Principles of Database Systems (CS307)

Lecture 4: More on Retrieving Data; Join

Yuxin Ma

Department of Computer Science and Engineering
Southern University of Science and Technology

- Most contents are from slides made by Stéphane Faroult and the authors of Database System Concepts (7th Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.

More on Retrieving Data

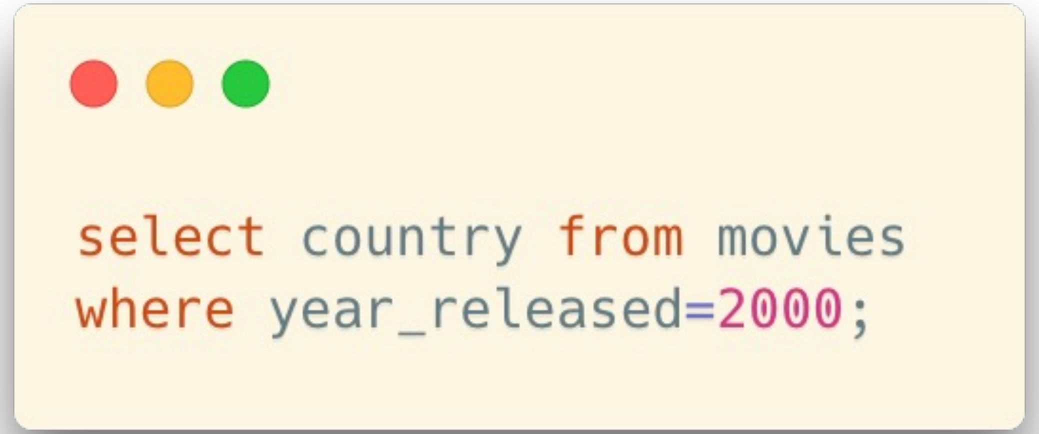
Distinct

Distinct

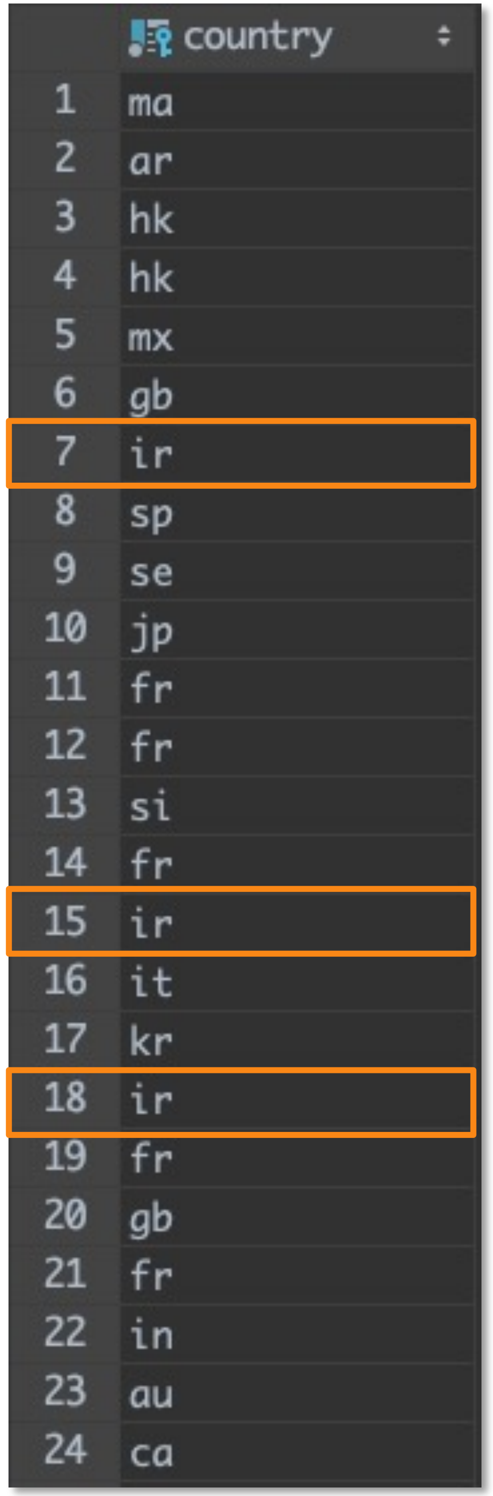
- No duplicated identifier
  - Some rules must be respected if you want to obtain valid results when you apply new operations to result sets
    - They must be mathematical sets, i.e., no duplicates

Distinct

- If we run a query such as the one below
  - Many identical rows
    - In other words, we may be obtaining a table, but it's not a relation because many rows cannot be distinguished



```
select country from movies
where year_released=2000;
```



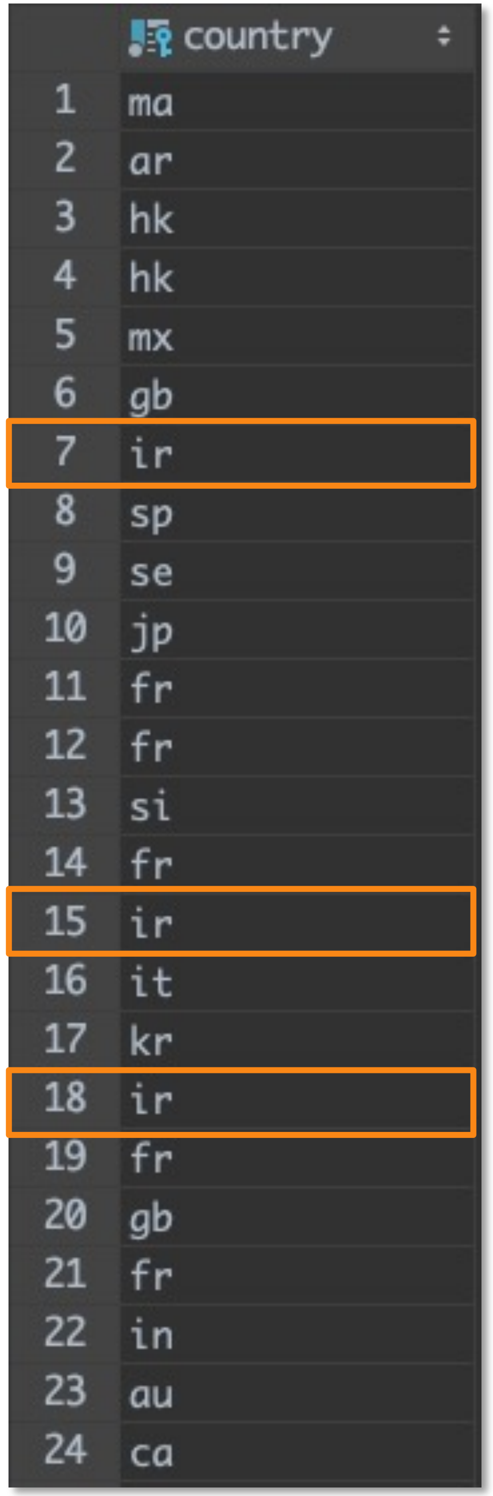
	country
1	ma
2	ar
3	hk
4	hk
5	mx
6	gb
7	ir
8	sp
9	se
10	jp
11	fr
12	fr
13	si
14	fr
15	ir
16	it
17	kr
18	ir
19	fr
20	gb
21	fr
22	in
23	au
24	ca

Duplicated country codes in the query result

- But their original rows are not considered duplicated tuples

Distinct

- The result of the query is in fact completely uninteresting
  - Whenever we are only interested in countries in table movies, it can only be for one of two reasons:
    - See a list of countries that have movies
    - Or, for instance, see which countries appear most often



