Kurs
Datenbankgrundlagen und Modellierung

Sebastian Maneth, Universität Bremen
maneth@uni-bremen.de

Sommersemester 2023

8.5.2023
Vorlesung 3: Introduction to SQL

Agenda

1.) RECAP
- the Relational Model
- creating tables

2.) Introduction to SQL

3.) Aggregate Functions

4.) `GROUP BY`

RECAP: The Relational Model

From a **mathematical point of view**, we can model a relational database D as follows:

D = (R, sch, dom, val)

A database consists of **four things**!

— R is a finite set of relation names

— for every r in R, sch(r) is a list of (pairwise distinct) attributes ("the **schema** of r")

— If sch(r)=(A_1,…,A_k) then dom(r, A_i) is a set of values for every i=1…k
("the **domain** of A_i in r")

and val(r) is a **relation** over dom(r,A_1), …, dom(r,A_k)

$$\text{i.e., } \mathrm{val}(r) \subseteq \mathrm{dom}(r, A_1) \times \cdots \times \mathrm{dom}(r, A_k).$$

There are Tables in RDBMSs that
do not correspond to relations in the Relational Model.

For exactly two reasons:

Tables in RDBMSs may contain

1.) duplicates

2.) NULL values.

NULL means: there is a value here, but we do not know it!

— we will **never** design tables that contain duplicates.
— try to avoid NULL values whenever possible!!!

RECAP: General Thumb Rules

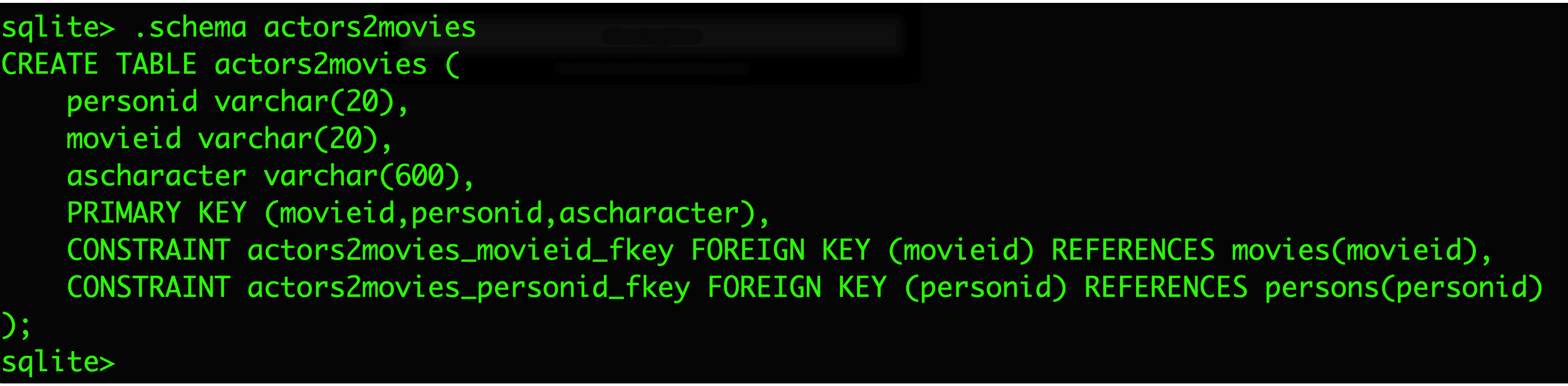
1.) in every table choose a PRIMARY KEY
(in the 'worst case' consisting of all attributes)

2.) use FOREIGN KEYs where appropriate

3.) for every attribute, forbid nulls by appending NOT NULL

```
CREATE Table Book(ISBN VARCHAR(40) PRIMARY KEY,
                  Title VARCHAR(100) NOT NULL);
CREATE Table Author(Name VARCHAR NOT NULL,
                    TaxNo VARCHAR PRIMARY KEY);
CREATE Table writtenBy(ISBN VARCHAR(40) REFERENCES Book,
                       TaxNo VARCHAR REFERENCES Author,
                       PRIMARY KEY (ISBN, TaxNo));
```

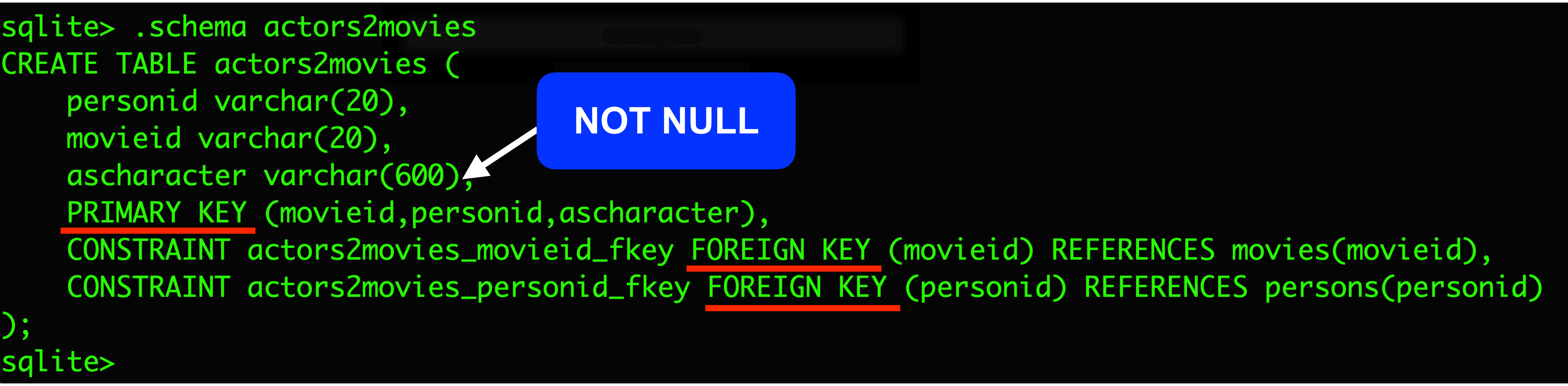
— attributes of PRIMARY KEYS may not contain NULLs by default



```
sqlite> .schema actors2movies
CREATE TABLE actors2movies (
    personid varchar(20),
    movieid varchar(20),
    ascharacter varchar(600),
    PRIMARY KEY (movieid,personid,ascharacter),
    CONSTRAINT actors2movies_movieid_fkey FOREIGN KEY (movieid) REFERENCES movies(movieid),
    CONSTRAINT actors2movies_personid_fkey FOREIGN KEY (personid) REFERENCES persons(personid)
);
sqlite>
```

NULL means: there is a value here, but we do not know it!

— we will **never** design tables that contain duplicates.
— try to avoid NULL values whenever possible!!!



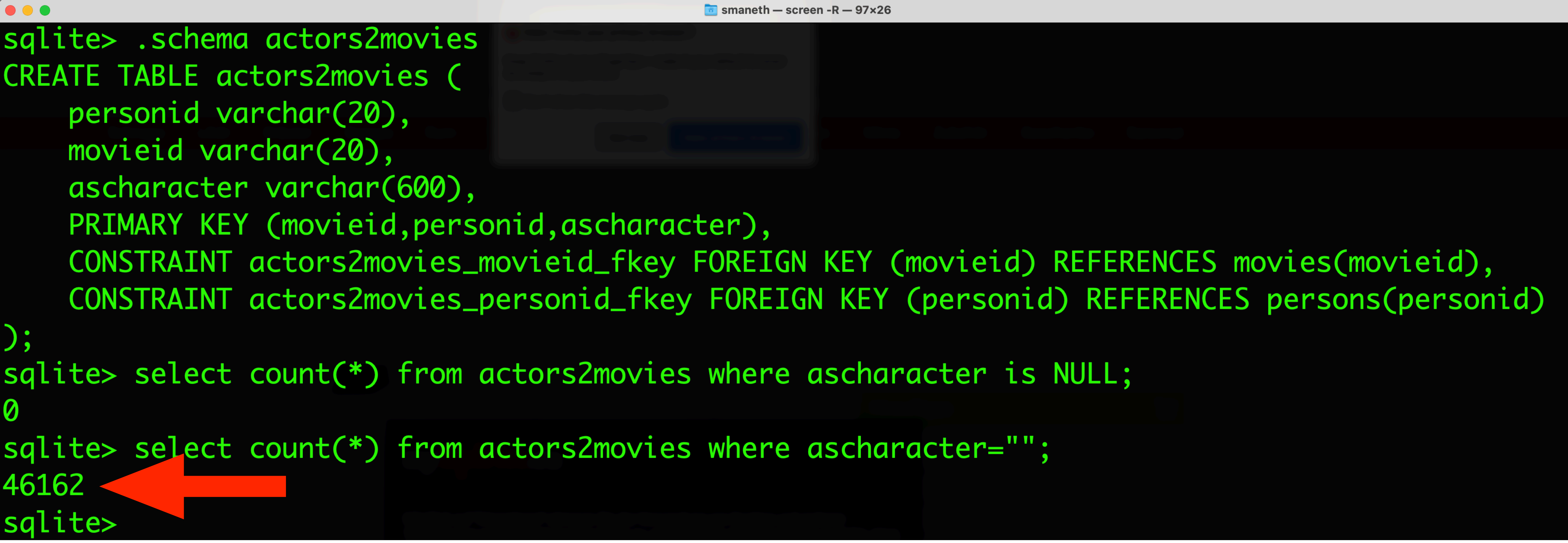
```
sqlite> .schema actors2movies
CREATE TABLE actors2movies (
    personid varchar(20),
    movieid varchar(20),
    ascharacter varchar(600),
    PRIMARY KEY (movieid,personid,ascharacter),
    CONSTRAINT actors2movies_movieid_fkey FOREIGN KEY (movieid) REFERENCES movies(movieid),
    CONSTRAINT actors2movies_personid_fkey FOREIGN KEY (personid) REFERENCES persons(personid)
);
sqlite>
```

— good: there is a PRIMARY KEY
— good: there are FOREIGN KEYs

— **not so good:** there are **no** NOT NULL constraints

NULL means: there is a value here, but we do not know it!

— we will **never** design tables that contain duplicates.
— try to avoid NULL values whenever possible!!!

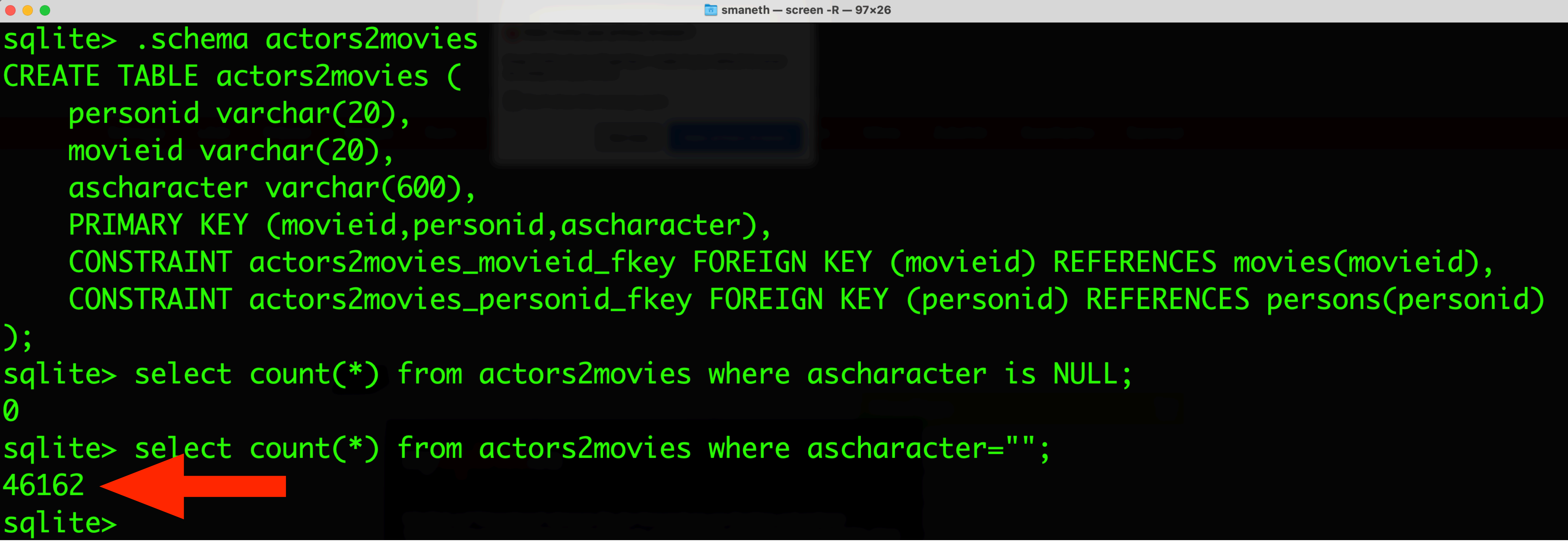


```
sqlite> .schema actors2movies
CREATE TABLE actors2movies (
    personid varchar(20),
    movieid varchar(20),
    ascharacter varchar(600),
    PRIMARY KEY (movieid,personid,ascharacter),
    CONSTRAINT actors2movies_movieid_fkey FOREIGN KEY (movieid) REFERENCES movies(movieid),
    CONSTRAINT actors2movies_personid_fkey FOREIGN KEY (personid) REFERENCES persons(personid)
);
sqlite> select count(*) from actors2movies where ascharacter is NULL;
0
sqlite> select count(*) from actors2movies where ascharacter="";
46162
sqlite>
```

— **super-bad:** NULL values, but they were inserted as empty strings!

NULL means: there is a value here, but we do not know it!

— we will **never** design tables that contain duplicates.
— try to avoid NULL values whenever possible!!!



```
sqlite> .schema actors2movies
CREATE TABLE actors2movies (
    personid varchar(20),
    movieid varchar(20),
    ascharacter varchar(600),
    PRIMARY KEY (movieid,personid,ascharacter),
    CONSTRAINT actors2movies_movieid_fkey FOREIGN KEY (movieid) REFERENCES movies(movieid),
    CONSTRAINT actors2movies_personid_fkey FOREIGN KEY (personid) REFERENCES persons(personid)
);
sqlite> select count(*) from actors2movies where ascharacter is NULL;
0
sqlite> select count(*) from actors2movies where ascharacter="";
46162
sqlite>
```

— **super-bad:** NULL values, but they were inserted as empty strings!

Solutions:

1.) change empty strings into NULL values (not so good)

2.) **better**:

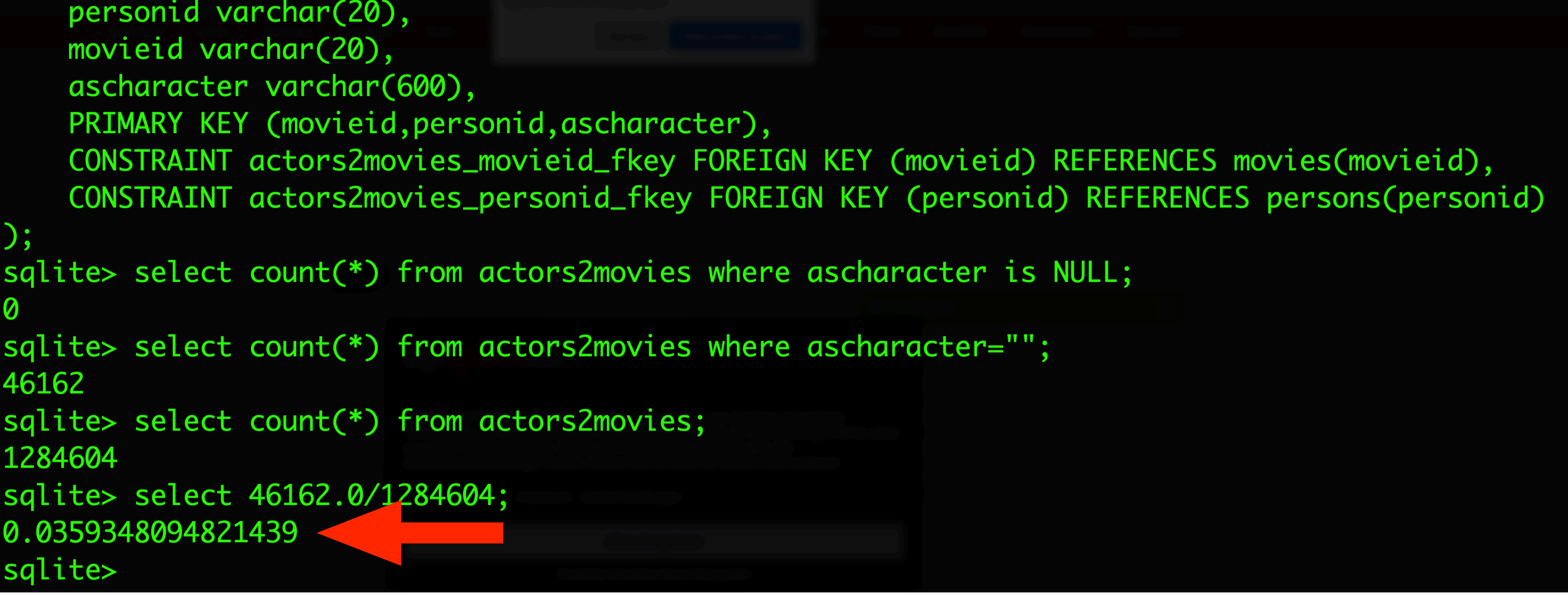
a2m:

pid	mid

am2c:

pid	mid	asc

Now there are **no** NULLs!



```
    personid varchar(20),
    movieid varchar(20),
    ascharacter varchar(600),
    PRIMARY KEY (movieid,personid,ascharacter),
    CONSTRAINT actors2movies_movieid_fkey FOREIGN KEY (movieid) REFERENCES movies(movieid),
    CONSTRAINT actors2movies_personid_fkey FOREIGN KEY (personid) REFERENCES persons(personid)
);
sqlite> select count(*) from actors2movies where ascharacter is NULL;
0
sqlite> select count(*) from actors2movies where ascharacter="";
46162
sqlite> select count(*) from actors2movies;
1284604
sqlite> select 46162.0/1284604;
0.0359348094821439
sqlite>
```

— **super-bad:** NULL values, but they were inserted as empty strings!

Solutions:

1.) change empty strings into NULL values (not so good)

2.) better:

a2m:

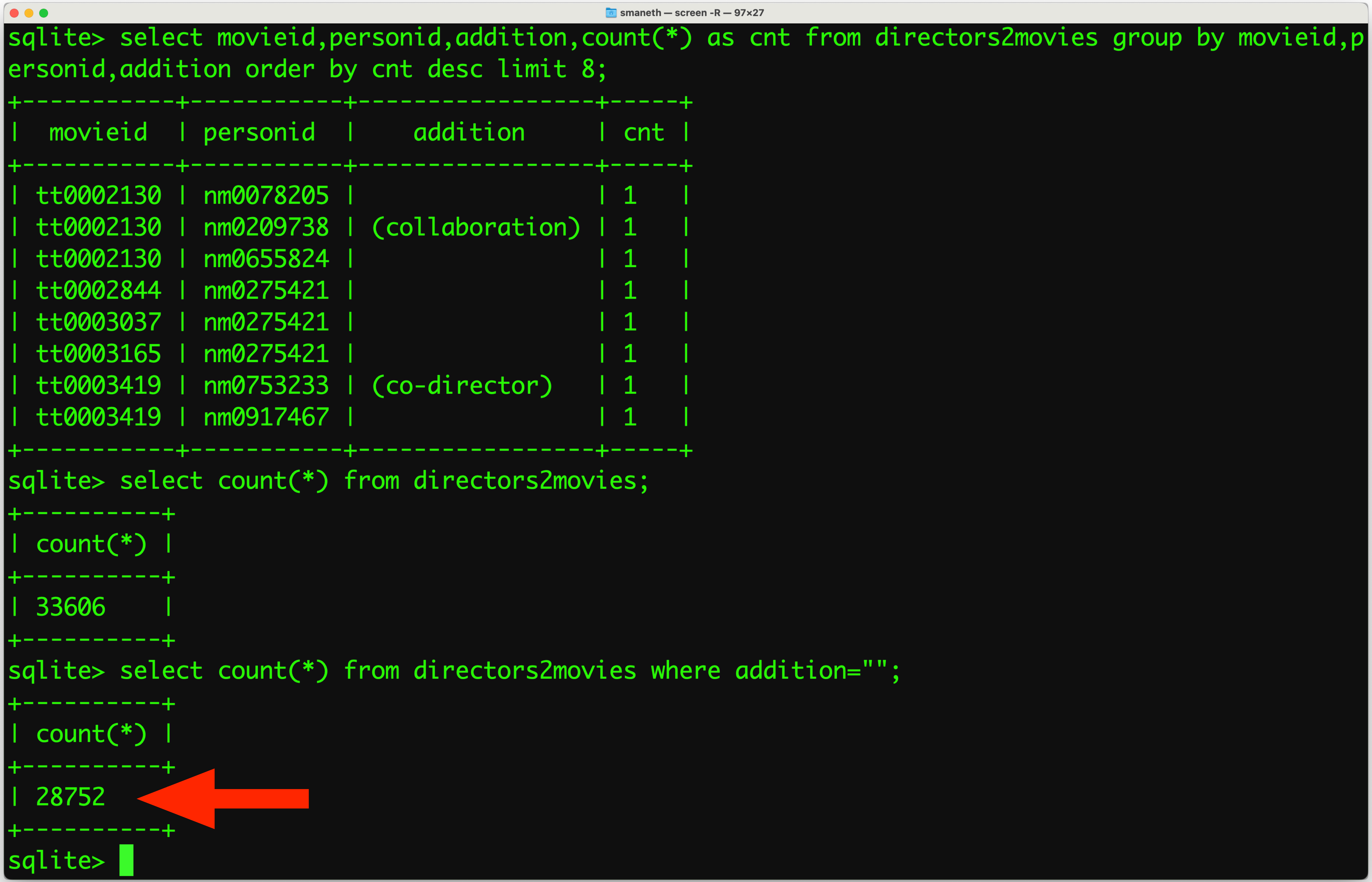
pid	mid

am2c:

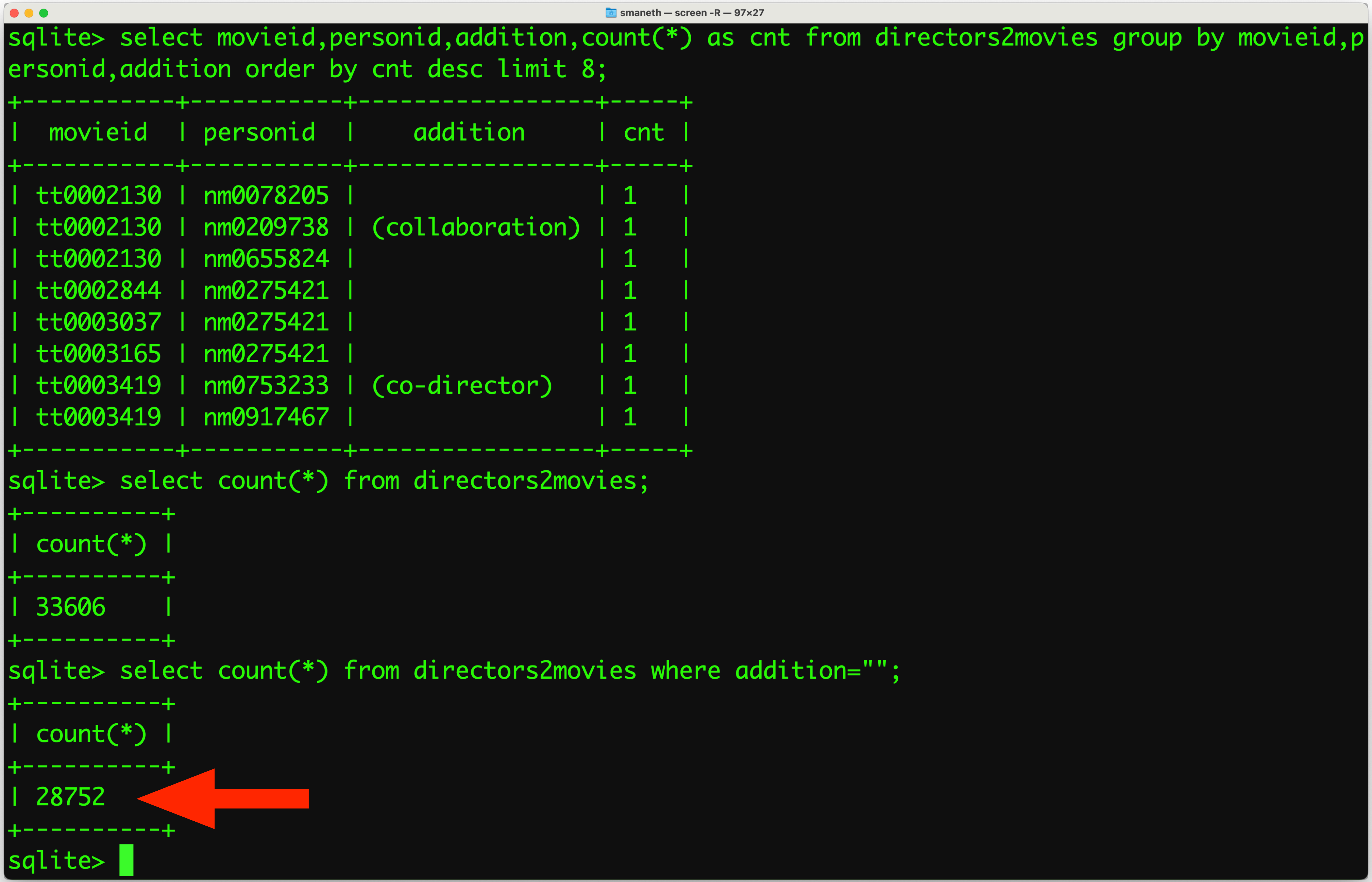
pid	mid	asc

Now there are **no** NULLs!

