SQL-MySQL

Inleiding

Wat is SQL / MySQL?

SQL = Structured Query Language

SQL componenten:

1. **Data Manipulation Language(DML)**
   DML dient om de gegevens in een tabel te bekijken, om nieuwe gegevens in te voeren, bestaande gegevens te wijzigen of om gegevens uit de tabel te verwijderen.
   Commando's: Select, Insert, Update, Delete
2. **Data Definition Language(DDL)**
   DDL dient om de tabel zelf aan te maken, te wijzigen of om de tabel uit de database te verwijderen. Commando's: Create table, Alter table, Drop table
3. **Data Control Language(DCL)**
   DCL dient voor het toekennen van machtigingen. Commando's: Grant, revoke

MySQL is een opensource Relationele Database Managment Systeem (RDMS)

Installatie MySQL: Demo

Selecteren data

DML

DML (Data Manipulation Language) bevat een aantal statements die het mogelijk maken om gegevens uit een tabel te manipuleren.

- ✓ SELECT: bekijken van gegevens in een tabel zonder dat men de inhoud van de tabel gaat wijzigen.
- ✓ UPDATE: wijzigen van bestaande gegevens.
- ✓ DELETE: verwijderen van gegevens uit de tabel.
- ✓ INSERT: nieuwe gegevens invoeren in de tabel.

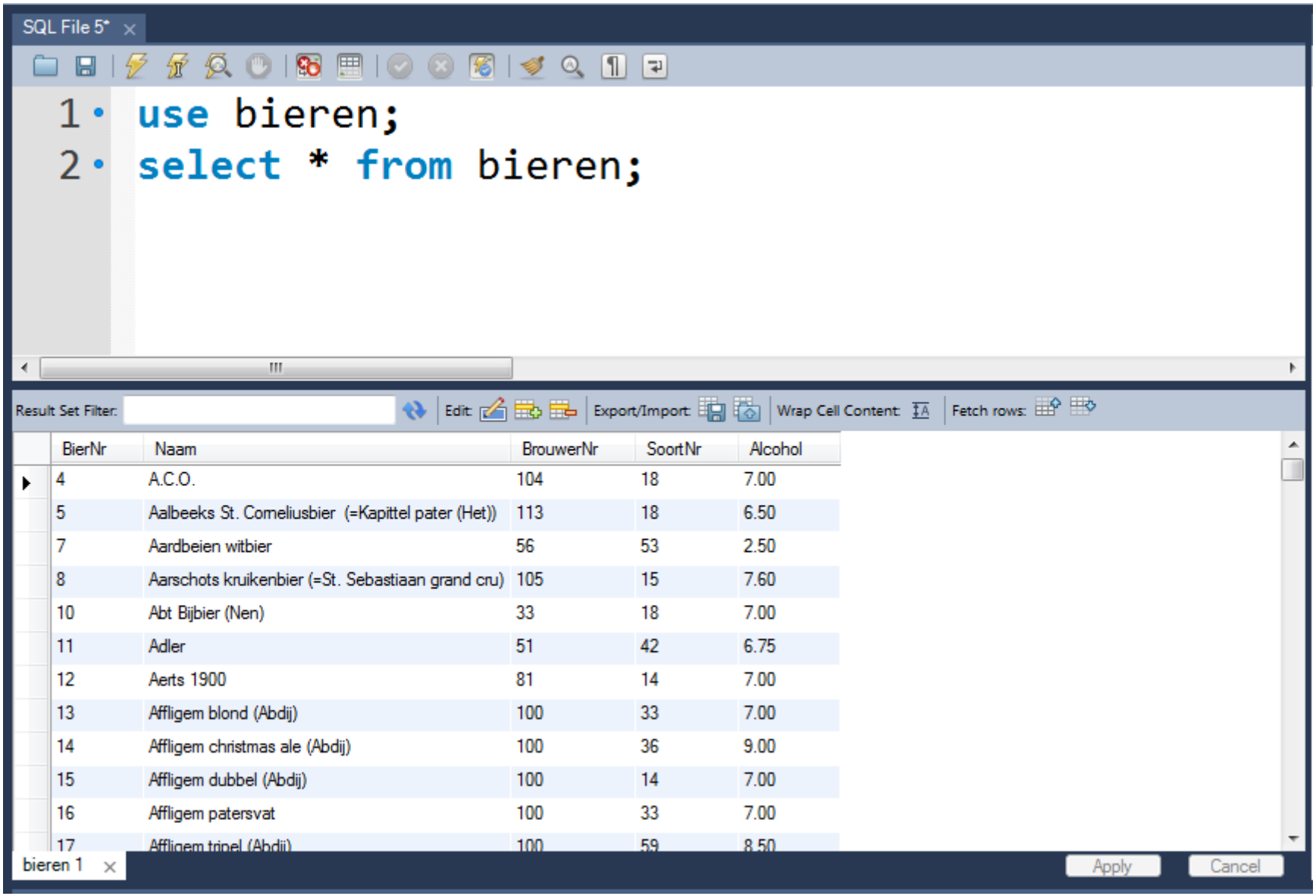
Query

Met SQL statements kan men tabellen aanmaken, veranderen of verwijderen. Men kan er ook de gegevens mee invoeren, verwijderen, veranderen of opvragen.

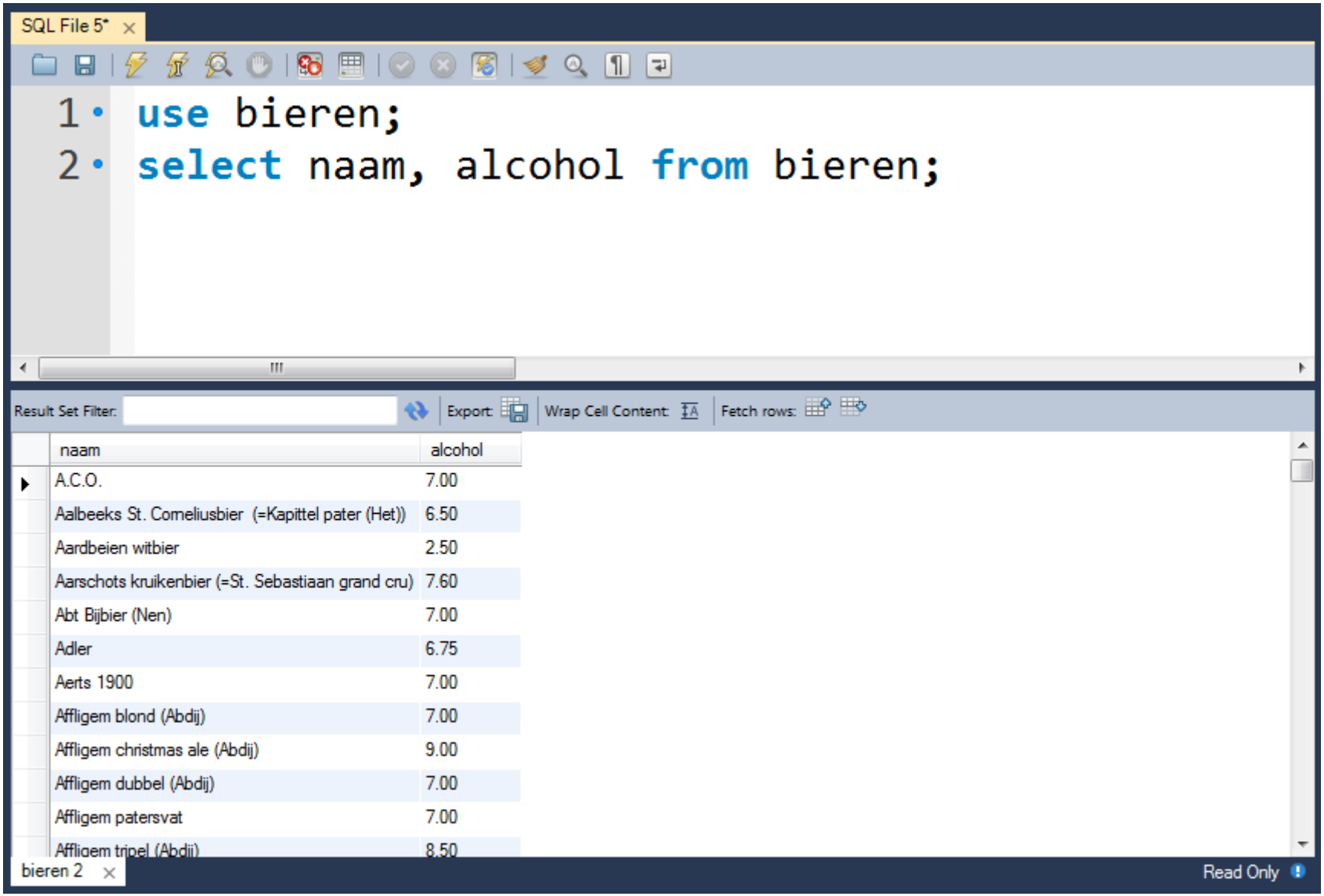
Vb
```
Select loon
From Loontab
Where stamnr = 4;
```