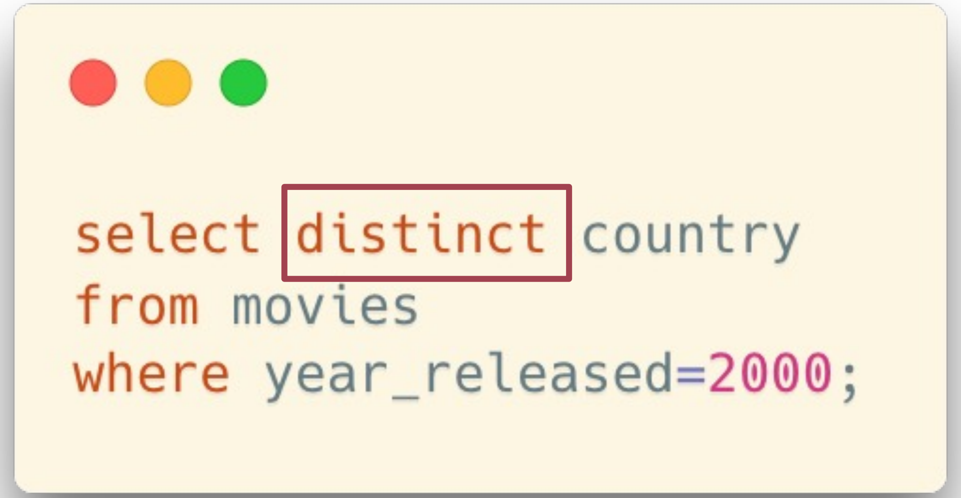
	country
1	ma
2	ar
3	hk
4	hk
5	mx
6	gb
7	ir
8	sp
9	se
10	jp
11	fr
12	fr
13	si
14	fr
15	ir
16	it
17	kr
18	ir
19	fr
20	gb
21	fr
22	in
23	au
24	ca

Duplicated country codes in the query result

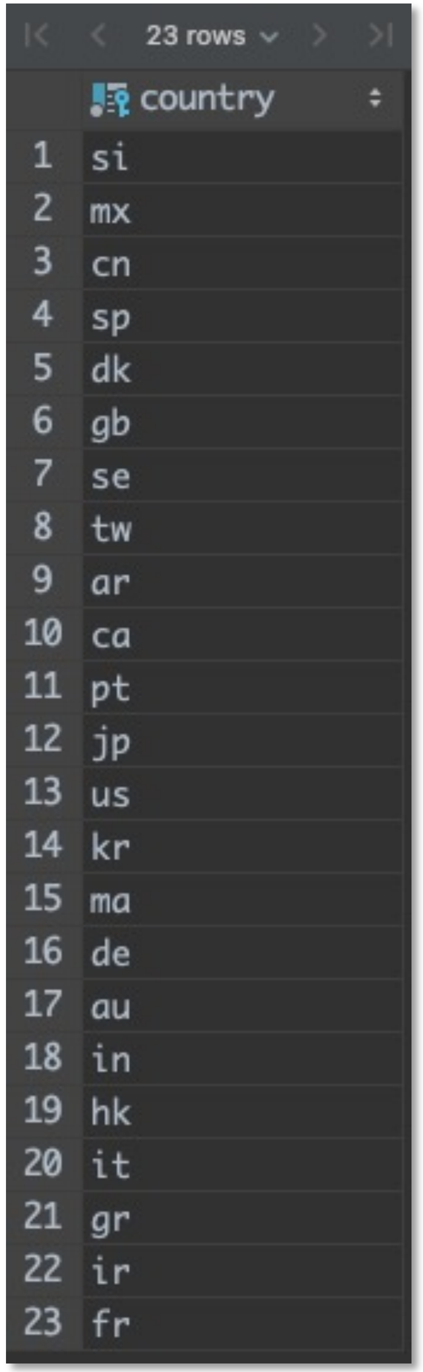
- But their original rows are not considered duplicated tuples

Distinct

- If we only are interested in the different countries, there is the special keyword `distinct`.



```
select distinct country
from movies
where year_released=2000;
```



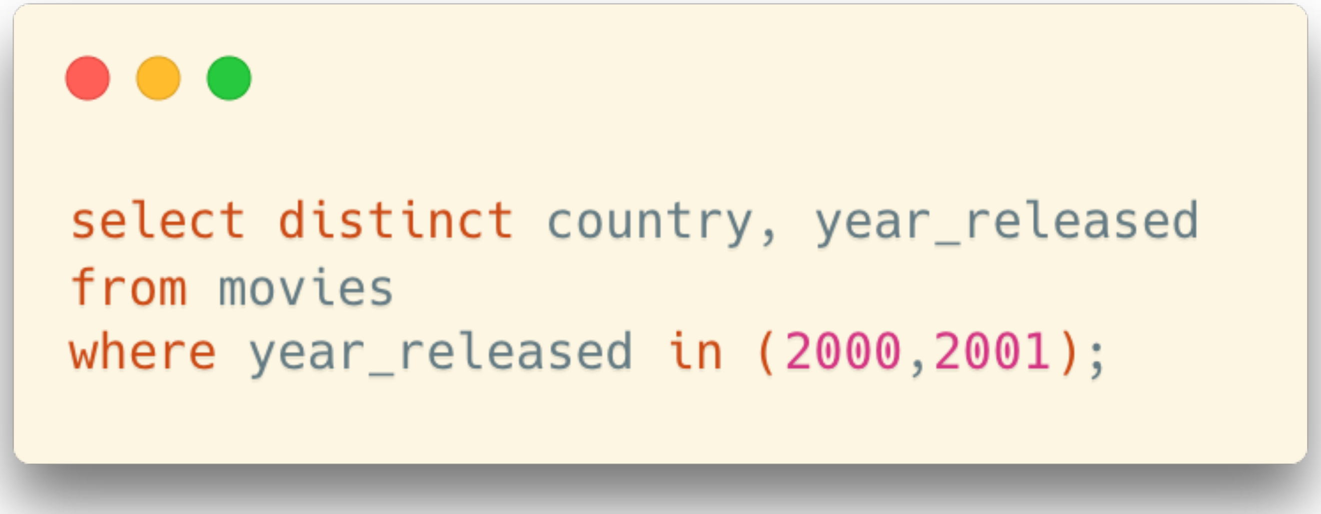
	country
1	si
2	mx
3	cn
4	sp
5	dk
6	gb
7	se
8	tw
9	ar
10	ca
11	pt
12	jp
13	us
14	kr
15	ma
16	de
17	au
18	in
19	hk
20	it
21	gr
22	ir
23	fr

No duplicated results in the country code list now

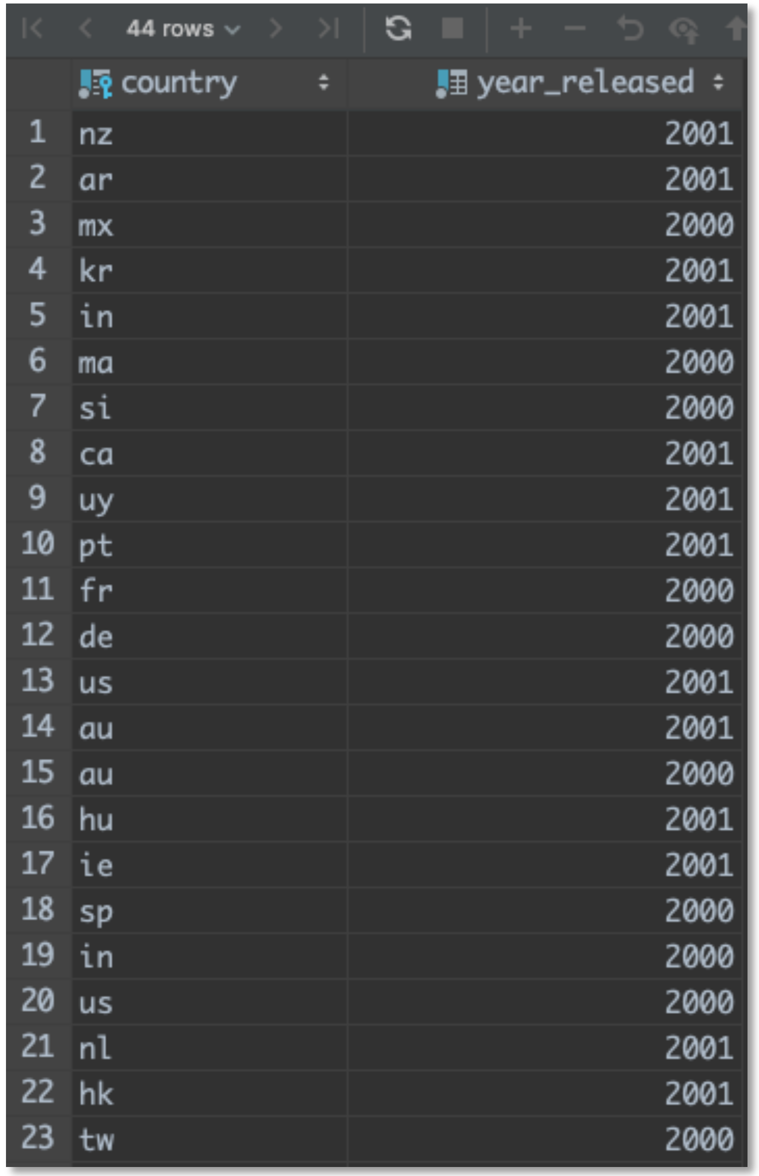
- All of them are different now, and hence it is a relation!