around **3.6%** of the table have NULL in the `ascharacter` column.



```
sqlite> select movieid,personid,addition,count(*) as cnt from directors2movies group by movieid,p
ersonid,addition order by cnt desc limit 8;
+-----------+-----------+-----------------+-----+
|  movieid  | personid  |    addition     | cnt |
+-----------+-----------+-----------------+-----+
| tt0002130 | nm0078205 |                 | 1   |
| tt0002130 | nm0209738 | (collaboration) | 1   |
| tt0002130 | nm0655824 |                 | 1   |
| tt0002844 | nm0275421 |                 | 1   |
| tt0003037 | nm0275421 |                 | 1   |
| tt0003165 | nm0275421 |                 | 1   |
| tt0003419 | nm0753233 | (co-director)   | 1   |
| tt0003419 | nm0917467 |                 | 1   |
+-----------+-----------+-----------------+-----+
sqlite> select count(*) from directors2movies;
+----------+
| count(*) |
+----------+
| 33606    |
+----------+
sqlite> select count(*) from directors2movies where addition="";
+----------+
| count(*) |
+----------+
| 28752    |
+----------+
sqlite> 
```



```
sqlite> select movieid,personid,addition,count(*) as cnt from directors2movies group by movieid,p
ersonid,addition order by cnt desc limit 8;
+-----------+-----------+-----------------+-----+
|  movieid  | personid  |    addition     | cnt |
+-----------+-----------+-----------------+-----+
| tt0002130 | nm0078205 |                 | 1   |
| tt0002130 | nm0209738 | (collaboration) | 1   |
| tt0002130 | nm0655824 |                 | 1   |
| tt0002844 | nm0275421 |                 | 1   |
| tt0003037 | nm0275421 |                 | 1   |
| tt0003165 | nm0275421 |                 | 1   |
| tt0003419 | nm0753233 | (co-director)   | 1   |
| tt0003419 | nm0917467 |                 | 1   |
+-----------+-----------+-----------------+-----+
sqlite> select count(*) from directors2movies;
+----------+
| count(*) |
+----------+
| 33606    |
+----------+
sqlite> select count(*) from directors2movies where addition="";
+----------+
| count(*) |
+----------+
| 28752    |
+----------+
sqlite> 
```

`directors2movies` table: **86% EMPTY STRINGS** in the `addition`-column.

RECAP: Altering Tables

→ DELETE FROM T1; - delete all rows from table T1

→ DELETE FROM T1 where c3=1; - delete rows with c3-value equals 1

→ DROP TABLE T1; - remove table T1

→ ALTER TABLE T1 ADD COLUMN col1 int; - adds a column to table T1

→ ALTER TABLE T1 DROP COLUMN col1; - removes a column from table T1

→ DESCRIBE T1; - lists the fields and types of table t1 (**MySQL**, not SQL!)
(in **sqlite3** this is done via ".schema t1" or "PRAGMA table_info(t1)")

→ SHOW tables; - lists tables of your database (**MySQL**, not SQL!)
(in **sqlite3** this is done via ".tables")

— \d lists all tables in PostgreSQL
\d tablename shows schema of the table

RECAP: Altering Tables

→ DELETE FROM T1; - delete all rows from table T1

→ DELETE FROM T1 where c3=1; - delete rows with c3-value equals 1

→ DROP TABLE T1; - remove table T1

Which value should be / is inserted?

→ ALTER TABLE T1 ADD COLUMN col1 int; - adds a column to table T1

→ ALTER TABLE T1 DROP COLUMN col1; - removes a column from table T1

→ DESCRIBE T1; - lists the fields and types of table t1 (**MySQL**, not SQL!)
(in **sqlite3** this is done via ".schema t1" or "PRAGMA table_info(t1)")

→ SHOW tables; - lists tables of your database (**MySQL**, not SQL!)
(in **sqlite3** this is done via ".tables")

— \d lists all tables in PostgreSQL
\d tablename shows schema of the table

RECAP: Altering Tables

→ DELETE FROM T1; - delete all rows from table T1

→ DELETE FROM T1 where c3=1; - delete rows with c3-value equals 1

→ DROP TABLE T1; - remove table T1

Which value should be / is inserted?

→ ALTER TABLE T1 ADD COLUMN col1 int DEFAULT 0;

Now the default value is 0.

Otherwise it is NULL!

2.) Introduction to SQL

2.) Introduction to SQL

Developed in the 1970's at IBM by Chamberlin and Boyce
(originally "SEQUEL" = Structured English Query Language")

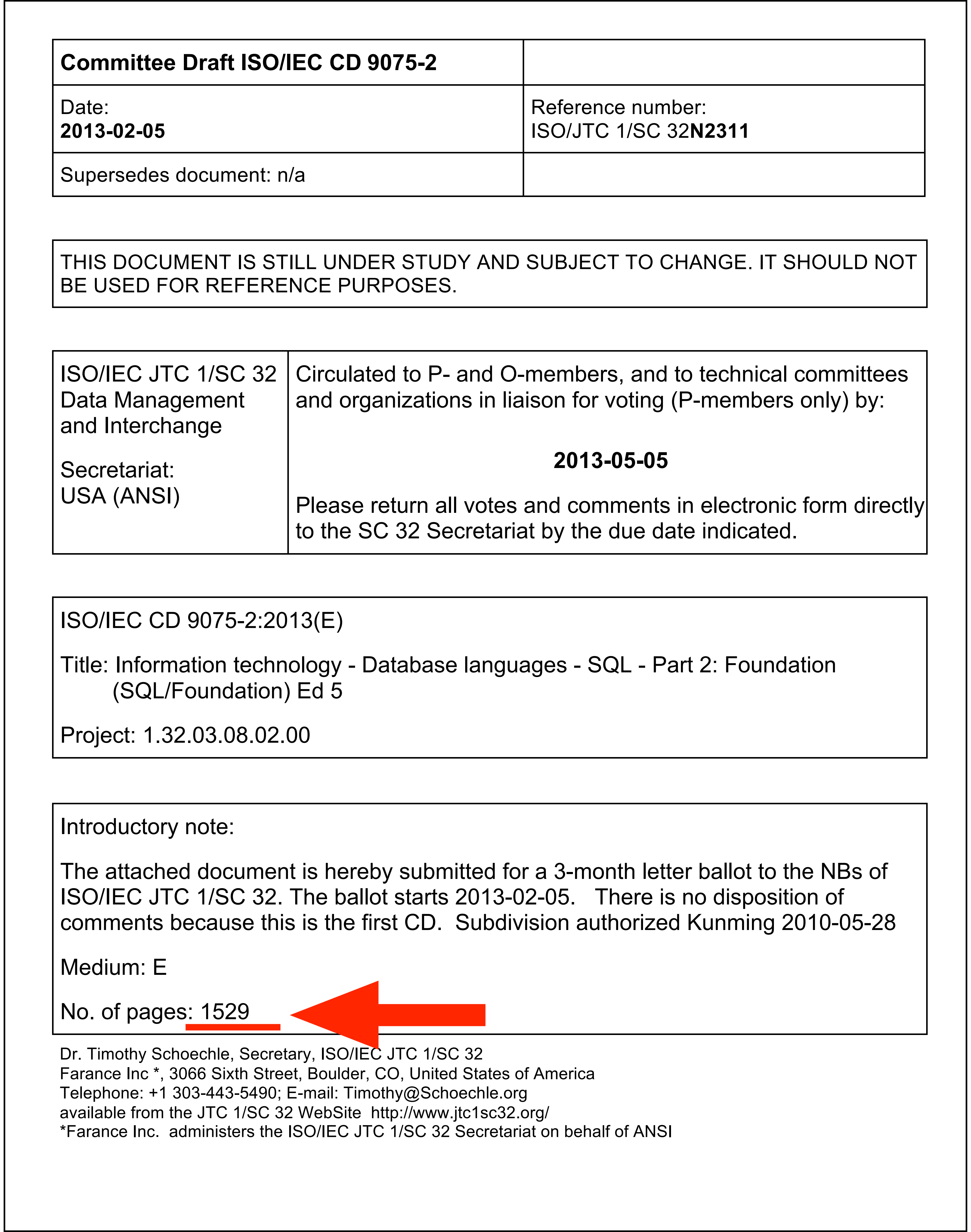
— June 1979 first commercial version by Relational Software Inc (later Oracle)

— ANSI standard in 1986

— ISO standard in 1987, important releases 1992, 1999, 2003, 2008, 2011, 2016

— MySQL attempts to comply with 2008 standard

→ Despite the existence of the SQL standards, most SQL code
is *not completely portable among different database systems*
without adjustments :-(



Committee Draft ISO/IEC CD 9075-2	
Date: **2013-02-05**	Reference number: ISO/JTC 1/SC 32**N2311**
Supersedes document: n/a	

THIS DOCUMENT IS STILL UNDER STUDY AND SUBJECT TO CHANGE. IT SHOULD NOT BE USED FOR REFERENCE PURPOSES.

ISO/IEC JTC 1/SC 32 Data Management and Interchange Secretariat: USA (ANSI)	Circulated to P- and O-members, and to technical committees and organizations in liaison for voting (P-members only) by: **2013-05-05** Please return all votes and comments in electronic form directly to the SC 32 Secretariat by the due date indicated.

ISO/IEC CD 9075-2:2013(E)

Title: Information technology - Database languages - SQL - Part 2: Foundation (SQL/Foundation) Ed 5

Project: 1.32.03.08.02.00

Introductory note:

The attached document is hereby submitted for a 3-month letter ballot to the NBs of ISO/IEC JTC 1/SC 32. The ballot starts 2013-02-05. There is no disposition of comments because this is the first CD. Subdivision authorized Kunming 2010-05-28

Medium: E

No. of pages: 1529

Dr. Timothy Schoechle, Secretary, ISO/IEC JTC 1/SC 32
Farance Inc *, 3066 Sixth Street, Boulder, CO, United States of America
Telephone: +1 303-443-5490; E-mail: Timothy@Schoechle.org
available from the JTC 1/SC 32 WebSite http://www.jtc1sc32.org/
*Farance Inc. administers the ISO/IEC JTC 1/SC 32 Secretariat on behalf of ANSI

SQL—Structured Query Language

By far the most important query language today is **SQL**.

- Structured Query Language
- Originally meant to be used by end users ☺
- Today supported by virtually any database system

SQL operates on **relational data**:

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

- Real databases may contain 100s or 1000s of **tables**, sometimes with billions of **rows** (also: **tuples**).

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

+

```
SELECT Name, Price
  FROM Ingredients
 WHERE Alcohol = 0
```

=

?

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

$+$

```
SELECT Name, Price
  FROM Ingredients
 WHERE Alcohol = 0
```

$=$

Name	Price
Mineral Water	1.49
Orange Juice	2.99

General Form of a SQL Query:

```
SELECT   list of attributes
FROM     list of tables
WHERE    condition over attributes
GROUP BY list of attributes
HAVING   condition over aggregates
ORDER BY list of attributes
LIMIT    number
```

aggregate functions:

```
COUNT    VARIANCE
SUM      STDDEV
AVG      BIT_OR
MAX      BIT_AND
MIN
```

General Form of a SQL Query:

```
SELECT   list of attributes
FROM     list of tables
WHERE    condition over attributes
GROUP BY list of attributes
HAVING   condition over aggregates
ORDER BY list of attributes
LIMIT    number
```

aggregate functions:

```
COUNT    VARIANCE
SUM      STDDEV
AVG      BIT_OR
MAX      BIT_AND
MIN
```

subqueries:

— where a table name may be placed, we may also place a query (which produces a table!)

General Form of a SQL Query:

```
SELECT    list of attributes
FROM      list of tables
WHERE     condition over attributes
GROUP BY  list of attributes
HAVING    condition over aggregates
ORDER BY  list of attributes
LIMIT     number
```

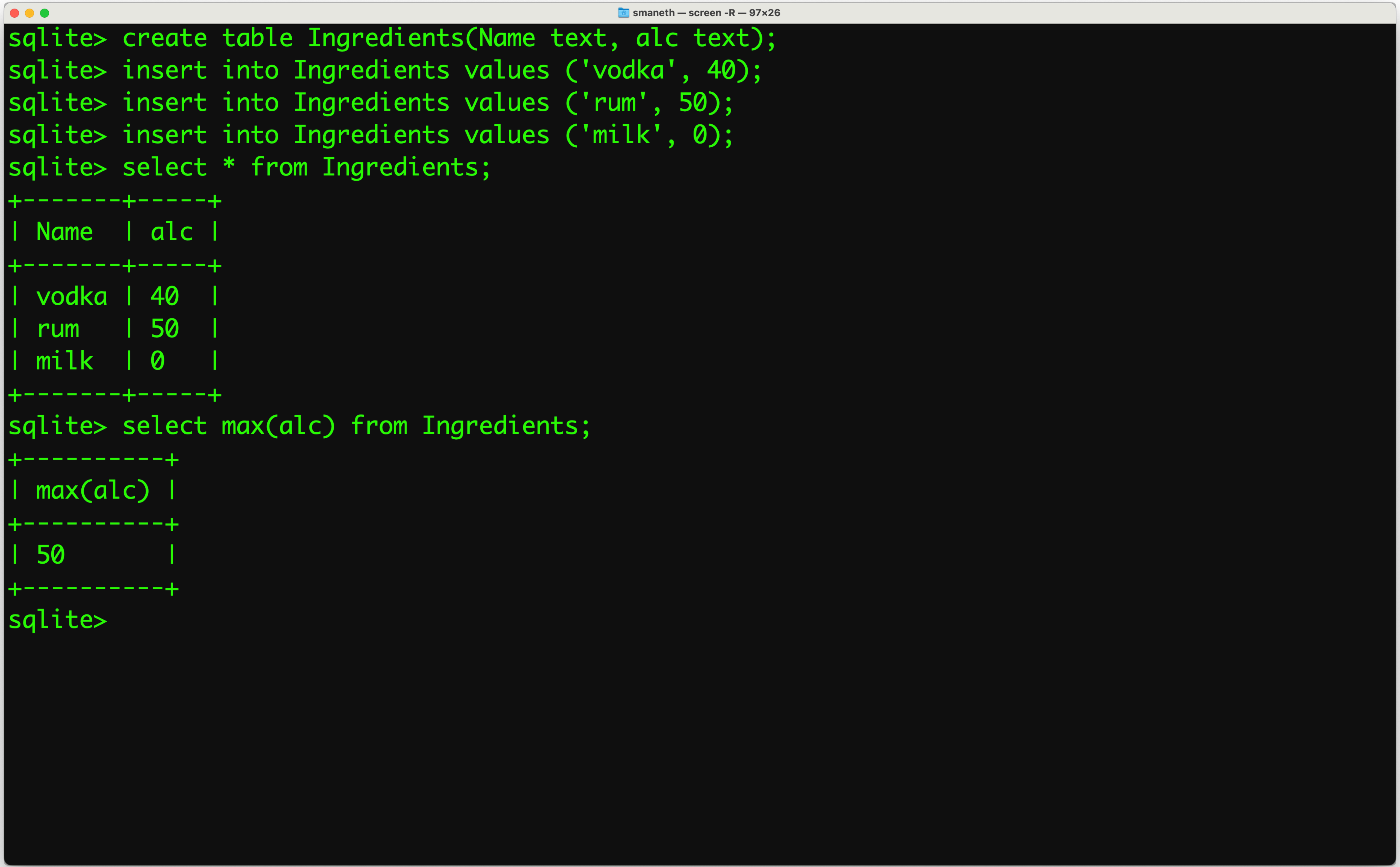
aggregate functions:

```
COUNT    VARIANCE
SUM      STDDEV
AVG      BIT_OR
MAX      BIT_AND
MIN
```

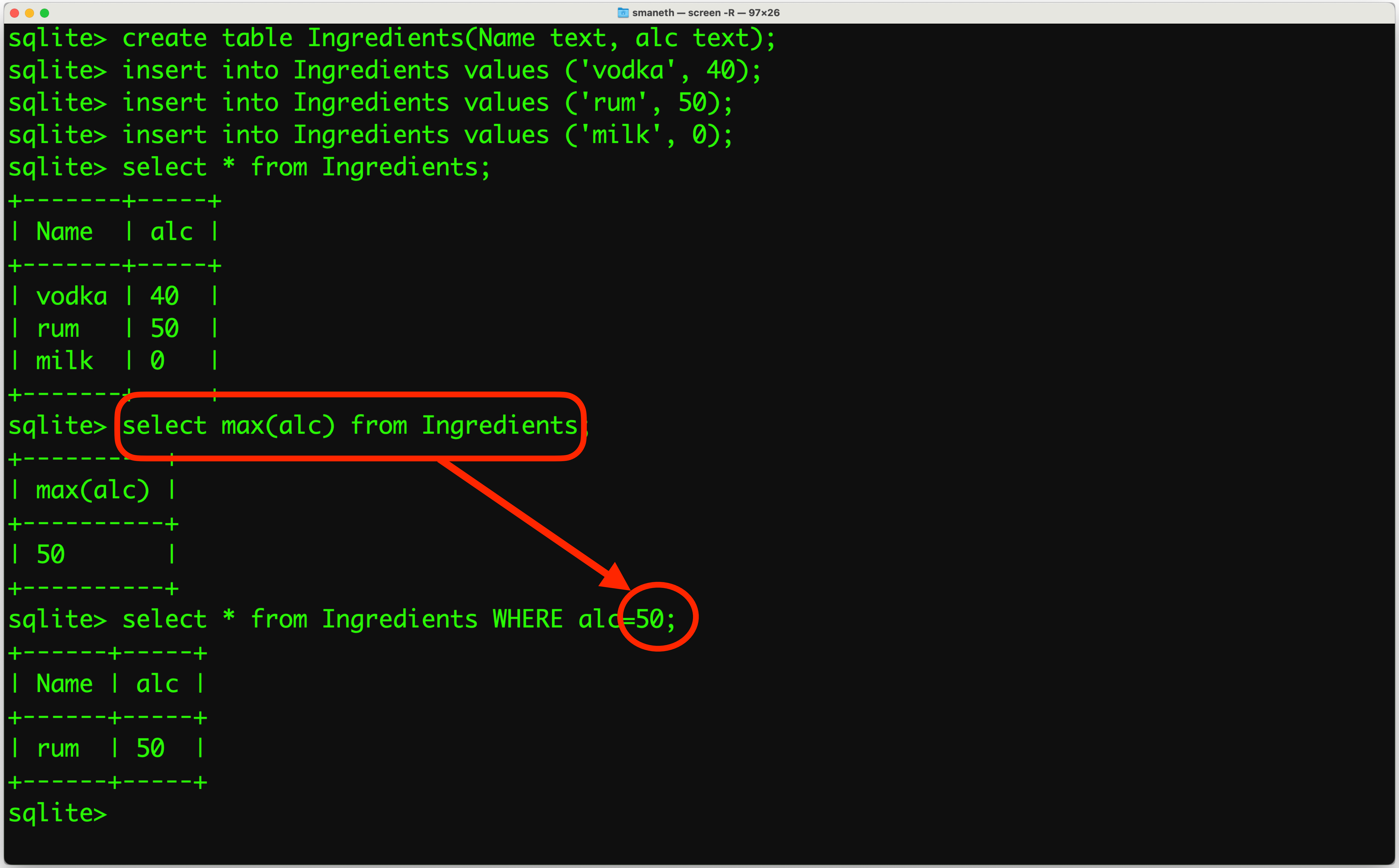
subqueries:

— where a table name may be placed, we may also place a query (which produces a table!)

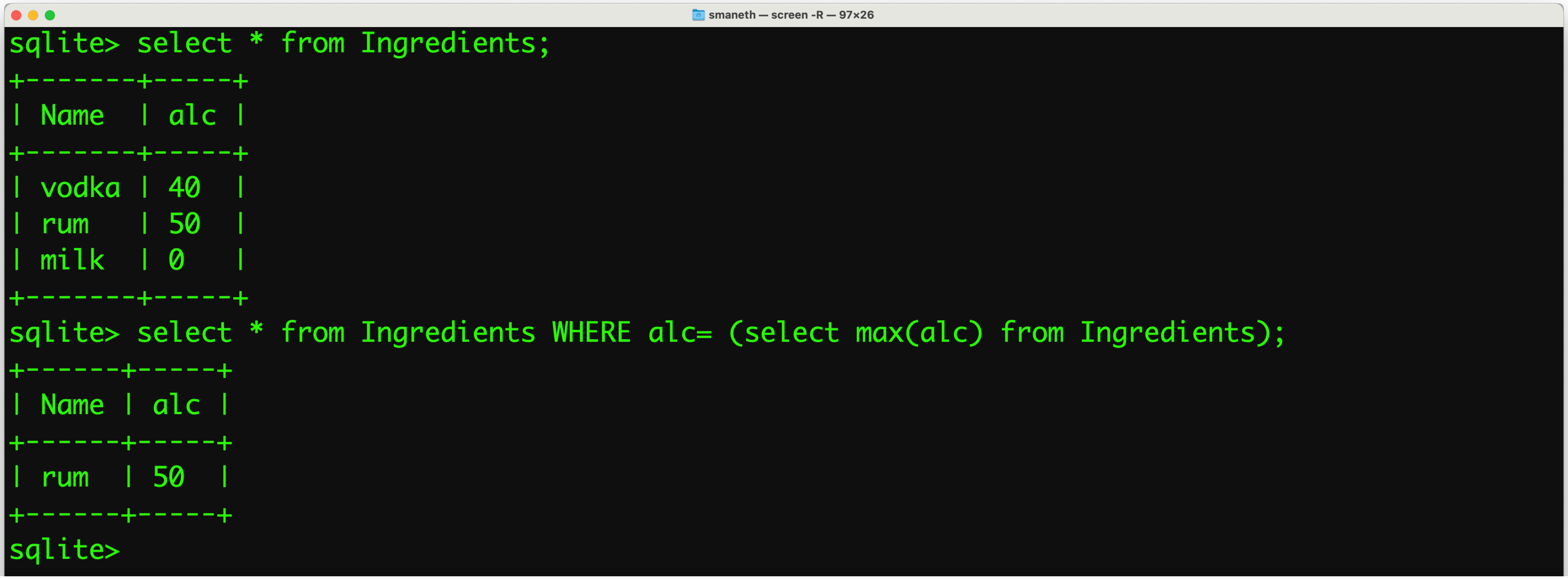
— where a value may be placed, we may also place a query which produced **one value**.


smaneth — screen -R — 97×26

```
sqlite> create table Ingredients(Name text, alc text);
sqlite> insert into Ingredients values ('vodka', 40);
sqlite> insert into Ingredients values ('rum', 50);
sqlite> insert into Ingredients values ('milk', 0);
sqlite> select * from Ingredients;
+-------+-----+
| Name  | alc |
+-------+-----+
| vodka | 40  |
| rum   | 50  |
| milk  | 0   |
+-------+-----+
sqlite> select max(alc) from Ingredients;
+----------+
| max(alc) |
+----------+
| 50       |
+----------+
sqlite>
```



```
sqlite> create table Ingredients(Name text, alc text);
sqlite> insert into Ingredients values ('vodka', 40);
sqlite> insert into Ingredients values ('rum', 50);
sqlite> insert into Ingredients values ('milk', 0);
sqlite> select * from Ingredients;
+-------+-----+
| Name  | alc |
+-------+-----+
| vodka | 40  |
| rum   | 50  |
| milk  | 0   |
+-------+-----+
sqlite> select max(alc) from Ingredients;
+----------+
| max(alc) |
+----------+
| 50       |
+----------+
sqlite> select * from Ingredients WHERE alc=50;
+------+-----+
| Name | alc |
+------+-----+
| rum  | 50  |
+------+-----+
sqlite>
```



```
sqlite> select * from Ingredients;
+-------+-----+
| Name  | alc |
+-------+-----+
| vodka | 40  |
| rum   | 50  |
| milk  | 0   |
+-------+-----+
sqlite> select * from Ingredients WHERE alc= (select max(alc) from Ingredients);
+------+-----+
| Name | alc |
+------+-----+
| rum  | 50  |
+------+-----+
sqlite>
```

List the ingredients that have
the highest (=maximum) alcohol value.

