
SELECT AVG([Prijs per eenheid] * Hoeveelheid) AS Gembedrag
FROM Orderinformatie
WHERE [Order-id] BETWEEN 10248 AND 10290
```

27. In hoeveel verschillende steden heeft Noordenwind klanten? In MS Acces moet deze met een subquery worden opgelost

```
SELECT COUNT(DISTINCT(Plaats))
FROM Klanten;
```

MS Access

```
SELECT COUNT(*)
FROM
(
SELECT DISTINCT Plaats
FROM Klanten
);
```

28. wat is het grootste orderregelbedrag? En het kleinste?

Oplossing Access

Maximum

```
SELECT MAX([Prijs per eenheid]*Hoeveelheid) AS MaxBedrag
FROM Orderinformatie;
```

Minimum

```
SELECT MIN([Prijs per eenheid]*Hoeveelheid) AS MinBedrag
FROM Orderinformatie;
```

Oplossing SQL Server

Maximum

```
SELECT MAX([Prijs per eenheid] * Hoeveelheid) AS MaxBedrag
FROM Orderinformatie;
```

Minimum

```
SELECT MIN([Prijs per eenheid] * Hoeveelheid) AS MinBedrag
FROM Orderinformatie;
```

29. Bereken het verschil in jaren tussen de datum in dienst en het huidige jaar

Oplossing Access

```
SELECT YEAR(DATE()) - YEAR(In dienst) AS VerschilInJaren
FROM werknemers
```

Oplossing SQL Server

```
SELECT YEAR(GETDATE()) – YEAR([In dienst]) AS VerschilInJaren
FROM werknemers
```

30. Maak een lijst van de dag en de maand waarop de werknemers jarig zijn. Sorteer de lijst op maand en daarbinnen op dag

```
SELECT Achternaam, Voornaam, MONTH(geboortedatum) AS Maand,
DAY(Geboortedatum) AS Dag
FROM Werknemers
ORDER BY 3, 4;
```

met casting en concateneren:

Oplossing Access

```
SELECT Achternaam & ,  & Voornaam & ,  & MONTH(geboortedatum) & - &
DAY(Geboortedatum) AS NaamVerjaardag
FROM Werknemers
ORDER BY 1;
```

Oplossing SQL Server

```
SELECT Achternaam + ,  + Voornaam + ,  + CAST(MONTH(geboortedatum) AS
VARCHAR) + '-'  + CAST(DAY(Geboortedatum) AS VARCHAR) AS NaamVerjaardag
FROM Werknemers
ORDER BY 1;
```

31. Maak een lijst van de klanten waarbij de bedrijfsnamen en plaatsnamen in hoofdletters worden weergegeven

Oplossing Access

```
SELECT UCASE(Bedrijf) AS Bedrijfsnaam, UCASE(Plaats) AS Vestigingsplaats
FROM Klanten;
```

Oplossing SQL Server

```
SELECT UPPER(Bedrijf) AS Bedrijfsnaam, UPPER(Plaats) AS Vestigingsplaats
FROM Klanten;
```

32. Maak een lijst van de ordernummers en de weeknummers waarin ze besteld zijn. Gebruik voor Access en SQL Server de datepart functie.

Oplossing Access

```
SELECT [Order-id], DATEPART(WW, Orderdatum, 2, 2) AS Weeknummer
FROM Orders;
```

Oplossing SQL Server

```
SELECT [Order-id], DATEPART(ISOWW, Orderdatum) AS Weeknummer
FROM Orders;
```

33. We willen een overzicht van het aantal orders dat in 1997 per klantnummer geplaatst is.

```
SELECT Klantnummer, COUNT(*)
FROM Orders
WHERE YEAR(Orderdatum) = 1997
GROUP BY Klantnummer
```

34. We willen een lijst met het aantal klanten per land

```
SELECT Land, COUNT(Klantnummer) AS Aantal
FROM Klanten
GROUP BY Land;
```

35. Maak een lijst van de verkochte hoeveelheid per productnummer

```
SELECT Productnummer, SUM(Hoeveelheid) AS Totaal
FROM Orderinformatie
GROUP BY Productnummer;
```

36. Geef een overzicht van de productnummers waarbij er voor meer dan 50.000 euro verkocht is

Oplossing Access

```
SELECT Productnummer, SUM([Prijs per eenheid]*Hoeveelheid) AS Totaalbedrag
FROM Orderinformatie
GROUP BY Productnummer
HAVING SUM([Prijs per eenheid]*Hoeveelheid) > 50000;
```

Oplossing SQl Server

```
SELECT Productnummer, SUM([Prijs per eenheid]*Hoeveelheid) AS Totaalbedrag
FROM Orderinformatie
GROUP BY Productnummer
HAVING SUM([Prijs per eenheid]*Hoeveelheid) > 50000;
```

37. Geef bij de vorige lijst ook de totale korting voor deze producten ([hoeveelheid]*[prijs per eenheid])*[korting])

Oplosssing Access

```
SELECT Productnummer, SUM(Prijs per eenheid*Hoeveelheid) AS Totaalbedrag,
SUM(([Prijs per eenheid]*Hoeveelheid)*(1-korting)) AS TotaalbedragMinKorting
FROM Orderinformatie
GROUP BY Productnummer
HAVING SUM([Prijs per eenheid]*Hoeveelheid) > 50000;
```

Oplosssing SQL Server

```
SELECT Productnummer, SUM([Prijs per eenheid]*Hoeveelheid) AS Totaalbedrag,
SUM(([Prijs per eenheid]*Hoeveelheid)*(1-korting)) AS TotaalbedragMinKorting
FROM Orderinformatie
GROUP BY Productnummer
HAVING SUM([Prijs per eenheid]*Hoeveelheid) > 50000;
```

38. Geef een lijst van klanten die nooit iets besteld hebben en dus niet in de ordertabel voorkomen

```
SELECT *
FROM Klanten
WHERE Klantnummer NOT IN
  (
  SELECT Klantnummer
  FROM Orders
  );
```

39. Maak een lijst van alle producten die op ordernummer 10248 zijn besteld

Oplossing Access

```
SELECT *
FROM Producten
WHERE Productnummer IN
  (
  SELECT Productnummer
  FROM Orderinformatie
  WHERE [Order-id] = 10248
  );
```

Oplossing SQl Server

```
SELECT *
FROM Producten
WHERE Productnummer IN
  (
  SELECT Productnummer
  FROM Orderinformatie
  WHERE [Order-id] = 10248
  );
```

40. Maak een lijst van klanten die in 1997 orders hebben geplaatst met een totaal bedrag van meer dan 5000 euro

Oplossing Access

```
SELECT *
FROM KLANTEN
WHERE Klantnummer IN
  (
  SELECT Klantnummer
  FROM Orders
  WHERE YEAR(Orderdatum) = 1997 AND [Order-id] IN
    (
    SELECT [Order-id]
    FROM Orderinformatie
    GROUP BY [Order-id]
    HAVING SUM([Prijs per eenheid] * Hoeveelheid) > 5000
    )
  );
```

Oplossing SQL Server