Distinct

- Multiple columns after the keyword `distinct`
  - It will eliminate those rows where all the selected fields are identical
  - The selected combination `(country, year_released)` will be identical



```
select distinct country, year_released
from movies
where year_released in (2000,2001);
```



	country	year_released
1	nz	2001
2	ar	2001
3	mx	2000
4	kr	2001
5	in	2001
6	ma	2000
7	si	2000
8	ca	2001
9	uy	2001
10	pt	2001
11	fr	2000
12	de	2000
13	us	2001
14	au	2001
15	au	2000
16	hu	2001
17	ie	2001
18	sp	2000
19	in	2000
20	us	2000
21	nl	2001
22	hk	2001
23	tw	2000

More on Retrieving Data

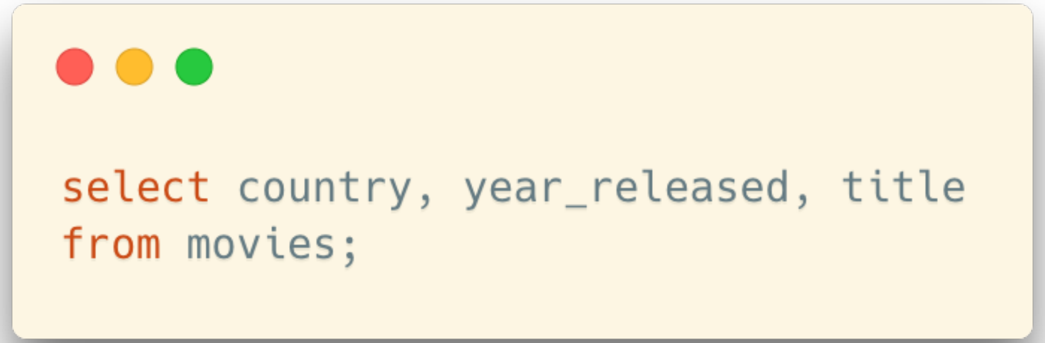
Aggregate Functions

Aggregate Functions

- Statistical functions
  - When we are interested in what we might call countrywide characteristics, such as how many movies released, we use Aggregate Functions.
  - Aggregate function will
    - aggregate all rows that share a feature (such as being movies from the same country)
    - … and return a characteristic of each group of aggregated rows

Aggregate Functions

- To compute an aggregated result, we'll first retrieve data
  - Here, all rows are in the table



```
select country, year_released, title
from movies;
```

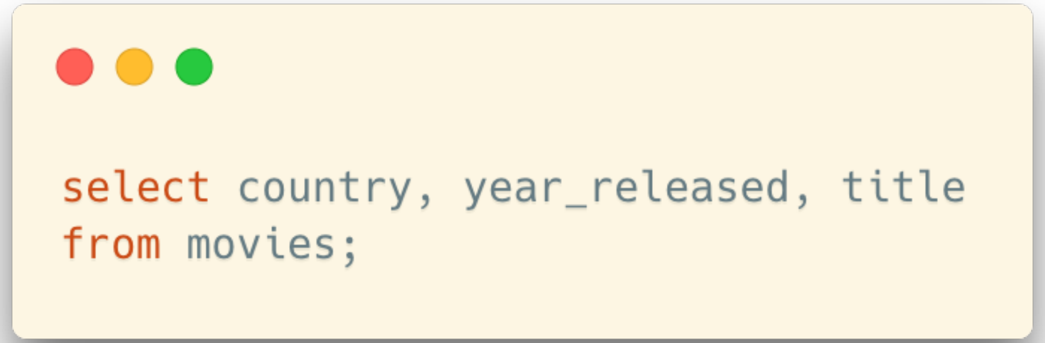
country	year_released	title
de	1985	Das Boot
fr	1997	Le cinquième élément
fr	1946	La belle et la bête
fr	1942	Les Visiteurs du Soir
gb	1962	Lawrence Of Arabia
gb	1949	The Third Man
in	1975	Sholay
in	1955	Pather Panchali
jp	1954	Shichinin no Samurai

Note: Just for demonstration purpose, not the real data in the table `movie`

Aggregate Functions

- To compute an aggregated result, we'll first retrieve data
  - Here, all rows are in the table

- Then, data will be regrouped according to the value in one or several columns



```
select country, year_released, title
from movies;
```

Grouped according to country

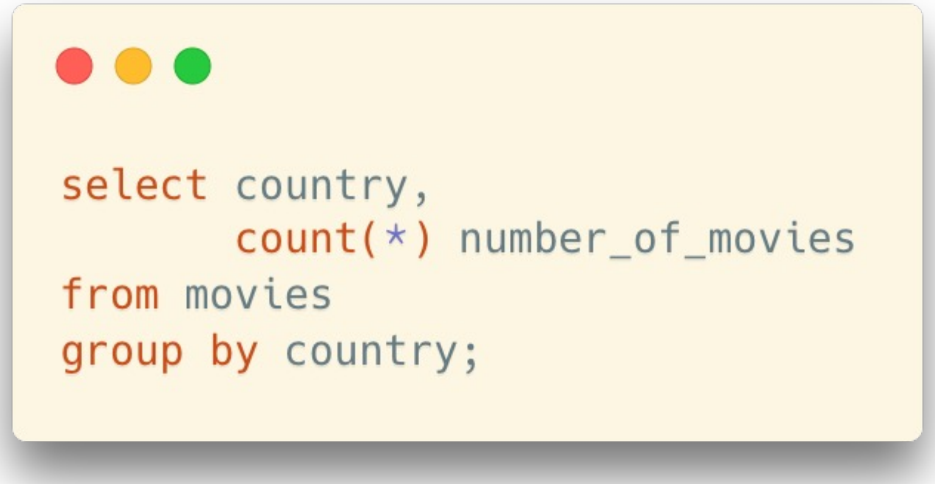
- Rows with the same value will be grouped together

country	year_released	title
de	1985	Das Boot
fr	1997	Le cinquième élément
fr	1946	La belle et la bête
fr	1942	Les Visiteurs du Soir
gb	1962	Lawrence Of Arabia
gb	1949	The Third Man
in	1975	Sholay
in	1955	Pather Panchali
jp	1954	Shichinin no Samurai

Note: Just for demonstration purpose, not the real data in the table `movie`

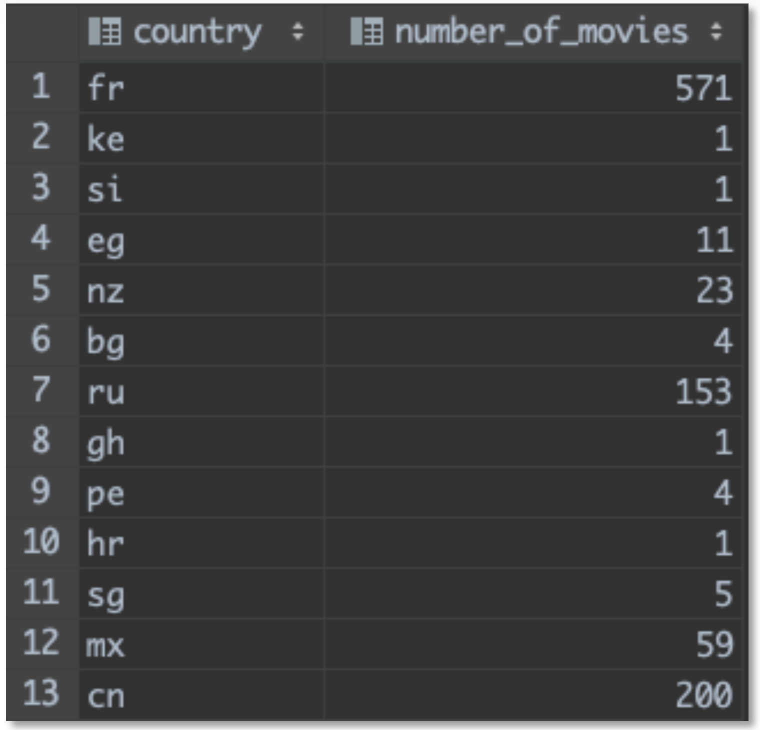
Aggregate Functions

- We say that we want to "group by country"
  - … and, for each country, the aggregate function `count(*)` says how many movies we have
    - "how many movies" = "how many rows"



```
select country,
       count(*) number_of_movies
from movies
group by country;
```