More on the WHERE clause

- Once we have types such as strings, numbers, we have type-specific operations, and hence type-specific selection conditions

-
```
create table finance (title char(20),
                      budget int,
                      gross int)

insert into finance values ('Shining', 19, 100)
insert into finance values ('Star wars', 11, 513)
insert into finance values ('Wild wild west', 170, 80)
```

More on the WHERE clause

- Find movies that lost money:

```
select title
from finance
where gross < budget
```

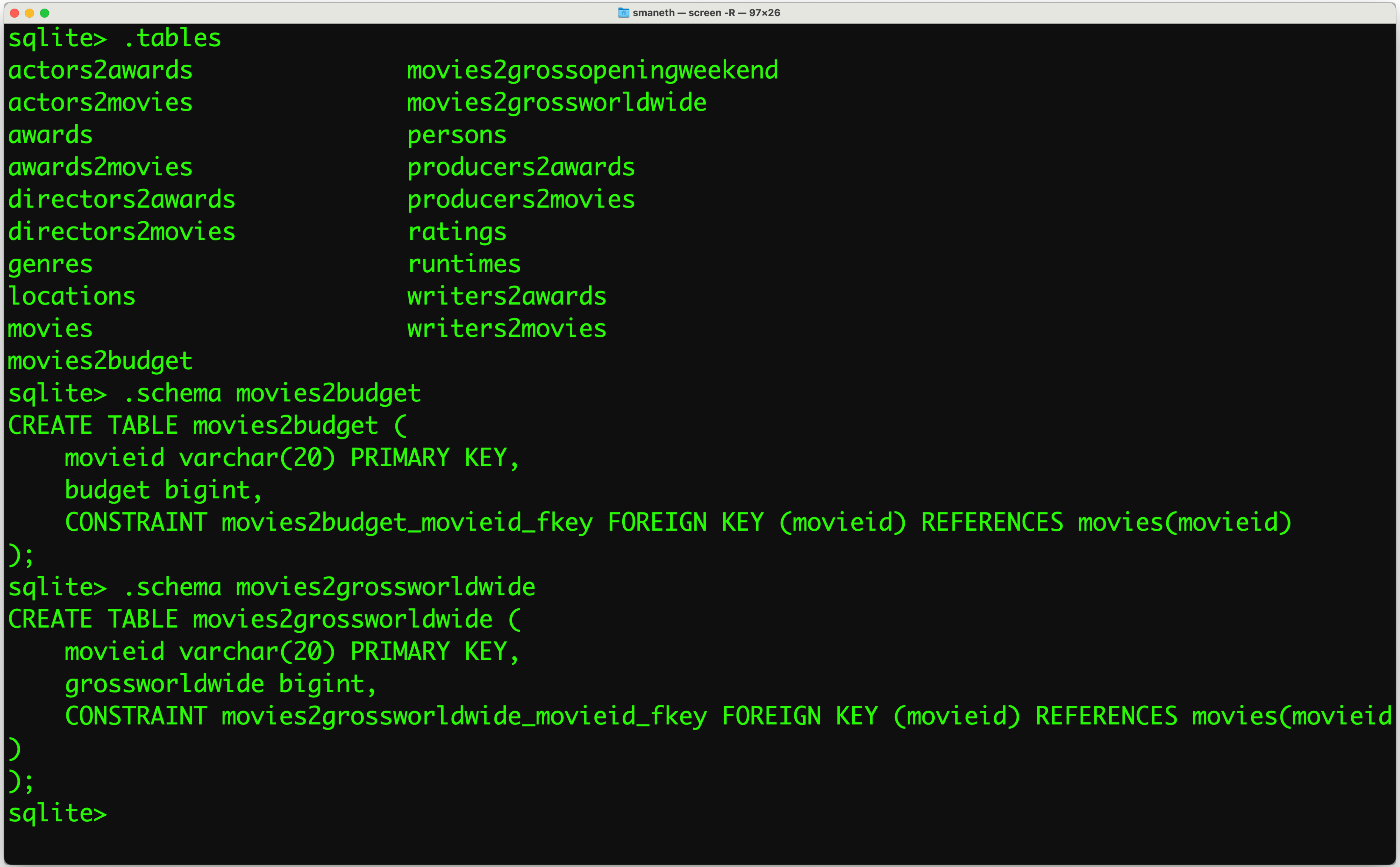
- Find movies that made at least 10 times as much as they cost:

```
select title
from finance
where gross > 10 * budget
```

- Find profit each movie made:

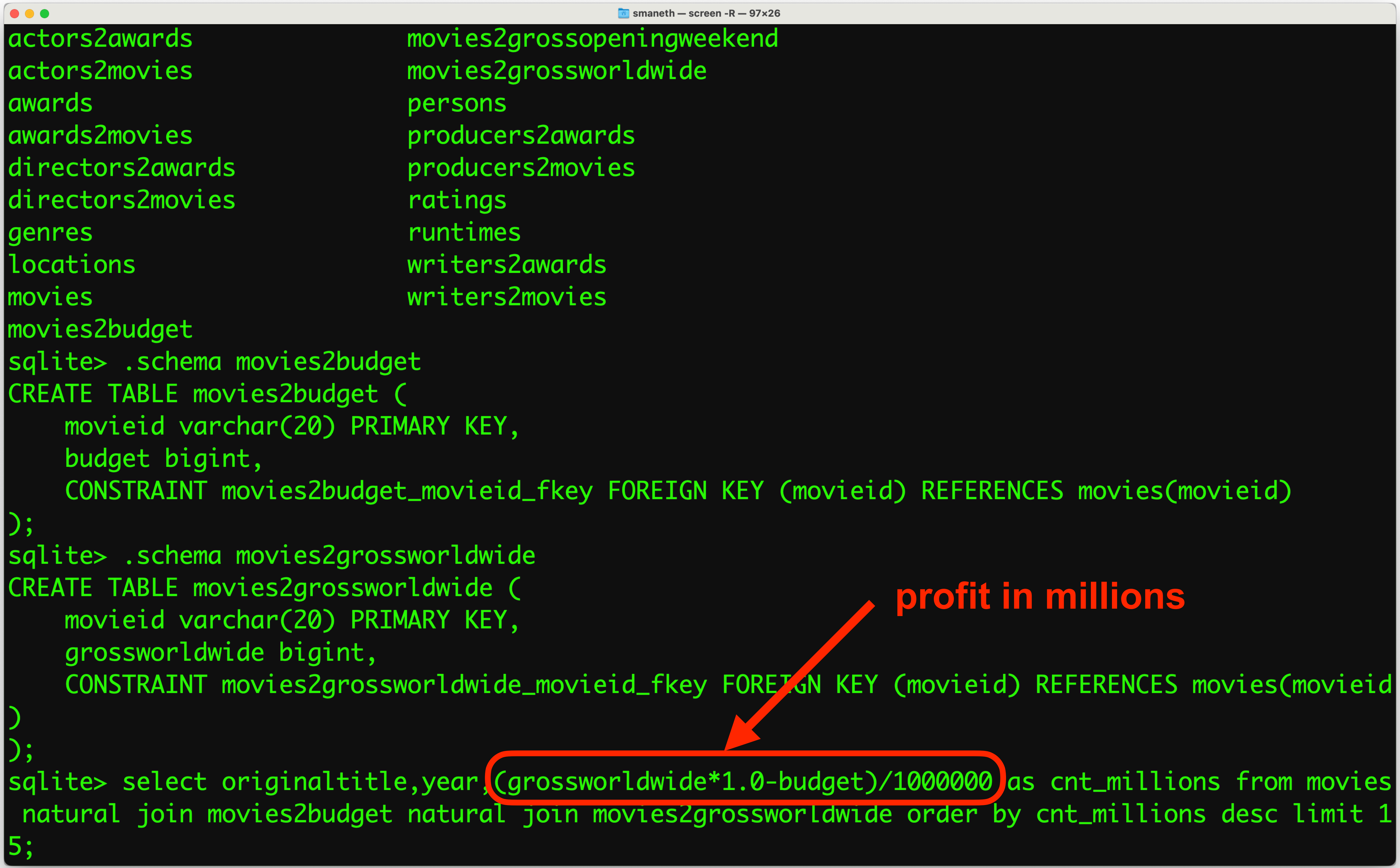
```
select title, gross - budget as profit
from finance
where gross - budget > 0
```

Which movies made the most profit?



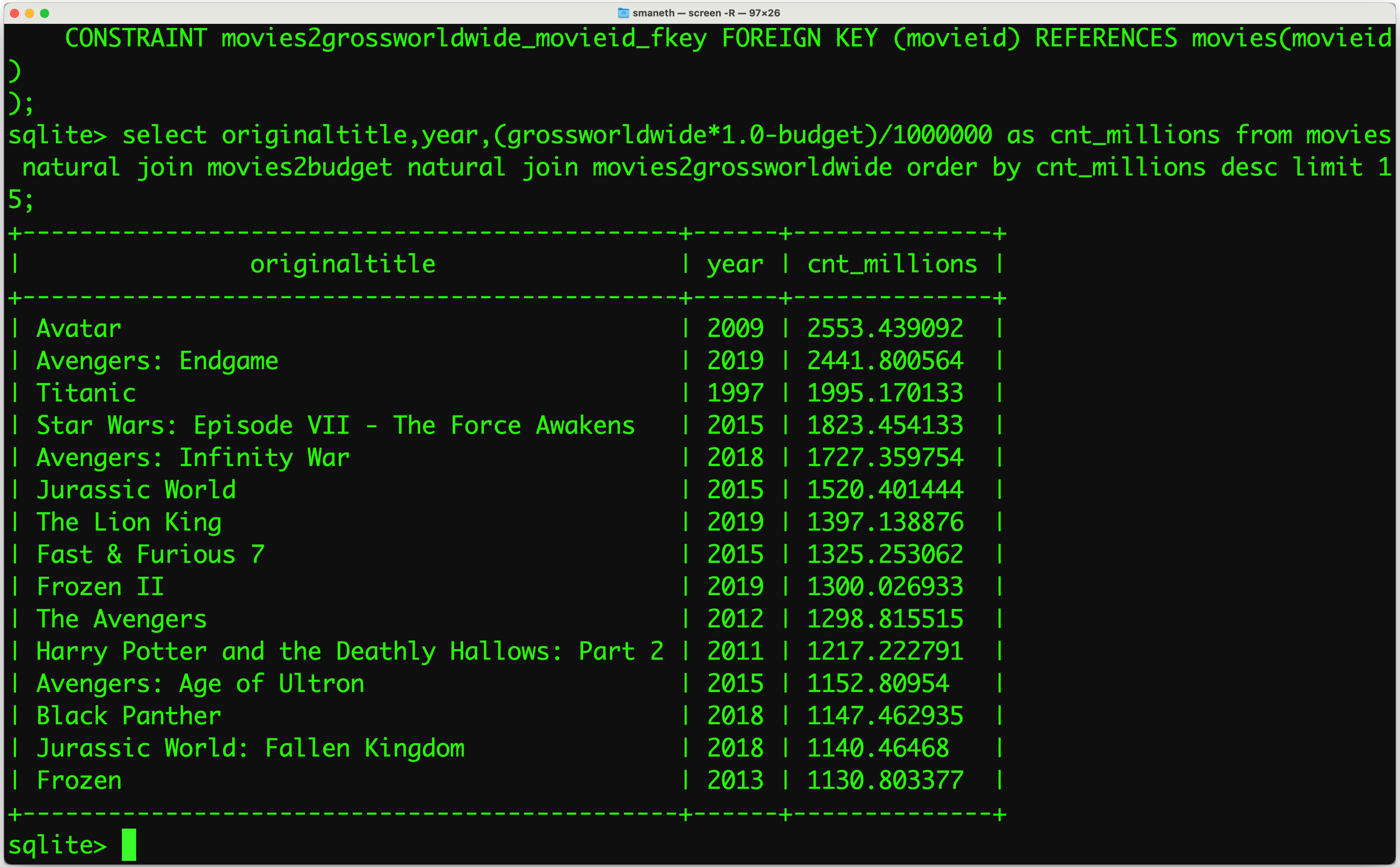
```
sqlite> .tables
actors2awards                 movies2grossopeningweekend
actors2movies                 movies2grossworldwide
awards                        persons
awards2movies                 producers2awards
directors2awards              producers2movies
directors2movies              ratings
genres                        runtimes
locations                     writers2awards
movies                        writers2movies
movies2budget
sqlite> .schema movies2budget
CREATE TABLE movies2budget (
    movieid varchar(20) PRIMARY KEY,
    budget bigint,
    CONSTRAINT movies2budget_movieid_fkey FOREIGN KEY (movieid) REFERENCES movies(movieid)
);
sqlite> .schema movies2grossworldwide
CREATE TABLE movies2grossworldwide (
    movieid varchar(20) PRIMARY KEY,
    grossworldwide bigint,
    CONSTRAINT movies2grossworldwide_movieid_fkey FOREIGN KEY (movieid) REFERENCES movies(movieid
)
);
sqlite>
```

Which movies made the most profit?



```
actors2awards                movies2grossopeningweekend
actors2movies                movies2grossworldwide
awards                       persons
awards2movies                producers2awards
directors2awards             producers2movies
directors2movies             ratings
genres                       runtimes
locations                    writers2awards
movies                       writers2movies
movies2budget
sqlite> .schema movies2budget
CREATE TABLE movies2budget (
    movieid varchar(20) PRIMARY KEY,
    budget bigint,
    CONSTRAINT movies2budget_movieid_fkey FOREIGN KEY (movieid) REFERENCES movies(movieid)
);
sqlite> .schema movies2grossworldwide
CREATE TABLE movies2grossworldwide (
    movieid varchar(20) PRIMARY KEY,
    grossworldwide bigint,
    CONSTRAINT movies2grossworldwide_movieid_fkey FOREIGN KEY (movieid) REFERENCES movies(movieid
)
);
sqlite> select originaltitle,year,(grossworldwide*1.0-budget)/1000000 as cnt_millions from movies
 natural join movies2budget natural join movies2grossworldwide order by cnt_millions desc limit 1
5;
```

Which movies made the most profit?



```
    CONSTRAINT movies2grossworldwide_movieid_fkey FOREIGN KEY (movieid) REFERENCES movies(movieid
)
);
sqlite> select originaltitle,year,(grossworldwide*1.0-budget)/1000000 as cnt_millions from movies
 natural join movies2budget natural join movies2grossworldwide order by cnt_millions desc limit 1
5;
+-----------------------------------------------+------+--------------+
|                 originaltitle                 | year | cnt_millions |
+-----------------------------------------------+------+--------------+
| Avatar                                        | 2009 | 2553.439092  |
| Avengers: Endgame                             | 2019 | 2441.800564  |
| Titanic                                       | 1997 | 1995.170133  |
| Star Wars: Episode VII - The Force Awakens    | 2015 | 1823.454133  |
| Avengers: Infinity War                        | 2018 | 1727.359754  |
| Jurassic World                                | 2015 | 1520.401444  |
| The Lion King                                 | 2019 | 1397.138876  |
| Fast & Furious 7                              | 2015 | 1325.253062  |
| Frozen II                                     | 2019 | 1300.026933  |
| The Avengers                                  | 2012 | 1298.815515  |
| Harry Potter and the Deathly Hallows: Part 2  | 2011 | 1217.222791  |
| Avengers: Age of Ultron                       | 2015 | 1152.80954   |
| Black Panther                                 | 2018 | 1147.462935  |
| Jurassic World: Fallen Kingdom                | 2018 | 1140.46468   |
| Frozen                                        | 2013 | 1130.803377  |
+-----------------------------------------------+------+--------------+
sqlite> 
```

-
  ```
  create table finance (title char(20),
                        budget int,
                        gross int)

  insert into finance values (‘Shining’, 19, 100)
  insert into finance values (‘Star wars’, 11, 513)
  insert into finance values (‘Wild wild west’, 170, 80)
  ```

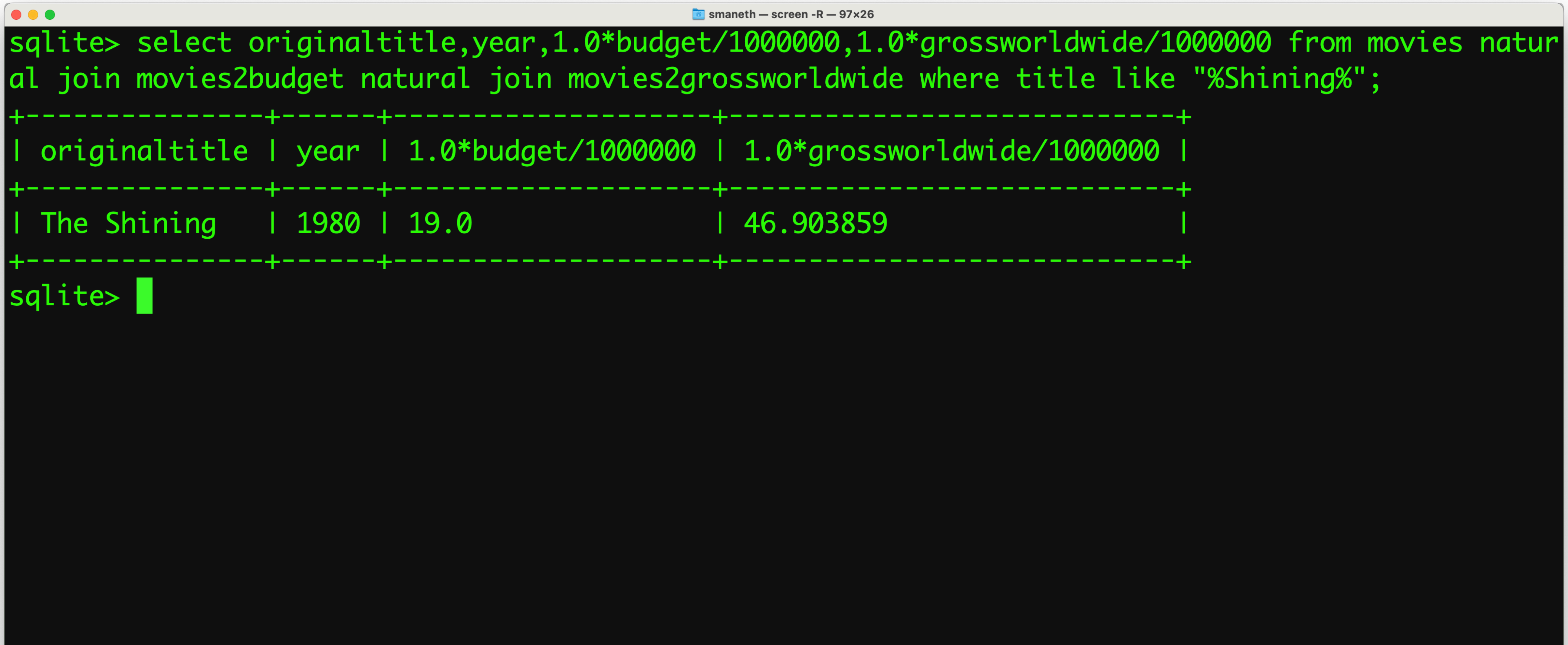
Are these numbers correct, wrt our IMDB-movie database?

-
  ```
  create table finance (title char(20),
                        budget int,
                        gross int)

  insert into finance values ('Shining', 19, 100)
  insert into finance values ('Star wars', 11, 513)
  insert into finance values ('Wild wild west', 170, 80)
  ```

Are these numbers correct, wrt our IMDB-movie database?

```
• create table finance (title char(20),
                        budget int,
                        gross int)

  insert into finance values ('Shining', 19, 100)
  insert into finance values ('Star wars', 11, 513)
  insert into finance values ('Wild wild west', 170, 80)
```

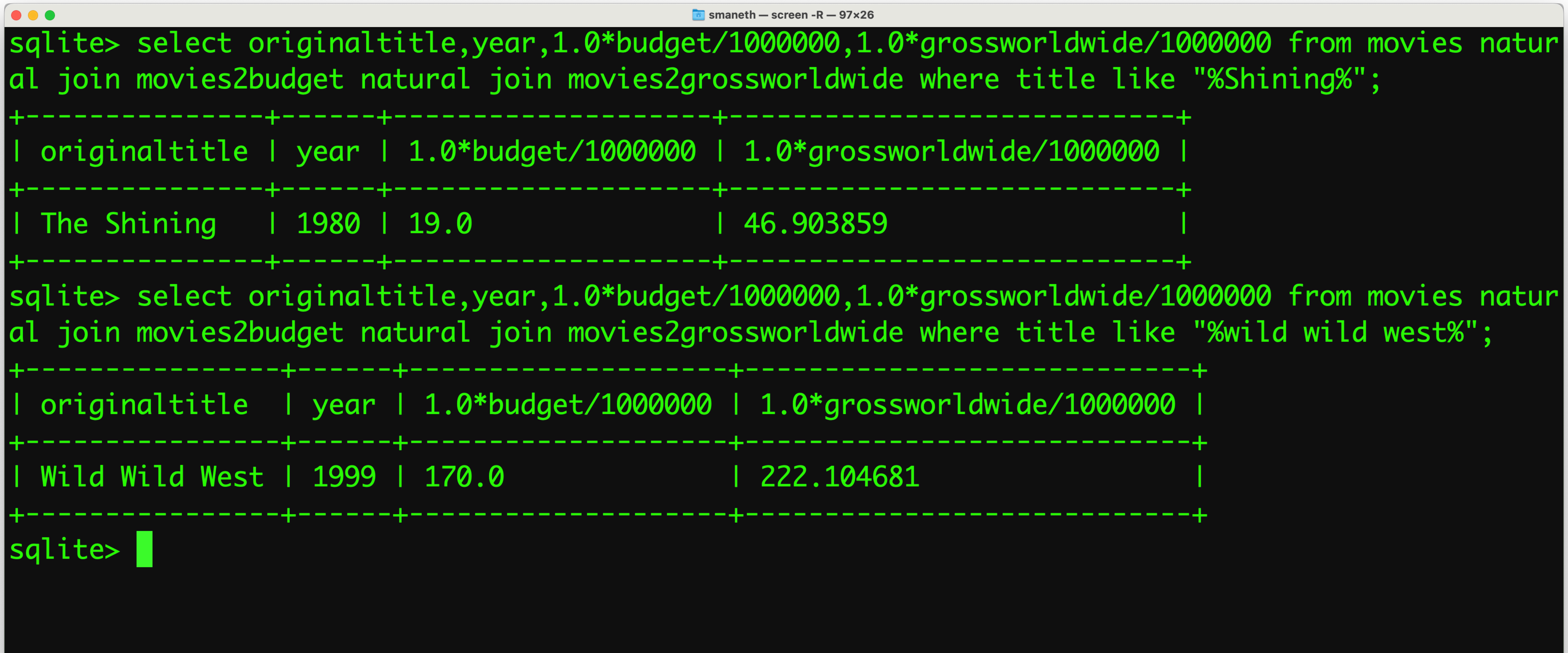
Are these numbers correct, wrt our IMDB-movie database?



```
sqlite> select originaltitle,year,1.0*budget/1000000,1.0*grossworldwide/1000000 from movies natur
al join movies2budget natural join movies2grossworldwide where title like "%Shining%";
+---------------+------+--------------------+---------------------------+
| originaltitle | year | 1.0*budget/1000000 | 1.0*grossworldwide/1000000 |
+---------------+------+--------------------+---------------------------+
| The Shining   | 1980 | 19.0               | 46.903859                 |
+---------------+------+--------------------+---------------------------+
sqlite> 
```

```
• create table finance (title char(20),
                        budget int,
                        gross int)

  insert into finance values (‘Shining’, 19, 100)
  insert into finance values (‘Star wars’, 11, 513)
  insert into finance values (‘Wild wild west’, 170, 80)
```

Are these numbers correct, wrt our IMDB-movie database?



smaneth — screen -R — 97×26

```
sqlite> select originaltitle,year,1.0*budget/1000000,1.0*grossworldwide/1000000 from movies natur
al join movies2budget natural join movies2grossworldwide where title like "%Shining%";
+---------------+------+--------------------+---------------------------+
| originaltitle | year | 1.0*budget/1000000 | 1.0*grossworldwide/1000000 |
+---------------+------+--------------------+---------------------------+
| The Shining   | 1980 | 19.0               | 46.903859                 |
+---------------+------+--------------------+---------------------------+
sqlite> select originaltitle,year,1.0*budget/1000000,1.0*grossworldwide/1000000 from movies natur
al join movies2budget natural join movies2grossworldwide where title like "%wild wild west%";
+----------------+------+--------------------+---------------------------+
| originaltitle  | year | 1.0*budget/1000000 | 1.0*grossworldwide/1000000 |
+----------------+------+--------------------+---------------------------+
| Wild Wild West | 1999 | 170.0              | 222.104681                |
+----------------+------+--------------------+---------------------------+
sqlite> 
```

More on the WHERE clause cont'd

- Is Kubrick spelled with a "k" or "ck" at the end?
- No need to remember.

matches *any string*

```
SELECT Title, Director
FROM Movies
WHERE director LIKE 'Kubr%'
```

- Is Polanski spelled with a "y" or with an "i"?

```
SELECT Title, Director
FROM Movies
WHERE director LIKE 'Polansk_'
```

matches *any character*

`LIKE` comparisons

- `attribute LIKE pattern`
- Patterns are built from:

  letters

  _ – stands for any letter

  % – stands for any substring, including empty
- Examples:

  `address LIKE '%Edinburgh%'`

  pattern `'_a_b_'` matches `cacbc`, `aabba`, etc

  pattern `'%a%b_'` matches `ccaccbc`, `aaaabcbcbbd`, `aba`, etc

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

$+$

```
SELECT Alcohol
  FROM Ingredients
 WHERE Alcohol=0;
```

$=$

?