```
SELECT *
FROM KLANTEN
WHERE Klantnummer IN
  (
  SELECT Klantnummer
  FROM Orders
  WHERE YEAR(Orderdatum) = 1997 AND [Order-id]IN
    (
    SELECT [Order-id]
    FROM Orderinformatie
    GROUP BY [Order-id]
    HAVING SUM([Prijs per eenheid] * Hoeveelheid) > 5000
    )
  );
```

41. Maak een lijst van alle klanten waarvan de orders zijn genoteerd door Davolio

Oplossing Access

```
SELECT *
FROM KLANTEN
WHERE Klantnummer IN
  (
  SELECT Klantnummer
  FROM Orders
  WHERE [Werknemer-id]IN
    (
    SELECT [Werknemer-id]
    FROM Werknemers
    WHERE Achternaam = 'Davolio'
    )
  );
```

Oplossing SQL Server

```
SELECT *
FROM KLANTEN
WHERE Klantnummer IN
  (
  SELECT Klantnummer
  FROM Orders
  WHERE [Werknemer-id]IN
    (
    SELECT [Werknemer-id]
    FROM Werknemers
    WHERE Achternaam = 'Davolio'
    )
  );
```

42. Maak een lijst van categorieʌn waarvan er producten uit het assortiment zijn genomen

Oplossing Access

```
SELECT Categorienaam
FROM Categorieën
WHERE Categorienummer IN
  (
  SELECT Categorienummer
  FROM Producten
  WHERE [Uit assortiment] = True
  );
```

Oplossing SQL Server

```
SELECT Categorienaam
FROM Categorieën
WHERE Categorienummer IN
  (
  SELECT Categorienummer
  FROM Producten
  WHERE [Uit assortiment] = 'True'
  );
```

43. Wat is het goedkoopste product dat Noordenwind verkoopt?

Oplossing Access

```
SELECT Productnaam
FROM Producten
WHERE [Prijs per eenheid] IN
  (
  SELECT MIN([Prijs per eenheid])
  FROM Producten
  );
```

Oplossing SQl Server

```
SELECT Productnaam
FROM Producten
WHERE [Prijs per eenheid] IN
  (
  SELECT MIN([Prijs per eenheid])
  FROM Producten
  );
```

44. Welke werknemer is het langst in dienst?

Oplossing Access

```
SELECT Achternaam, Voornaam
FROM Werknemers
WHERE [In dienst] IN
  (
  SELECT MIN([In dienst])
  FROM Werknemers
  );
```

Oplossing SQL Server

```
SELECT Achternaam, Voornaam
FROM Werknemers
WHERE [In dienst] IN
  (
  SELECT MIN([In dienst])
  FROM Werknemers
  );
```

45. Wat is de grootste order qua totaal bedrag die ooit bij Noordenwind is geplaatst?

Oplossing Access

Met ALL

```
SELECT [Order-id]
FROM Orderinformatie
GROUP BY [Order-id]
HAVING SUM([Prijs per eenheid]*Hoeveelheid)  >= ALL
  (
  SELECT SUM([Prijs per eenheid]*Hoeveelheid) as MaxBedrag
  FROM Orderinformatie
  GROUP BY [Order-id]
  );
```

Met een subquery in de **FROM** component

```
SELECT [Order-id]
FROM Orderinformatie
GROUP BY [Order-id]
HAVING SUM([Prijs per eenheid]*Hoeveelheid)  IN
  (
  SELECT MAX(MaxBedrag)
  FROM
    (
    SELECT SUM([Prijs per eenheid]*Hoeveelheid) as MaxBedrag
    FROM Orderinformatie
    GROUP BY [Order-id]
    ) AS MaxBedrag
  );
```

Oplossing SQL Server

Met ALL

```
SELECT [Order-id]
FROM Orderinformatie
GROUP BY [Order-id]
HAVING SUM([Prijs per eenheid] * Hoeveelheid)  >= ALL
  (
  SELECT SUM([Prijs per eenheid] * Hoeveelheid) as MaxBedrag
  FROM Orderinformatie
  GROUP BY [Order-id]
  );
```

Met een subquery in de **FROM** component

```
SELECT [Order-id]
FROM Orderinformatie
GROUP BY [Order-id]
HAVING SUM([Prijs per eenheid] * Hoeveelheid)  IN
  (
  SELECT MAX(MaxBedrag)
  FROM
    (
    SELECT SUM([Prijs per eenheid] * Hoeveelheid) as MaxBedrag
    FROM Orderinformatie
    GROUP BY [Order-id]
    ) AS MaxBedrag
  );
```

46. Welke werknemer heeft er in de eerste maand van 1997 geen enkele order genoteerd?

Oplossing Access

```
SELECT *
FROM Werknemers
WHERE [Werknemer-id] NOT IN
  (
  SELECT [Werknemer-id]
  FROM Orders
  WHERE Orderdatum BETWEEN #1/1/1997# AND #1/31/1997#
  );
```

Oplossing SQL Server

```
SELECT *
FROM Werknemers
WHERE [Werknemer-id] NOT IN
  (
  SELECT [Werknemer-id]
  FROM Orders
  WHERE Orderdatum BETWEEN '1/1/1997' AND '1/31/1997'
  );
```

47. Geeft een lijst van producten die niet zijn verkocht in 1998.

Oplossing Access

```
SELECT *
FROM Producten
WHERE Productnummer NOT IN
  (
  SELECT Productnummer
  FROM Orderinformatie
  WHERE [Order-id] IN
    (
    SELECT [Order-id]
    FROM Orders
    WHERE Orderdatum BETWEEN #1/1/1998# AND #12/31/1998#
    )
  );
```

Oplossing SQL Server

```
SELECT *
FROM Producten
WHERE Productnummer NOT IN
  (
  SELECT Productnummer
  FROM Orderinformatie
  WHERE [Order-id] IN
    (
    SELECT [Order-id]
    FROM Orders
    WHERE Orderdatum BETWEEN '1/1/1998' AND '12/31/1998'
    )
  );
```

48. Geef een lijst van producten waarvan er in 1997 minder dan 1000 zijn verkocht

```
SELECT Productnaam
FROM Producten
WHERE Productnummer IN
 (
  SELECT Productnummer
  FROM Orderinformatie
  WHERE [Order-id] IN
  (
      SELECT [Order-id]
      FROM ORDERS
      WHERE YEAR(Orderdatum)=1997
  )
  GROUP BY Productnummer
  HAVING SUM(hoeveelheid)<1000
);
```

49. Selecteer alle gegevens van klanten die wel eens een product uit de categorie dranken hebben besteld

```
SELECT *
FROM Klanten
WHERE Klantnummer IN
  (
  SELECT Klantnummer
  FROM Orders
  WHERE [Order-id] IN
    (
    SELECT [Order-id]
    FROM Orderinformatie
    WHERE Productnummer IN
      (
      SELECT Productnummer
      FROM Producten
      WHERE Categorienummer IN
        (
        SELECT Categorienummer
        FROM Categorieën
        WHERE Categorienaam = 'Dranken'
        )
      )
    )
  )
```

50. Wat is het duurste product per categorie?

```
SELECT productnaam,[Prijs per eenheid]
FROM Producten AS P
WHERE [Prijs per eenheid] >= ALL
  (
  SELECT [Prijs per eenheid]
  FROM Producten AS PS
  WHERE PS.Categorienummer = P.Categorienummer
  )
```

51. Geef een overzicht van de producten met hun prijzen en totaal verkochte aantal

```
SELECT Productnaam, [Prijs per eenheid],
  (
  SELECT SUM(Hoeveelheid)
  FROM Orderinformatie AS O
  WHERE O.productnummer = P.productnummer
  ) AS AantalVerkocht
FROM Producten AS P;
```

52. Welke orderregels bevatten totaalbedragen die minder dan 10 procent van het gemiddelde bedragen?