- The query result
  - One row for each group
  - The statistical value is attached in another column

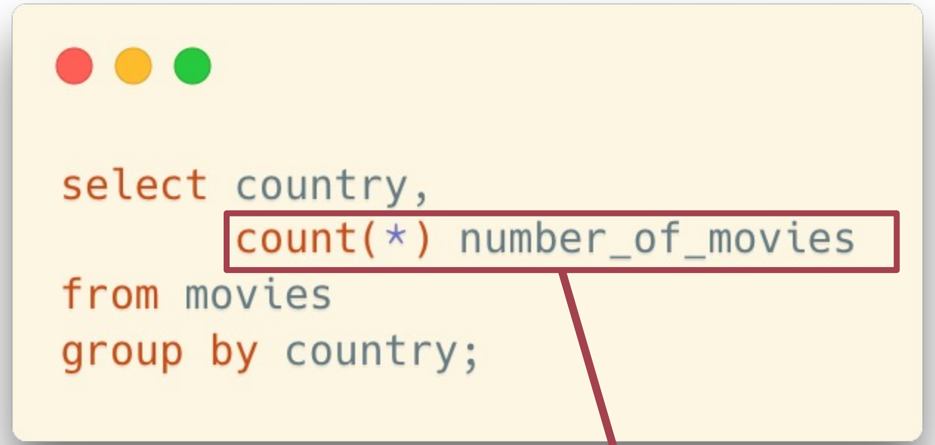


	country	number_of_movies
1	fr	571
2	ke	1
3	si	1
4	eg	11
5	nz	23
6	bg	4
7	ru	153
8	gh	1
9	pe	4
10	hr	1
11	sg	5
12	mx	59
13	cn	200

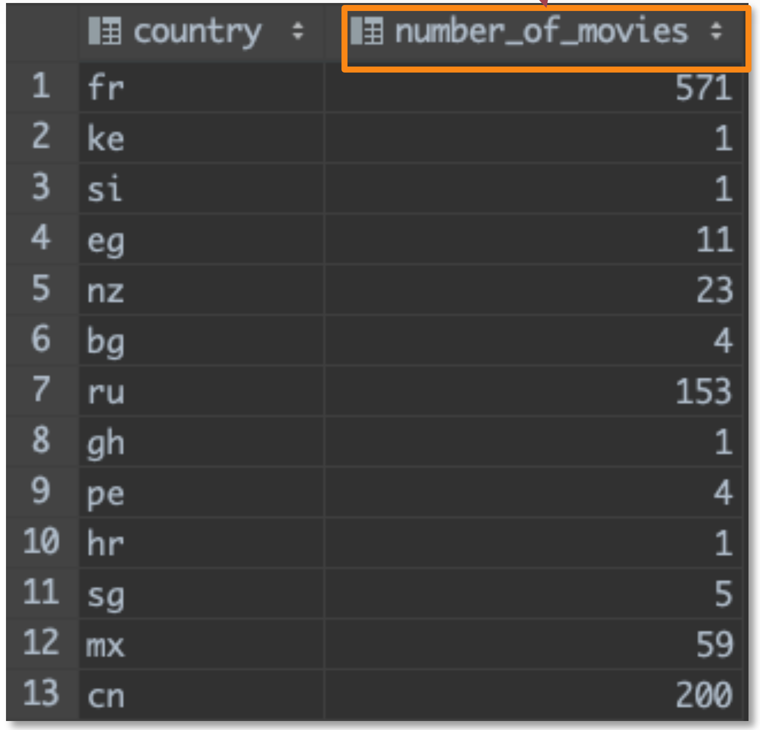
Aggregate Functions

By the way, we can **rename** the column of the aggregate function, like below

- We say that we want to "group by country"
  - … and, for each country, the aggregate function `count(*)` says how many movies we have
    - "how many movies" = "how many rows"
- The query result
  - One row for each group
  - The statistical value is attached in another column



```
select country,
       count(*) number_of_movies
from movies
group by country;
```



	country	number_of_movies
1	fr	571
2	ke	1
3	si	1
4	eg	11
5	nz	23
6	bg	4
7	ru	153
8	gh	1
9	pe	4
10	hr	1
11	sg	5
12	mx	59
13	cn	200

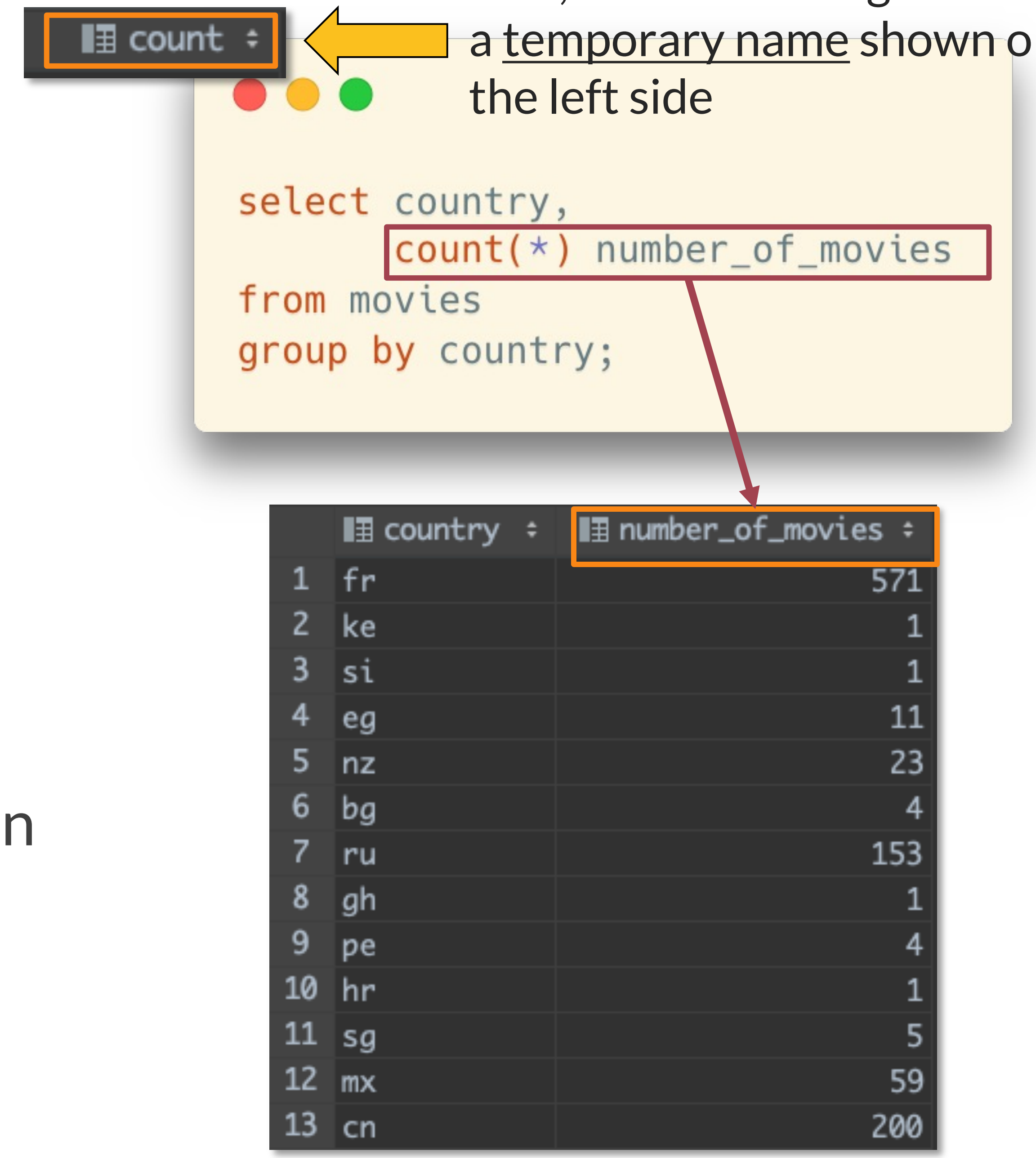
Aggregate Functions

- We say that we want to "group by country"
  - … and, for each country, the aggregate function `count(*)` says how many movies we have
    - "how many movies" = "how many rows"