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

$+$

```
SELECT Alcohol
  FROM Ingredients
 WHERE Alcohol=0;
```

$=$

Alcohol
0.0
0.0

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

$+$

```
SELECT Alcohol
  FROM Ingredients
 WHERE Alcohol=0;
```

$=$

Alcohol
0.0
0.0

This is **not** a table as we want them (contains duplicates)!!

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

$+$

```
SELECT Alcohol
  FROM Ingredients
 WHERE Alcohol=0;
```

$=$

Alcohol
0.0
0.0

This is **not** a table as we want them (contains duplicates)!!

Note:

— result tables are not like ordinary DB tables
— they may contain **duplicates**
— they may be **ordered** in a particular way

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

Imagine there were hundrets of entries with Alcohol=0.

Typically we would rather **count them**.

—> show each distinct Alcohol value,
together with the **number of ingredients** that have that Alcohol value

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

Imagine there were hundrets of entries with Alcohol=0.

Typically we would rather **count them**.

—> show each distinct Alcohol value,
together with the **number of ingredients** that have the Alcohol value

—> this is precisely what you can do with
GROUP BY (we will learn more about this later)

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

```
SELECT Alcohol,count(*) AS c
  FROM Ingredients
 GROUP BY Alcohol
```

=

?

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

```
SELECT Alcohol,count(*) AS c
      FROM Ingredients
      GROUP BY Alcohol
```

Alcohol	c
0.0	2
25.0	1
37.5	1

Data in Multiple Tables

Cocktail ingredients are sold by various suppliers (for a certain price), which could be represented as

SoldBy			
Ingredient	Supplier	DelTim[3]	Price
Orange Juice	A&P Supermarket	1	2.49
Orange Juice	Shop Rite	3	2.79
Campari	Joe's Liquor Store	2	14.95
Bacardi	Liquor's & More	5	13.99
Mineral Water	Shop Rite	3	1.89
Bacardi	Joe's Liquor Store	2	14.99

[3]Delivery time in days.

Data in Multiple Tables

Cocktail ingredients are sold by various suppliers (for a certain price), which could be represented as

SoldBy			
Ingredient	Supplier	DelTim[3]	Price
Orange Juice	A&P Supermarket	1	2.49
Orange Juice	Shop Rite	3	2.79
Campari	Joe's Liquor Store	2	14.95
Bacardi	Liquor's & More	5	13.99
Mineral Water	Shop Rite	3	1.89
Bacardi	Joe's Liquor Store	2	14.99

many-to-many relationship!

[3]Delivery time in days.

When multiple tables are reference in the `FROM` clause, this is interpreted as the **Cartesian product** of the referenced tables:[4]

```
SELECT *
  FROM Ingredients, SoldBy
```

Ingredients				SoldBy			
Name	Alcohol	InStock	Price	Ingredient	Supplier	DelTim	Price
Orange Juice	0.0	12	2.99	Orange Juice	A&P Supermarket	1	2.49
Orange Juice	0.0	12	2.99	Orange Juice	Shop Rite	3	2.79
Orange Juice	0.0	12	2.99	Campari	Joe's Liquor Store	2	14.95
Orange Juice	0.0	12	2.99	Bacardi	Liquors & More	5	13.99
Orange Juice	0.0	12	2.99	Mineral Water	Shop Rite	3	1.89
Orange Juice	0.0	12	2.99	Bacardi	Joe's Liquor Store	2	14.99
Campari	25.0	5	12.95	Orange Juice	A&P Supermarket	1	2.49
Campari	25.0	5	12.95	Orange Juice	Shop Rite	3	2.79
Campari	25.0	5	12.95	Campari	Joe's Liquor Store	2	14.95
Campari	25.0	5	12.95	Bacardi	Liquors & More	5	13.99
Campari	25.0	5	12.95	Mineral Water	Shop Rite	3	1.89
Campari	25.0	5	12.95	Bacardi	Joe's Liquor Store	2	14.99
Mineral Water	0.0	10	1.49	Orange Juice	A&P Supermarket	1	2.49
Mineral Water	0.0	10	1.49	Orange Juice	Shop Rite	3	2.79
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

[4] Use `*` in the `SELECT` clause when you simply want to choose all columns.

When multiple tables are reference in the `FROM` clause, this is interpreted as the **Cartesian product** of the referenced tables:[4]

```
SELECT *
  FROM Ingredients, SoldBy
```

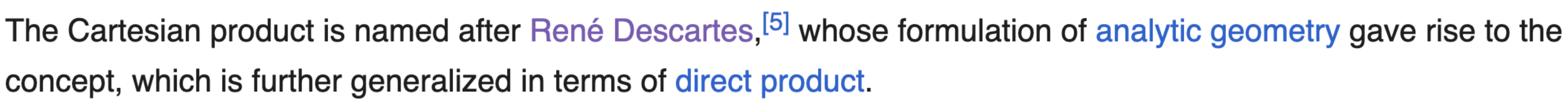
Ingredients				SoldBy			
Name	Alcohol	InStock	Price	Ingredient	Supplier	DelTim	Price
Orange Juice	0.0	12	2.99	Orange Juice	A&P Supermarket	1	2.49
Orange Juice	0.0	12	2.99	Orange Juice	Shop Rite	3	2.79
Orange Juice	0.0	12	2.99	Campari	Joe's Liquor Store	2	14.95
Orange Juice	0.0				& More	5	13.99
Orange Juice	0.0				te	3	1.89
Orange Juice	0.0				quor Store	2	14.99
Campari	25.0				permarket	1	2.49
Campari	25.0				te	3	2.79
Campari	25.0				quor Store	2	14.95
Campari	25.0				& More	5	13.99
Campari	25.0				ite	3	1.89
Campari	25.0	5	12.95	Bacardi	Joe's Liquor Store	2	14.99
Mineral Water	0.0	10	1.49	Orange Juice	A&P Supermarket	1	2.49
Mineral Water	0.0	10	1.49	Orange Juice	Shop Rite	3	2.79
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Poll

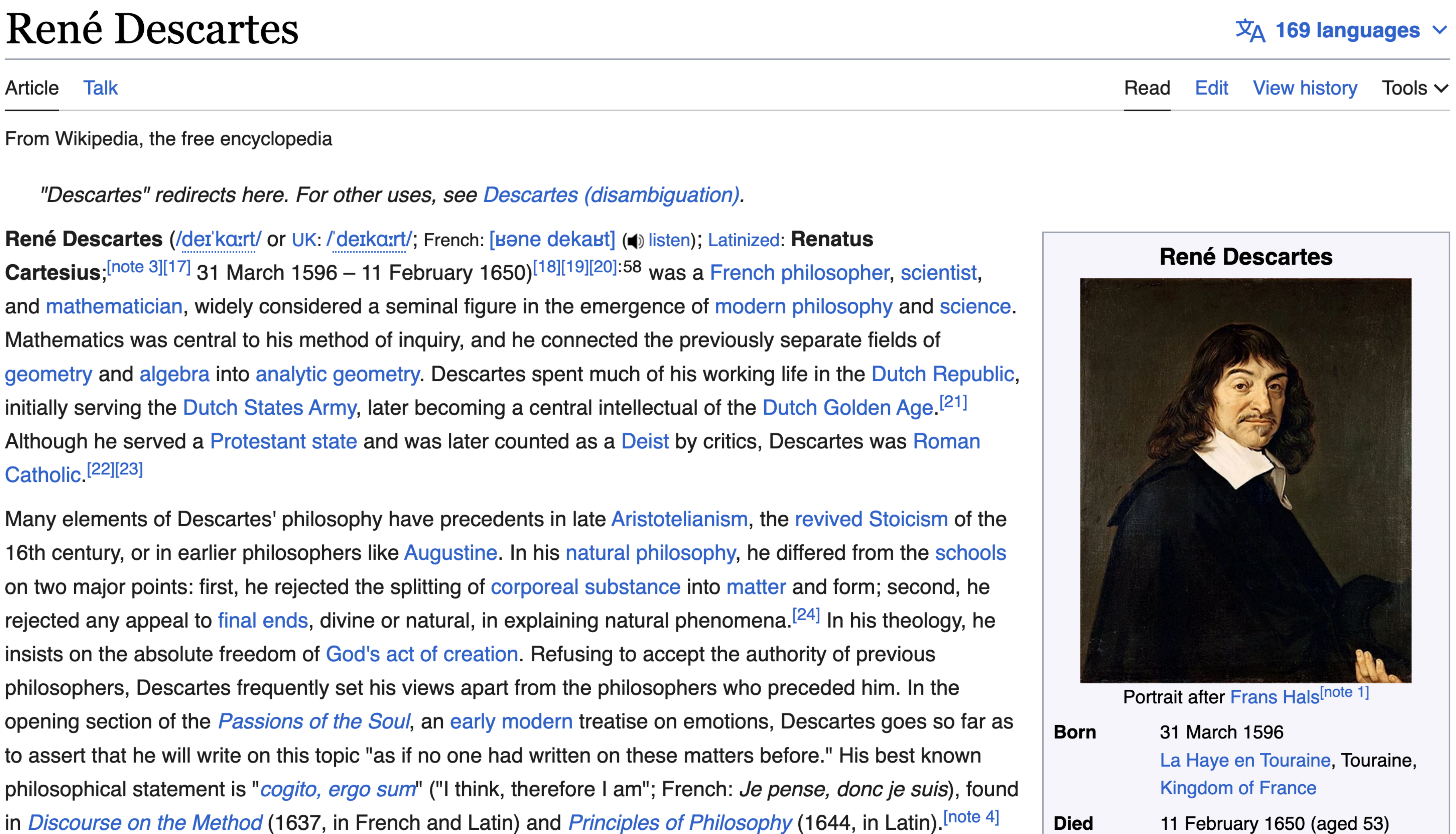
how many tuples (rows) are in this result table?

[4]Use `*` in the `SELECT` clause when you simply want to choose all columns.

When multiple tables are reference in the `FROM` clause, this is interpreted as the **Cartesian product** of the referenced tables:[4]



The Cartesian product is named after René Descartes,[5] whose formulation of analytic geometry gave rise to the concept, which is further generalized in terms of direct product.



René Descartes

169 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

"Descartes" redirects here. For other uses, see Descartes (disambiguation).

René Descartes (/deɪˈkɑːrt/ or UK: /ˈdeɪkɑːrt/; French: [ʁəne dekaʁt] (listen); Latinized: **Renatus Cartesius**;[note 3][17] 31 March 1596 – 11 February 1650)[18][19][20]:58 was a French philosopher, scientist, and mathematician, widely considered a seminal figure in the emergence of modern philosophy and science. Mathematics was central to his method of inquiry, and he connected the previously separate fields of geometry and algebra into analytic geometry. Descartes spent much of his working life in the Dutch Republic, initially serving the Dutch States Army, later becoming a central intellectual of the Dutch Golden Age.[21] Although he served a Protestant state and was later counted as a Deist by critics, Descartes was Roman Catholic.[22][23]

Many elements of Descartes' philosophy have precedents in late Aristotelianism, the revived Stoicism of the 16th century, or in earlier philosophers like Augustine. In his natural philosophy, he differed from the schools on two major points: first, he rejected the splitting of corporeal substance into matter and form; second, he rejected any appeal to final ends, divine or natural, in explaining natural phenomena.[24] In his theology, he insists on the absolute freedom of God's act of creation. Refusing to accept the authority of previous philosophers, Descartes frequently set his views apart from the philosophers who preceded him. In the opening section of the *Passions of the Soul*, an early modern treatise on emotions, Descartes goes so far as to assert that he will write on this topic "as if no one had written on these matters before." His best known philosophical statement is "*cogito, ergo sum*" ("I think, therefore I am"; French: *Je pense, donc je suis*), found in *Discourse on the Method* (1637, in French and Latin) and *Principles of Philosophy* (1644, in Latin).[note 4]

René Descartes

Portrait after Frans Hals[note 1]

Born	31 March 1596 La Haye en Touraine, Touraine, Kingdom of France
Died	11 February 1650 (aged 53)

These should be **equal**!

Ingredients				SoldBy			
Name	Alcohol	InStock	Price	Ingredient	Supplier	DelTim	Price
Orange Juice	0.0	12	2.99	Orange Juice	A&P Supermarket	1	2.49
Orange Juice	0.0	12	2.99	Orange Juice	Shop Rite	3	2.79
Orange Juice	0.0	12	2.99	Campari	Joe's Liquor Store	2	14.95
Orange Juice	0.0	12	2.99	Bacardi	Liquors & More	5	13.99
Orange Juice	0.0	12	2.99	Mineral Water	Shop Rite	3	1.89
Orange Juice	0.0	12	2.99	Bacardi	Joe's Liquor Store	2	14.99
Campari	25.0	5	12.95	Orange Juice	A&P Supermarket	1	2.49
Campari	25.0	5	12.95	Orange Juice	Shop Rite	3	2.79
Campari	25.0	5	12.95	Campari	Joe's Liquor Store	2	14.95
Campari	25.0	5	12.95	Bacardi	Liquors & More	5	13.99
Campari	25.0	5	12.95	Mineral Water	Shop Rite	3	1.89
Campari	25.0	5	12.95	Bacardi	Joe's Liquor Store	2	14.99
Mineral Water	0.0	10	1.49	Orange Juice	A&P Supermarket	1	2.49
Mineral Water	0.0	10	1.49	Orange Juice	Shop Rite	3	2.79
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

[4] Use * in the **SELECT** clause when you simply want to choose all columns.

Queries over Multiple Tables

In practice, you rarely want to see this Cartesian product in the final result.

→ Use a WHERE clause to select only semantically related data.

```
SELECT Name, InStock, Supplier
  FROM Ingredients, SoldBy
 WHERE Name = Ingredient
```

↓

Name	InStock	Supplier
Orange Juice	12	A&P Supermarket
Orange Juice	12	Shop Rite
Campari	5	Joe's Liquor Store
Mineral Water	10	Shop Rite
Bacardi	3	Liquors & More
Bacardi	3	Joe's Liquor Store

Queries over Multiple Tables

In practice, you rarely want to see this Cartesian product in the final result.

→ Use a `WHERE` clause to select only semantically related data.

```
SELECT Name, InStock, Supplier
  FROM Ingredients, SoldBy
 WHERE Name = Ingredient
```

This type of query is very common and is called a **JOIN**.

We will learn a lot about JOINS:

— NATURAL **JOIN**
— THETA **JOIN**
— SEMI **JOIN**
— OUTER **JOIN**
etc.

Queries over Multiple Tables

Resolve ambiguities by prepending column names with their table name:

```
SELECT Name, InStock, Supplier, SoldBy.Price
  FROM Ingredients, SoldBy
 WHERE Name = Ingredient
   AND SoldBy.Price < Ingredients.Price
```

↓

Name	InStock	Supplier	Price
Orange Juice	12	A&P Supermarket	2.49
Orange Juice	12	Shop Rite	2.79
Bacardi	3	Liquors & More	13.99
Bacardi	3	Joe's Liquor Store	14.99

Tuple Variables

...or introduce **tuple variables** for easier reference:

```
SELECT Name, InStock, Supplier, s.Price
  FROM Ingredients AS i, SoldBy AS s
 WHERE Name = Ingredient
   AND s.Price < i.Price
```

↓

Name	InStock	Supplier	Price
Orange Juice	12	A&P Supermarket	2.49
Orange Juice	12	Shop Rite	2.79
Bacardi	3	Liquors & More	13.99
Bacardi	3	Joe's Liquor Store	14.99

(The keyword `AS` is optional; '`SoldBy s`' would mean just the same.)

Semantics of SQL `SELECT-FROM-WHERE` Expressions

Conceptually, the query

```
SELECT AttList
  FROM TableName₁ , TableName₂ , . . .
 WHERE Condition
```

does the following:

$$\begin{tikzcd}
\mathit{TableName}_1 \arrow[dr] & & & & \\
\mathit{TableName}_2 \arrow[r] & \times \arrow[r] & \mathit{Condition} \arrow[r] & \mathit{AttList} \arrow[r] & {} \\
\vdots \arrow[ur] & & & &
\end{tikzcd}$$

`FROM` `WHERE` `SELECT`

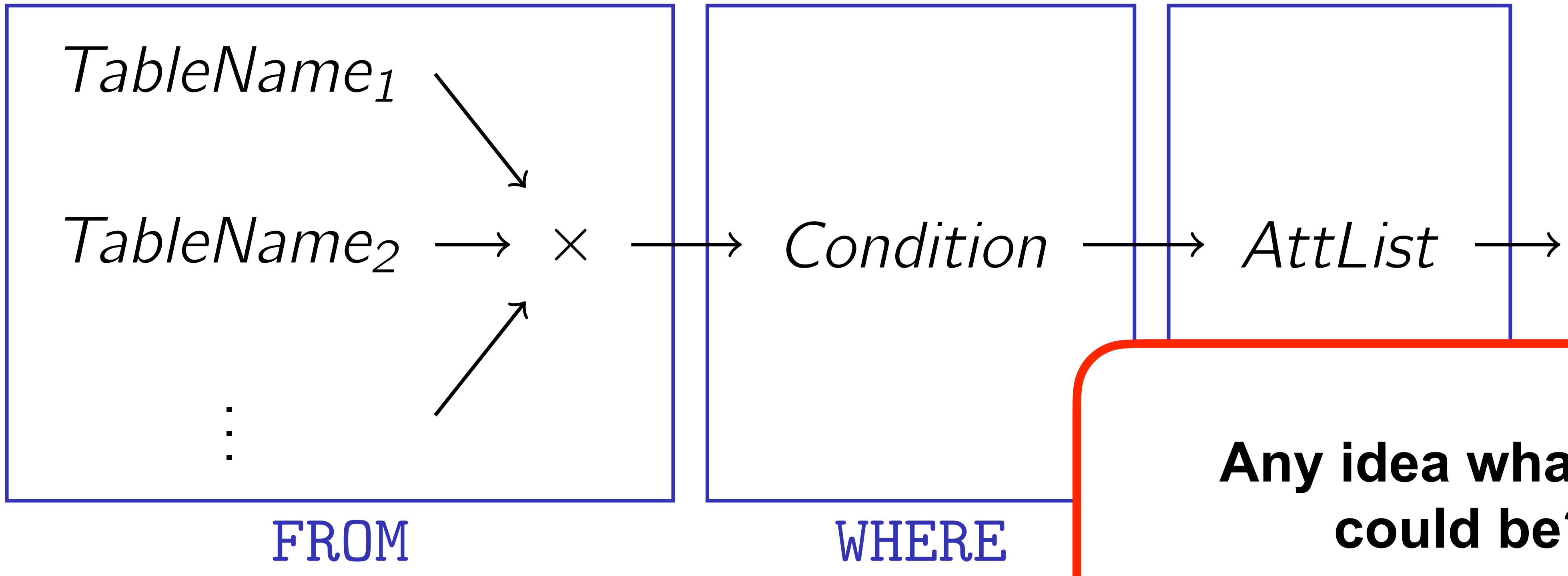
(But most likely, the database system will choose a better strategy to actually execute the query.)

Semantics of SQL SELECT-FROM-WHERE Expressions

Conceptually, the query

```
SELECT AttList
  FROM TableName1 , TableName2 , . . .
 WHERE Condition
```

does the following:



Any idea what that could be?

(But most likely, the database system will choose a better strategy to actually execute the query.)

Concluding Remarks

- SQL is **case insensitive**; use ’ as a **string delimiter**.
- It is okay to reference the **same table multiple times** in a `FROM` clause (→ “self-join”). Use **tuple variables** then to tell things apart.

Never, **never ever**, write queries where the correctness depends on the current table contents.

E.g., the correct answer to “give me names and prices of all non-alcoholic ingredients” is **not**

```
SELECT Name, Price
  FROM Ingredients
 WHERE Name = 'Orange Juice' OR Name = 'Mineral Water'
```

Concluding Remarks

- SQL is **case insensitive**; use ' as a **string delimiter**.
- It is okay to reference the **same table multiple times** in a `FROM` clause (→ "self-join"). Use **tuple variables** then to tell things apart.

Never, **never ever**, write queries where the correctness depends on the current table contents.