Zo'n SQL-programma noemt men een ***Query.***

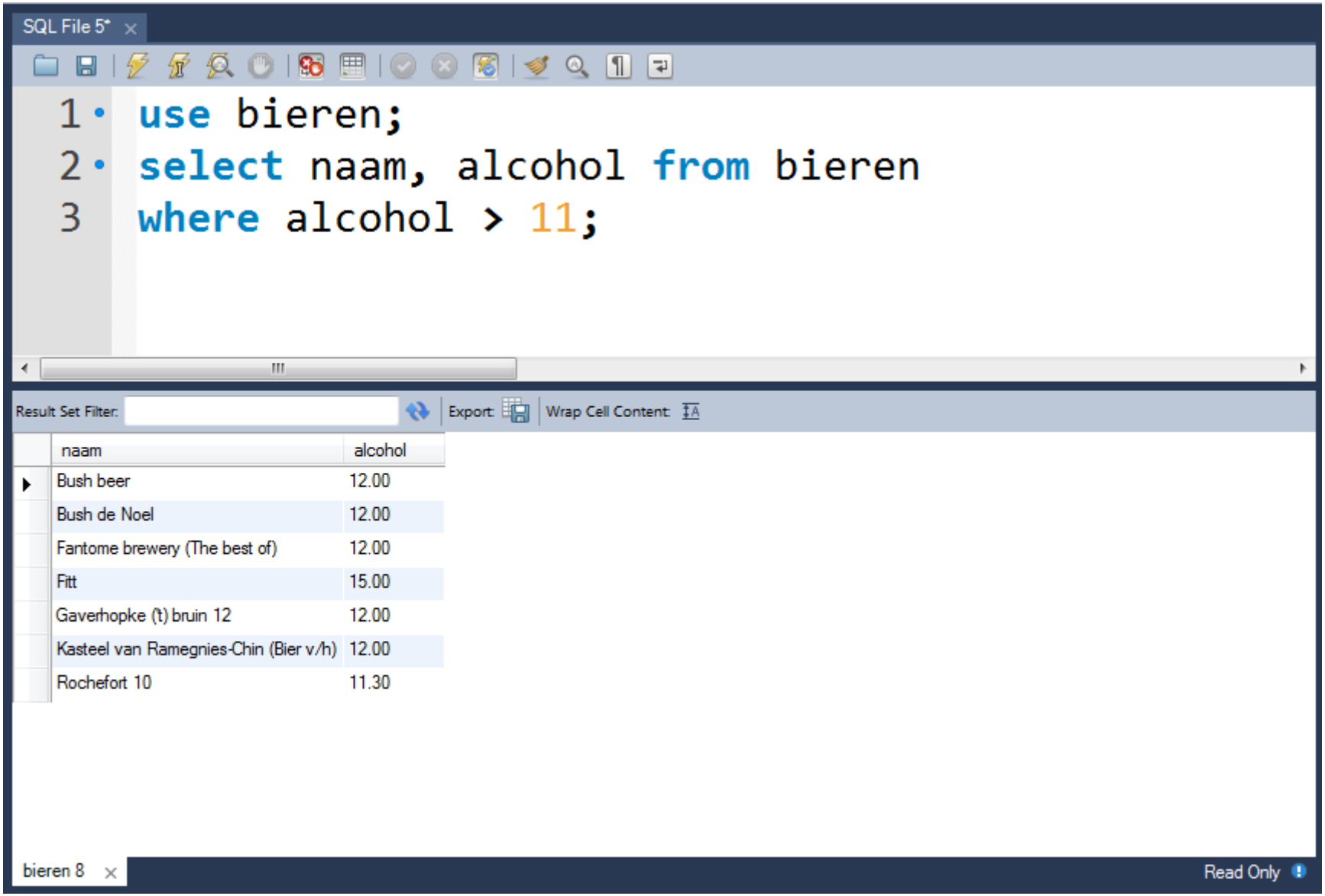
Lijst opvragen : alle velden



Lijst opvragen : selectie van velden



Selectie maken m.b.v. WHERE



Selectie maken m.b.v. WHERE

De **WHERE** clausule wordt gevolgd door een conditie. Een conditie is samengesteld uit:

attribuut operator constante* | *attribuut

De operator kan zijn:

- =, > , >=, <, <=, <> : Deze vergelijkingen zijn van toepassing op numerieke en alfanumerieke waarden (alfanumerieke waardes tussen enkele quotes).
- LIKE: Bij alfanumerieke waardes kan men karakterstrings afkorten met de wildcards % en _
  (% vervangt meerdere tekens en _ vervangt een enkel teken).

  Vb:
  SELECT Naam
  FROM Perstab
  WHERE Naam ***like*** 'P%' ; (geeft zowel Pol als Piet)
- AND, OR, NOT
  (uiteraard enkel van toepassing op booleaanse variabelen)

SELECT

- Na de SELECT mag men de kolomnaam ook laten voorafgaan door de tabelnaam en een .

vb:

```
SELECT perstab.naam
FROM perstab;
```

is hetzelfde als

```
SELECT naam
FROM perstab ;
```

- Zolang er geen twijfel bestaat van welke tabel een bepaalde kolom komt, mag men beide schrijfwijzen toepassen. Bestaat er wel twijfel, moet men de ‘uitgebreide versie’ gebruiken (zie verder join)

Selectie maken m.b.v. WHERE

```
SELECT naam
FROM bieren
WHERE alcohol < 5
```

geeft als resultaat een lijst met de naam van alle bieren met een alcoholpercentage lager dat 5%

```
SELECT brnaam
FROM brouwers
WHERE gemeente = 'Brussel'
```

geeft als resultaat een lijst van alle brouwerijen gelegen in Brussel

```
SELECT naam
FROM bieren
WHERE naam LIKE '%ale%'
```

geeft als resultaat een lijst van alle bieren waar het woord `ale` voorkomt in de naam.

WHERE: wildcards

Aard van selectie	Patroon	Waarden die in het patroon passen	Waarden die niet voldoen aan de voorwaarden
Meerdere karakters	a%a %ab% ab%	aa, aBa, aBBBa abc, AABB, Xab abcdefg, abc	aBC aZb, bac cab, aab
Speciaal teken	a[@]a	a@a	aaa
Eén karakter	a_a	aaa, a3a, aBa	aBBBa
Karakter moet voorkomen in de reeks	[a-z]	f, p, j	2, &
Karakter moet voorkomen in de benoemde lijst	[agm]	a, g, m	b, c, n, 4
Combinatie van hiervoor vermelde formaten	a[b-m]_	Ab9, af0	aacfd, a90

WHERE: wildcards

```
SELECT naam
FROM bieren
WHERE alcohol BETWEEN 5 AND 7
```

geeft een lijst van alle bieren met een alcoholpercentage vanaf 5% tot en met 7%. Merk op dat dit inclusief de grenswaarden is.

```
SELECT naam
FROM bieren
WHERE alcohol IN (0, 5, 8)
```

geeft een lijst van alle bieren met een alcoholpercentage van 0%, 5% of 8%.

```
SELECT brnaam
FROM brouwers
WHERE gemeente IN ('Leuven', 'Genk', 'Antwerpen', 'Dendermonde',
'Wevelgem')
```

geeft een lijst van alle brouwerijen gevestigd in de gemeenten Leuven, Hasselt, Genk, Antwerpen, Dendermonde en Wevelgem.

```
SELECT naam
FROM bieren
WHERE alcohol IS NULL
```

geeft een lijst van alle bieren waarvan de kolom alcohol niet ingevuld is.
De operator IS NULL geeft lege velden.
Om de kolommen te controleren die niet leeg zijn gebruik je IS NOT NULL.

LIMIT: Het aantal records beperken

De LIMIT clausule in SELECT laat ons toe het aantal records te beperken. LIMIT kan gebruikt worden met één of twee argumenten:

```
SELECT *
FROM TABLE
LIMIT aantalrecords
```

of

```
SELECT *
FROM TABLE
LIMIT voorbijRij aantalrecords
```

Voorbeelden:

```
SELECT *
FROM bieren
ORDER BY alcohol DESC
LIMIT 5
```

geeft de top vijf van meest alcoholische bieren.

LIMIT: Het aantal records beperken

LIMIT kan je ook met twee argumenten gebruiken waarbij het eerste argument de rij is vanaf waar getoond zal worden (niet inclusief) en het tweede argument het aantal rijen vanaf die plaats.
De allereerste rij van een recordset heeft de index 0.

```
SELECT *
FROM bieren
ORDER BY alcohol DESC
LIMIT 5, 10
```

geeft 10 bieren beginnend vanaf record 6.

Een gesorteerde lijst opvragen: ORDER BY

SQL File 5*

```
1  use bieren;
2  select naam, alcohol from bieren
3  order by naam asc;
```

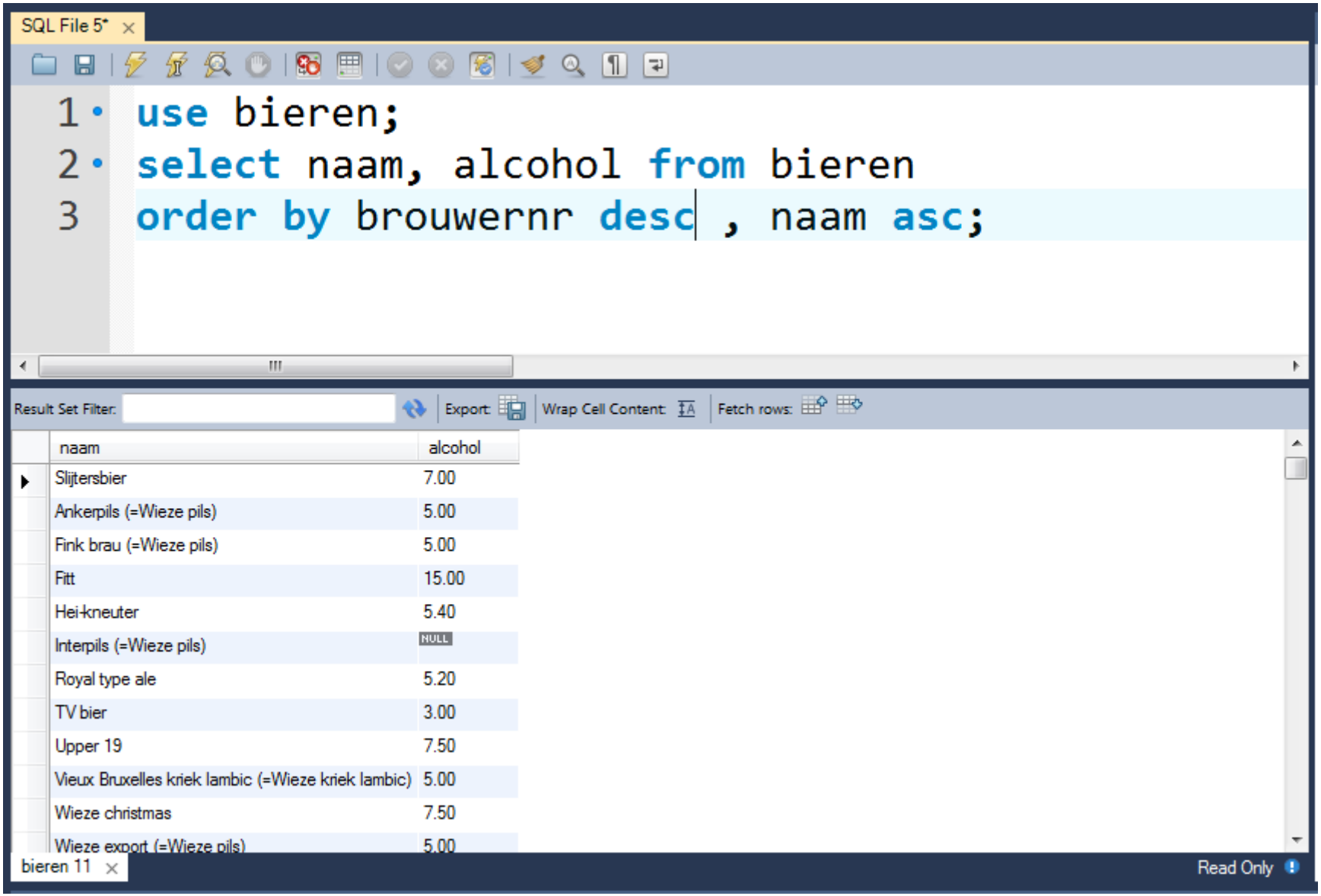
Result Set Filter: Export: Wrap Cell Content: Fetch rows:

naam	alcohol
A.C.O.	7.00
Aalbeeks St. Corneliusbier (=Kapittel pater (Het))	6.50
Aardbeien witbier	2.50
Aarschots kruikenbier (=St. Sebastiaan grand cru)	7.60
Abt Bijbier (Nen)	7.00
Adler	6.75
Aerts 1900	7.00
Affligem blond (Abdij)	7.00
Affligem christmas ale (Abdij)	9.00
Affligem dubbel (Abdij)	7.00
Affligem patersvat	7.00
Affligem tripel (Abdij)	8.50

bieren 10

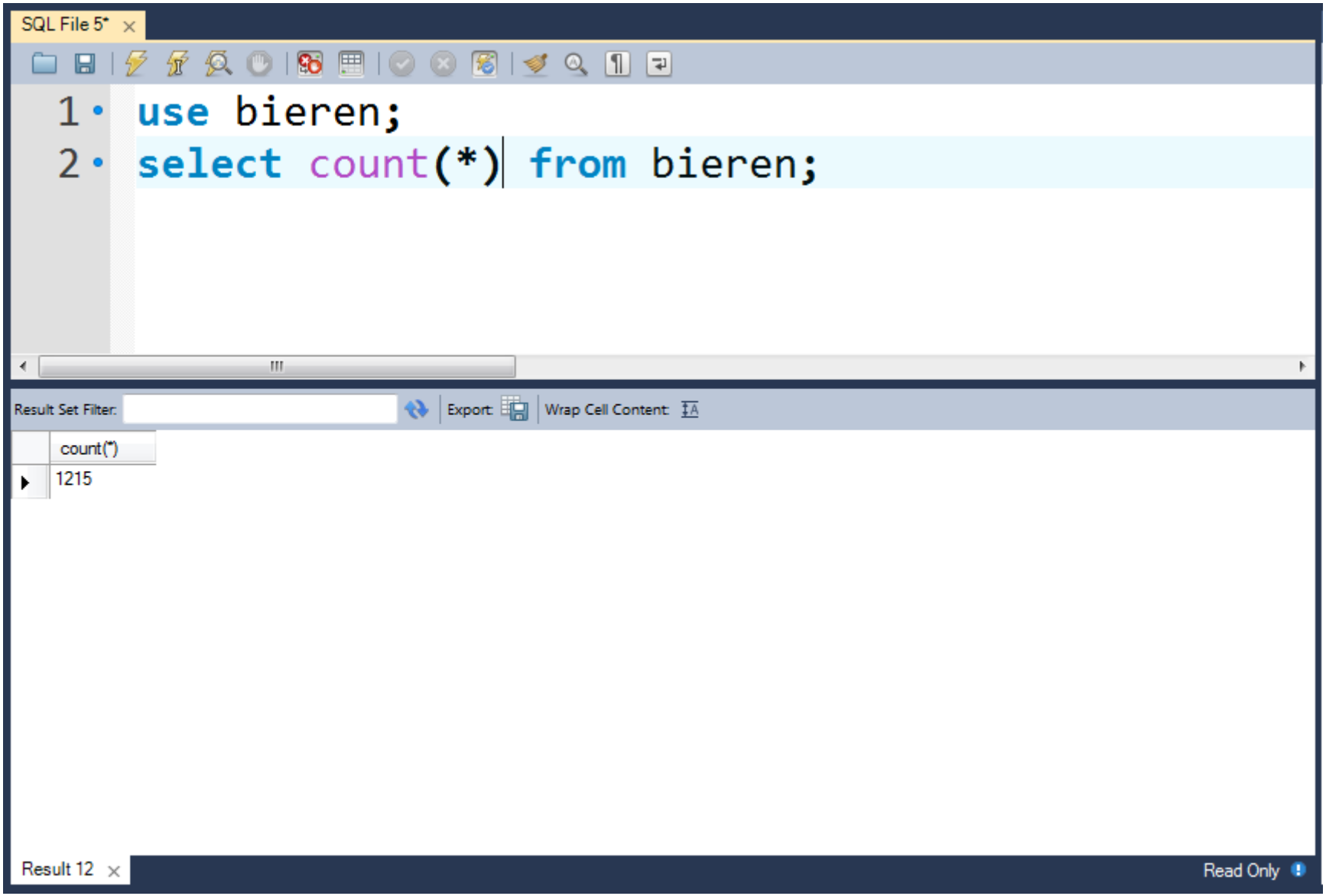
Read Only

Een gesorteerde lijst opvragen: ORDER BY

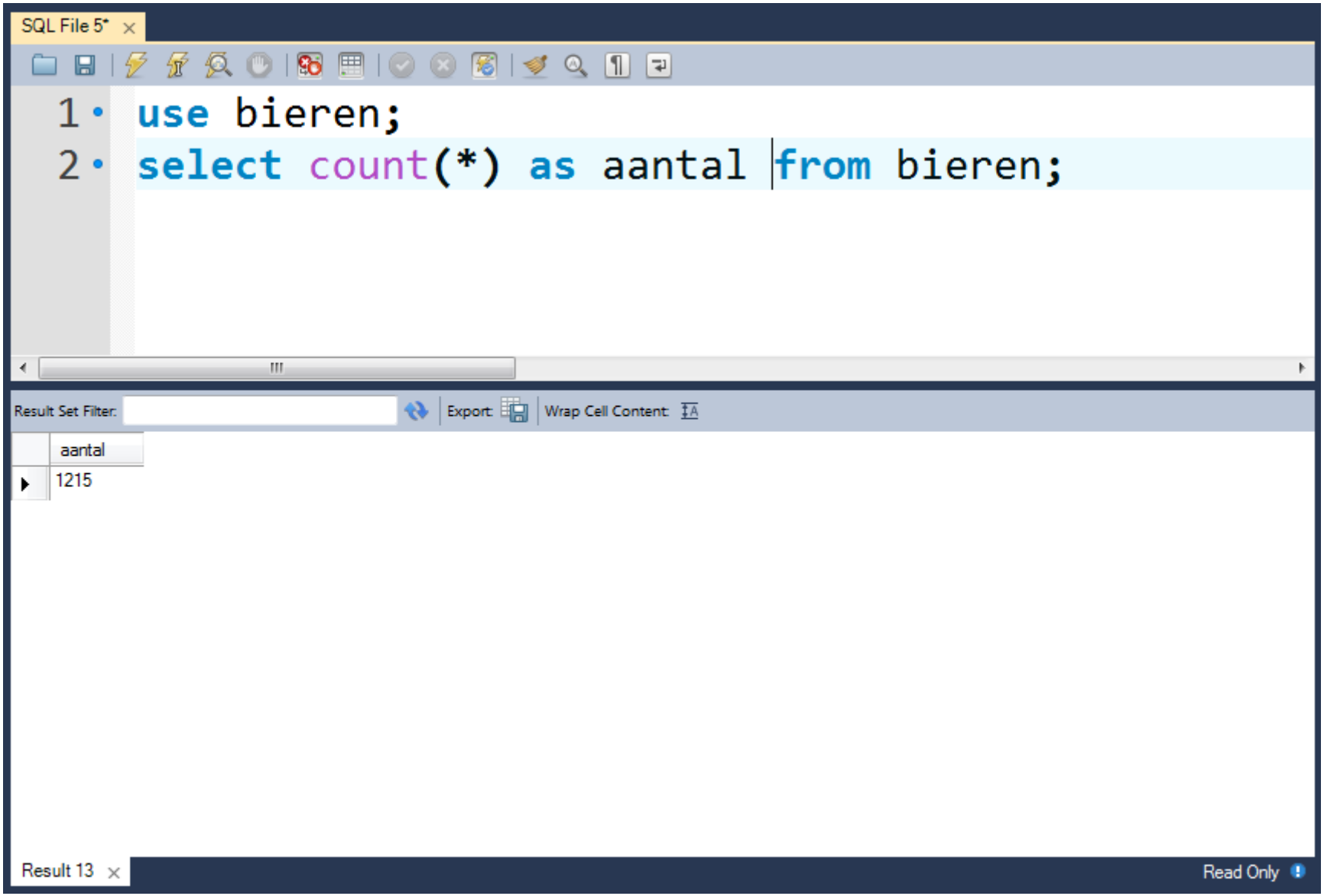


Oefeningen SELECT deel 1

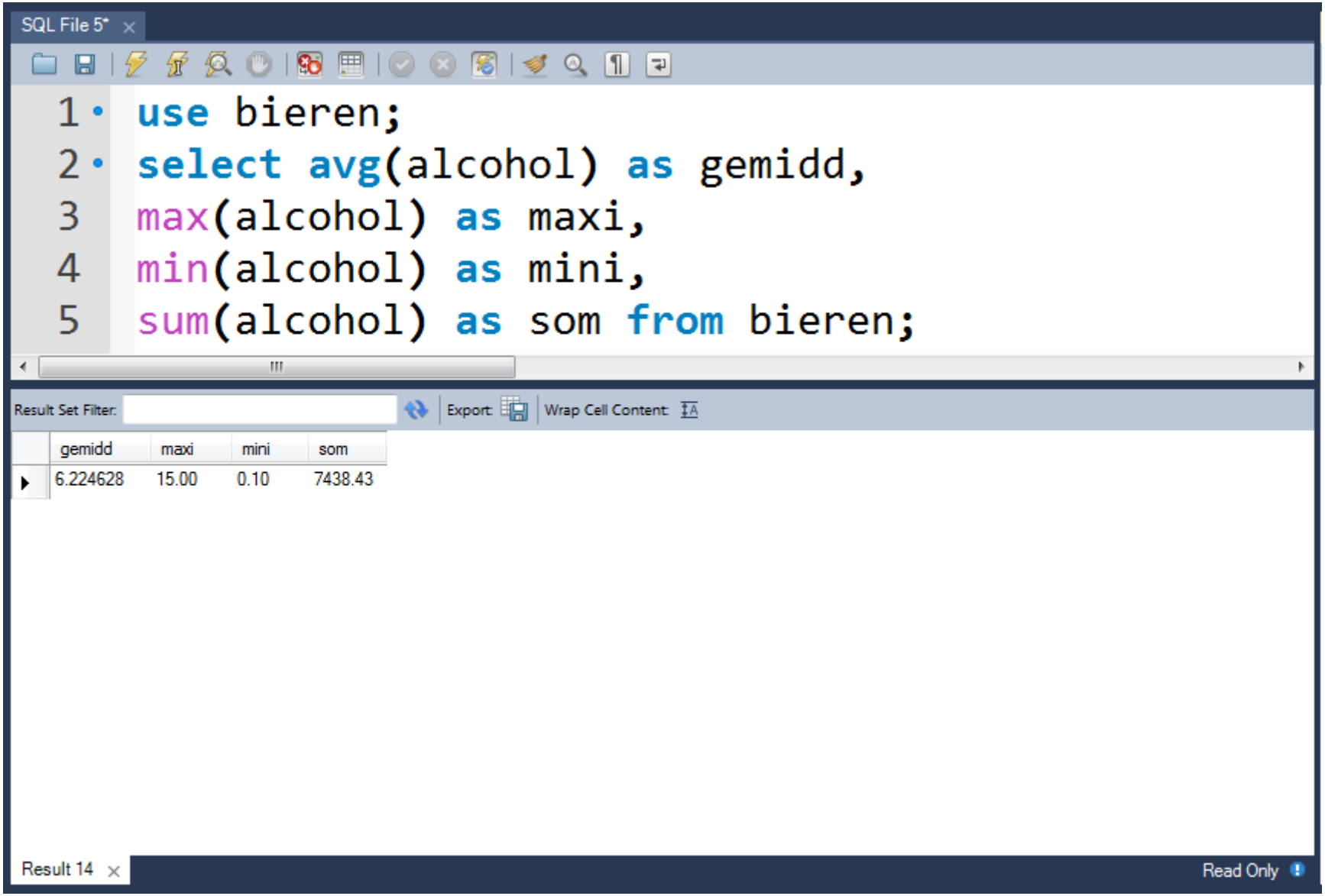
Aggregate functions



Aggregate functions



Aggregate functions



Berekeningen maken

SQL File 5*

```
1  use bieren;
2  select brnaam, omzet * 0.9118 as omzet_dollar,
3  omzet * 116.6 as omzet_yen from brouwers;
```

Result Set Filter: Export: Wrap Cell Content:

bmaam	omzet_dollar	omzet_yen
Achouffe	9118.0000	1166000.0
Alken	866210.0000	110770000.0
Ambly	455.9000	58300.0
Anker	2735.4000	349800.0
Artois	3647200.0000	466400000.0
Bavik	100298.0000	12826000.0
Belle Vue - Molenbeek	NULL	NULL
Belle Vue - Zuun	NULL	NULL
Belle Vue	273540.0000	34980000.0
Bie (De)	255.3040	32648.0
Binchoise	638.2600	81620.0
Bios	36472.0000	4664000.0

Result 15

Read Only

Oefeningen SELECT deel 2

Gegevens groeperen: GROUP BY

SQL File 5*

```
1  use bieren;
2  select brouwernr, avg(alcohol) as gemidd