- The query result
  - One row for each group
  - The statistical value is attached in another column

By the way, we can **rename** the column of the aggregate function, like below

- … or, the client will generate a temporary name shown on the left side



```
select country,
       count(*) number_of_movies
from movies
group by country;
```

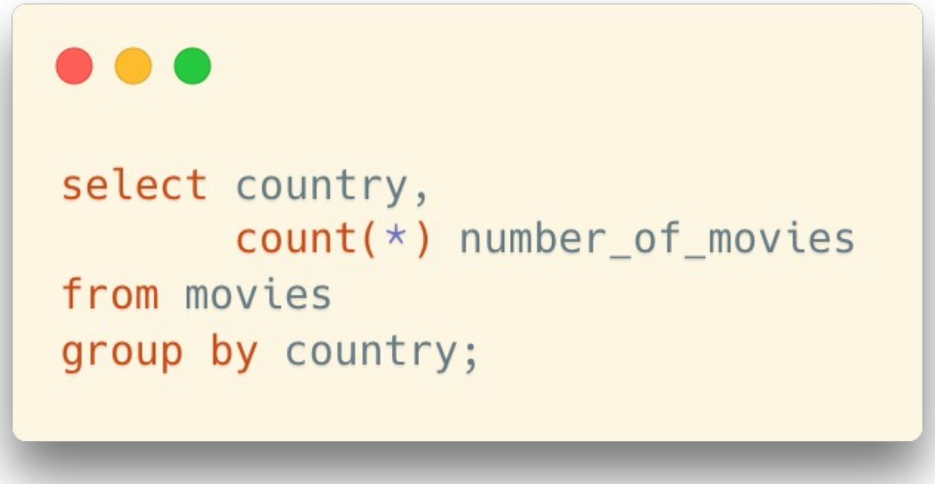
	country	number_of_movies
1	fr	571
2	ke	1
3	si	1
4	eg	11
5	nz	23
6	bg	4
7	ru	153
8	gh	1
9	pe	4
10	hr	1
11	sg	5
12	mx	59
13	cn	200

Aggregate Functions

- We say that we want to "group by country"
  - … and, for each country, the aggregate function `count(*)` says <u>how many movies we have</u>
    - "how many movies" = "how many rows"

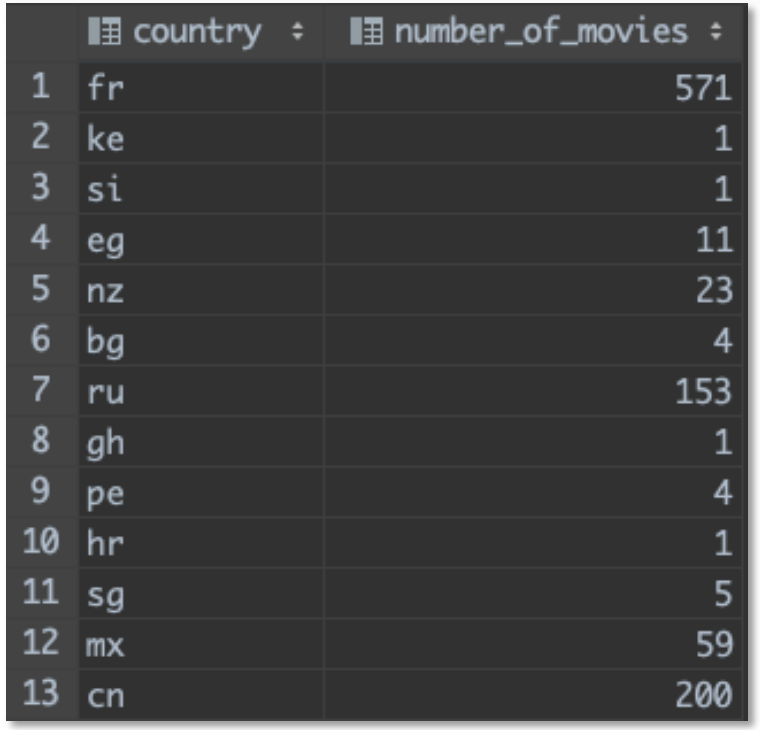
Caution: The table `movie` must be a relation (no duplicated movie records)

- … or, the counting result will not reflect the actual number of movies



```
select country,
       count(*) number_of_movies
from movies
group by country;
```

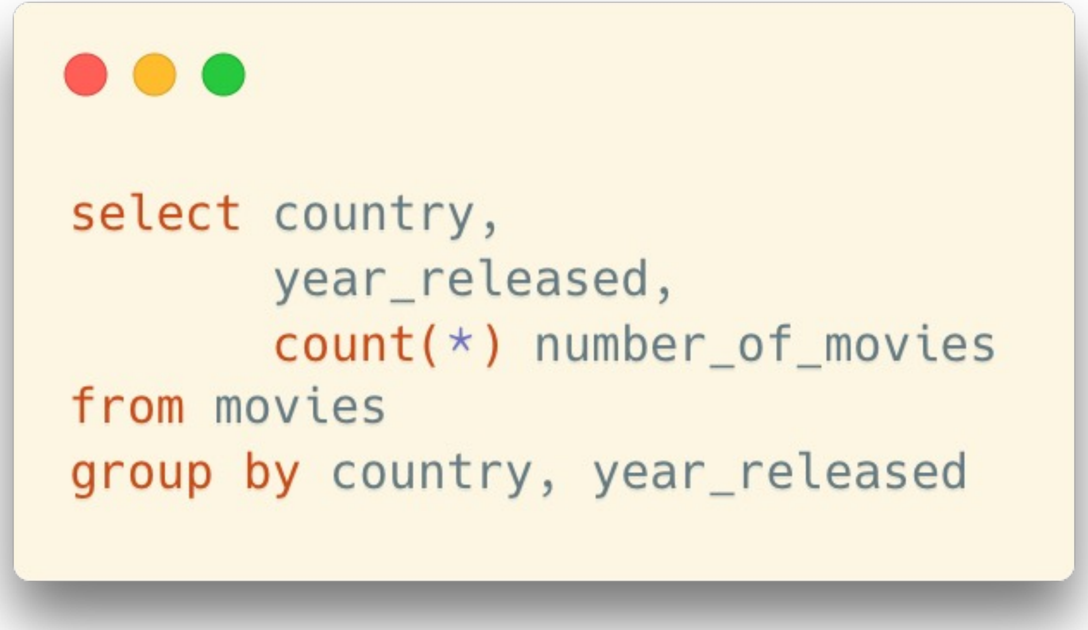
- The query result
  - One row for each group
  - The statistical value is attached in another column



	country	number_of_movies
1	fr	571
2	ke	1
3	si	1
4	eg	11
5	nz	23
6	bg	4
7	ru	153
8	gh	1
9	pe	4
10	hr	1
11	sg	5
12	mx	59
13	cn	200