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

Give the name of the ingredients that have the highest Alcohol value

```
SELECT Name
  FROM Ingredients
 WHERE Alcohol=37.5;
```

4.) Aggregate Funtions

4.) Aggregate Funtions

aggregare (lat.) zusammenhäufen, aufhäufen

4.) Aggregate Funtions

aggregare (lat.) zusammenhäufen, aufhäufen

e.g. build the **sum** of all price-values

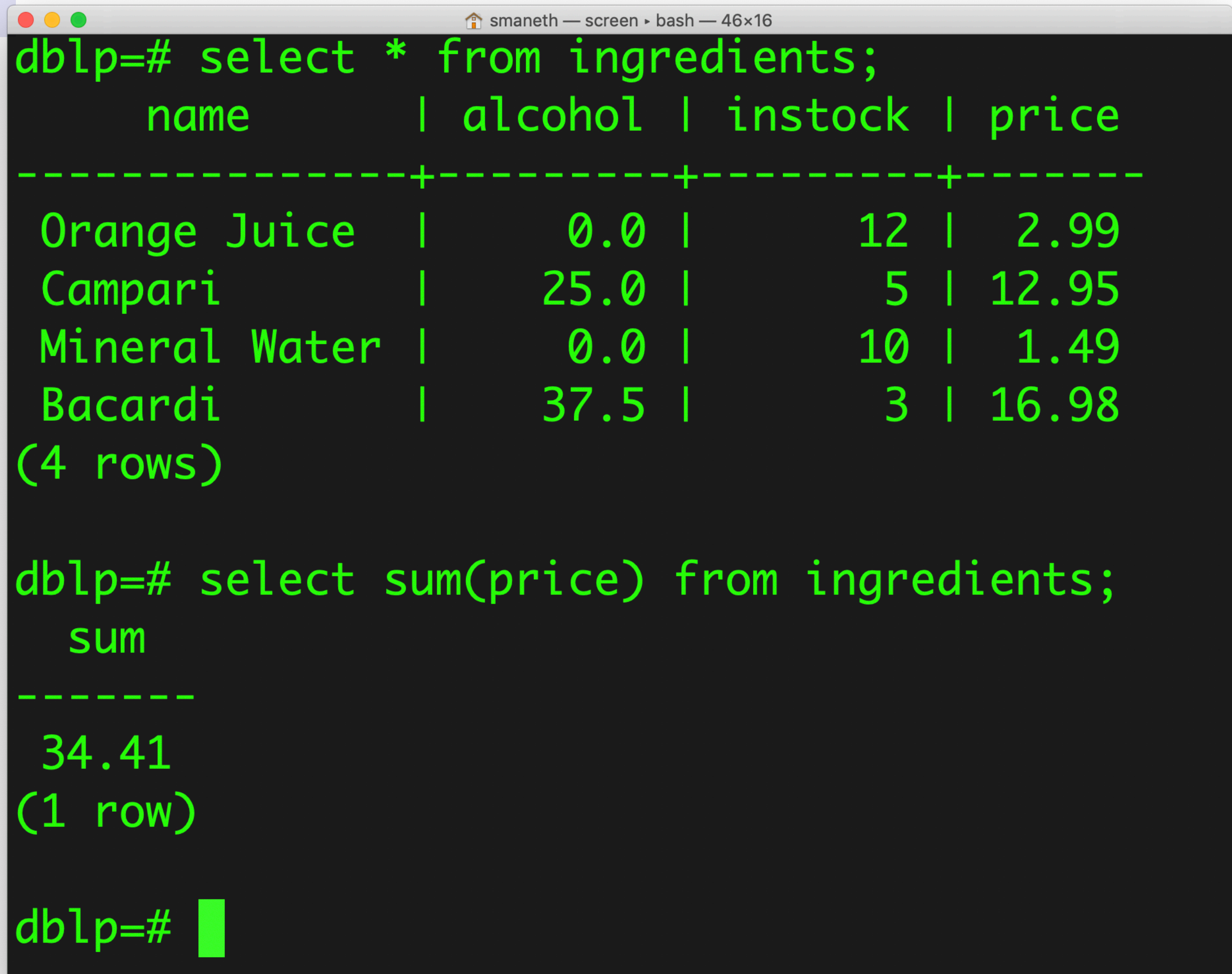
4.) Aggregate Funtions

aggregare (lat.) zusammenhäufen, aufhäufen

e.g. build the **sum** of all price-values

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

```
SELECT SUM(Price)
  FROM Ingredients;
```



```
dblp=# select * from ingredients;
     name      | alcohol | instock | price
---------------+---------+---------+-------
 Orange Juice  |     0.0 |      12 |  2.99
 Campari       |    25.0 |       5 | 12.95
 Mineral Water |     0.0 |      10 |  1.49
 Bacardi       |    37.5 |       3 | 16.98
(4 rows)

dblp=# select sum(price) from ingredients;
  sum
-------
 34.41
(1 row)

dblp=# 
```

4.) Aggregate Funtions

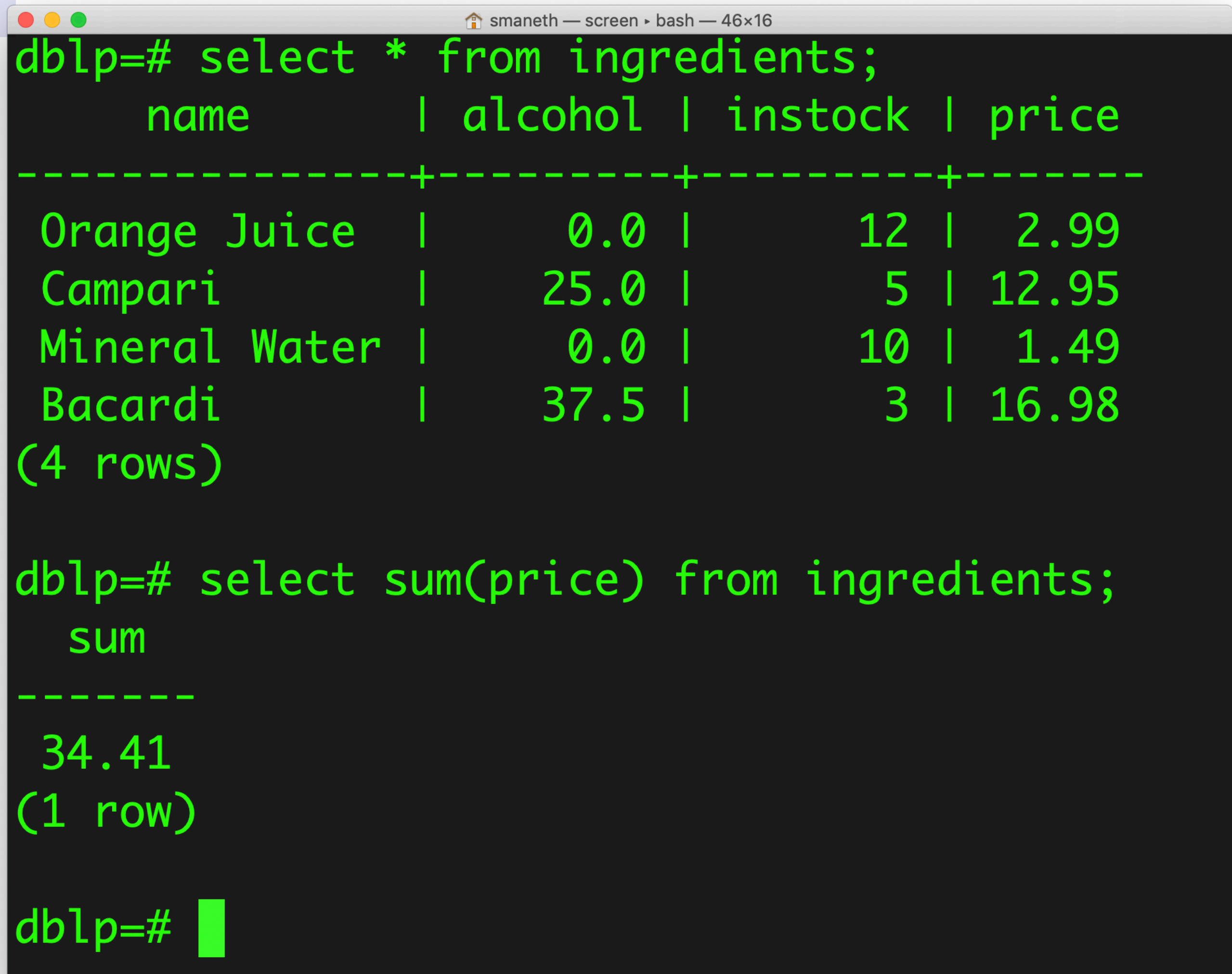
aggregare (lat.) zusammenhäufen, aufhäufen

e.g. build the **sum** of all price-values

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

If we assume Price is our selling-price, we may want to multiply by the InStock value!

```
SELECT SUM(Price)
   FROM Ingredients;
```



```
dblp=# select * from ingredients;
     name      | alcohol | instock | price
---------------+---------+---------+-------
 Orange Juice  |     0.0 |      12 |  2.99
 Campari       |    25.0 |       5 | 12.95
 Mineral Water |     0.0 |      10 |  1.49
 Bacardi       |    37.5 |       3 | 16.98
(4 rows)

dblp=# select sum(price) from ingredients;
  sum
-------
 34.41
(1 row)

dblp=# 
```

4.) Aggregate Funtions

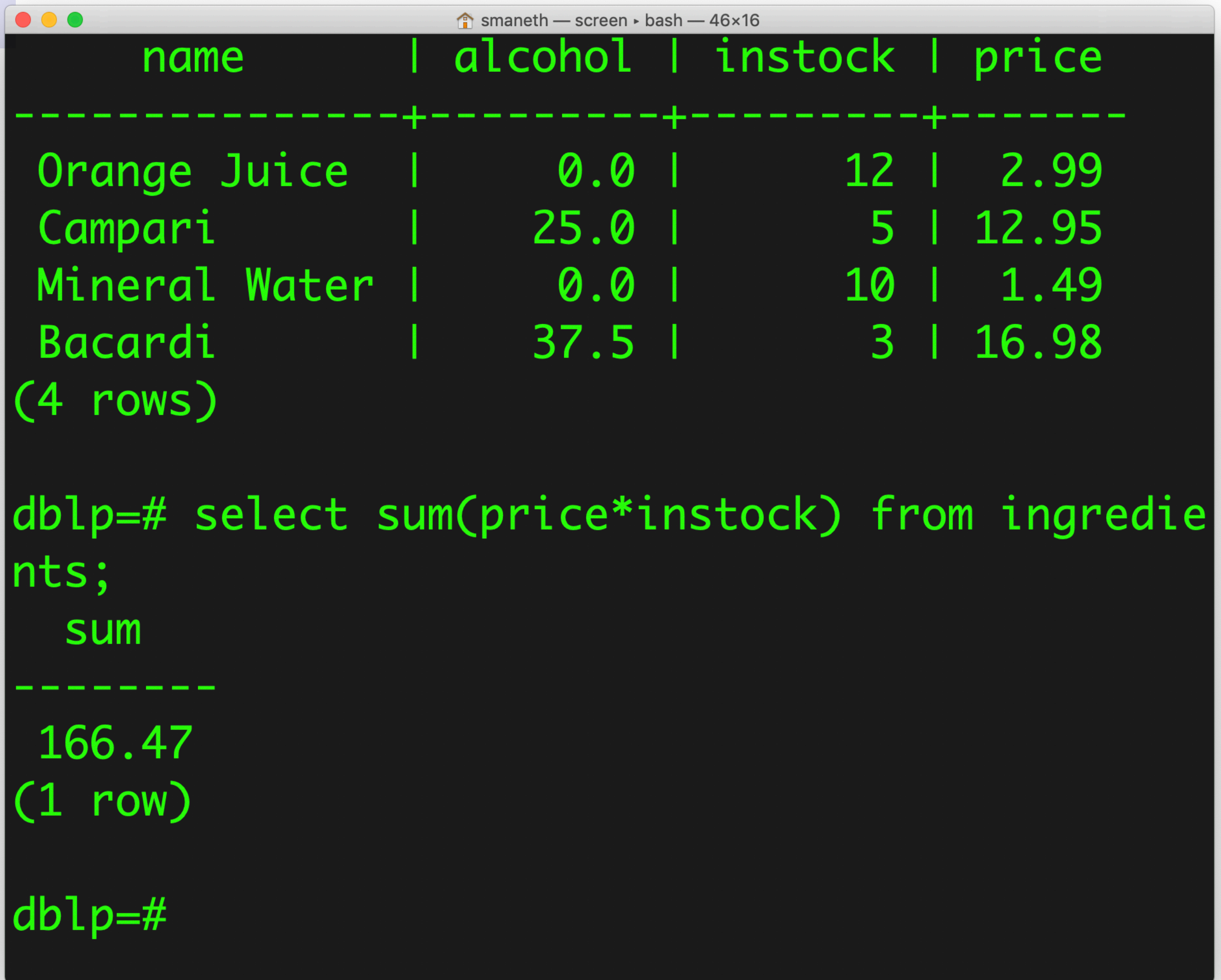
aggregare (lat.) zusammenhäufen, aufhäufen

e.g. build the **sum** of all price-values

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

If we assume Price is our selling-price, we may want to multiply by the InStock value!

```
SELECT SUM(Price*InStock)
  FROM Ingredients;
```



```
     name      | alcohol | instock | price
---------------+---------+---------+-------
 Orange Juice  |     0.0 |      12 |  2.99
 Campari       |    25.0 |       5 | 12.95
 Mineral Water |     0.0 |      10 |  1.49
 Bacardi       |    37.5 |       3 | 16.98
(4 rows)

dblp=# select sum(price*instock) from ingredie
nts;
   sum
--------
 166.47
(1 row)

dblp=#
```

4.) Aggregate Funtions

aggregare (lat.) zusammenhäufen, aufhäufen

e.g. build the **average** of all price-values

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

```
SELECT AVG(Price)
  FROM Ingredients;
```

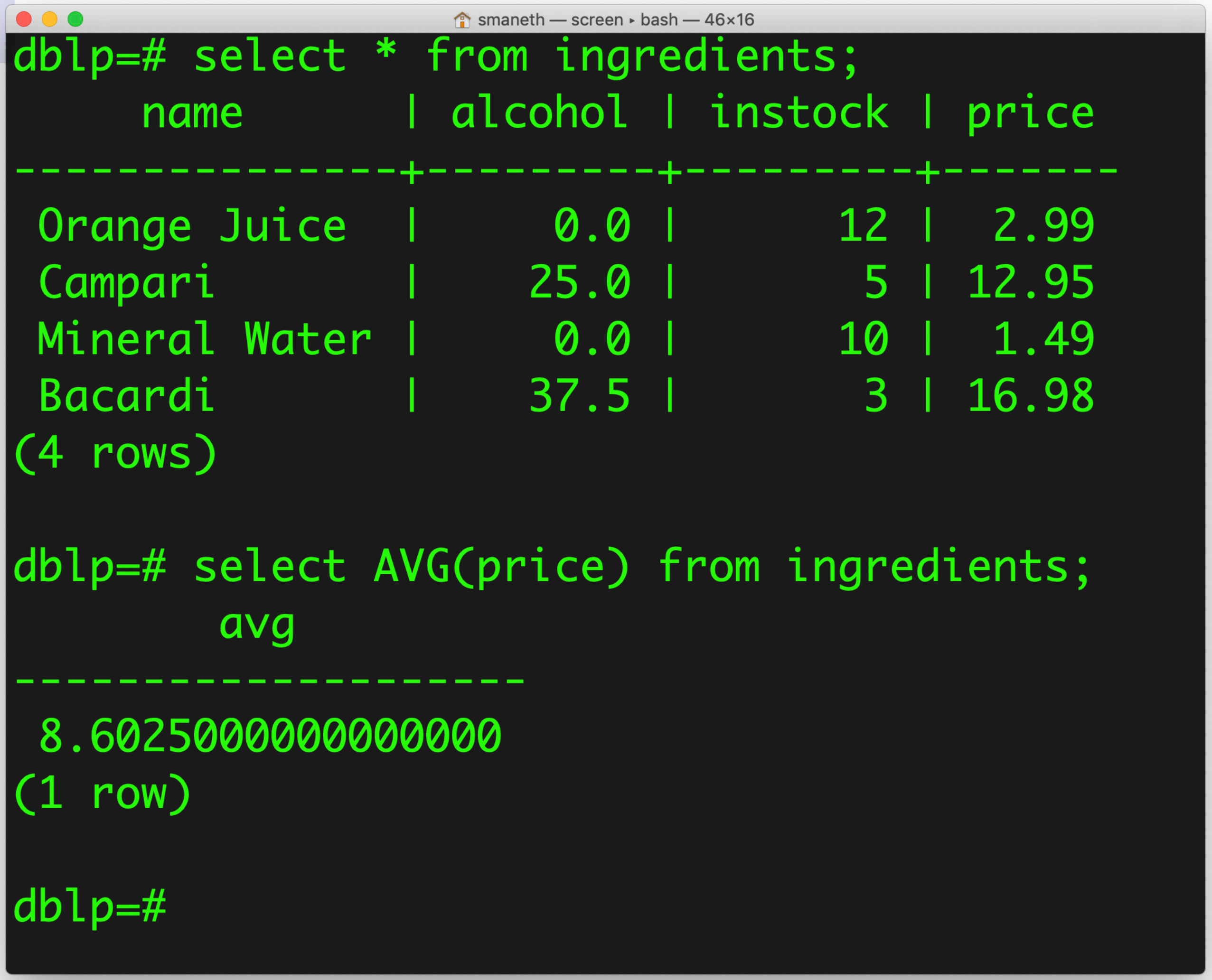
4.) Aggregate Funtions

aggregare (lat.) zusammenhäufen, aufhäufen

e.g. build the **average** of all price-values

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

```
SELECT AVG(Price)
  FROM Ingredients;
```



```
dblp=# select * from ingredients;
     name      | alcohol | instock | price
---------------+---------+---------+-------
 Orange Juice  |     0.0 |      12 |  2.99
 Campari       |    25.0 |       5 | 12.95
 Mineral Water |     0.0 |      10 |  1.49
 Bacardi       |    37.5 |       3 | 16.98
(4 rows)

dblp=# select AVG(price) from ingredients;
        avg
--------------------
 8.6025000000000000
(1 row)

dblp=#
```

4.) Aggregate Funtions

aggregare (lat.) zusammenhäufen, aufhäufen

e.g. build the **count** of all tuples

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

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

4.) Aggregate Funtions

aggregare (lat.) zusammenhäufen, aufhäufen

e.g. build the **count** of all tuples

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

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

Either **one attribute** here, or the star (*).

Same result here, for any of the four attributes.

DISTINCT keyword

Find the number of directors.

Naive approach:

```
SELECT COUNT(director)
  FROM Movies;
```

This is the same as just counting
the number of rows
in the table Movies.

DISTINCT keyword

Find the number of directors.

Naive approach:

```
SELECT COUNT(director)
  FROM Movies;
```

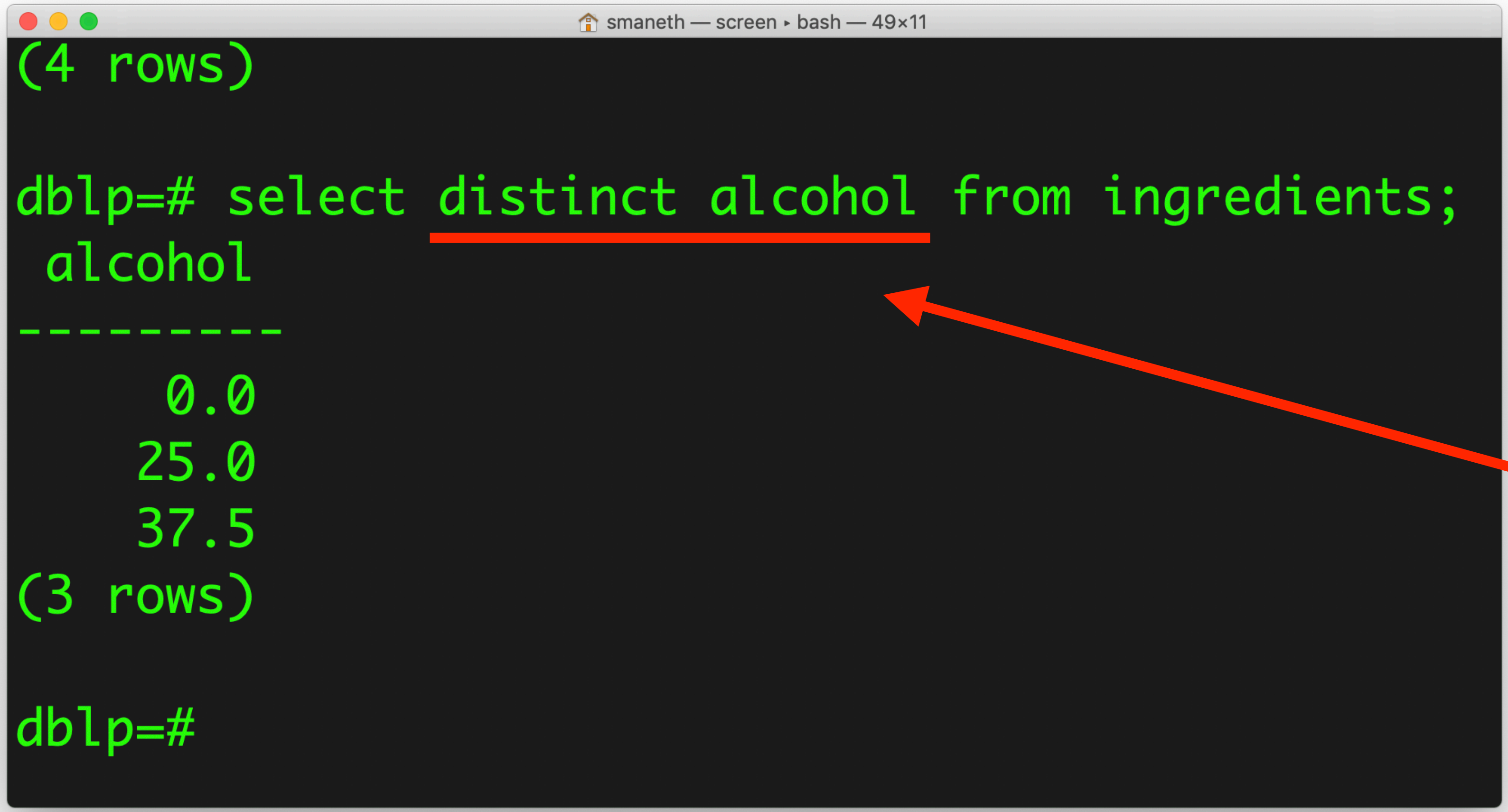
This is the same as just counting the number of rows in the table Movies.

Correct:

```
SELECT COUNT(DISTINCT director)
  FROM Movies;
```

DISTINCT keyword

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98



```
(4 rows)

dblp=# select distinct alcohol from ingredients;
 alcohol
---------
     0.0
    25.0
    37.5
(3 rows)

dblp=#
```

select all distinct alcohol values

DISTINCT keyword

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

After DISTINCT we may have several attributes!

```
SELECT DISTINCT alcohol, instock
  FROM Ingredients;
```

What is the result of this query?

4.) Aggregate Funtions

aggregare (lat.) zusammenhäufen, aufhäufen

e.g. build the **count** of all tuples

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

```
SELECT COUNT(DISTINCT Alcohol)
  FROM Ingredients;
```

What is the result of this query?

4.) Aggregate Funtions

aggregare (lat.) zusammenhäufen, aufhäufen

e.g. build the **count** of all tuples

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

```
SELECT SUM(Price)/COUNT(*)
  FROM Ingredients;
```

What is the result of this query?