3  from bieren
4  group by brouwernr;
```

Result Set Filter: Export: Wrap Cell Content:

brouwernr	gemidd
1	8.295556
2	3.011111
3	7.000000
4	7.050000
6	4.788235
8	4.742857
9	5.033333
10	5.000000
11	4.925000
12	7.375000
13	7.804167
14	7.467347

Result 16

Read Only

Gegevens groeperen: GROUP BY

OPGELET!

```
SELECT brouwernr, AVG(alcohol) AS gemidd
FROM bieren
GROUP BY brouwernr
```

berekent het gemiddelde alcoholpercentage per brouwernr

In de lijst van de te tonen kolommen mogen enkel bewerkingen met een aggregate functie staan en kolommen die vermeld staan na de group by.

```
SELECT art_code, art_lev, AVG(off_prijs)
FROM offertes
GROUP BY art_code
```

is niet toegelaten omdat bij art_lev geen aggregaat functie gebruikt wordt of omdat art_lev niet na de group by staat

Gegevens groeperen: HAVING

```
SELECT brouwernr, MIN(alcohol) AS mini
FROM bieren
GROUP BY brouwernr
HAVING MIN(alcohol)<5
```

→ aggregaat functie

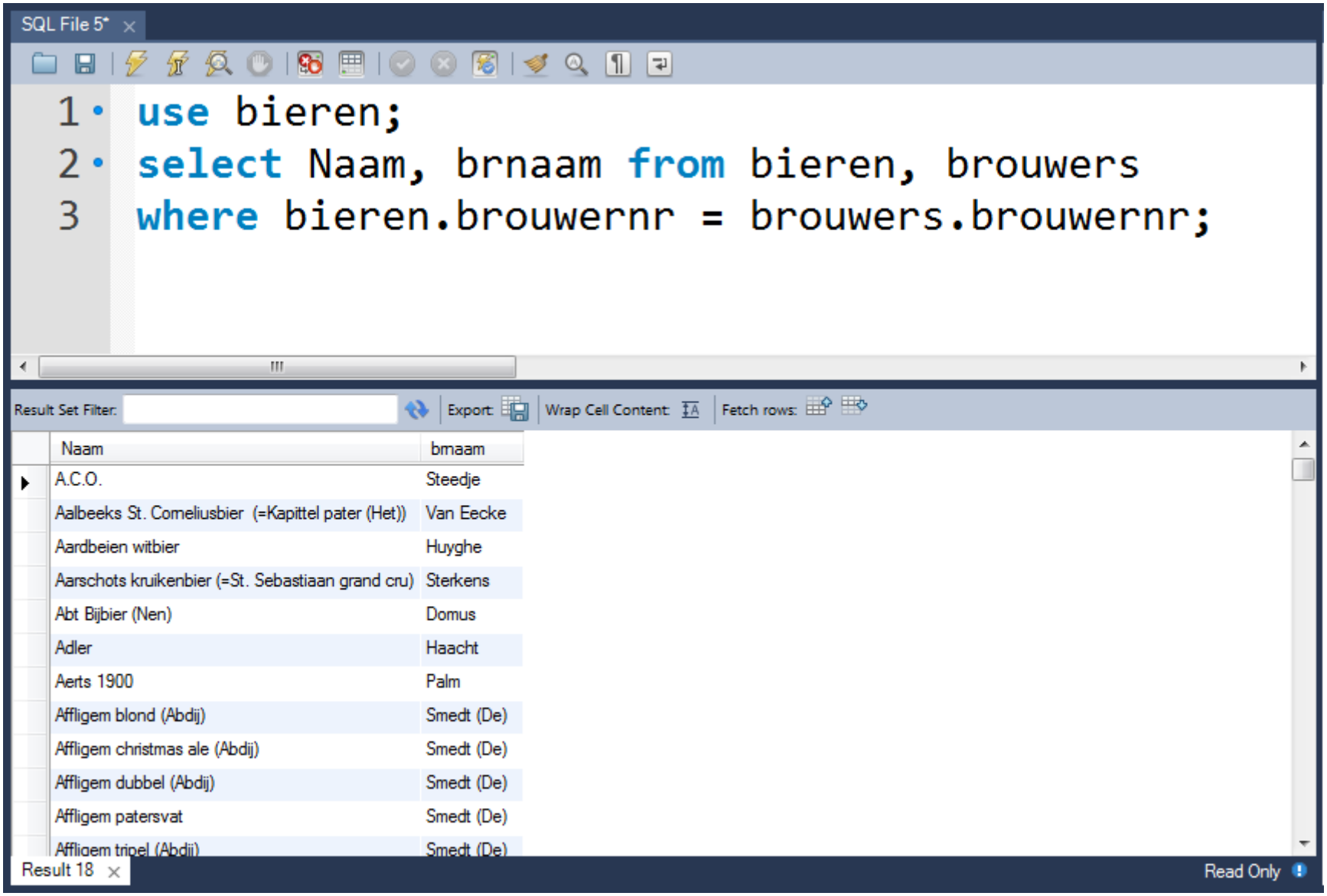
bepaalt het minimum alcoholpercentage per brouwernr, de lijst toont enkel de brouwernr's en percentages die kleiner zijn dan 5%. Je gebruikt "having" indien de selectie gebaseerd is op het resultaat van een bewerking met een aggregaat functie. In alle andere gevallen gebruik je "where".

```
SELECT brouwernr, AVG(alcohol) AS mini
FROM bieren
GROUP BY brouwernr
HAVING COUNT(*)>10
```