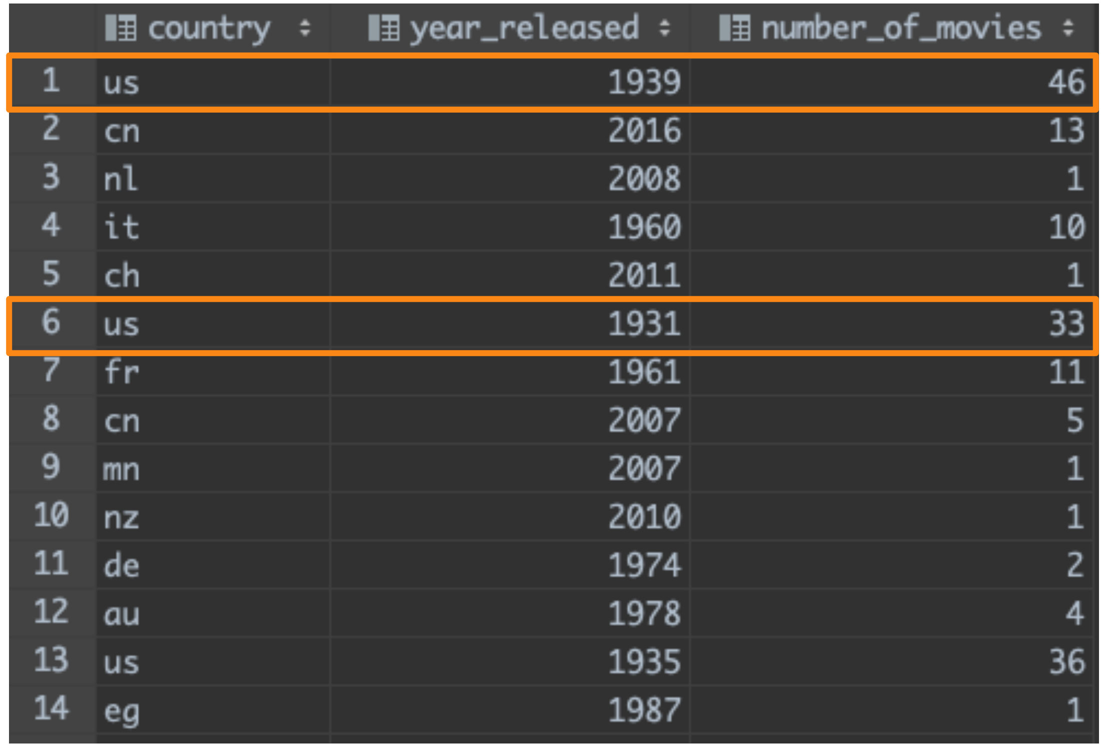
Aggregate Functions

- Group on several columns
  - Every column that <u>isn't an aggregate function</u> and <u>appears after `select`</u> must also appear after `group by`



```
select country,
       year_released,
       count(*) number_of_movies
from movies
group by country, year_released
```

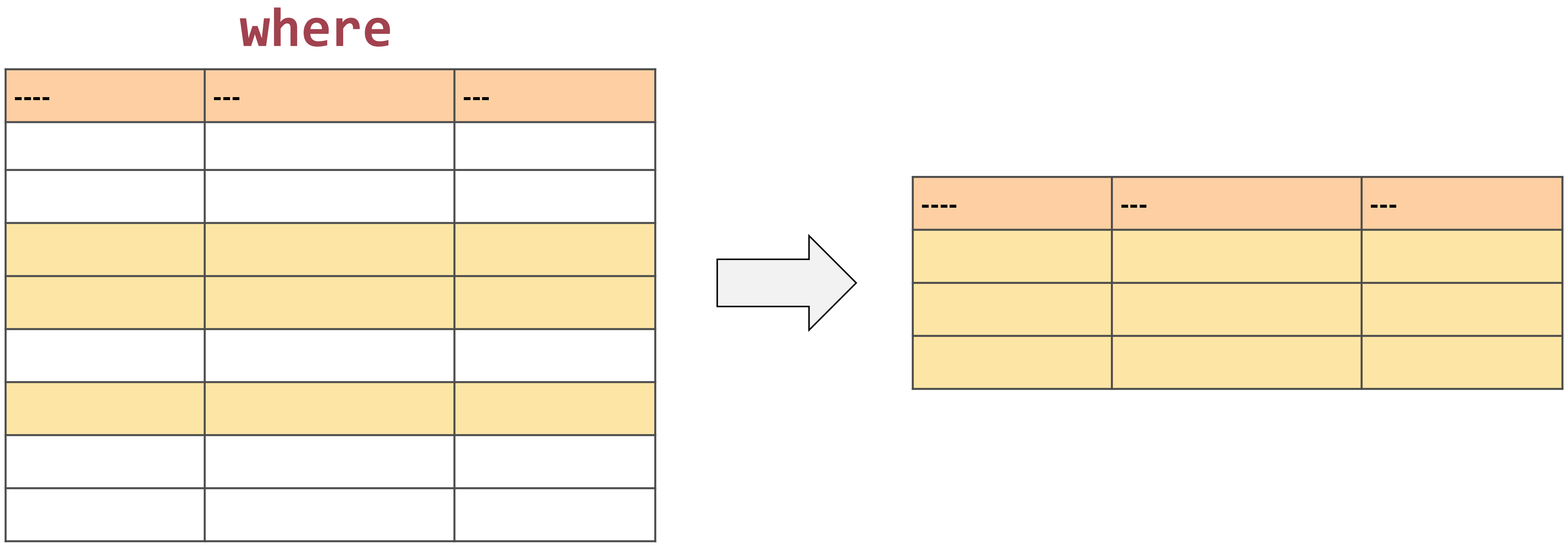
The combination of the countries and released years will appear in the result



	country	year_released	number_of_movies
1	us	1939	46
2	cn	2016	13
3	nl	2008	1
4	it	1960	10
5	ch	2011	1
6	us	1931	33
7	fr	1961	11
8	cn	2007	5
9	mn	2007	1
10	nz	2010	1
11	de	1974	2
12	au	1978	4
13	us	1935	36
14	eg	1987	1

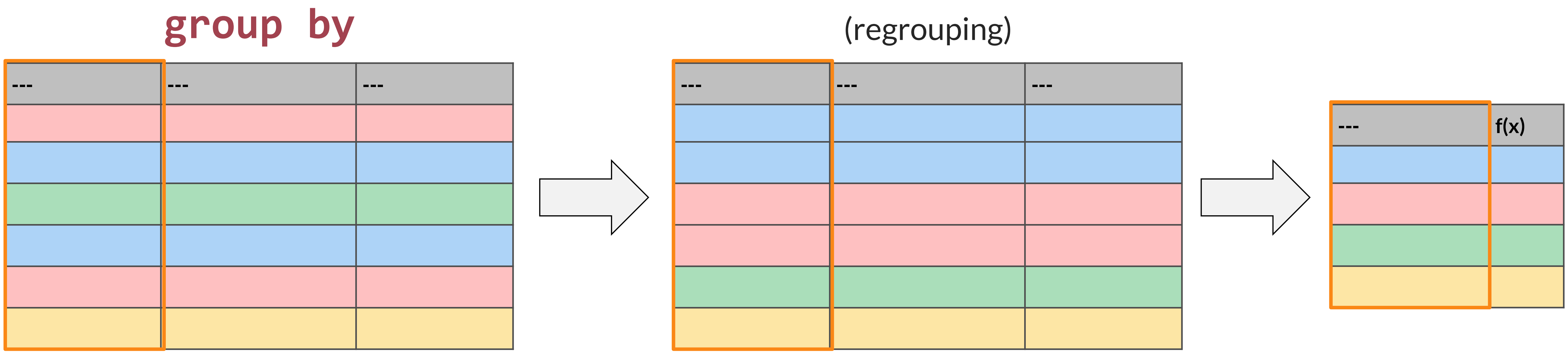
Aggregate Functions

- Beware of some performance implications
  - When you apply a simple `where` filter, you can start returning rows as soon as you have found a match.



Aggregate Functions

- Beware of some performance implications
  - With a group by, you must regroup rows before you can aggregate them and return results.
    - In other words, you have a preparatory phase that may take time, even if you return few rows in the end.
    - In interactive applications, end-users don't always understand it well.