4.) Aggregate Funtions

aggregare (lat.) zusammenhäufen, aufhäufen

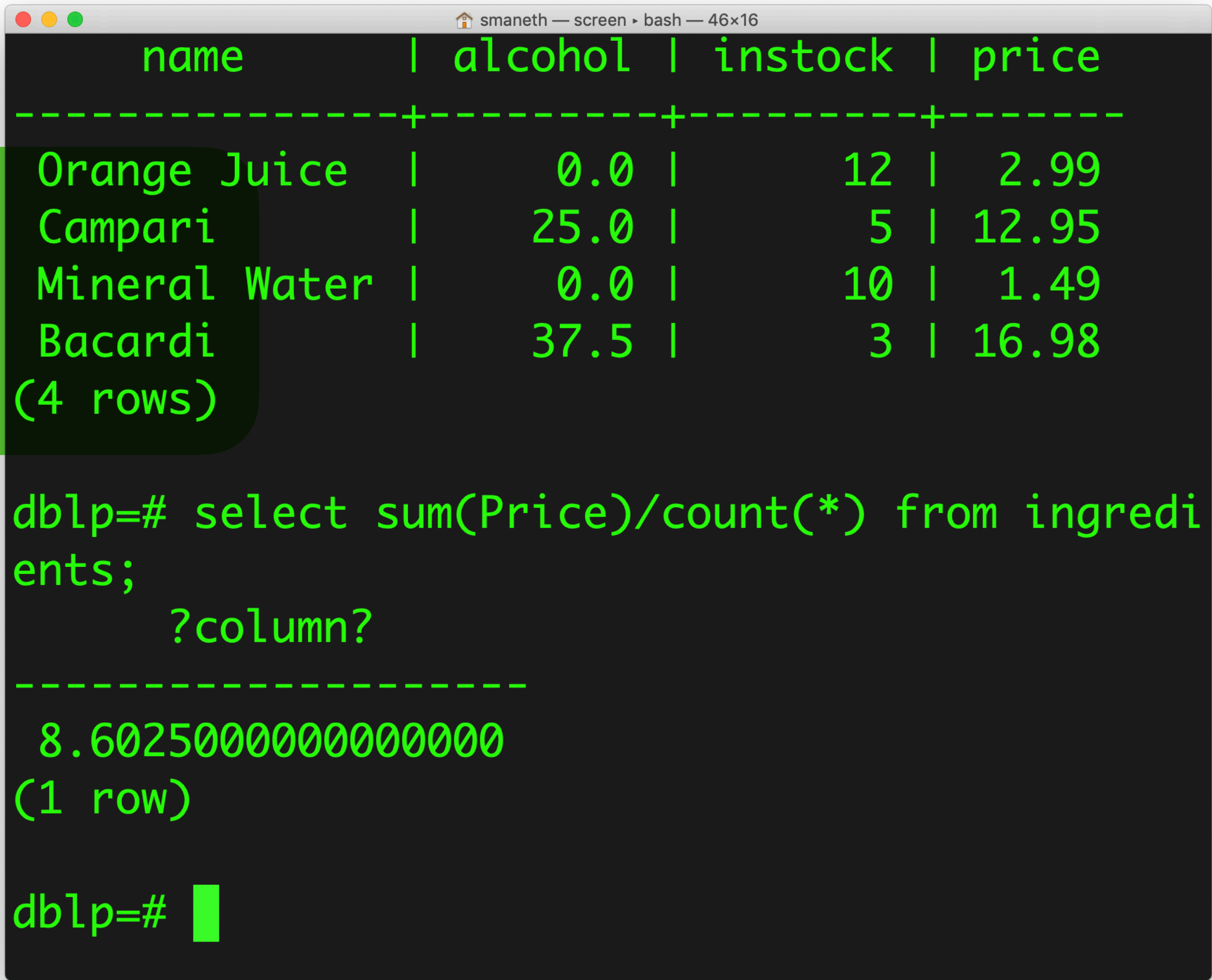
e.g. build the **count** of all tuples

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Mineral Water	0.0	10	1.49
Bacardi	37.5	3	16.98

Same as AVG( )!

```
SELECT SUM(Price)/COUNT(*)
  FROM Ingredients;
```

What is the result of this query?



smaneth — screen ▸ bash — 46×16

```
    name        | alcohol | instock | price
----------------+---------+---------+-------
 Orange Juice   |     0.0 |      12 |  2.99
 Campari        |    25.0 |       5 | 12.95
 Mineral Water  |     0.0 |      10 |  1.49
 Bacardi        |    37.5 |       3 | 16.98
(4 rows)

dblp=# select sum(Price)/count(*) from ingredi
ents;
      ?column?
--------------------
 8.6025000000000000
(1 row)

dblp=# 
```

4.) Aggregate Funtions

aggregare (lat.) zusammenhäufen, aufhäufen

```
COUNT
SUM
AVG
MAX
MIN

VARIANCE
STDDEV
BIT_OR
BIT_AND
```

4.) Aggregate Funtions

aggregare (lat.) zusammenhäufen, aufhäufen

```
COUNT
SUM
AVG
MAX
MIN

VARIANCE
STDDEV
BIT_OR
BIT_AND
```

```
SELECT Name
  FROM Ingredients
 WHERE Alcohol=37.5;
```

4.) Aggregate Funtions

aggregare (lat.) zusammenhäufen, aufhäufen

```
COUNT
SUM
AVG
MAX
MIN

VARIANCE
STDDEV
BIT_OR
BIT_AND
```

```
SELECT Name
  FROM Ingredients
 WHERE Alcohol=37.5;
```

```
SELECT MAX(Alcohol)
  FROM Ingredients;
```

returns 37.5 for our Ingredients table.

4.) Aggregate Funtions

aggregare (lat.) zusammenhäufen, aufhäufen

```
COUNT
SUM
AVG
MAX
MIN

VARIANCE
STDDEV
BIT_OR
BIT_AND
```

```
SELECT Name
  FROM Ingredients
 WHERE Alcohol=37.5;
```

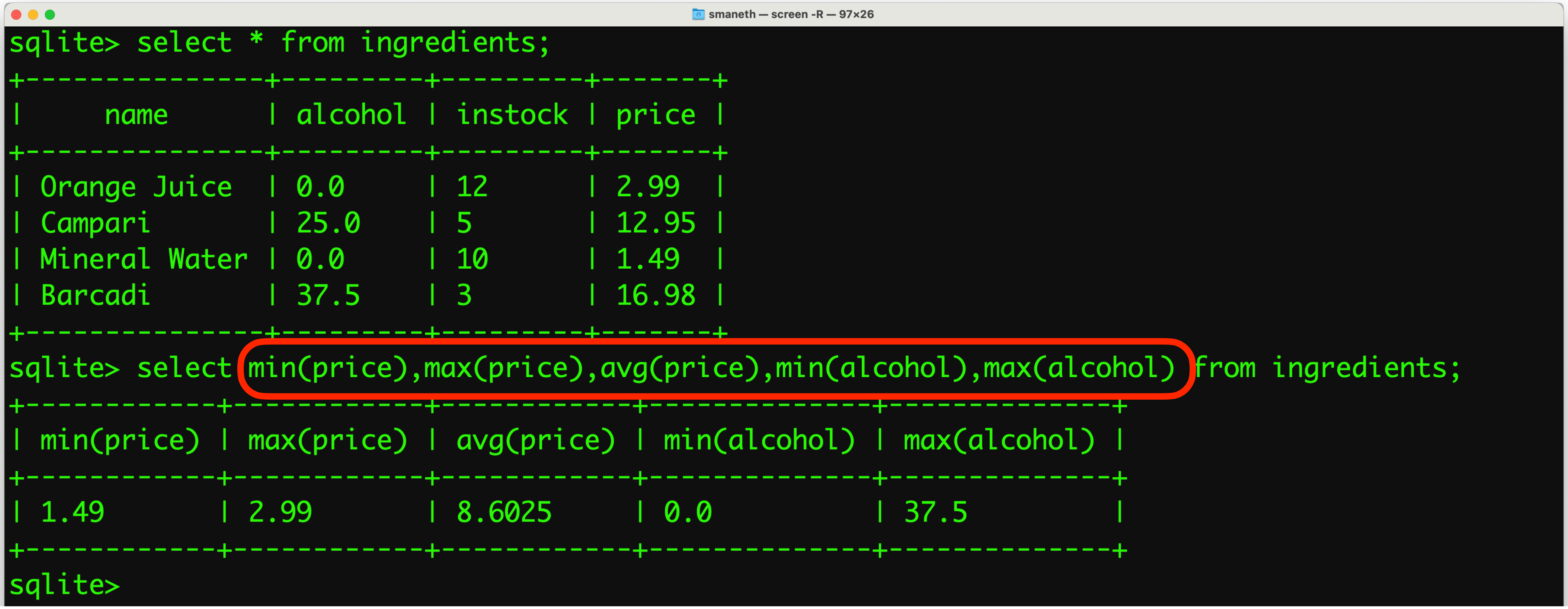
```
SELECT MAX(Alcohol)
  FROM Ingredients;
```

returns 37.5 for our Ingredients table.

Return the corresponding names:

```
SELECT Name from Ingredients
 WHERE Alcohol=(SELECT MAX(Alcohol)
                  FROM Ingredients);
```

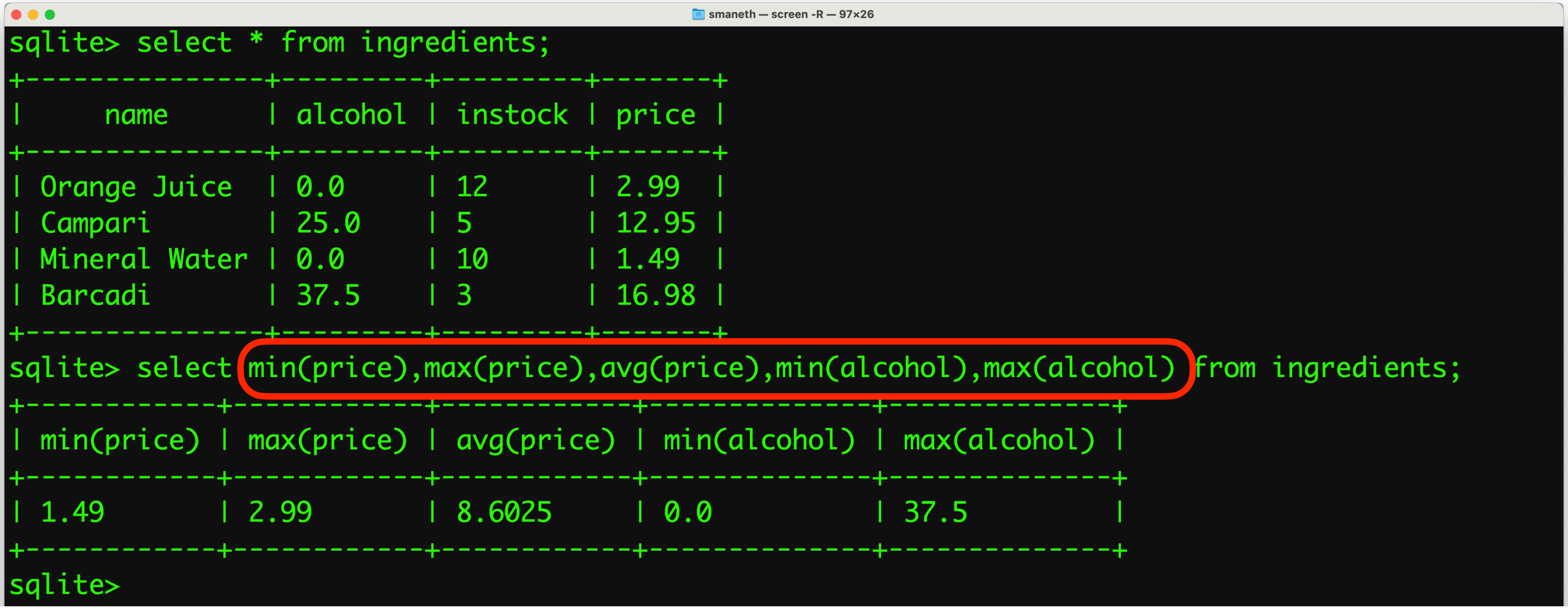
a **nested query**
(returning exactly one value)



smaneth — screen -R — 97×26

```
sqlite> select * from ingredients;
+---------------+---------+---------+-------+
|     name      | alcohol | instock | price |
+---------------+---------+---------+-------+
| Orange Juice  | 0.0     | 12      | 2.99  |
| Campari       | 25.0    | 5       | 12.95 |
| Mineral Water | 0.0     | 10      | 1.49  |
| Barcadi       | 37.5    | 3       | 16.98 |
+---------------+---------+---------+-------+
sqlite> select min(price),max(price),avg(price),min(alcohol),max(alcohol) from ingredients;
+------------+------------+------------+--------------+--------------+
| min(price) | max(price) | avg(price) | min(alcohol) | max(alcohol) |
+------------+------------+------------+--------------+--------------+
| 1.49       | 2.99       | 8.6025     | 0.0          | 37.5         |
+------------+------------+------------+--------------+--------------+
sqlite>
```

We may select **any number** of AGGREGATES or any combination of AGGREGATES.



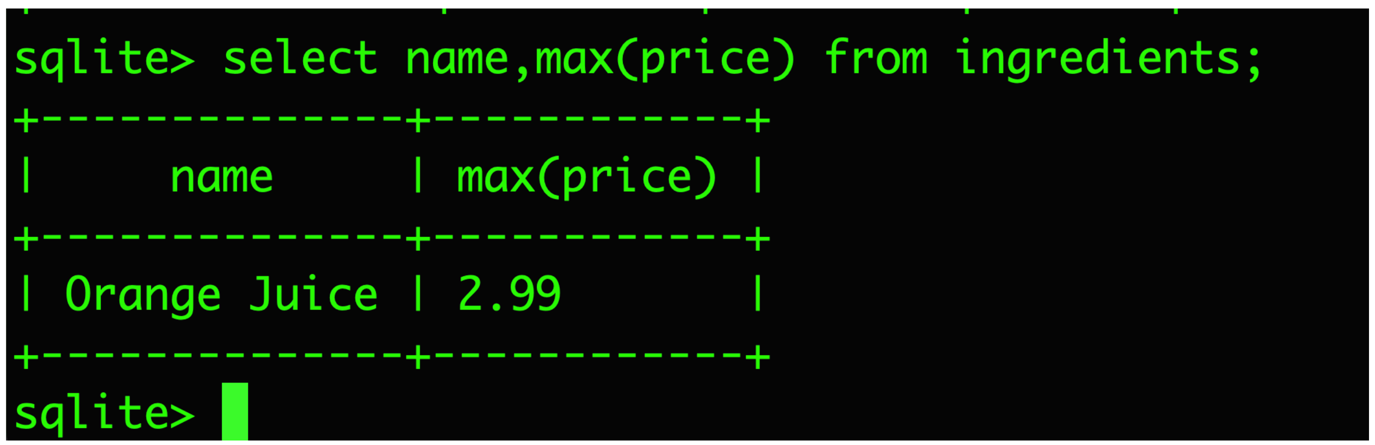
```
sqlite> select * from ingredients;
+---------------+---------+---------+-------+
|     name      | alcohol | instock | price |
+---------------+---------+---------+-------+
| Orange Juice  | 0.0     | 12      | 2.99  |
| Campari       | 25.0    | 5       | 12.95 |
| Mineral Water | 0.0     | 10      | 1.49  |
| Barcadi       | 37.5    | 3       | 16.98 |
+---------------+---------+---------+-------+
sqlite> select min(price),max(price),avg(price),min(alcohol),max(alcohol) from ingredients;
+------------+------------+------------+--------------+--------------+
| min(price) | max(price) | avg(price) | min(alcohol) | max(alcohol) |
+------------+------------+------------+--------------+--------------+
| 1.49       | 2.99       | 8.6025     | 0.0          | 37.5         |
+------------+------------+------------+--------------+--------------+
sqlite>
```

We may select **any number** of AGGREGATES or any combination of AGGREGATES.

BUT, we **may not mix** attributes and AGGREGATES!

```
SELECT name,min(price) from ingredients;
```

What should be the semantics of this query??



```
sqlite> select name,max(price) from ingredients;
+--------------+------------+
|     name     | max(price) |
+--------------+------------+
| Orange Juice | 2.99       |
+--------------+------------+
sqlite>
```

Careful:

This is an ill-formed (=syntactically wrong) query.
But **sqlite3** does not care and produces SOMETHING.
==> not good!!!

Try this on **MySQL** or **PostgreSQL** or **Oracle**.
All of them will produce an ERROR.

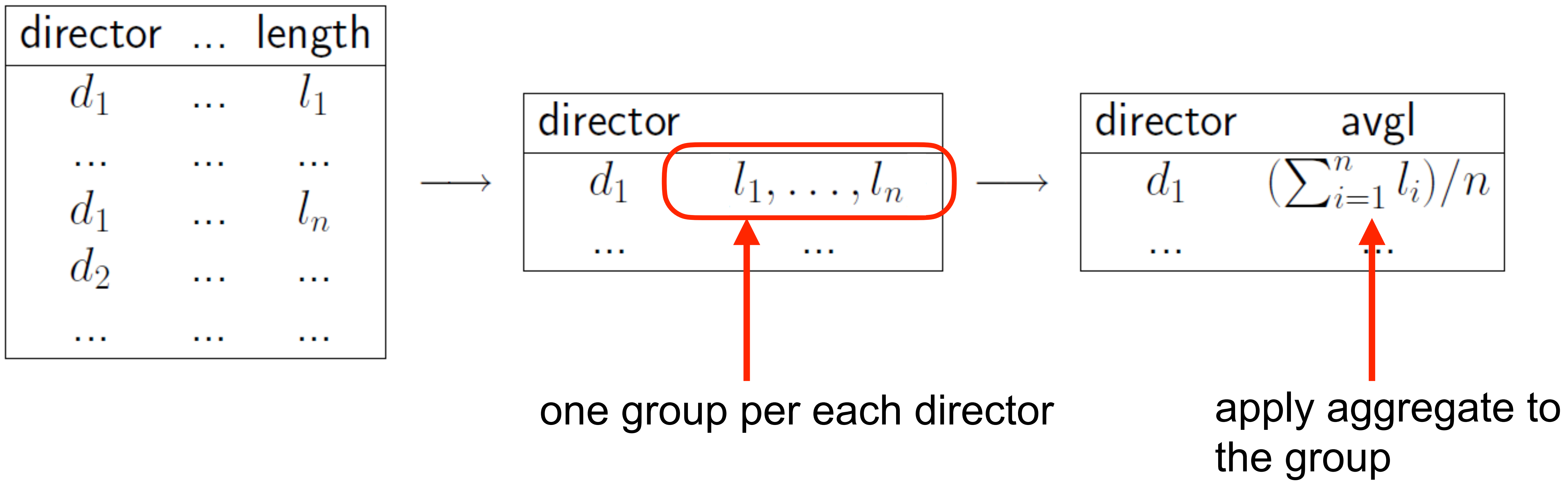
5.) GROUP BY

5.) GROUP BY

→ for each director return the average running time of his/her movies

```
SELECT director, AVG(length) AS avgl
FROM Movies
GROUP BY director;
```

How does grouping work?



director	...	length
d_1	...	l_1
...	...	...
d_1	...	l_n
d_2	...	...
...	...	...

$\longrightarrow$

director	
d_1	$l_1, \ldots, l_n$
...	...

$\longrightarrow$

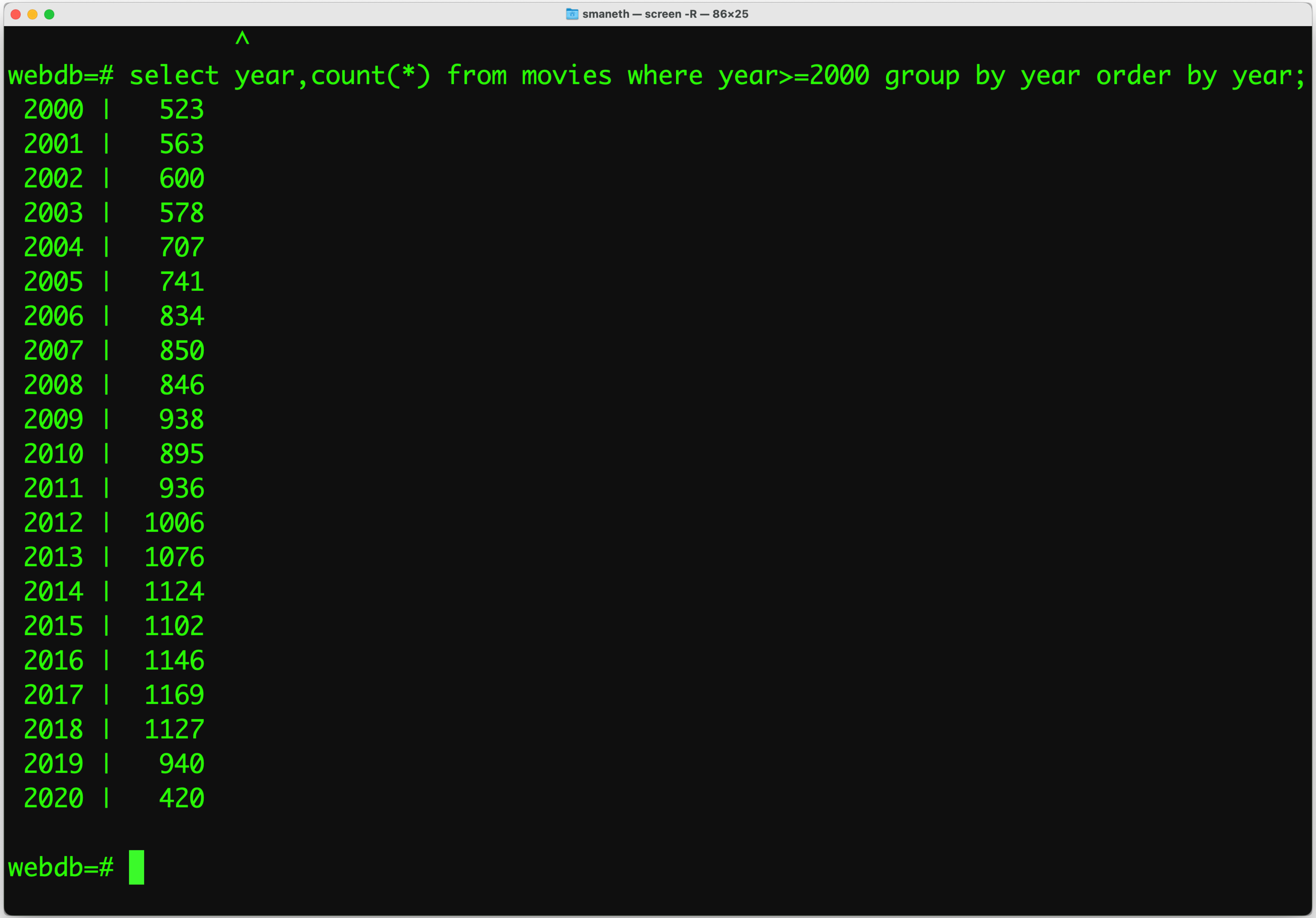
director	avgl
d_1	$(\sum_{i=1}^{n} l_i)/n$
...	...

5.) GROUP BY

```
                          Table "public.movies"
    Column     |          Type          | Collation | Nullable | Default
---------------+------------------------+-----------+----------+---------
 movieid       | character varying(20)  |           | not null |
 originaltitle | character varying(150) |           |          |
 title         | character varying(150) |           |          |
 year          | integer                |           |          |
 country       | text                   |           |          |
```

```
SELECT year,count(*)
  FROM movies GROUP BY year;
```

What does this query compute?



```
               ^
webdb=# select year,count(*) from movies where year>=2000 group by year order by year;
 2000 |   523
 2001 |   563
 2002 |   600
 2003 |   578
 2004 |   707
 2005 |   741
 2006 |   834
 2007 |   850
 2008 |   846
 2009 |   938
 2010 |   895
 2011 |   936
 2012 |  1006
 2013 |  1076
 2014 |  1124
 2015 |  1102
 2016 |  1146
 2017 |  1169
 2018 |  1127
 2019 |   940
 2020 |   420

webdb=# 
```

5.) GROUP BY

```
Cocktails
   Name          | Ingr          | Amount_cl
-------------------------------------------------
White Russian | Vodka          |         4
White Russian | Milk           |         4
White Russian | Kahlua         |         4
Black Russian | Vodka          |         8
Black Russian | Kahlua         |         4
```

```
SELECT Name,count(*)
  FROM Cocktails GROUP BY Name;
```

What is the result of this query?

5.) GROUP BY

```
Cocktails
  Name           | Ingr          | Amount_cl
--------------------------------------------
White Russian    | Vodka         |         4
White Russian    | Milk          |         4
White Russian    | Kahlua        |         4
Black Russian    | Vodka         |         8
Black Russian    | Kahlua        |         4
```

```
SELECT Name,count(*) AS num
  FROM Cocktails GROUP BY Name;
```

```
  Name           | num
-------------------------
White Russian    | 3
Black Russian    | 2
```

Each cocktail, together with its number of ingredients.