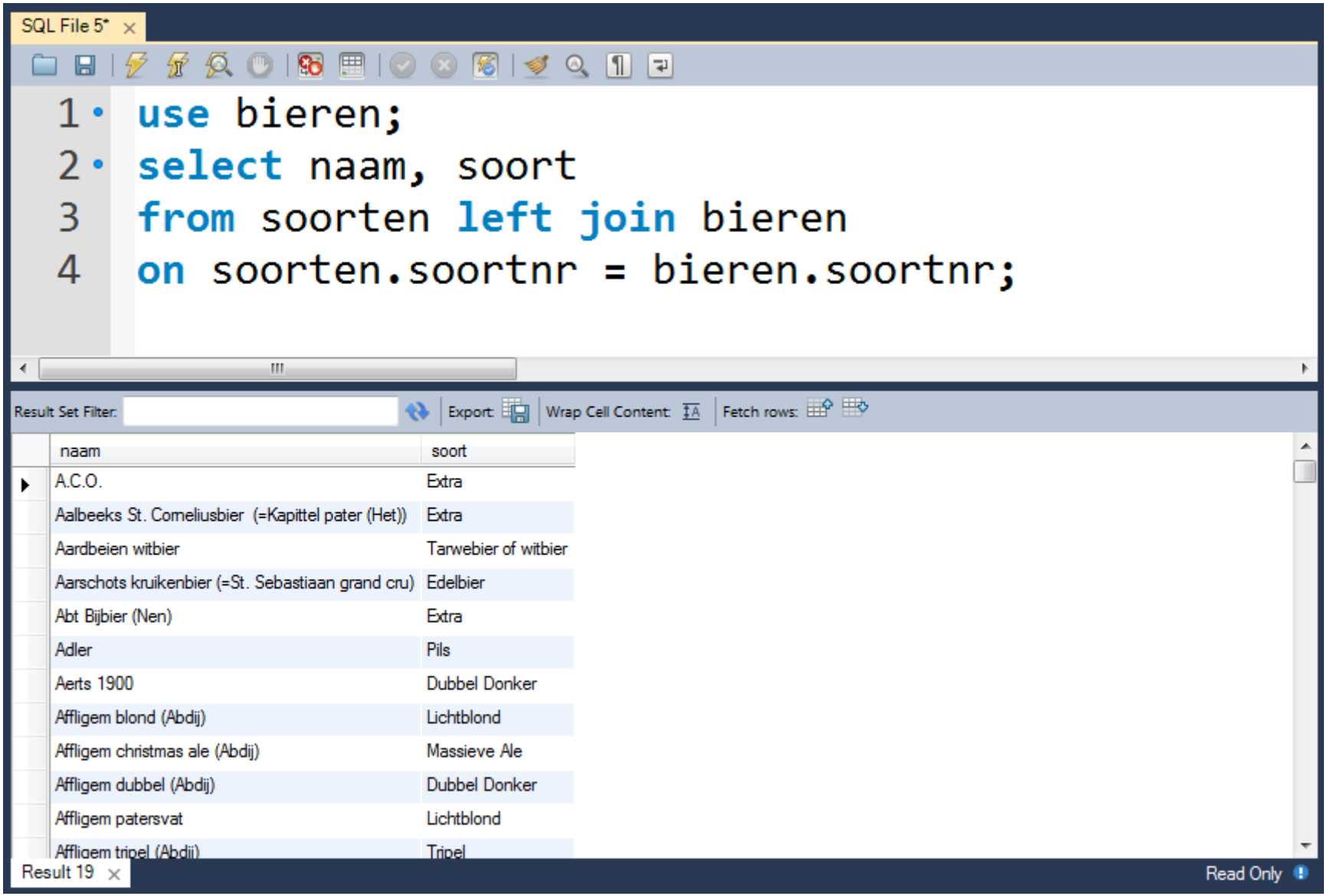
toont het gemiddelde alcoholpercentage per brouwernr voor alle brouwers die minimum 10 bieren produceren.