Aggregate Functions

```
count(*)/count(col), min(col), max(col), stddev(col), avg(col)
```

- These aggregate function examples exist in almost all products
  - Most products implement other functions
  - Some work with any datatype, others only work with numerical columns

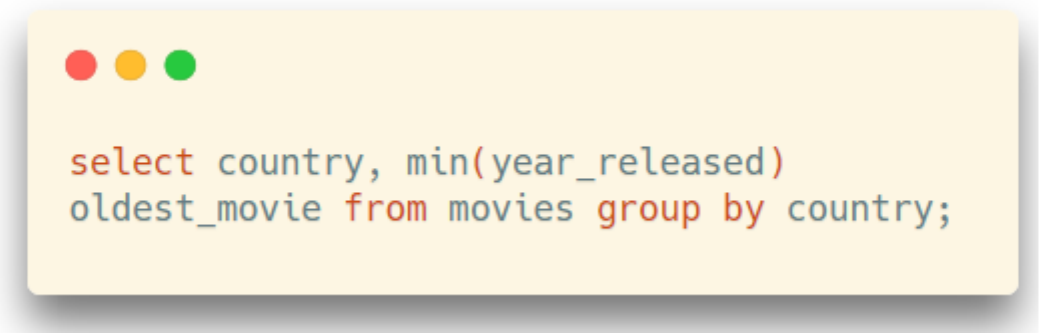
  - It is strongly recommended to refer to the database manual for details
    - For example, SQLite doesn’t have `stddev()` which computes the standard deviation

Aggregate Functions

- *Earliest release year by country?*

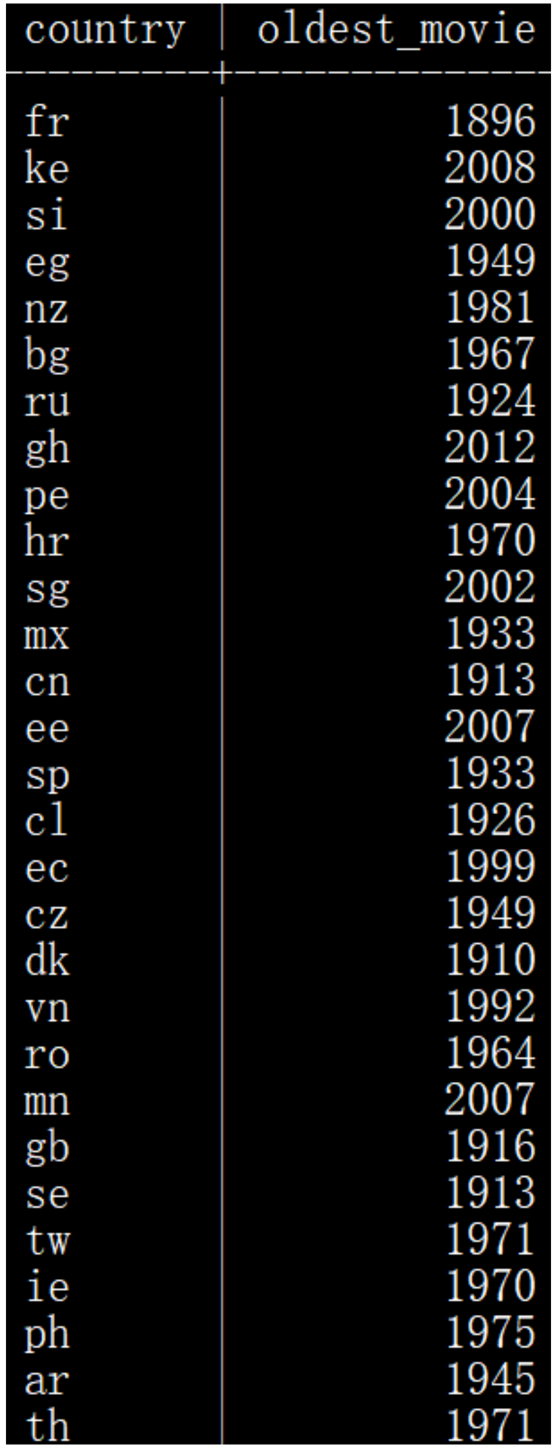
Aggregate Functions

- ***Earliest release year by country?***



```
select country, min(year_released)
oldest_movie from movies group by country;
```

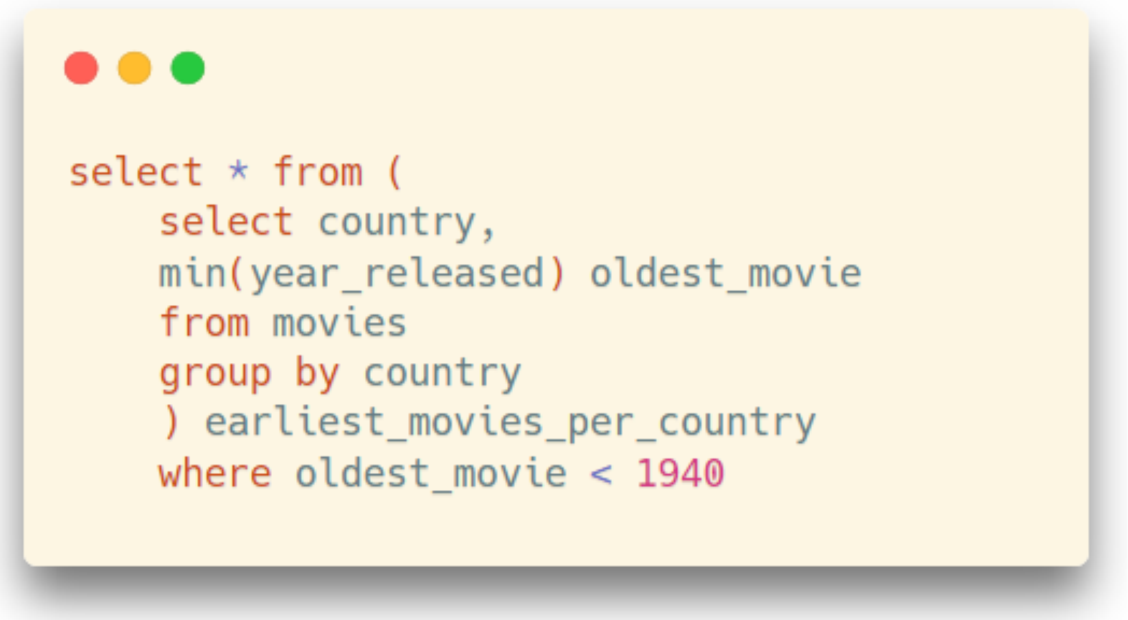
- Such a query answers the question
  - Note that in the demo database years are simple numerical values, but generally speaking min() applied to a date logically returns the earliest one.
  - The result will be a relation: no duplicates, and the key that identifies each row will be the country code (generally speaking, what follows GROUP BY).



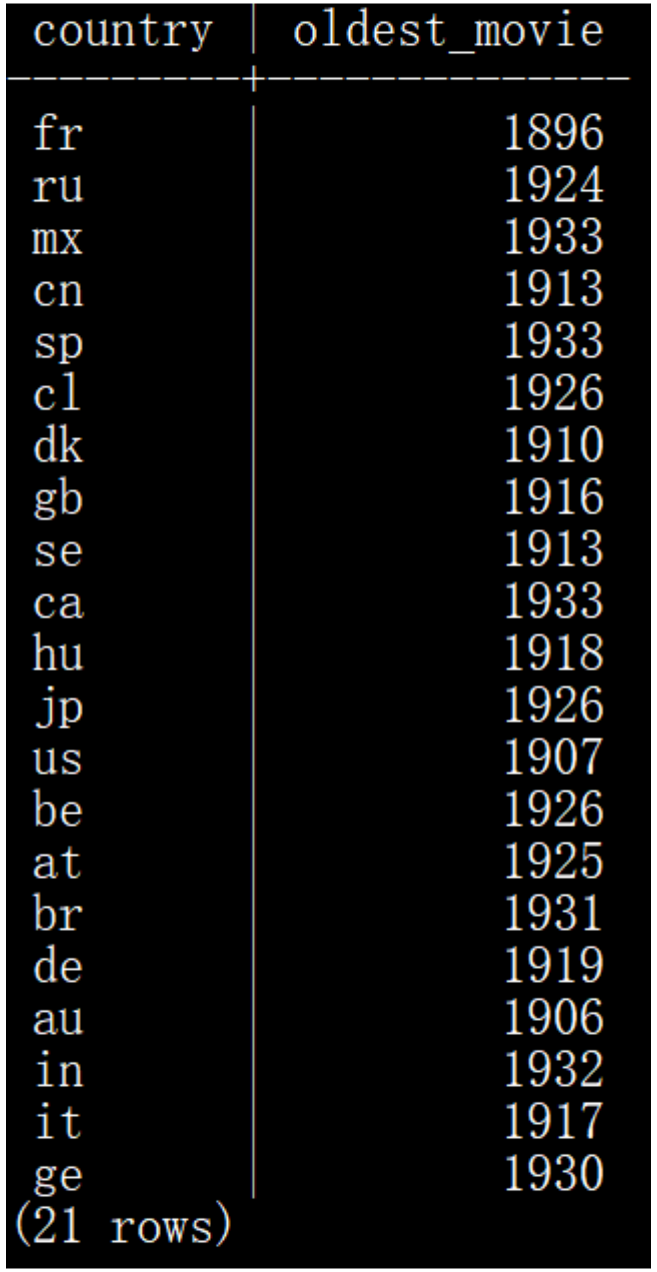
```
 country | oldest_movie
---------+--------------
 fr      |         1896
 ke      |         2008
 si      |         2000
 eg      |         1949
 nz      |         1981
 bg      |         1967
 ru      |         1924
 gh      |         2012
 pe      |         2004
 hr      |         1970
 sg      |         2002
 mx      |         1933
 cn      |         1913
 ee      |         2007
 sp      |         1933
 cl      |         1926
 ec      |         1999
 cz      |         1949
 dk      |         1910
 vn      |         1992
 ro      |         1964
 mn      |         2007
 gb      |         1916
 se      |         1913
 tw      |         1971
 ie      |         1970
 ph      |         1975
 ar      |         1945
 th      |         1971
```

Aggregate Functions

- Therefore, we can validly apply another relational operation such as the “select” operation (row filtering) and only return countries for which the earliest movie was released before 1940.