```
SELECT ([Prijs per eenheid]*Hoeveelheid)  AS Totaal,
  (
  SELECT AVG([Prijs per eenheid]*Hoeveelheid)
  FROM Orderinformatie
  ) AS GemBedrag
FROM Orderinformatie
WHERE ([Prijs per eenheid]*Hoeveelheid)  /
  (
  SELECT AVG([Prijs per eenheid]*Hoeveelheid)
  FROM Orderinformatie
  )  < 0.1
```

53. Maak een lijst met het totale orderbedrag per werknemers

```
SELECT w.[Werknemer-id], w.Achternaam, SUM(oi.[Prijs per
eenheid]*oi.Hoeveelheid) AS OrderBedrag
FROM (Werknemers AS w INNER JOIN Orders AS o ON w.[Werknemer-id] =
o.[Werknemer-id]) INNER JOIN Orderinformatie AS oi ON o.[Order-id] =
oi.[Order-id]
GROUP BY w.[Werknemer-id], w.Achternaam
```

54. Geef een overzicht van de producten waarvan er in 1997 meer dan 500 verkocht werden

```
SELECT p.Productnaam, Sum(oi.Hoeveelheid) AS SomVanHoeveelheid
FROM (Producten AS p INNER JOIN Orderinformatie AS oi ON p.Productnummer =
oi.Productnummer) INNER JOIN Orders AS o ON o.[Order-id] = oi.[Order-id]
WHERE YEAR(o.orderdatum) = 1997
GROUP BY p.Productnaam
HAVING Sum(Hoeveelheid)>500;
```

55. Geef een overzicht van de categorieʌn die meer dan 10 verschillende producten kennen

```
SELECT c.Categorienaam, Count(Productnummer) AS AantalVanProductnummer
FROM Categorieën AS c INNER JOIN Producten AS p ON c.Categorienummer =
p.Categorienummer
GROUP BY c.Categorienaam
HAVING Count(Productnummer) > 10;
```

56. Geef de naam en de jaaromzet van de oudste vertegenwoordiger van Noordenwind

```
SELECT w.Achternaam, YEAR(o.Orderdatum) AS jaar, Sum(oi.[Prijs per
eenheid]*oi.hoeveelheid) AS JaarOmzet
FROM (Werknemers AS w INNER JOIN Orders AS O ON w.[Werknemer-id] =
o.[Werknemer-id]) INNER JOIN Orderinformatie AS oi ON o.[Order-id] =
oi.[Order-id]
WHERE w.Geboortedatum IN
  (
  SELECT MIN(Geboortedatum)
  FROM Werknemers
  WHERE Functie='Vertegenwoordiger'
  )
GROUP BY Achternaam, YEAR(o.Orderdatum);
```

57. Welke vertegenwoordiger had in 1997 de hoogste omzet?

```
SELECT TOP 1  Achternaam,  Sum(oi.[Prijs per eenheid]*oi.hoeveelheid) AS
JaarOmzet
FROM (Werknemers AS w INNER JOIN Orders AS O ON w.[Werknemer-id] =
o.[Werknemer-id]) INNER JOIN Orderinformatie AS oi ON o.[Order-id] =
oi.[Order-id]
WHERE  YEAR(o.Orderdatum) = 1997
AND Functie='Vertegenwoordiger'
GROUP BY w.Achternaam
ORDER BY 2 DESC;
```

58. De afdeling Promotie wil alle grote klanten een mailing sturen. Deze afdeling wil een lijst met daarop de namen en adressen van alle klanten die in 1997 een order hebben geplaatst van minimaal 8.500,-. Hoe kan deze lijst worden gemaakt?

Oplossing Access

```
SELECT klnaam, klplaats
FROM klantenbestand
WHERE klantnr IN
  (
  SELECT klantnr
  FROM orderbestand
  WHERE orderdat BETWEEN #01/01/98# AND #12/31/98#
  AND orderbdr >= 20000
  );
```

Oplossing SQL Server

```
SELECT klnaam, klplaats
FROM klantenbestand
WHERE klantnr IN
  (
  SELECT klantnr
  FROM orderbestand
  WHERE orderdat BETWEEN '01/01/98' AND '12/31/98'
  AND orderbdr >= 20000
  );
```

59. Kun je een lijst maken van alle artikelen die op de grootste order van 1997 zijn besteld? (Drie subqueries)

Oplossing Access

```
SELECT artnr, omschr
FROM artikelenbestand
WHERE artnr IN
  (
  SELECT artnr
  FROM orderregelsbestand
  WHERE ordernr IN
    (
    SELECT ordernr
    FROM orderbestand
    WHERE orderbdr IN
      (
      SELECT MAX(orderbdr)
      FROM orderbestand
      WHERE orderdat BETWEEN #01/01/1997# AND #12/31/1997#
      )
    )
  );
```

Oplossing SQL Server

```
SELECT artnr, omschr
FROM artikelenbestand
WHERE artnr IN
  (
  SELECT artnr
  FROM orderregelsbestand
  WHERE ordernr IN
    (
    SELECT ordernr
    FROM orderbestand
    WHERE orderbdr IN
      (
      SELECT MAX(orderbdr)
      FROM orderbestand
      WHERE orderdat BETWEEN '01/01/1997' AND '12/31/1997'
      )
    )
  );
```

60. De chef van de afdeling Verkoop is het niet eens met de vraag van oefening 1. Klanten die eenmalig een grote order plaatsen, zij niet altijd grote klanten. Hij zou liever een lijst hebben waarop de naam en adresgegevens zijn vermeld van alle klanten die in 1997 in totaal voor meer dan 50.000 bij Relate hebben gekocht.

Oplossing Access

```
SELECT *
FROM klantenbestand
WHERE klantnr IN
  (
  SELECT klantnr
  FROM orderbestand
  WHERE orderdat BETWEEN #01/01/97# AND  #12/31/97#
  GROUP BY klantnr
  HAVING SUM(orderbdr) > 50000
  );
```

Oplossing SQL Server

```
SELECT *
FROM klantenbestand
WHERE klantnr IN
  (
  SELECT klantnr
  FROM orderbestand
  WHERE orderdat BETWEEN '01/01/97' AND  '12/31/97'
  GROUP BY klantnr
  HAVING SUM(orderbdr) > 50000
  );
```

61. Dick de Bruin heeft bij het skeeleren zijn pols gebroken en zal twee weken lang niet bij klanten op bezoek kunnen gaan. Martin Cohen valt voor hem in. Hij wil graag een lijst met alle klanten van Dick waarop naam, adres en telefoonnummer staan vermeld. Rangschik de klanten alfabetisch.

```
SELECT klnaam, klstraat, klplaats, klnetn, kltel
FROM klantenbestand
WHERE klantnr IN
  (
  SELECT klantnr
  FROM orderbestand
  WHERE persnr IN
    (
    SELECT persnr
    FROM personeelsbestand
    WHERE anaam = 'Bruin'
    )
  )
ORDER BY 1;
```

62. De indeling van het magazijn is niet altijd overzichtelijk. De magazijnchef zou graag weten welk artikel en hoeveel daarvan ligt opgeslagen in de vakken VC 12 tot en met VC 16.