Oefeningen SELECT deel 3

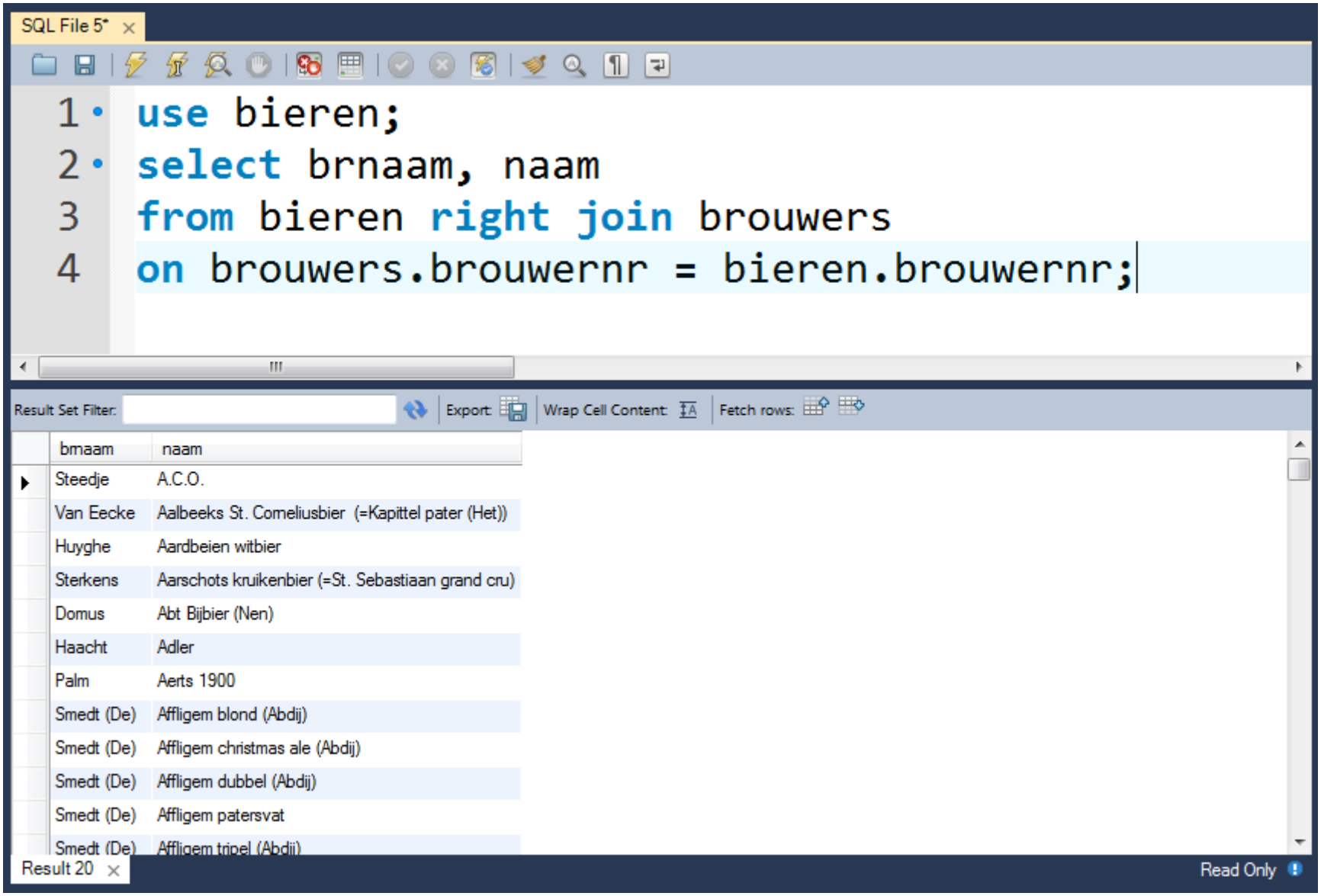
Uit meerdere tabellen tegelijkertijd



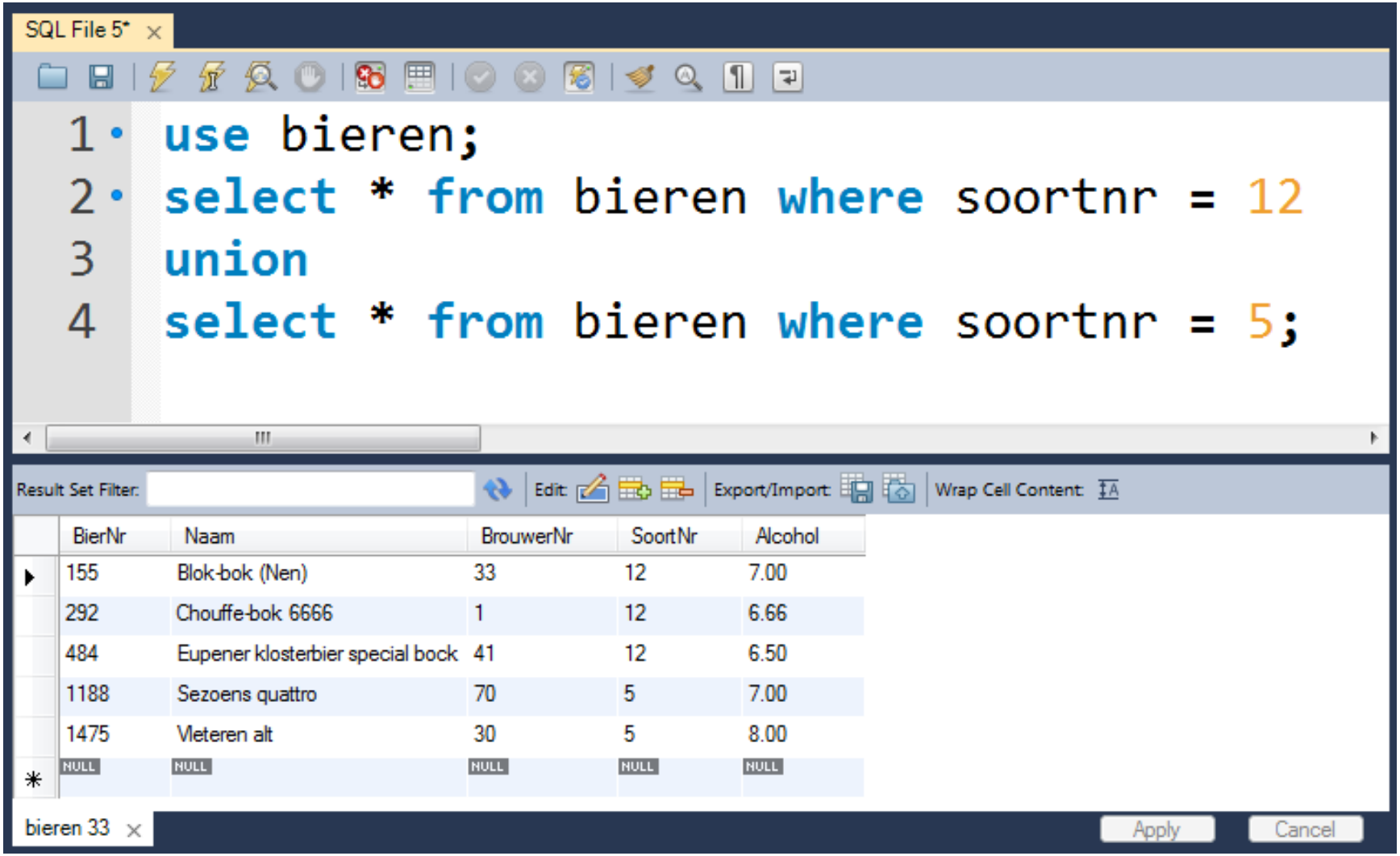
JOIN : left join



JOIN : right join

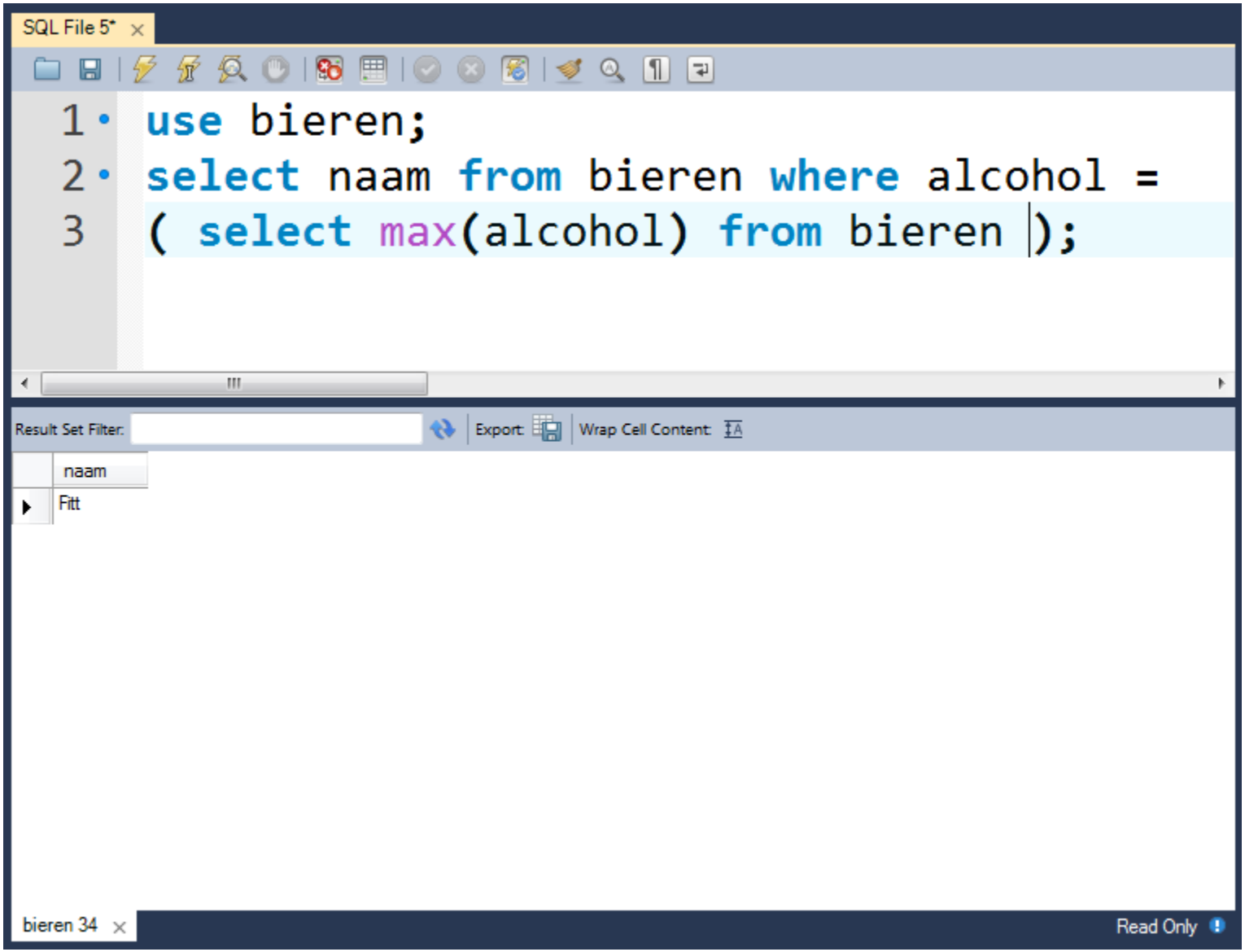


UNION

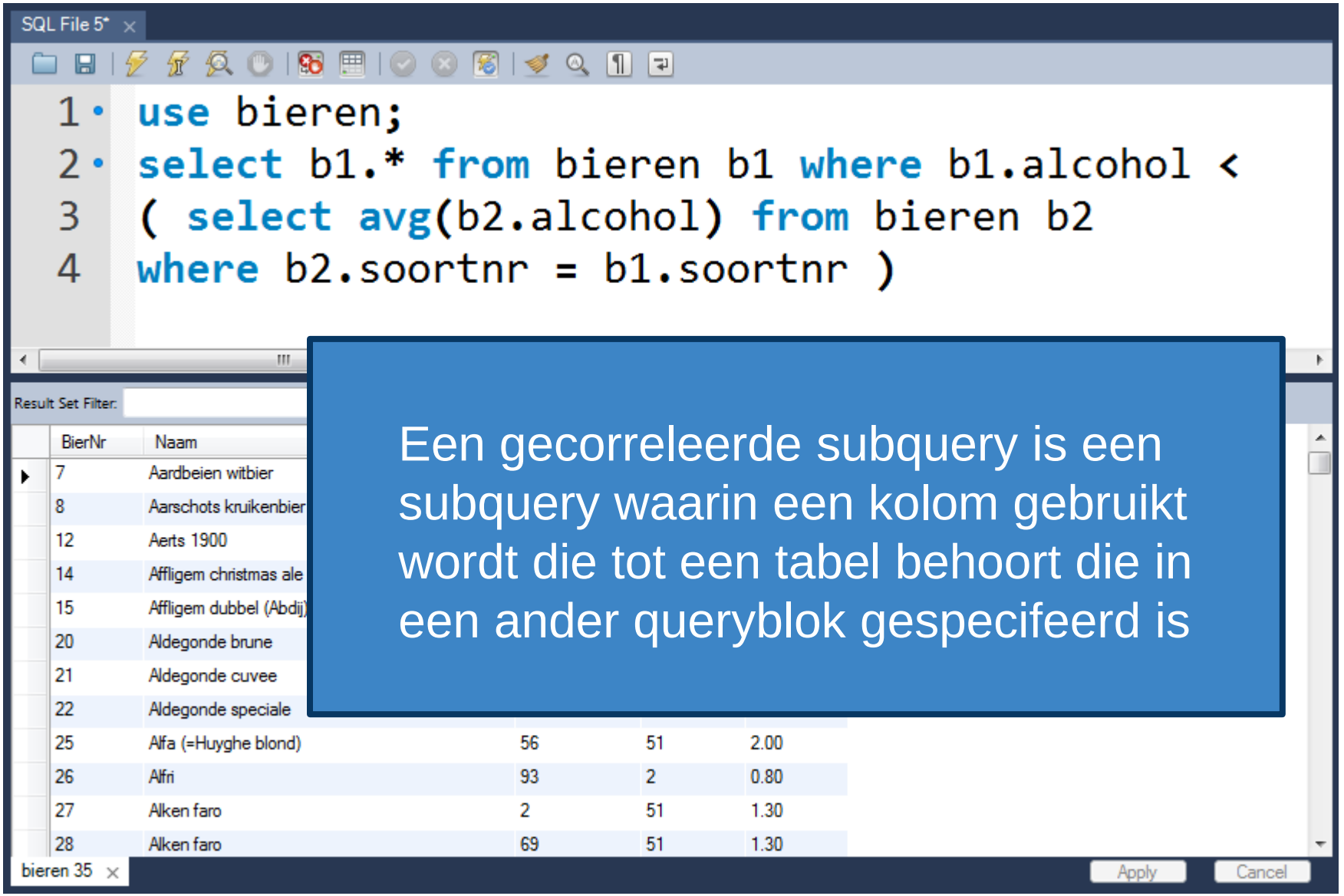

SQL File 5*
use bieren;
select * from bieren where soortnr = 12
union
select * from bieren where soortnr = 5;
Result Set Filter:
Edit:
Export/Import:
Wrap Cell Content:
BierNr Naam BrouwerNr SoortNr Alcohol
155 Blok-bok (Nen) 33 12 7.00
292 Chouffe-bok 6666 1 12 6.66
484 Eupener klosterbier special bock 41 12 6.50
1188 Sezoens quattro 70 5 7.00
1475 Vleteren alt 30 5 8.00
NULL NULL NULL NULL NULL
bieren 33
Apply
Cancel

Oefeningen SELECT deel 4

Subqueries


SQL File 5*
use bieren;
select naam from bieren where alcohol =
(select max(alcohol) from bieren);
Result Set Filter:
Export:
Wrap Cell Content:
naam
Fitt
bieren 34
Read Only

Gecorreleerde subqueries



Een gecorreleerde subquery is een subquery waarin een kolom gebruikt wordt die tot een tabel behoort die in een ander queryblok gespecifeerd is

Gecorreleerde subqueries

SQL File 5*

```
1  use bieren;
2  select b1.* from bieren b1 where b1.alcohol <
3  ( select avg(b2.alcohol) from bieren b2
4  where b2.soortnr = b1.soortnr )
```

Result Set Filter: Edit: Export/Import: Wrap Cell Content:

BierNr	Naam	BrouwerNr	SoortNr	Alcohol
7	Aardbeien witbier	56	53	2.50
8	Aarschots kruikenbier (=St. Sebastiaan grand cru)	105	15	7.60
12	Aerts 1900	81	14	7.00
14	Affligem christmas ale (Abdij)	100	36	9.00
15	Affligem dubbel (Abdij)	100	14	7.00
20	Aldegonde brune	72	36	8.50
21	Aldegonde cuvee	72	15	7.50
22	Aldegonde speciale	72	36	8.50
25	Alfa (=Huyghe blond)	56	51	2.00
26	Alfri	93	2	0.80
27	Alken faro	2	51	1.30
28	Alken faro	69	51	1.30

bieren 35

Apply Cancel

Oefeningen SELECT deel 5

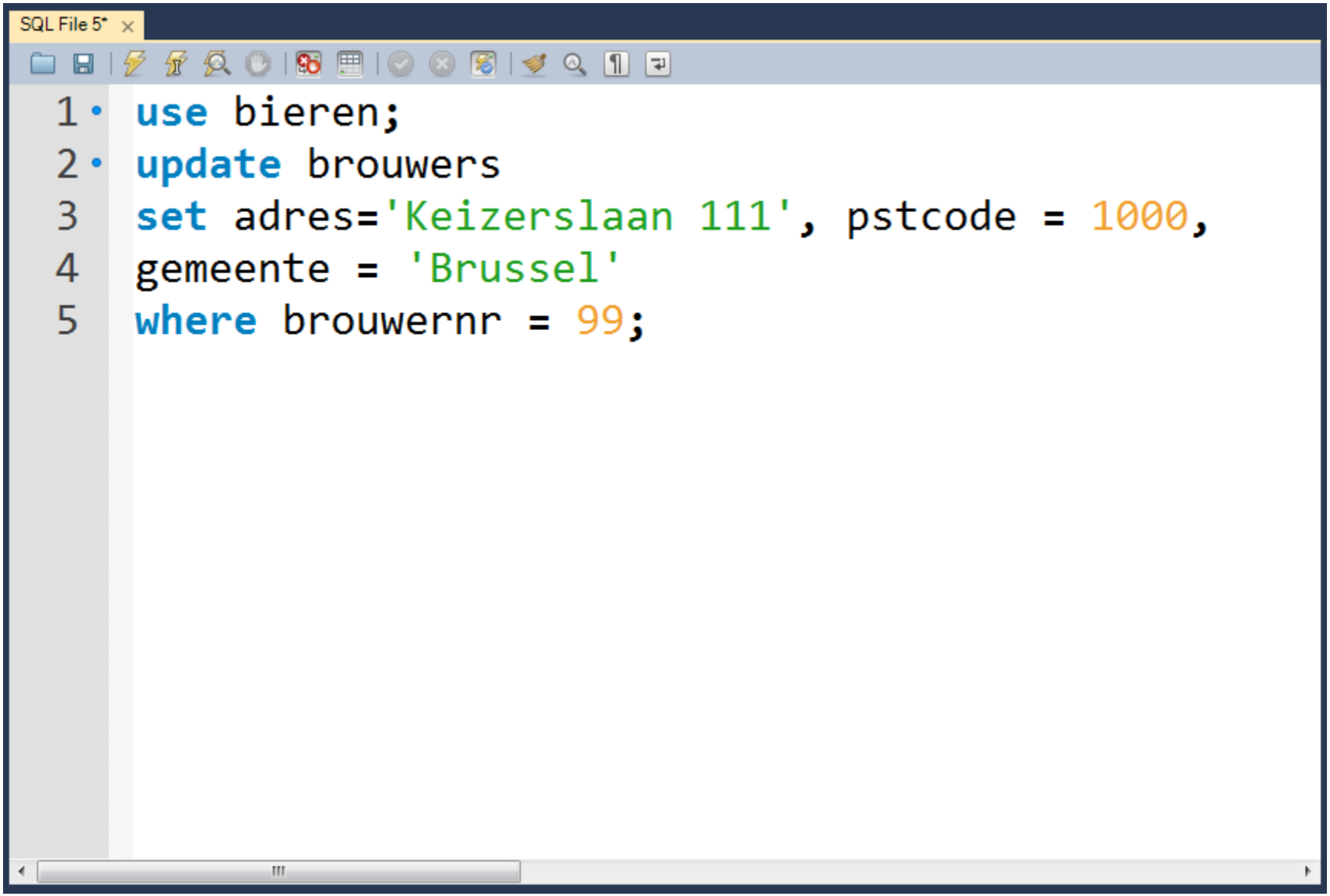
Aanpassen data

INSERT

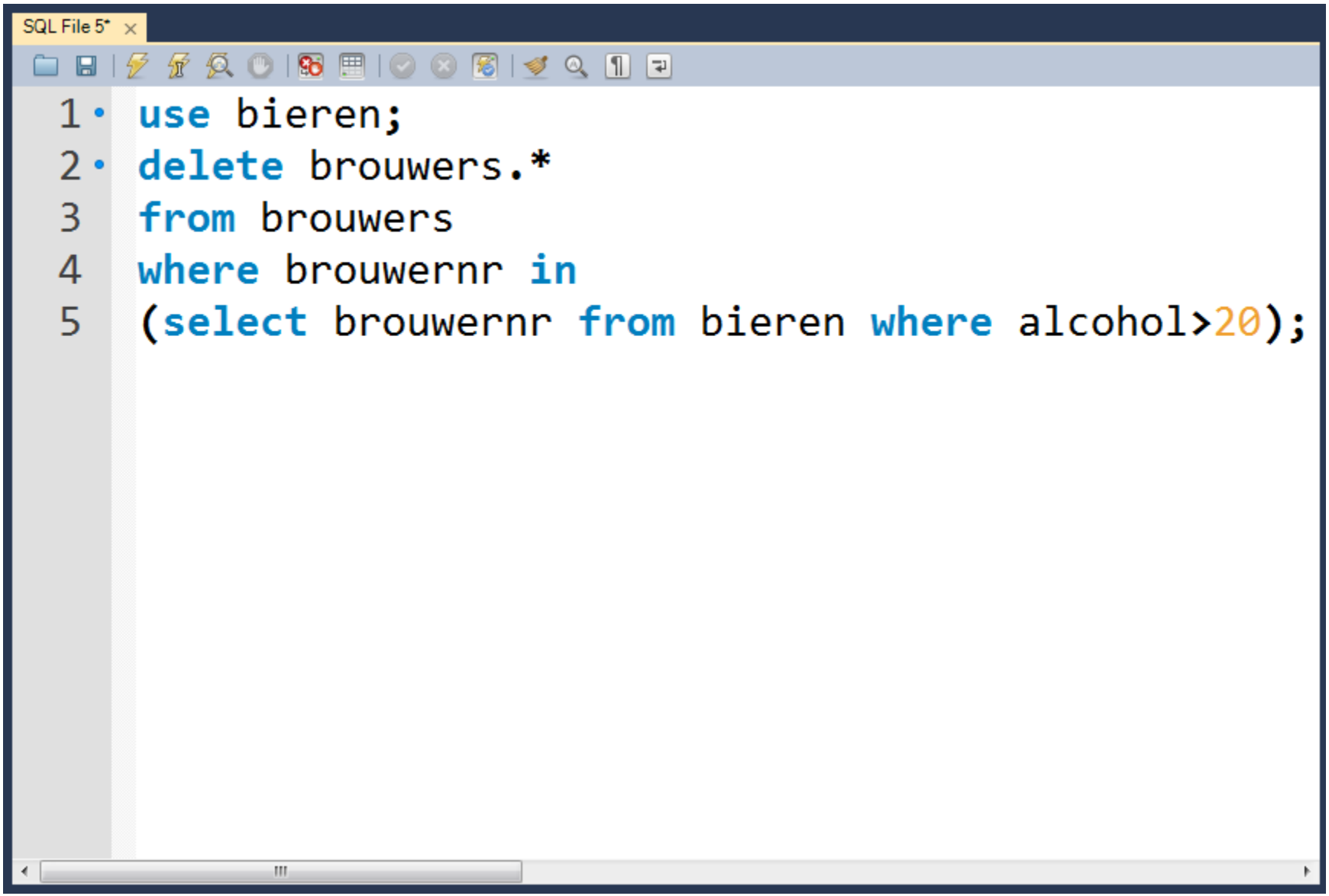
SQL File 5*

```
1  use bieren;
2
3  insert into soorten (soortnr, soort)
4  values (30, 'Extra donker');
5
6  insert into Brouwers
7  values (99, 'Brouwerij Vaattappers',
8  'Interleuvenlaan 2', 3000, 'Heverlee', 1000);
```

UPDATE

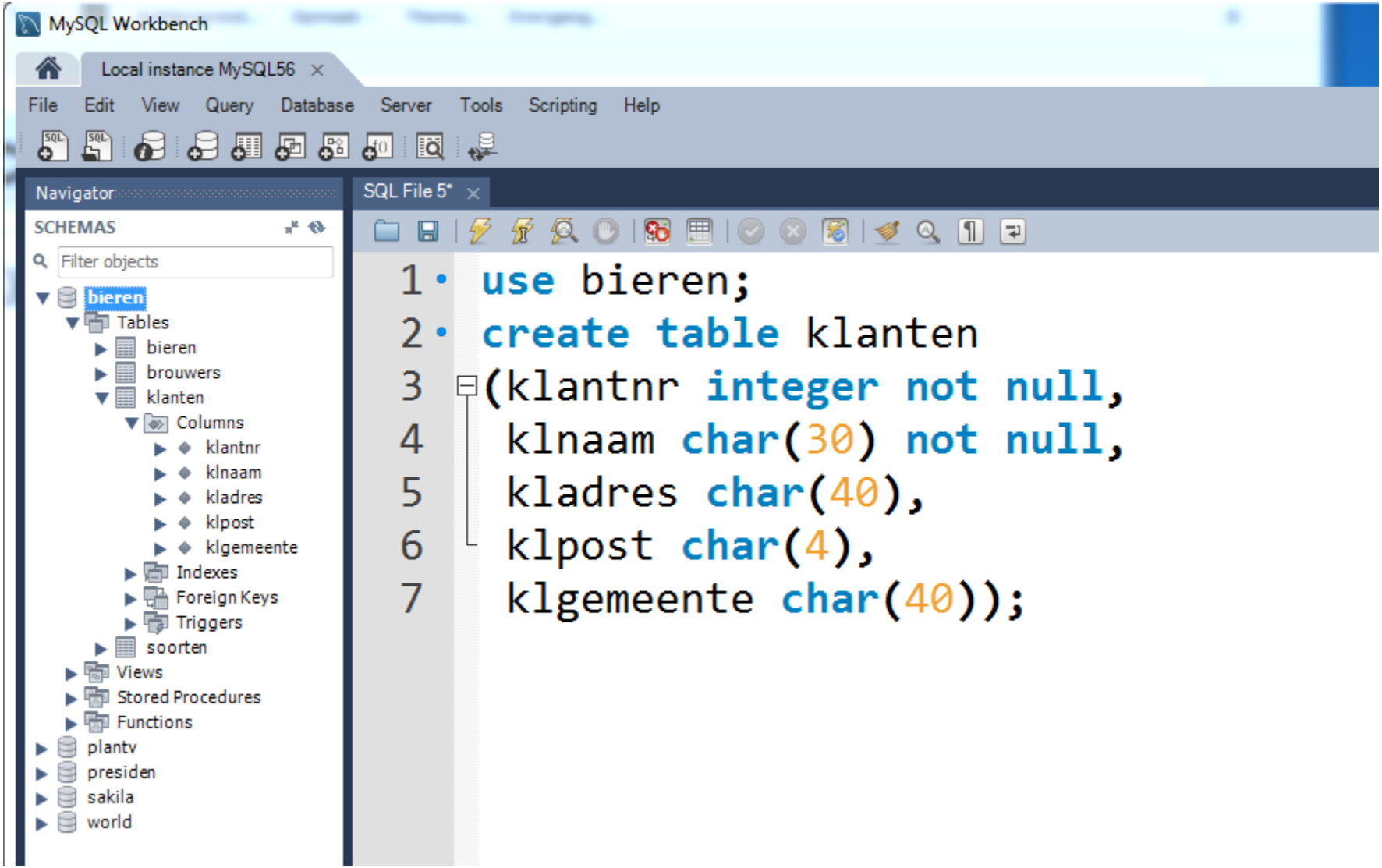

SQL File 5*
1 use bieren;
2 update brouwers
3 set adres='Keizerslaan 111', pstcode = 1000,
4 gemeente = 'Brussel'
5 where brouwernr = 99;