5.) GROUP BY

```
Cocktails
  Name          | Ingr          | Amount_cl
________________________________________
White Russian | Vodka         |        4
White Russian | Milk          |        4
White Russian | Kahlua        |        4
Black Russian | Vodka         |        8
Black Russian | Kahlua        |        4
```

```
Ingredients
  Ingr   | Alcohol
__________________
  Vodka |     40.0
   Milk |      0.0
 Kaluha |     20.0
```

How can we compute for each cocktail its alcohol content?

5.) GROUP BY

```
Cocktails
  Name           | Ingr          | Amount_cl
------------------------------------------------
White Russian | Vodka         |          4
White Russian | Milk          |          4
White Russian | Kahlua        |          4
Black Russian | Vodka         |          8
Black Russian | Kahlua        |          4
```

```
Ingredients
  Ingr   | Alcohol
--------------------
  Vodka |     40.0
   Milk |      0.0
 Kaluha |     20.0
```

How can we compute for each cocktail
its alcohol content?

— idea: **multiply** each Amount-value by the
corresponding Alcohol-value,
sum those values
and then **divide** by the total Amount.

NATURAL JOIN

```
Cocktails
  Name         | Ingr          | Amount_cl
---------------------------------------------
White Russian | Vodka         |        4
White Russian | Milk          |        4
White Russian | Kahlua        |        4
Black Russian | Vodka         |        8
Black Russian | Kahlua        |        4
```

```
Ingredients
  Ingr    | Alcohol
-----------------------
  Vodka  |     40.0
   Milk  |      0.0
 Kaluha  |     20.0
```

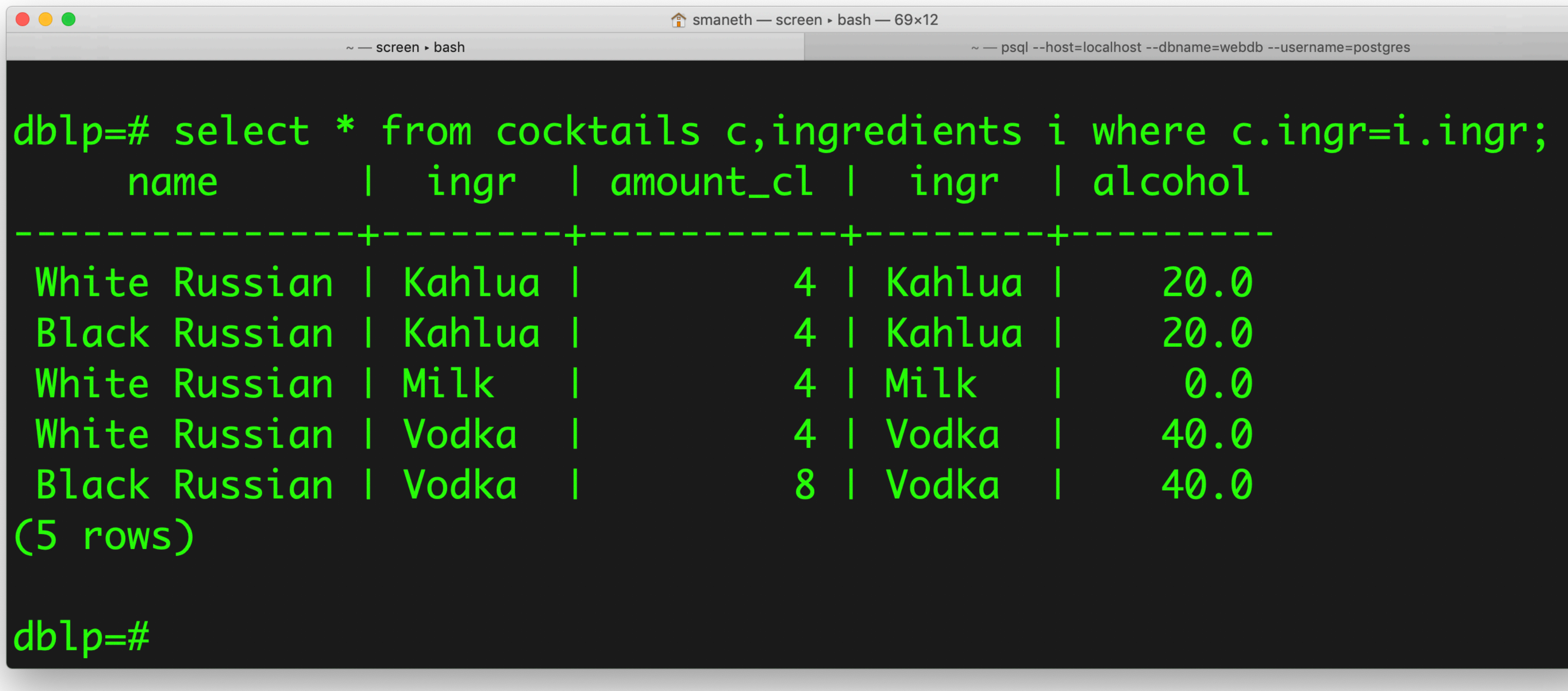
How do we attach to each Ingr-value the corresponding alcohol value?

NATURAL JOIN

How do we attach to each Ingr
the corresponding alcohol value?

```
Cocktails
  Name          | Ingr         | Amount_cl
─────────────────────────────────────────────
White Russian | Vodka        |         4
White Russian | Milk         |         4
White Russian | Kahlua       |         4
Black Russian | Vodka        |         8
Black Russian | Kahlua       |         4
```

```
Ingredients
  Ingr   | Alcohol
──────────────────
  Vodka |    40.0
   Milk |     0.0
 Kaluha |    20.0
```


smaneth — screen ▸ bash — 69×12
~ — screen ▸ bash
~ — psql --host=localhost --dbname=webdb --username=postgres

```
dblp=# select * from cocktails c,ingredients i where c.ingr=i.ingr;
     name      |  ingr  | amount_cl |  ingr  | alcohol
---------------+--------+-----------+--------+---------
 White Russian | Kahlua |         4 | Kahlua |    20.0
 Black Russian | Kahlua |         4 | Kahlua |    20.0
 White Russian | Milk   |         4 | Milk   |     0.0
 White Russian | Vodka  |         4 | Vodka  |    40.0
 Black Russian | Vodka  |         8 | Vodka  |    40.0
(5 rows)

dblp=#
```

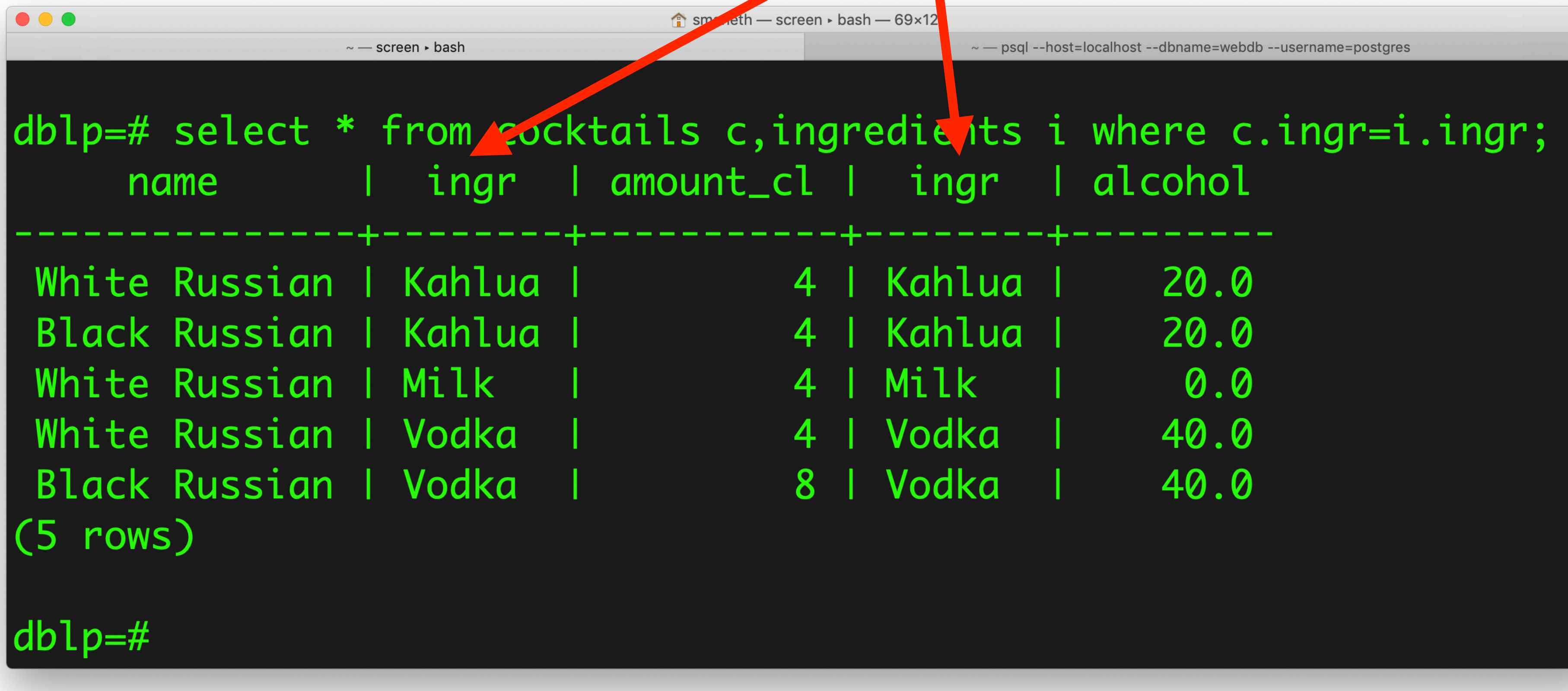
NATURAL JOIN

How do we attach to each Ingredient the corresponding alcohol value?

```
Cocktails
  Name              | Ingr            | Amount_cl
--------------------------------------------------
White Russian | Vodka           |            4
White Russian | Milk            |            4
White Russian | Kahlua          |            4
Black Russian | Vodka           |            8
Black Russian | Kahlua          |            4
```

```
Ingredients
  Ingr   | Alcohol
-------------------
  Vodka  |    40.0
   Milk  |     0.0
 Kaluha  |    20.0
```

We don’t need this column twice.
(**not** a correct relation!!!)


~ — screen ▸ bash
~ — psql --host=localhost --dbname=webdb --username=postgres

```
dblp=# select * from cocktails c,ingredients i where c.ingr=i.ingr;
     name      |  ingr   | amount_cl |  ingr   | alcohol
---------------+---------+-----------+---------+---------
 White Russian | Kahlua  |         4 | Kahlua  |    20.0
 Black Russian | Kahlua  |         4 | Kahlua  |    20.0
 White Russian | Milk    |         4 | Milk    |     0.0
 White Russian | Vodka   |         4 | Vodka   |    40.0
 Black Russian | Vodka   |         8 | Vodka   |    40.0
(5 rows)

dblp=#
```

NATURAL JOIN

How do we attach to each Ingredient the corresponding alcohol value?

```
Cocktails
  Name          | Ingr          | Amount_cl
---------------------------------------------
White Russian | Vodka         |         4
White Russian | Milk          |         4
White Russian | Kahlua        |         4
Black Russian | Vodka         |         8
Black Russian | Kahlua        |         4
```

```
Ingredients
  Ingr   | Alcohol
------------------
  Vodka |     40.0
   Milk |      0.0
 Kahlua |     20.0
```

The NATURAL JOIN checks equality on the common attributes!
The common attributes are listed only once, as first attributes of the result.

NATURAL JOIN

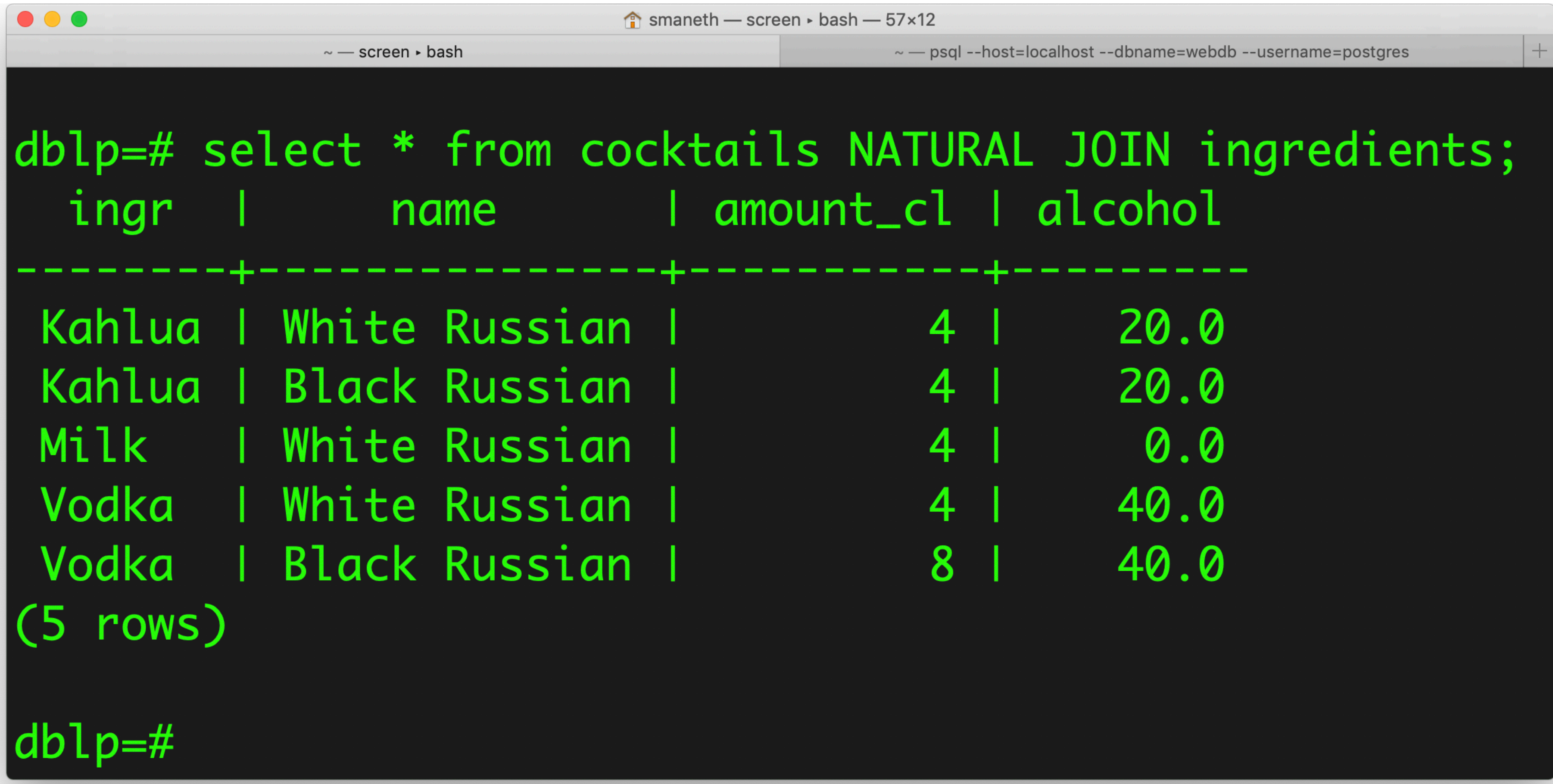
How do we attach to each Ingredient the corresponding alcohol value?

Cocktails

Name	Ingr	Amount_cl
White Russian	Vodka	4
White Russian	Milk	4
White Russian	Kahlua	4
Black Russian	Vodka	8
Black Russian	Kahlua	4

Ingredients

Ingr	Alcohol
Vodka	40.0
Milk	0.0
Kahlua	20.0


smaneth — screen ▸ bash — 57×12
~ — screen ▸ bash
~ — psql --host=localhost --dbname=webdb --username=postgres

```
dblp=# select * from cocktails NATURAL JOIN ingredients;
  ingr  |     name      | amount_cl | alcohol
--------+---------------+-----------+---------
 Kahlua | White Russian |         4 |    20.0
 Kahlua | Black Russian |         4 |    20.0
 Milk   | White Russian |         4 |     0.0
 Vodka  | White Russian |         4 |    40.0
 Vodka  | Black Russian |         8 |    40.0
(5 rows)

dblp=#
```

NATURAL JOIN

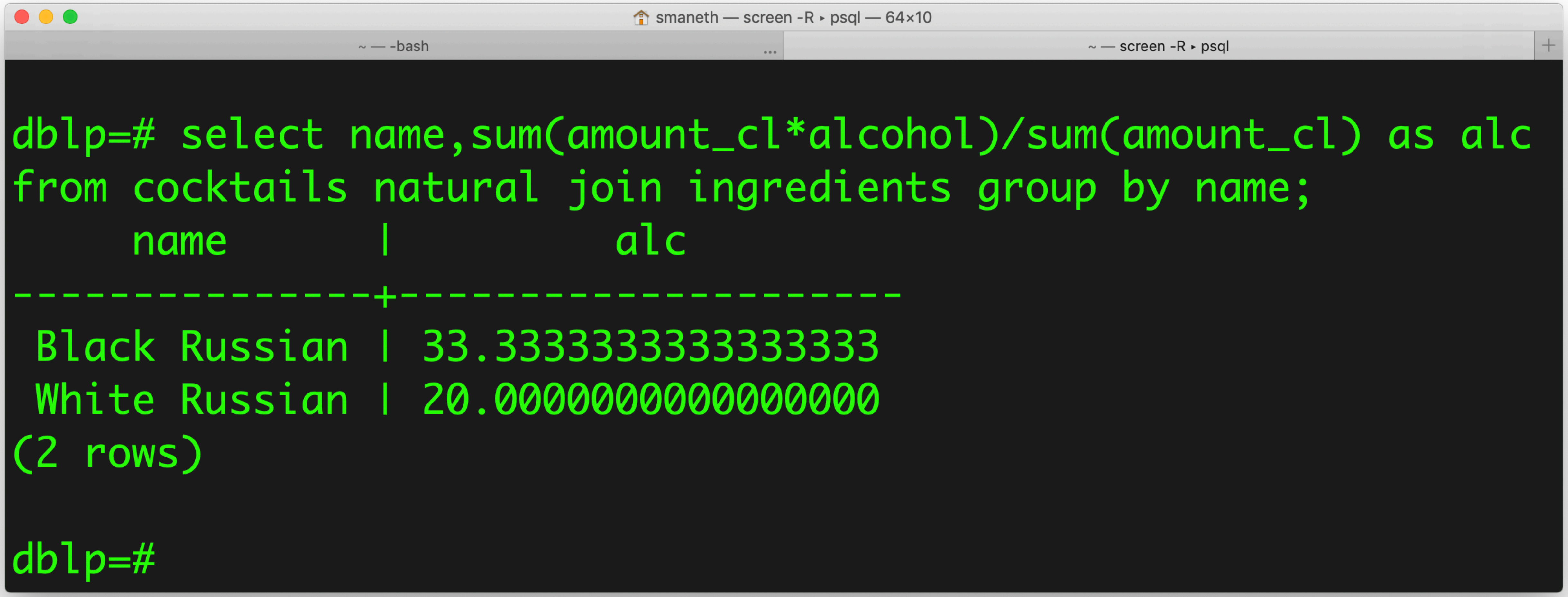
How can we compute for each cocktail its alcohol content?

Cocktails

Name	Ingr	Amount_cl
White Russian	Vodka	4
White Russian	Milk	4
White Russian	Kahlua	4
Black Russian	Vodka	8
Black Russian	Kahlua	4

Ingredients

Ingr	Alcohol
Vodka	40.0
Milk	0.0
Kaluha	20.0



```
dblp=# select name,sum(amount_cl*alcohol)/sum(amount_cl) as alc
from cocktails natural join ingredients group by name;
     name      |          alc
---------------+---------------------
 Black Russian | 33.3333333333333333
 White Russian | 20.0000000000000000
(2 rows)

dblp=#
```

5.) GROUP BY

How can we compute for each cocktail
its alcohol content?

```
Cocktails
  Name           | Ingr          | Amount_cl
-------------------------------------------------
White Russian | Vodka          |           4
White Russian | Milk           |           4
White Russian | Kahlua         |           4
Black Russian | Vodka          |           8
Black Russian | Kahlua         |           4
```

```
Ingredients
  Ingr    | Alcohol
------------------
  Vodka  |    40.0
   Milk  |     0.0
 Kaluha  |    20.0
```

Now we only want to include those cocktails
where the alc-amount ist >25.

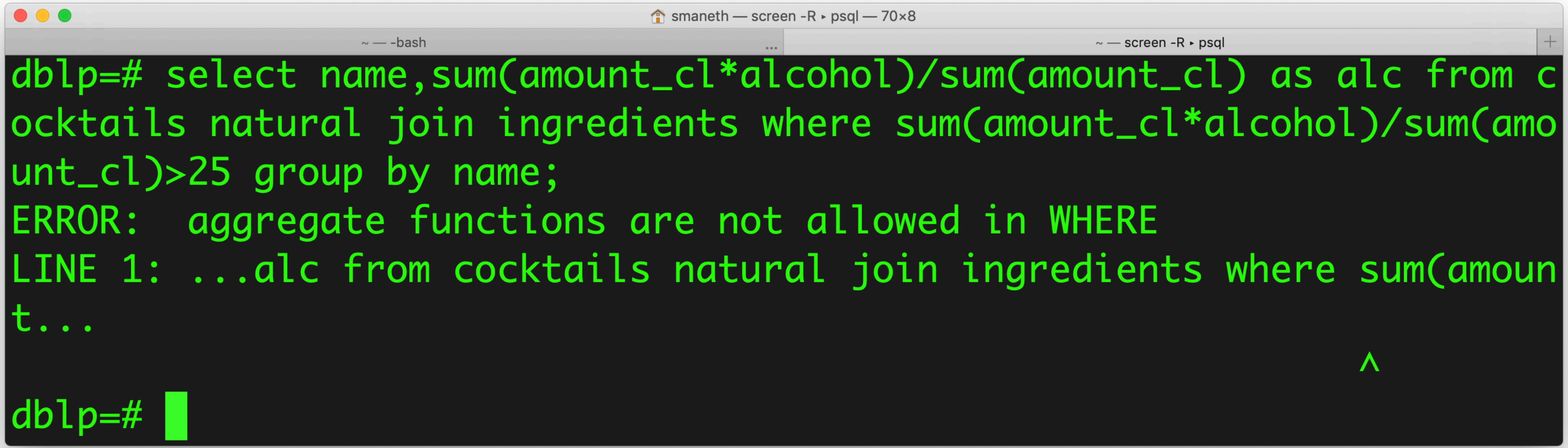
5.) GROUP BY

How can we compute for each cocktail its alcohol content?

```
Cocktails
   Name          | Ingr          | Amount_cl
-----------------------------------------------
White Russian | Vodka         |          4
White Russian | Milk          |          4
White Russian | Kahlua        |          4
Black Russian | Vodka         |          8
Black Russian | Kahlua        |          4
```

```
Ingredients
   Ingr   | Alcohol
------------------
   Vodka |    40.0
    Milk |     0.0
  Kaluha |    20.0
```

Now we only want to include those cocktails where the alc-amount ist >25.



```
dblp=# select name,sum(amount_cl*alcohol)/sum(amount_cl) as alc from c
ocktails natural join ingredients where sum(amount_cl*alcohol)/sum(amo
unt_cl)>25 group by name;
ERROR:  aggregate functions are not allowed in WHERE
LINE 1: ...alc from cocktails natural join ingredients where sum(amoun
t...
                                                              ^
dblp=# 
```

5.) GROUP BY

NOT ALLOWED to have aggregates in the WHERE-clause!!!