```
SELECT artnr, omschr, voorraad
FROM artikelenbestand
WHERE lokatie IN
  (
  SELECT lokatie
  FROM magazijnlokatie
  WHERE stellingcode = 'VC'
  AND vaknummer IN (12,13,14,15,16)
  );
```

63. De financiële administratie wil graag een overzicht van alle betalingen van de firma Bakker B.V. in Dordrecht in de maanden januari tot en met mei van 1998 (drie subqueries)

Oplossing Access:

```
SELECT *
FROM betalingenbestand
WHERE betdat BETWEEN #01/01/1997# AND #05/31/1997#
AND factuurnr IN
  (
  SELECT factuurnr
  FROM facturenbestand
  WHERE ordernr IN
    (
    SELECT ordernr
    FROM orderbestand
    WHERE klantnr IN
      (
      SELECT klantnr
      FROM klantenbestand
      WHERE klnaam LIKE 'Bakker*'
      AND klplaats = 'Dordrecht'
      )
    )
  );
```

Oplossing SQL Server:

```
SELECT *
FROM betalingenbestand
WHERE betdat BETWEEN '01/01/1997' AND '05/31/1997'
AND factuurnr IN
  (
  SELECT factuurnr
  FROM facturenbestand
  WHERE ordernr IN
    (
    SELECT ordernr
    FROM orderbestand
    WHERE klantnr IN
      (
      SELECT klantnr
      FROM klantenbestand
      WHERE klnaam LIKE 'Bakker%'
      AND klplaats = 'Dordrecht'
      )
    )
  );
```

64. Wat is het goedkoopste product dat Relate verkoopt?

```
SELECT *
FROM artikelenbestand
WHERE prijs IN
  (
  SELECT MIN(prijs)
  FROM artikelenbestand
  );
```

65. Welke werknemer is het langst in dienst bij Relate?

```
SELECT *
FROM personeelsbestand
WHERE startdat IN
  (
  SELECT MIN(startdat)
  FROM personeelsbestand
  );
```

66. Wat is uitgedrukt in geld, de grootste order die ooit bij Relate geplaatst is?

```
SELECT ordernr, orderdat, klantnr, orderbdr
FROM orderbestand
WHERE orderbdr IN
  (
  SELECT MAX(orderbdr)
  FROM orderbestand
  );
```

67. Welke verkoper heeft in de eerste twee weken van 1997 helemaal geen orders geboekt?

Oplossing Access:

```
SELECT vnaam, tv, anaam
FROM personeelsbestand
WHERE fte IN ('Vertegenwoordiger','Binnendienst','Chef verkoop')
AND persnr NOT IN
  (
  SELECT persnr
  FROM orderbestand
  WHERE orderdat BETWEEN #01/01/1997# AND #01/14/1997#
  );
```

Oplossing SQL Server:

```
SELECT vnaam, tv, anaam
FROM personeelsbestand
WHERE fte IN ('Vertegenwoordiger','Binnendienst','Chef verkoop')
AND persnr NOT IN
  (
  SELECT persnr
  FROM orderbestand
  WHERE orderdat BETWEEN '01/01/1997' AND '01/14/1997'
  );
```

68. De verkoopmanager denkt een soort regelmaat te hebben ontdekt in de ordergrootte. Vooral bij ййn klant is deze regelmaat opvallend; deze heeft klantnummer 100004. De manager zou nu graag weten hoeveel orders er zijn met een orderbedrag dat gelijk is aan een van de orderbedragen van die klant.

```
SELECT orderbdr, COUNT(*)
FROM orderbestand
WHERE klantnr <> 100004
AND orderbdr = ANY
  (
  SELECT orderbdr
  FROM orderbestand
  WHERE klantnr = 100004
  )
GROUP BY orderbdr;
```

69. Begin 1998 heeft Relate de administratiekosten voor kleine orders afgeschaft. Hoeveel geld heeft Relate hiermee in 1998 verspeeld? Een kleine order is een order onder de 200; de kosten die in rekening werden gebracht waren 23,50.

Oplossing Access:

```
SELECT COUNT(*)*50 AS verloren_geld
FROM orderbestand
WHERE orderbdr <500
AND orderdat BETWEEN #01/01/98# AND #12/31/98#;
```

Oplossing SQL Server:

```
SELECT COUNT(*)*50 AS verloren_geld
FROM orderbestand
WHERE orderbdr <500
AND orderdat BETWEEN '01/01/98' AND '12/31/98';
```

70. De klant Expert Computerservice heeft in totaal 6 kleine orders geplaatst. De vraag is of er andere klanten zijn die nog meer kleine orders hebben geplaatst. Toon de namen van die klanten.

```
SELECT klnaam
FROM klantenbestand
WHERE klantnr IN
  (
  SELECT klantnr
  FROM orderbestand
  WHERE orderbdr < 500
  GROUP BY klantnr
  HAVING COUNT(ordernr)>11
  );
```

71. Maak een lijst waarop per medewerker het PERSNR en het totaal aantal dagen verzuimd in 1999 komt te staan. Zet de medewerker met het hoogste verzuim bovenaan.

Oplossing Access:

```
SELECT persnr, SUM(einde-start) AS verzuimd
FROM verzuim
WHERE start > #01/01/1999#
AND einde < #12/31/1999#
GROUP BY persnr
ORDER BY 2 DESC;
```

Oplossing SQL Server:

```
SELECT persnr, SUM(DATEDIFF(d,start,einde)) AS verzuimd
FROM verzuim
WHERE start > '01/01/1999'
AND einde < '12/31/1999'
GROUP BY persnr
ORDER BY 2 DESC;
```

72. Maak een lijst van personeelsnummers die in 1999 meer verzuimd hebben dan Johan Boom.

Oplossing Access:

```
SELECT persnr, SUM (einde-start)
FROM verzuim
WHERE start >= #01/01/1999#
AND einde <= #12/31/1999#
GROUP BY persnr
HAVING SUM(einde-start)>
  (
  SELECT SUM(einde-start)
  FROM verzuim
  WHERE start >= #01/01/1999#
  AND einde <= #12/31/1999#
  AND persnr IN
    (
    SELECT persnr
    FROM personeelsbestand
    WHERE anaam = 'BOOM'
    )
  );
```

Oplossing SQL Server:

```
SELECT persnr, SUM(DATEDIFF(d,start,einde))
FROM verzuim
WHERE start >= '01/01/1999'
AND einde <= '12/31/1999'
GROUP BY persnr
HAVING SUM(DATEDIFF(d,start,einde))>
  (
  SELECT SUM(DATEDIFF(d,start,einde))
  FROM verzuim
  WHERE start >= '01/01/1999'
  AND einde <= '12/31/1999'
  AND persnr IN
    (
    SELECT persnr
    FROM personeelsbestand
    WHERE anaam = 'BOOM'
    )
  );
```

73. De chef van het magazijn wil opruiming houden. Zijn vraag is of er ook artikelen zijn die ooit wel zijn ingekocht, maar tot nu toe nooit zijn verkocht.