DELETE


SQL File 5*
1 use bieren;
2 delete brouwers.*
3 from brouwers
4 where brouwernr in
5 (select brouwernr from bieren where alcohol>20);

Tabellen & relaties

CREATE TABLE



DROP TABLE

Deze instructie verwijdert een bestaande tabel uit een database.

```
DROP TABLE tabel
```

De instructie DROP bevat de volgende onderdelen:

Onderdeel	Beschrijving
Tabel	De naam van de tabel die je wilt verwijderen

Voordat je een tabel kunt verwijderen, moet je de tabel sluiten.

```
DROP TABLE klanten
```

verwijdert de tabel klanten uit database.

ALTER TABLE

Met deze instructie wijzig je het ontwerp van een tabel die reeds is aangemaakt.

```
ALTER TABLE tabel
{
ADD {[COLUMN] veld [(column_definition)]
    | CONSTRAINT index} |
DROP {[COLUMN] veld
    | CONSTRAINT index} |
MODIFY [COLUMN] veld column_definition
}
```

De instructie ALTER TABLE bevat de volgende onderdelen:

Onderdeel	**Beschrijving**
`Tabel`	De naam van de tabel die je wilt wijzigen.
`Veld`	De naam van het veld dat je wilt toevoegen aan of verwijderen uit de tabel.
`Column_definition`	De specificaties voor het veld zoals het type, de grootte, etc…

ALTER TABLE: uitgebreide mogelijkheden

- Een nieuw veld toevoegen aan de tabel met ADD COLUMN.

  U geeft de veldnaam, het gegevenstype en (voor tekst- en binaire velden) een optionele grootte op. De volgende instructie voegt bijvoorbeeld aan de tabel Brouwers een veld van 25 karakters met de naam Opmerkingen toe:

```
ALTER TABLE Brouwers ADD COLUMN Opmerkingen CHAR(25)
```

- Een veld verwijderen met DROP COLUMN.
  U hoeft alleen de naam van het veld op te geven.

```
ALTER TABLE Brouwers DROP COLUMN Opmerkingen
```

- De velddefinitie van een kolom wijzigen met MODIFY:

```
ALTER TABLE klanten MODIFY COLUMN naam VARCHAR(100) NOT NULL
```

- Je kan ook een index definiëren op een veld met ADD INDEX

```
ALTER TABLE klanten ADD INDEX (naam), ADD UNIQUE (adres)
```

CONSTRAINT

= een **beperkende voorwaarde** voor één of meerdere kolommen in de instructies **ALTER TABLE** en **CREATE TABLE**

CREATE TABLE + CONSTRAINT Syntax:

CREATE TABLE *table_name*
(
column_name1 data_type(*size*) *constraint_name*,
column_name2 data_type(*size*) *constraint_name*,
column_name3 data_type(*size*) *constraint_name*,
....
);

CONSTRAINTS

- **NOT NULL** - Een kolom kan geen NULL waarde bevatten
- **UNIQUE** - Iedere rij heeft een uniek veld in deze kolom.
- **PRIMARY KEY** - Een combinatie van NOT NULL en UNIQUE.
- **FOREIGN KEY** - referentiedata naar een veld in een ander tabel.
- **CHECK** - De veldwaardes in deze kolom moeten aan bepaalde voorwaarden voldoen.
- **DEFAULT** - Specifieert de standaardwaarde wanneer een veld niet werd ingevuld.

FOREIGN KEY CONSTRAINTS

Om relaties tussen tabellen te leggen kunnen we ook CONSTRAINT gebruiken.
Opmerking: enkel `INNODB` tabellen ondersteunen foreign key constraints.

```
[CONSTRAINT [symbol]]
FOREIGN KEY[INDEX_name] (INDEX_col_name, ...)
REFERENCES tbl_name (INDEX_col_name,...)
[ON DELETE (RESTRICT | CASCADE | SET NULL | NO ACTION)]
[ON UPDATE (RESTRICT | CASCADE | SET NULL | NO ACTION)]
```

Met FOREIGN KEY kan je een veld (in de *parent table*) aanwijzen als refererende sleutel.

Met REFERENCES geef je het/de veld(en) aan van de refererende tabel (*child table*) waarnaar wordt verwezen. Als het veld of de velden waarnaar wordt verwezen, de primaire sleutel van de refererende tabel vormen, hoef je de velden niet op te geven.

De foreign key en reference veld(en) moeten van een vergelijkbaar datatype zijn.

FOREIGN KEY CONSTRAINTS

De optionele clausule ON DELETE bepaalt wat er gebeurt als het record in de *parent table* verwijderd wordt, ON UPDATE bepaalt wat er gebeurt als de foreign key waarde in de *parent table* gewijzigd wordt. De mogelijke settings zijn:

- RESTRICT: de `DELETE` of `UPDATE` voor de parent table wordt geweigerd, er gebeurt niets.
  Dit is ook de standaardinstelling als de clausules ONDELETE/UPDATE niet gespecificeerd zijn.

- NO ACTION: in MySQL is dit hetzelfde als RESTRICT

- SET NULL: bij een `DELETE` of `UPDATE` van de *parent table* wordt de waarde van de gerefereerde kolom(men) in de *child table* op `NULL` gezet. Deze kolommen mogen uiteraard geen NOT NULL ingesteld hebben

- CASCADE: bij een `DELETE` van een record in de *parent table* wordt dit record verwijderd en automatisch ook de gerelateerde rijen van de child table. Bij een update van een record in de *parent table* wordt dit record gewijzigd en automatisch ook de gerelateerde rijen van de child table bijgewerkt.

FOREIGN KEY: voorbeeld

De 1-N relatie tussen klant en bestellingen.aanmaak klanten:

```
CREATE TABLE klanten
(
  klantnr INT NOT NULL PRIMARY KEY,
  naam VARCHAR(50)  NOT NULL,
  adres VARCHAR(50) NOT NULL,
  postcode CHAR(4) NOT NULL,
  woonplaats VARCHAR(50) NOT NULL
) ENGINE = INNODB
```

We kunnen de relatie onmiddellijk inbouwen in de aanmaak van bestellingen.

```
CREATE TABLE bestellingen
(
  bestelnr INT(11) NOT NULL PRIMARY KEY,
  klantnr INT(11) NOT NULL,
  besteldatum datetime NOT NULL,
  CONSTRAINT fk_klantnr FOREIGN KEY (klantnr)
  REFERENCES klanten(klantnr)
  ON DELETE CASCADE
  ON UPDATE CASCADE
) ENGINE=InnoDB
```

FOREIGN KEY: voorbeeld

Het kan ook korter:

```
CREATE TABLE klanten
(
  klantnr INT NOT NULL PRIMARY KEY,
  naam VARCHAR(50)  NOT NULL,
  adres VARCHAR(50) NOT NULL,
  postcode CHAR(4) NOT NULL,
  woonplaats VARCHAR(50) NOT NULL
) ENGINE = INNODB
```

```
CREATE TABLE bestellingen
(
  bestelnr INT(11) NOT NULL PRIMARY KEY,
  klantnr INT(11) NOT NULL REFERENCES klanten(klantnr)
  ON DELETE CASCADE ON UPDATE CASCADE,
  besteldatum datetime NOT NULL
) ENGINE=INNODB
```

FOREIGN KEY: voorbeeld

De relatie later toevoegen:

```
CREATE TABLE bestellingen
(
  bestelnr INTEGER NOT NULL PRIMARY KEY,
  klantnr INTEGER NOT NULL,
  besteldatum datetime NOT NULL
)ENGINE = INNODB
```

```
ALTER TABLE bestellingen
ADD CONSTRAINT fk_klanten FOREIGN KEY (klantnr)
REFERENCES klanten (klantnr)
ON DELETE CASCADE
ON UPDATE CASCADE
```

OF

```
ALTER TABLE bestellingen
ADD FOREIGN KEY (klantnr)
REFERENCES klanten (klantnr)
ON DELETE CASCADE
ON UPDATE CASCADE
```

DROP FOREIGN KEY

We hoeven geen CONSTRAINTS te gebruiken. (zie ons voorbeeld) maar het is wel een voordeel:
De constraint heeft nu een naam **‘fk_klanten’**
Als we die willen wissen is het eenvoudig:

```
ALTER TABLE bestellingen DROP FOREIGN KEY fk_klanten
```

CREATE INDEX

Indexen laten de database toe de data sneller te vinden zonder de gehele tabel te lezen.

SQL CREATE INDEX Syntax

(maakt een index aan. Dubbele waarden zijn toegestaan)

CREATE INDEX index_name
ON table_name (column_name)

SQL CREATE UNIQUE INDEX Syntax

(maakt een unieke index aan. Geen dubbele waarden !)

CREATE UNIQUE INDEX index_name
ON table_name (column_name)

Views

VIEWS

= Een virtuele tabel die het resultaat is van een SQL-statement.

Een virtuele tabel is als een echte tabel met rijen, kolommen en velden.

De velden in een view kunnen uit één of meerdere echte tabellen komen

CREATE VIEW

Maakt een nieuwe view aan

SQL CREATE VIEW Syntax:

CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition

DROP VIEW

Verwijdert een view

SQL DROP VIEW Syntax:

DROP VIEW view_name