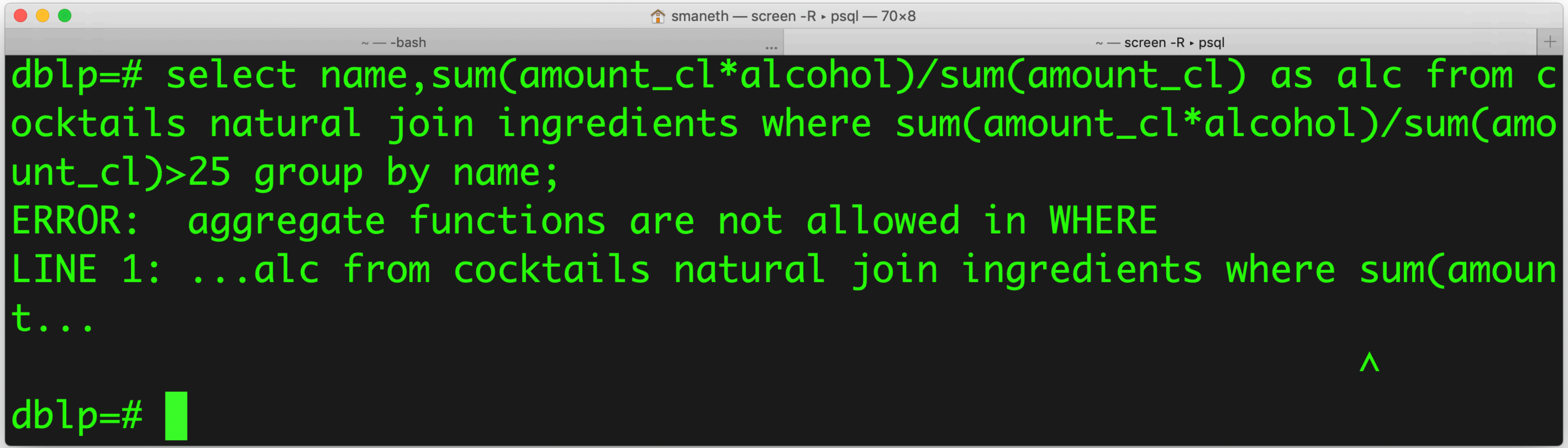
Cocktails

Name	Ingr	Amount_cl
White Russian	Vodka	4
White Russian	Milk	4
White Russian	Kahlua	4
Black Russian	Vodka	8
Black Russian	Kahlua	4

Ingredients

Ingr	Alcohol
Vodka	40.0
Milk	0.0
Kaluha	20.0

Now we only want to include those cocktails where the alc-amount ist >25.



```
dblp=# select name,sum(amount_cl*alcohol)/sum(amount_cl) as alc from c
ocktails natural join ingredients where sum(amount_cl*alcohol)/sum(amo
unt_cl)>25 group by name;
ERROR:  aggregate functions are not allowed in WHERE
LINE 1: ...alc from cocktails natural join ingredients where sum(amoun
t...
                                                             ^

dblp=#
```

5.) GROUP BY

— use the **HAVING** keyword
It checks the groups AFTER the grouping!

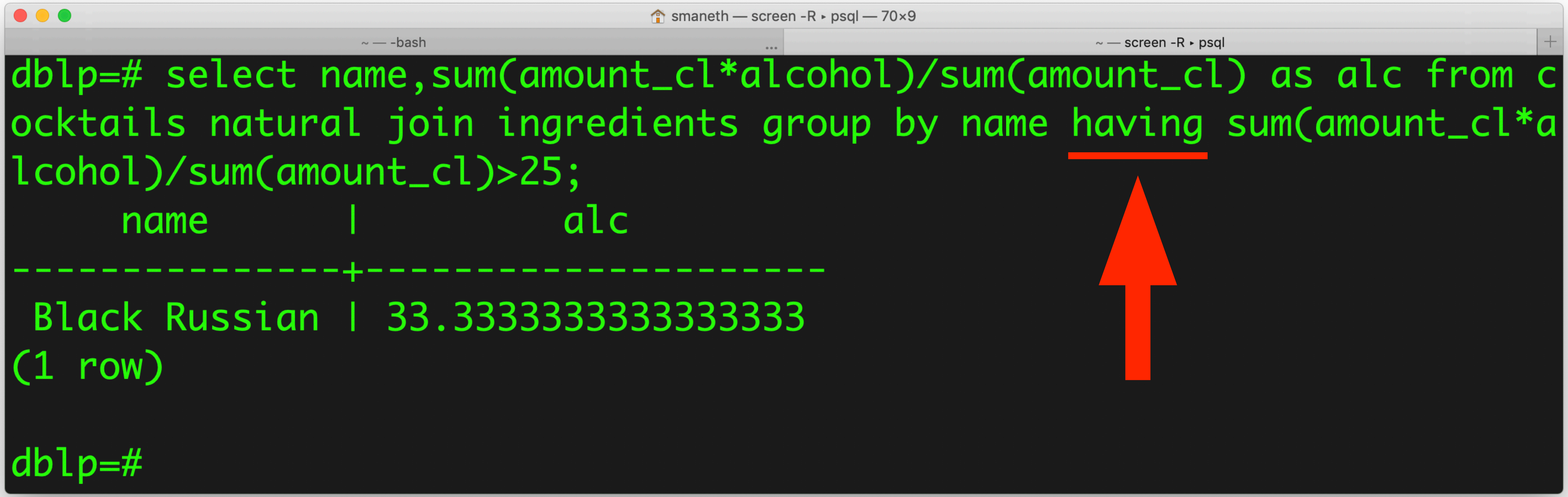
Cocktails

Name	Ingr	Amount_cl
White Russian	Vodka	4
White Russian	Milk	4
White Russian	Kahlua	4
Black Russian	Vodka	8
Black Russian	Kahlua	4

Ingredients

Ingr	Alcohol
Vodka	40.0
Milk	0.0
Kaluha	20.0

Now we only want to include those cocktails
where the alc-amount ist >25.



```
dblp=# select name,sum(amount_cl*alcohol)/sum(amount_cl) as alc from c
ocktails natural join ingredients group by name having sum(amount_cl*a
lcohol)/sum(amount_cl)>25;
     name      |         alc
---------------+---------------------
 Black Russian | 33.3333333333333333
(1 row)

dblp=#
```

5.) GROUP BY

How can we give the names of cocktails that contain **no alcohol?**

```
Cocktails
  Name          | Ingr         | Amount_cl
----------------------------------------
White Russian | Vodka        |        4
White Russian | Milk         |        4
White Russian | Kahlua       |        4
Black Russian | Vodka        |        8
Black Russian | Kahlua       |        4
```

```
Ingredients
  Ingr   | Alcohol
------------------
  Vodka |    40.0
   Milk |     0.0
 Kaluha |    20.0
```

select name from cocktails natural join ingredients group by name
HAVING sum(alcohol)=0;

5.) GROUP BY

```
SELECT name,year,count(*) as cnt
  FROM movies
       NATURAL JOIN directors2movies
       NATURAL JOIN persons
 GROUP BY name,year
 ORDER BY cnt desc;
```

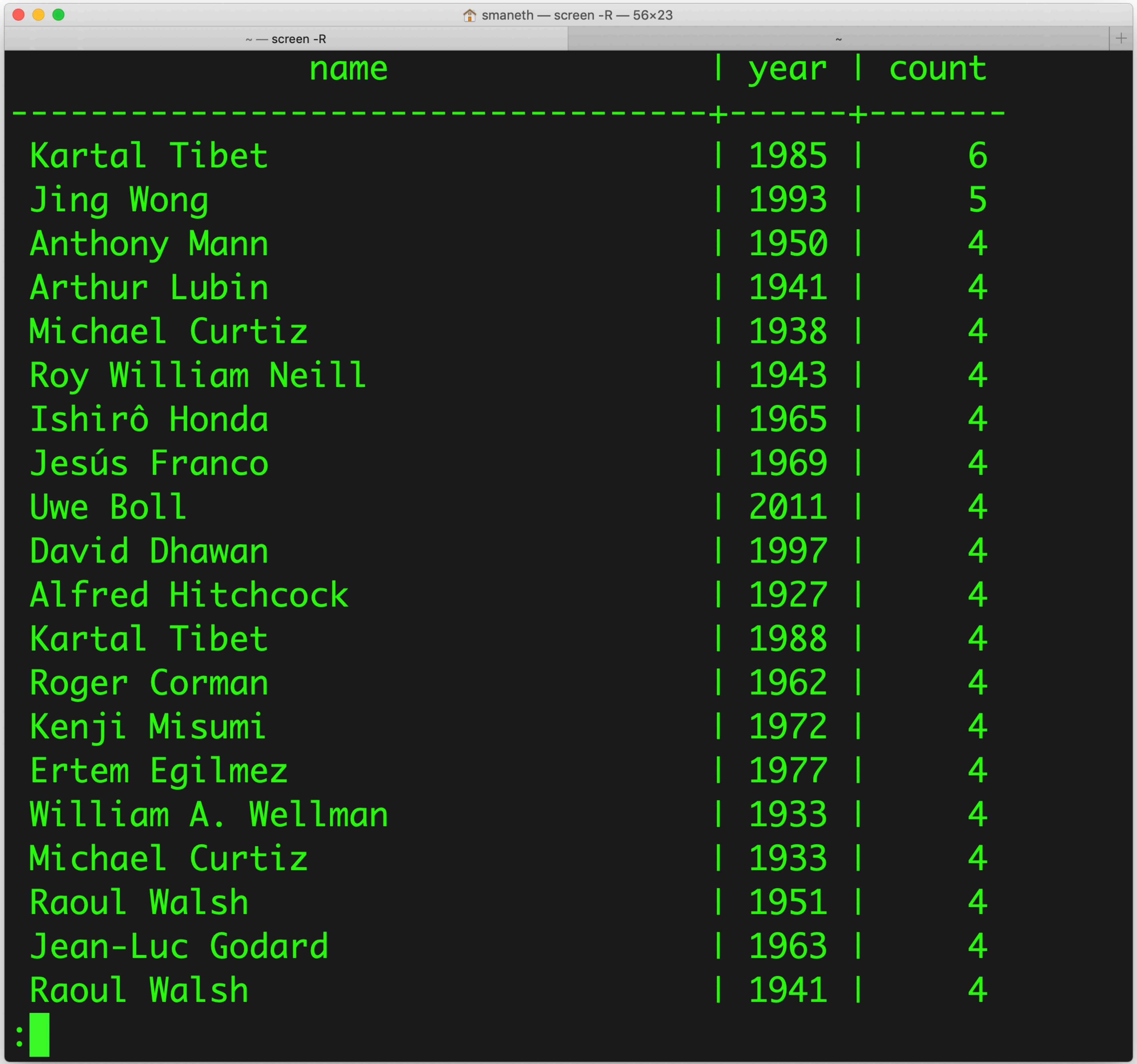
What does this query compute?

5.) GROUP BY

```
SELECT name,year,count(*) as cnt
  FROM movies
       NATURAL JOIN directors2movies
       NATURAL JOIN persons
 GROUP BY name,year
 ORDER BY cnt desc;
```

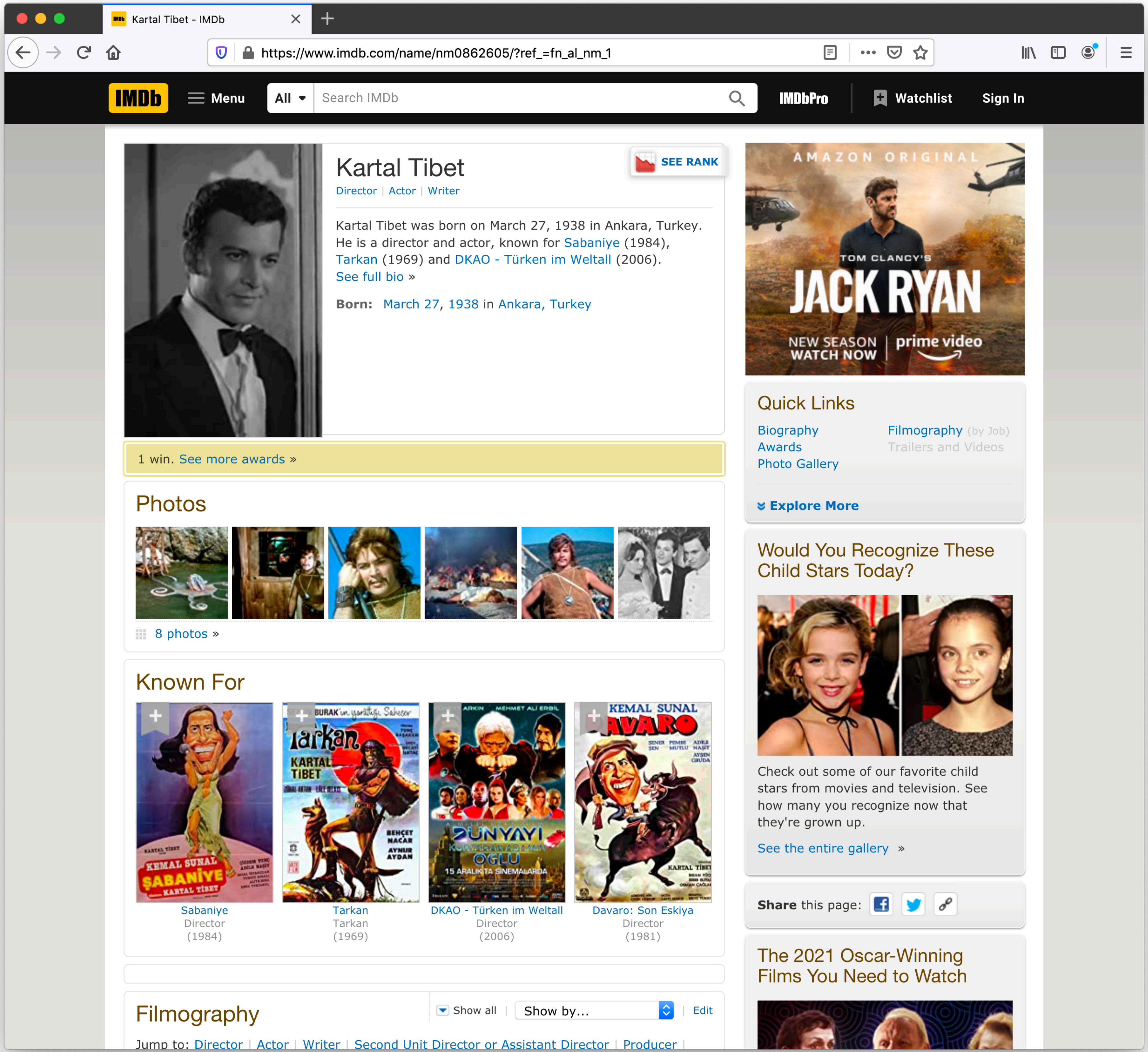
What does this query compute?

Show how many movies each director made in each year
(sort it from largest to smallest number of movies)


smaneth — screen -R — 56×23

~ — screen -R   ~

```
               name                 | year | count
------------------------------------+------+-------
 Kartal Tibet                       | 1985 |     6
 Jing Wong                          | 1993 |     5
 Anthony Mann                       | 1950 |     4
 Arthur Lubin                       | 1941 |     4
 Michael Curtiz                     | 1938 |     4
 Roy William Neill                  | 1943 |     4
 Ishirô Honda                       | 1965 |     4
 Jesús Franco                       | 1969 |     4
 Uwe Boll                           | 2011 |     4
 David Dhawan                       | 1997 |     4
 Alfred Hitchcock                   | 1927 |     4
 Kartal Tibet                       | 1988 |     4
 Roger Corman                       | 1962 |     4
 Kenji Misumi                       | 1972 |     4
 Ertem Egilmez                      | 1977 |     4
 William A. Wellman                 | 1933 |     4
 Michael Curtiz                     | 1933 |     4
 Raoul Walsh                        | 1951 |     4
 Jean-Luc Godard                    | 1963 |     4
 Raoul Walsh                        | 1941 |     4
:
```



Kartal Tibet - IMDb

https://www.imdb.com/name/nm0862605/?ref_=fn_al_nm_1

IMDb Menu All Search IMDb IMDbPro Watchlist Sign In

Kartal Tibet

SEE RANK

Director | Actor | Writer

Kartal Tibet was born on March 27, 1938 in Ankara, Turkey. He is a director and actor, known for Sabaniye (1984), Tarkan (1969) and DKAO - Türken im Weltall (2006).
See full bio »

Born: March 27, 1938 in Ankara, Turkey

1 win. See more awards »

Photos

8 photos »

Known For

Sabaniye	Tarkan	DKAO - Türken im Weltall	Davaro: Son Eskiya
Director	Tarkan	Director	Director
(1984)	(1969)	(2006)	(1981)

Filmography

Show all | Show by... | Edit

Jump to: Director | Actor | Writer | Second Unit Director or Assistant Director | Producer |

AMAZON ORIGINAL TOM CLANCY'S JACK RYAN NEW SEASON WATCH NOW prime video

Quick Links

Biography
Awards
Photo Gallery
Filmography (by Job)
Trailers and Videos

Explore More

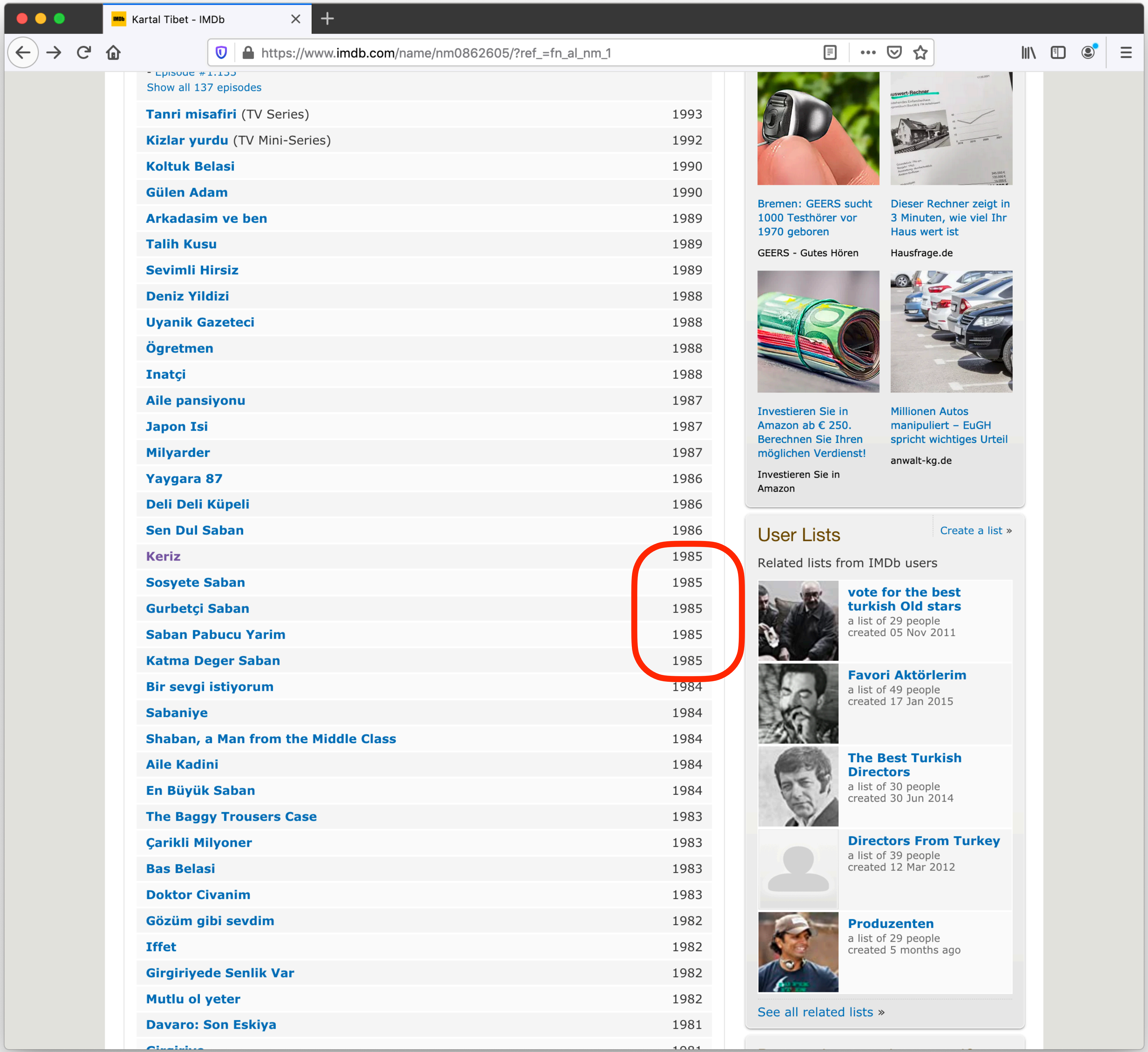
Would You Recognize These Child Stars Today?

Check out some of our favorite child stars from movies and television. See how many you recognize now that they're grown up.

See the entire gallery »

Share this page:

The 2021 Oscar-Winning Films You Need to Watch



Kartal Tibet - IMDb

https://www.imdb.com/name/nm0862605/?ref_=fn_al_nm_1

- Episode #1.135

Show all 137 episodes

Tanri misafiri (TV Series)	1993
Kizlar yurdu (TV Mini-Series)	1992
Koltuk Belasi	1990
Gülen Adam	1990
Arkadasim ve ben	1989
Talih Kusu	1989
Sevimli Hirsiz	1989
Deniz Yildizi	1988
Uyanik Gazeteci	1988
Ögretmen	1988
Inatçi	1988
Aile pansiyonu	1987
Japon Isi	1987
Milyarder	1987
Yaygara 87	1986
Deli Deli Küpeli	1986
Sen Dul Saban	1986
Keriz	1985
Sosyete Saban	1985
Gurbetçi Saban	1985
Saban Pabucu Yarim	1985
Katma Deger Saban	1985
Bir sevgi istiyorum	1984
Sabaniye	1984
Shaban, a Man from the Middle Class	1984
Aile Kadini	1984
En Büyük Saban	1984
The Baggy Trousers Case	1983
Çarikli Milyoner	1983
Bas Belasi	1983
Doktor Civanim	1983
Gözüm gibi sevdim	1982
Iffet	1982
Girgiriyede Senlik Var	1982
Mutlu ol yeter	1982
Davaro: Son Eskiya	1981
Girgiriye	1981

Bremen: GEERS sucht 1000 Testhörer vor 1970 geboren

GEERS - Gutes Hören

Dieser Rechner zeigt in 3 Minuten, wie viel Ihr Haus wert ist

Hausfrage.de

Investieren Sie in Amazon ab € 250. Berechnen Sie Ihren möglichen Verdienst!

Investieren Sie in Amazon

Millionen Autos manipuliert – EuGH spricht wichtiges Urteil

anwalt-kg.de

User Lists

Create a list »

Related lists from IMDb users

vote for the best turkish Old stars
a list of 29 people
created 05 Nov 2011

Favori Aktörlerim
a list of 49 people
created 17 Jan 2015

The Best Turkish Directors
a list of 30 people
created 30 Jun 2014

Directors From Turkey
a list of 39 people
created 12 Mar 2012

Produzenten
a list of 29 people
created 5 months ago

See all related lists »

General Form of an SQL Query:

```
SELECT    list of attributes
FROM      list of tables
WHERE     condition over attributes
GROUP BY  list of attributes
HAVING    condition over aggregates
ORDER BY  list of attributes
LIMIT     number
```

aggregate functions:

```
COUNT    VARIANCE
SUM      STDDEV
AVG      BIT_OR
MAX      BIT_AND
MIN
```

General Form of an SQL Query:

```
SELECT    list of attributes
FROM      list of tables
WHERE     condition over attributes
GROUP BY  list of attributes
HAVING    condition over aggregates
ORDER BY  list of attributes
LIMIT     number
```

aggregate functions:

```
COUNT     VARIANCE
SUM       STDDEV
AVG       BIT_OR
MAX       BIT_AND
MIN
```

More Queries
E.g: show movies and directors where the director acted in that movie.

==> **formulate interesting queries on your own!**
==> **try them on the movies.sqlite3 database!**

Kurs
Datenbankgrund
und Modell

Sebast sität Bremen

bremen.de

ersemester 2023

8.5.2023

Vorlesung 3: Introduction to SQL

End of this Lecture