```
SELECT artnr, omschr
FROM artikelenbestand
WHERE artnr NOT IN
(SELECT artnr
FROM orderregelsbestand);
```

74. De vraag wordt nu als volgt gesteld: zijn er ook artikelen geweest waarvan er niet meer dan 1000 zijn verkocht?

```
SELECT artnr, omschr
FROM artikelenbestand
WHERE artnr NOT IN
(SELECT artnr
FROM orderregelsbestand
GROUP BY artnr
HAVING SUM([aantal geleverd])>1000);
```

75. De financiʌle administratie wil graag een overzicht van alle orders onder de 500. De lijst moet per order naast de orderdatum en het orderbedrag ook de naam, het adres en de plaats van vestiging van de klant tonen.

```
SELECT o.orderdat, o.orderbdr, k.klantnr, k.klnaam, k.klplaats
FROM klantenbestand AS k
INNER JOIN orderbestand AS o
ON k.klantnr=o.klantnr
WHERE orderbdr < 500;
```

76. De chef van de afdeling verkoop wil graag een overzicht van de prestaties van zijn vertegenwoordigers. De lijst moet de volledige naam van de vertegenwoordiger bevatten, zijn jaarsalaris, het totaal aantal orders dat de vertegenwoordiger in 1999 heeft geboekt en de omzet die hij daarmee gehaald heeft.

Oplossing Access:

```
SELECT p.vnaam, p.tv, p.anaam, salaris, COUNT(ordernr) AS [aantal orders],
SUM(orderbdr) AS [totaal orderbedrag]
FROM personeelsbestand AS p
INNER JOIN orderbestand AS o
ON p.persnr = o.persnr
WHERE fte = 'Vertegenwoordiger'
AND orderdat BETWEEN #01/01/1999# AND #12/31/1999#
GROUP BY p.vnaam, p.tv, p.anaam, salaris;
```

Oplossing SQL Server:

```
SELECT p.vnaam, p.tv, p.anaam, salaris, COUNT(ordernr) AS [aantal orders],
SUM(orderbdr) AS [totaal orderbedrag]
FROM personeelsbestand AS p
INNER JOIN orderbestand AS o
ON p.persnr = o.persnr
WHERE fte = 'Vertegenwoordiger'
AND orderdat BETWEEN '01/01/1999' AND '12/31/1999'
GROUP BY p.vnaam, p.tv, p.anaam, salaris;
```

77. De financiële administratie is op zoek naar de klant die de order met ordernummer 970019 heeft geplaatst. Maak een uitdraai waarop alle gegevens van de klant en de order inclusief de orderregels staan vermeld.

Oplossing Access:

```
SELECT k.*, o.*, r.*
FROM (klantenbestand AS k
INNER JOIN orderbestand AS o
ON o.klantnr = k.klantnr)
INNER JOIN orderregelsbestand AS r
ON r.ordernr = o.ordernr
WHERE o.ordernr = 970019;
```

Oplossing SQL SErver:

```
SELECT k.*, o.*, r.*
FROM klantenbestand AS k
INNER JOIN orderbestand AS o
ON o.klantnr = k.klantnr
INNER JOIN orderregelsbestand AS r
ON r.ordernr = o.ordernr
WHERE o.ordernr = 970019;
```

78. Breid de query van 77 zodanig uit dat ook de omschrijving van de gekochte artikelen worden getoond.

Oplossing Access:

```
SELECT k.*, o.*, r.*, a.omschr
FROM ((klantenbestand AS k
INNER JOIN orderbestand AS o
ON o.klantnr = k.klantnr)
INNER JOIN orderregelsbestand AS r
ON o.ordernr = r.ordernr)
INNER JOIN artikelenbestand AS a
ON o.artnr = r.artnr
WHERE o.ordernr = 970019;
```

Oplossing SQL Server:

```
SELECT k.*, o.*, r.*, a.omschr
FROM klantenbestand AS k
INNER JOIN orderbestand AS o
ON o.klantnr = k.klantnr
INNER JOIN orderregelsbestand AS r
ON o.ordernr = r.ordernr
INNER JOIN artikelenbestand AS a
ON o.artnr = r.artnr
WHERE o.ordernr = 970019;
```

79. De verkopers zouden graag een overzicht hebben waarop per artikelgroep staat vermeld welke artikelen er zijn en wat de prijs van die artikelen is. Vermeld ook het artikelnummer op de lijst. Per artikelgroep wil men graag de prijs oplopend gerangschikt zien.

```
SELECT g.artgroepnr, g.omschrijving, a.artnr, a.omschr, a.prijs
FROM artikelgroepen AS g
INNER JOIN artikelenbestand AS a
ON g.artgroepnr = a.artgroepnr
ORDER BY 1, 5;
```

80. De afdeling promotie wil graag een lijst van de artikelen die in 1998 tijdens de decembermaand zijn verkocht op basis van de artikelgroepen. De lijst moet dus de te onderscheiden artikelgroepen weergeven die in december 1998 zijn verkocht.

Oplossing Access:

```
SELECT DISTINCT g.artgroepnr, g.omschrijving AS artikelgroep
FROM ((artikelgroepen AS g
INNER JOIN artikelenbestand AS a
ON g.artgroepnr = a.artgroepnr)
INNER JOIN orderregelsbestand AS r
ON a.artnr = r.artnr)
INNER JOIN orderbestand AS o
ON r.ordernr = o.ordernr
WHERE o.orderdat BETWEEN #12/01/98# AND #12/31/98#;
```

Oplossing SQL Server:

```
SELECT DISTINCT g.artgroepnr, g.omschrijving AS artikelgroep
FROM ((artikelgroepen AS g
INNER JOIN artikelenbestand AS a
ON g.artgroepnr = a.artgroepnr)
INNER JOIN orderregelsbestand AS r
ON a.artnr = r.artnr)
INNER JOIN orderbestand AS o
ON r.ordernr = o.ordernr
WHERE o.orderdat BETWEEN '12/01/98' AND '12/31/98';
```

81. Voor het maken van een speciale mailing wil de afdeling promotie graag een lijst van klanten die in de decembermaand van 1998 beauty-artikelen, luxe artikelen of horloges hebben gekocht. De lijst moet het volledige klantadres bevatten en gerangschikt zijn op klantnaam en het soort artikel dat door de klant is besteld.

Oplossing Access:

```
SELECT DISTINCT k.klnaam, k.klstraat, k.klpostcode, k.klplaats,
g.omschrijving
FROM (((klantenbestand AS k
INNER JOIN orderbestand AS o
ON k.klantnr = o.klantnr)
INNER JOIN orderregelsbestand AS r
ON o.ordernr = r.ordernr)
INNER JOIN artikelenbestand AS a
ON r.artnr = a.artnr)
INNER JOIN artikelgroepen AS g
ON a.artgroepnr = g.artgroepnr
WHERE (omschrijving LIKE '*eauty*'
OR omschrijving LIKE '*uxe*'
OR omschrijving LIKE '*orloge*')
AND o.orderdat BETWEEN #12/01/1998# AND #12/31/1998#
ORDER BY 1, 5;
```

Oplossing SQL Server:

```
SELECT DISTINCT k.klnaam, k.klstraat, k.klpostcode, k.klplaats,
g.omschrijving
FROM klantenbestand AS k
INNER JOIN orderbestand AS o
ON k.klantnr = o.klantnr
INNER JOIN orderregelsbestand AS r
ON o.ordernr = r.ordernr
INNER JOIN artikelenbestand AS a
ON r.artnr = a.artnr
INNER JOIN artikelgroepen AS g
ON a.artgroepnr = g.artgroepnr
WHERE (omschrijving LIKE '%eauty%'
OR omschrijving LIKE '%uxe%'
OR omschrijving LIKE '%orloge%')
AND o.orderdat BETWEEN '12/01/1998' AND '12/31/1998'
ORDER BY 1, 5;
```

82. Een klant belt op over een order van 9 februari 1999. De financiʌle administratie wil alle gegevens van die order op papier hebben. Maak een lijst met daarop alle ordergegevens en de daarbij horende orderregels. Er is op die dag maar ййn order geplaatst.

Oplossing Access:

```
SELECT o.*, r.*
FROM orderbestand AS o
INNER JOIN orderregelsbestand AS r
ON r.ordernr = o.ordernr
WHERE o.orderdat = #02/09/99#;
```

Oplossing SQL Server:

```
SELECT o.*, r.*
FROM orderbestand AS o
INNER JOIN orderregelsbestand AS r
ON r.ordernr = o.ordernr
WHERE o.orderdat = '02/09/99';
```

83. Mick Dirksen van de afdeling verkoop wil een lijst van de grote klanten. Op deze lijst wil hij de voornaamste klantgegevens zoals naam, adres en telefoonnummer gevolgd door het aantal orders en het totale orderbedrag per klant in 2000. Een grote klant is een klant die in 2000 voor meer dan 45.000 heeft ingekocht. Zet de beste klant bovenaan de lijst.

Oplossing Access:

```
SELECT klnaam, klstraat, klpostcode, klplaats, klnetn, kltel, COUNT(ordernr)
AS aantal_orders, SUM(orderbdr) AS ordertotaal
FROM orderbestand AS o
INNER JOIN klantenbestand AS k
ON k.klantnr = o.klantnr
WHERE o.orderdat BETWEEN #01/01/99# AND #12/31/1999#
GROUP BY klnaam, klstraat, klpostcode, klplaats, klnetn, kltel
HAVING SUM(orderbdr)>100000
ORDER BY 8 DESC;
```

Oplossing SQL Server:

```
SELECT klnaam, klstraat, klpostcode, klplaats, klnetn, kltel, COUNT(ordernr)
AS aantal_orders, SUM(orderbdr) AS ordertotaal
FROM orderbestand AS o
INNER JOIN klantenbestand AS k
ON k.klantnr = o.klantnr
WHERE o.orderdat BETWEEN '01/01/99' AND '12/31/1999'
GROUP BY klnaam, klstraat, klpostcode, klplaats, klnetn, kltel
HAVING SUM(orderbdr)>100000
ORDER BY 8 DESC;
```

84. Maak een query die het totaal verkoopbedrag per verkopende medewerker in 1997 laat zien. Sorteer op functie en zet daarbij de beste verkoper bovenaan.

Oplossing Access:

```
SELECT vnaam, tv, anaam, fte, SUM(orderbdr) AS [omzet 1997]
FROM orderbestand AS o INNER JOIN personeelsbestand AS p
ON p.persnr = o.persnr
WHERE o.orderdat BETWEEN #01/01/97# AND #12/31/1997#
AND (fte = 'Binnendienst' OR fte = 'Vertegenwoordiger')
GROUP BY vnaam, tv, anaam, fte
ORDER BY 4, 5 DESC;
```

Oplossing SQL Server:

```
SELECT vnaam, tv, anaam, fte, SUM(orderbdr) AS [omzet 1997]
FROM orderbestand AS o INNER JOIN personeelsbestand AS p
ON p.persnr = o.persnr
WHERE o.orderdat BETWEEN '01/01/97' AND '12/31/1997'
AND (fte = 'Binnendienst' OR fte = 'Vertegenwoordiger')
GROUP BY vnaam, tv, anaam, fte
ORDER BY 4, 5 DESC;
```

85. Runners zijn artikelen die veel en regelmatig worden verkocht. Stel dat een artikel een runner is als er meer dan 1000 stuks per jaar worden verkocht. Welke artikelen van Relate waren in 1998 dan runners?

Oplossing Access:

```
SELECT a.artnr, omschr, SUM([aantal geleverd]) AS [verkocht in 1998]
FROM artikelenbestand AS a
INNER JOIN orderregelsbestand AS r
ON a.artnr = r.artnr
INNER JOIN orderbestand AS o
ON r.ordernr = o.ordernr
WHERE orderdat BETWEEN #01/01/98# AND #12/31/98#
GROUP BY a.artnr, omschr
HAVING SUM([aantal geleverd])>10000;
```

Oplossing Access:

```
SELECT a.artnr, omschr, SUM([aantal geleverd]) AS [verkocht in 1998]
FROM (artikelenbestand AS a
INNER JOIN orderregelsbestand AS r
ON a.artnr = r.artnr)
INNER JOIN orderbestand AS o
ON r.ordernr = o.ordernr
WHERE orderdat BETWEEN '01/01/98' AND '12/31/98'
GROUP BY a.artnr, omschr
HAVING SUM([aantal geleverd])>10000;
```

86. Maar een overzicht van alle kantoorartikelen waarvan er in 1997 meer dan 500 zijn verkocht.

Oplossing Access:

```
SELECT a.artnr, a.omschr, a.prijs
FROM ((artikelenbestand AS a
INNER JOIN artikelgroepen AS g
ON a.artgroepnr = g.artgroepnr)
INNER JOIN orderregelsbestand AS r
ON a.artnr = r.artnr)
INNER JOIN orderbestand AS o
ON r.ordernr = o.ordernr
WHERE g.omschrijving LIKE '*antoo*'
AND orderdat BETWEEN #01/01/97# AND #12/31/97#
GROUP BY a.artnr, a.omschr, a.prijs
HAVING SUM([aantal geleverd]) >500;
```

Oplossing SQL Server:

```
SELECT a.artnr, a.omschr, a.prijs
FROM artikelenbestand AS a
INNER JOIN artikelgroepen AS g
ON a.artgroepnr = g.artgroepnr
INNER JOIN orderregelsbestand AS r
ON a.artnr = r.artnr
INNER JOIN orderbestand AS o
ON r.ordernr = o.ordernr
WHERE g.omschrijving LIKE '%antoo%'
AND orderdat BETWEEN '01/01/97' AND '12/31/97'
GROUP BY a.artnr, a.omschr, a.prijs
HAVING SUM([aantal geleverd]) >500;
```

87. Mevrouw Doornbos van de financiële administratie heeft het idee dat de klanten steeds slechter gaan betalen. Daarom wil zij graag een overzicht waarop vermeld staat wat de gemiddelde betalingstermijn per jaar is. Met betalingstermijn bedoelen we de periode tussen het versturen van de factuur en het ontvangen van het totale factuurbedrag.

Oplossing Access:

```
SELECT YEAR(o.orderdat) AS jaar, AVG(b.betdat-f.factuurdat) AS
gem_betaaltermijn
FROM (orderbestand AS o
INNER JOIN facturenbestand AS f
ON  o.ordernr = f.ordernr)
INNER JOIN betalingenbestand AS b
ON f.factuurnr = b.factuurnr
GROUP BY YEAR(o.orderdat);
```

Oplossing SQL Server:

```
SELECT YEAR(o.orderdat) AS jaar, AVG(DATEDIFF(d,f.factuurdat, b.betdat)
AS gem_betaaltermijn
FROM (orderbestand AS o
INNER JOIN facturenbestand AS f
ON  o.ordernr = f.ordernr)
INNER JOIN betalingenbestand AS b
ON f.factuurnr = b.factuurnr
GROUP BY YEAR(o.orderdat);
```

88. Om een beter inzicht te krijgen in het betalingsgedrag wil Willy Doornbos bovendien nog eens per klant de maximale en minimale betalingstermijn van het jaar 1998 weten.

Oplossing Access:

```
SELECT YEAR(o.orderdat) AS jaar, k.klnaam, MAX(b.betdat-f.factuurdat) AS
grootste_betaaltermijn, MIN(b.betdat-f.factuurdat) AS kleinste_betaaltermijn
FROM ((orderbestand AS o
INNER JOIN facturenbestand AS f
ON o.ordernr = f.ordernr)
INNER JOIN betalingenbestand AS b
ON f.factuurnr = b.factuurnr)
INNER JOIN klantenbestand AS k
ON o.klantnr = k.klantnr
WHERE orderdat BETWEEN #01/01/98# AND #12/31/99#
GROUP BY YEAR(o.orderdat), k.klnaam
ORDER BY 1, 2;
```

Oplossing SQL Server:

```
SELECT YEAR(o.orderdat) AS jaar, k.klnaam, MAX(DATEDIFF(d, f.factuurdat,
b.betdat)) AS grootste_betaaltermijn, MIN(DATEDIFF(d, f.factuurdat,
b.betdat)) AS kleinste_betaaltermijn
FROM orderbestand AS o
INNER JOIN facturenbestand AS f
ON o.ordernr = f.ordernr
INNER JOIN betalingenbestand AS b
ON f.factuurnr = b.factuurnr
INNER JOIN klantenbestand AS k
ON o.klantnr = k.klantnr
WHERE orderdat BETWEEN '01/01/98' AND '12/31/99'
GROUP BY YEAR(o.orderdat), k.klnaam
ORDER BY 1, 2;
```

89. Welke facturen staan op dit moment langer dan twee weken open? Gebruik de actuele datum.

Oplossing Access:

```
SELECT f.factuurnr, f.factuurdat
FROM ((orderbestand AS o
INNER JOIN facturenbestand AS f
ON o.ordernr = f.ordernr)
INNER JOIN betalingenbestand AS b
ON f.factuurnr = b.factuurnr)
INNER JOIN klantenbestand AS k
ON o.klantnr = k.klantnr
WHERE DATE()-f.factuurdat > 14
AND b.betdat > DATE();
```

Oplossing SQL Server:

```
SELECT f.factuurnr, f.factuurdat
FROM orderbestand AS o
INNER JOIN facturenbestand AS f
ON o.ordernr = f.ordernr
INNER JOIN betalingenbestand AS b
ON f.factuurnr = b.factuurnr
INNER JOIN klantenbestand AS k
ON o.klantnr = k.klantnr
WHERE GETDATE()-f.factuurdat > 14
AND b.betdat > GETDATE();
```

90. Liselot van der Laan houdt de verjaardagen en de jubilea bij. Zij wil graag elke morgen even handig opvragen wie er de volgende dag jarig zijn. Daarnaast wil zij graag een overzicht kunnen maken waarop per werknemer het aantal dienstjaren staat vermeld met de dag en de maand van indiensttreding. Ontwerp de juiste queries.

Oplossing Access:

```
SELECT vnaam, tv, anaam
FROM personeelsbestand
WHERE MONTH(gebdat)=MONTH(DATE())
AND DAY(gebdat)=DAY(DATE()+1);
```

```
SELECT vnaam, tv, anaam, YEAR(DATE())-YEAR(startdat) AS dienstjaren,
DAY(startdat) AS jubileumdag, MONTH(startdat) AS jubileummaand
FROM personeelsbestand;
```

Oplossing SQL Server:

```
SELECT vnaam, tv, anaam
FROM personeelsbestand
WHERE MONTH(gebdat)=MONTH(GETDATE())
AND DAY(gebdat)=DAY(GETDATE()+1);
```

```
SELECT vnaam, tv, anaam, YEAR(GETDATE())-YEAR(startdat) AS dienstjaren,
DAY(startdat) AS jubileumdag, MONTH(startdat) AS jubileummaand
FROM personeelsbestand;
```

91. Toon de naam en de jaaromzet per jaar van de oudste vertegenwoordiger van Relate.

```
SELECT vnaam, tv, anaam, YEAR(orderdat) AS jaar, SUM(orderbdr) AS omzet
FROM personeelsbestand AS p
INNER JOIN orderbestand AS o
ON p.persnr = o.persnr
WHERE gebdat IN
  (
  SELECT MIN(gebdat)
  FROM personeelsbestand
  WHERE afd ='Verkoop'
  AND fte NOT IN ('Orderverwerking','Promotie')
  )
GROUP BY vnaam, tv, anaam, YEAR(orderdat);
```

92. Jaarlijks krijgt de vertegenwoordiger met de hoogste omzet van de directie een prijs. Toon de naam en de klanten van de beste vertegenwoordiger van Relate van 1999.

Oplossing Access:

```
SELECT DISTINCT vnaam, tv, anaam, klnaam
FROM (klantenbestand AS k
INNER JOIN orderbestand AS o
ON o.klantnr = k.klantnr)
INNER JOIN personeelsbestand AS p
ON o.persnr = p.persnr
WHERE fte = 'Vertegenwoordiger'
AND p.persnr IN
  (
  SELECT persnr
  FROM orderbestand
  WHERE orderdat BETWEEN #01/01/99# AND #12/31/99#
  GROUP BY persnr
  HAVING SUM(orderbdr) >= ALL
    (
    SELECT SUM(orderbdr)
    FROM orderbestand
    WHERE orderdat BETWEEN #01/01/99# AND #12/31/99#
    AND persnr IN
      (
      SELECT persnr
      FROM personeelsbestand
      WHERE fte = 'Vertegenwoordiger'
      )
    GROUP BY persnr
    )
  );
```

Oplossing SQL Server:

```
SELECT DISTINCT vnaam, tv, anaam, klnaam
FROM klantenbestand AS k
INNER JOIN orderbestand AS o
ON o.klantnr = k.klantnr
INNER JOIN personeelsbestand AS p
ON o.persnr = p.persnr
WHERE fte = 'Vertegenwoordiger'
AND p.persnr IN
  (
  SELECT persnr
  FROM orderbestand
  WHERE orderdat BETWEEN '01/01/99' AND '12/31/99'
  GROUP BY persnr
  HAVING SUM(orderbdr) >= ALL
    (
    SELECT SUM(orderbdr)
    FROM orderbestand
    WHERE orderdat BETWEEN '01/01/99' AND '12/31/99'
    AND persnr IN
      (
      SELECT persnr
      FROM personeelsbestand
      WHERE fte = 'Vertegenwoordiger'
      )
    GROUP BY persnr
    )
  );
```

93. In de oplossing van 85 kunnen we de regelmaat van de bestellingen niet controleren. Liever had de inkoopster een overzicht gehad van de artikelen waarvan er in 1998 meer dan 1000 per maand zijn verkocht.

Oplossing Access:

```
SELECT artnr, omschr
FROM artikelenbestand
WHERE NOT EXISTS
  (
  SELECT a.artnr,MONTH(orderdat),SUM([aantal geleverd])
  FROM (orderbestand AS o
  INNER JOIN orderregelsbestand AS ob
  ON o.ordernr = ob.ordernr)
  INNER JOIN artikelenbestand AS a
  ON ob.artnr = a.artnr
  WHERE orderdat BETWEEN #01/01/98# AND #12/31/98#
  GROUP BY a.artnr,MONTH(orderdat)
  HAVING SUM([aantal geleverd]) < 1000
  );
```

Oplossing SQL Server:

```
SELECT artnr, omschr
FROM artikelenbestand
WHERE NOT EXISTS
  (
  SELECT a.artnr,MONTH(orderdat),SUM([aantal geleverd])
  FROM orderbestand AS o
  INNER JOIN orderregelsbestand AS ob
  ON o.ordernr = ob.ordernr
  INNER JOIN artikelenbestand AS a
  ON ob.artnr = a.artnr
  WHERE orderdat BETWEEN '01/01/98' AND '12/31/98'
  GROUP BY a.artnr,MONTH(orderdat)
  HAVING SUM([aantal geleverd]) < 1000
  );
```

94. Het overzicht van oefening 87 verschaft te weinig duidelijkheid. Misschien veroorzaken enkele uitschieters het verschil in de gemiddelden. Daarom wil Willy Doornbos nu ook een overzicht van de gemiddelde betalingstermijn per klant voor het jaar 1999 en voor het jaar 1998 (slechtste betaler vooraan).

Oplossing Access:

```
SELECT YEAR(o.orderdat) AS jaar, k.klnaam, AVG(b.betdat-f.factuurdat) AS
gem_betaaltermijn
FROM ((orderbestand AS o
INNER JOIN facturenbestand AS f
ON o.ordernr = f.ordernr)
INNER JOIN betalingenbestand AS b
ON f.factuurnr = b.factuurnr)
INNER JOIN klantenbestand AS k
ON o.klantnr = k.klantnr
WHERE orderdat BETWEEN #01/01/98# AND #12/31/99#
GROUP BY YEAR(o.orderdat), k.klnaam
ORDER BY 1, 3 DESC;
```

Oplossing SQL Server:

```
SELECT YEAR(o.orderdat) AS jaar, k.klnaam, AVG(DATEDIFF(d,f.factuurdat,
b.betdat) AS gem_betaaltermijn
FROM orderbestand AS o
INNER JOIN facturenbestand AS f
ON o.ordernr = f.ordernr
INNER JOIN betalingenbestand AS b
ON f.factuurnr = b.factuurnr
INNER JOIN klantenbestand AS k ON o.klantnr = k.klantnr
WHERE orderdat BETWEEN '01/01/98' AND '12/31/99'
GROUP BY YEAR(o.orderdat), k.klnaam
ORDER BY 1, 3 DESC;
```

95. Maak een lijst van klanten en leveranciers met bedrijfsnaam, plaats en land en een zelfgemaakte kolom waarin we een K of L zetten

Oplossing Access:

```
SELECT Bedrijf, Plaats, Land, 'K' as Categorie
FROM Klanten
UNION
SELECT Bedrijf, Plaats, Land, 'L'
FROM Leveranciers
```

Oplossing SQL Server:

```
SELECT Bedrijf, Plaats, Land, 'K' as Categorie
FROM Klanten
UNION
SELECT Bedrijf, Plaats, Land, 'L'
FROM Leveranciers
```

96. Maak een lijst van plaatsnamen die zowel in de klanten als de leveranciers tabel voorkomen

Oplossing SQL Server

```
SELECT Plaats
FROM Klanten
INTERSECT
SELECT Plaats
FROM Leveranciers
```

97. Maak een lijst van plaatsen uit de klantentabel die niet in de leverancierstabel voorkomen

Oplossing SQL Server

```
SELECT Plaats
FROM Klanten
EXCEPT
SELECT Plaats
FROM Leveranciers
```

98. Maak een overzicht van plaatsen met alleen klanten en alleen leveranciers

Oplossing SQL Server

```
SELECT plaats, 'K' as typje
FROM klanten
EXCEPT
SELECT plaats, 'K' as typje
FROM leveranciers
UNION
SELECT plaats, 'L' as typje
FROM leveranciers
EXCEPT
SELECT plaats, 'L' as typje
FROM klanten
```

99. Maak een overzicht van de landen met alleen klanten, de landen met alleen leveranciers en de landen met beide op volgorde van land

Oplossing SQL Server

```
SELECT land, 'alleen klanten'
FROM
  (
  SELECT land
  FROM klanten
  EXCEPT
  SELECT land
  FROM leveranciers
  ) AS alleenklanten
UNION
SELECT land, 'alleen leveranciers'
FROM
  (
  SELECT land
  FROM leveranciers
  EXCEPT
  SELECT land
  FROM klanten
  ) AS alleenleveranciers
UNION
SELECT land, 'klanten en leveranciers'
FROM
  (
  SELECT land
  FROM leveranciers
  INTERSECT
  SELECT land
  FROM klanten
  ) AS klantenleveranciers
ORDER BY 1
```

100. Alle plaatsnamen in de klantentabel moeten voortaan met hoofdletters beschreven worden; maakt een **UPDATE** script om dit aan te passen

Oplossing Access

```
UPDATE Klanten SET Plaats = UCASE(Plaats);
```

Oplossing SQL Server

```
UPDATE Klanten SET Plaats = UPPER(Plaats);
```

101. Maak met **SELECT INTO** een kopie van de producten tabel en verhoog vervolgens alle prijzen met 10 procent

```
SELECT * INTO ProductenKopie
FROM Producten;
```

en

Oplossing Access

```
UPDATE ProductenKopie SET [Prijs per eenheid] = [Prijs per eenheid] * 0.1
```

Oplossing SQL Server

```
UPDATE ProductenKopie SET [Prijs per eenheid] = [Prijs per eenheid] * 0.1
```

102. Maak een kopie van de tabel orders en vervang alle lege leverdatums door de orderdatum + 30 dagen

```
SELECT * INTO OrdersKopie
FROM Orders;
```

en

```
UPDATE OrdersKopie SET Leverdatum = Orderdatum + 30
WHERE Leverdatum IS NULL;
```

103. Wis uit deze laatste kopie alle orders waarvan de levering korter dan 10 dagen heeft geduurd

```
DELETE FROM OrdersKopie
WHERE Leverdatum – Orderdatum < 10;
```

104. Maak een kopie van de klantentabel, leeg deze kopie vervolgens en voeg uit de tabellen leveranciers en klanten de naam van het bedrijf en de plaatsnaam toe aan deze kopie

```
SELECT * INTO KlantenKopie
FROM Klanten;
```

```
DELETE FROM KlantenKopie;
```

```
SELECT bedrijf, plaats INTO KlantenKopie
FROM Klanten;
```

```
SELECT bedrijf, plaats INTO KlantenKopie
FROM Leveranciers;
```