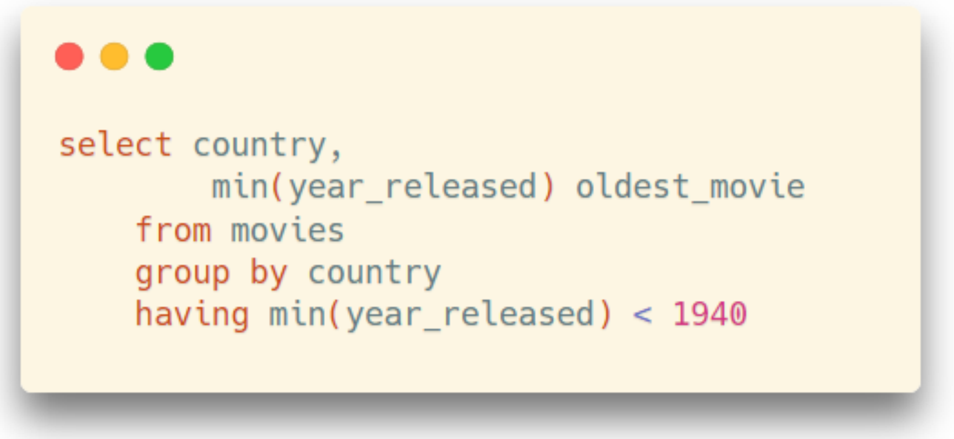
```
select * from (
    select country,
    min(year_released) oldest_movie
    from movies
    group by country
    ) earliest_movies_per_country
    where oldest_movie < 1940
```



```
 country | oldest_movie
---------+--------------
 fr      |         1896
 ru      |         1924
 mx      |         1933
 cn      |         1913
 sp      |         1933
 cl      |         1926
 dk      |         1910
 gb      |         1916
 se      |         1913
 ca      |         1933
 hu      |         1918
 jp      |         1926
 us      |         1907
 be      |         1926
 at      |         1925
 br      |         1931
 de      |         1919
 au      |         1906
 in      |         1932
 it      |         1917
 ge      |         1930
(21 rows)
```

Aggregate Functions

- There is a short-hand that makes nesting queries unnecessary (in the same way as AND allows multiple filters). You can have a condition on the result of an aggregate with **having.**



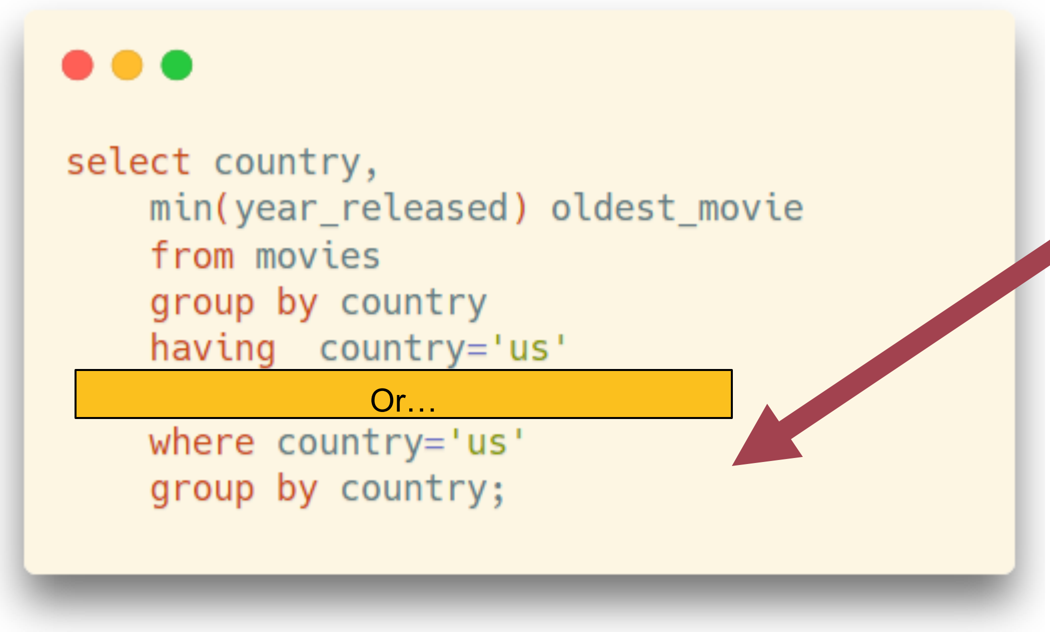
```
select country,
        min(year_released) oldest_movie
    from movies
    group by country
    having min(year_released) < 1940
```

- Now, keep in mind that aggregating rows requires sorting them in a way or another, and that sorts are always costly operations that don't scale well (cost increases faster than the number of rows sorted).

Aggregate Functions

SORT： Time complexity of sorting algorithms: O(n*log(n))

- The following query is perfectly valid in SQL. What you are doing is aggregating movies for all countries, then discarding everything that isn't American:



```
select country,
    min(year_released) oldest_movie
    from movies
    group by country
    having  country='us'
```

Or…

```
    where country='us'
    group by country;
```

The efficient way to proceed is of course to select American movies first, and only aggregate them.

- SQL Server will do the right thing behind your back.
- Oracle will assume that you have some obscure reason for writing your query that way and will do as told. It can hurt.

Aggregate Functions

- All database management systems have a highly important component that we'll see again, called the "query optimizer".
  - It takes your query and tries to find the most efficient way to run it.
  - Sometimes it tries to outsmart you, with from time to time unintended consequences
  - Sometimes it optimistically assumes that you know what you are doing
  - ... In all, optimizers don't all behave the same.

Aggregate Functions

- *Nulls?*

- When you apply a function or operators to a null, with very few exceptions the result is null because the result of a transformation applied to something unknown is an unknown quantity. What happens with aggregates?

- known + unknown = unknown

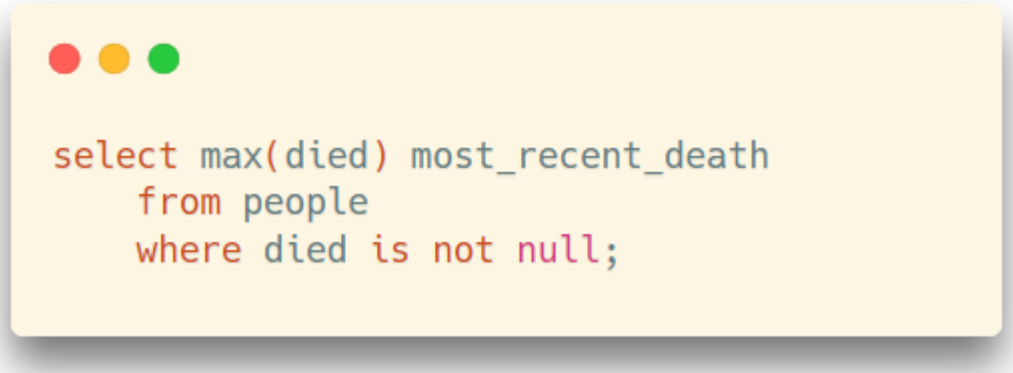
Aggregate Functions

- ***Nulls?***

- Aggregate functions ***ignore*** Nulls.

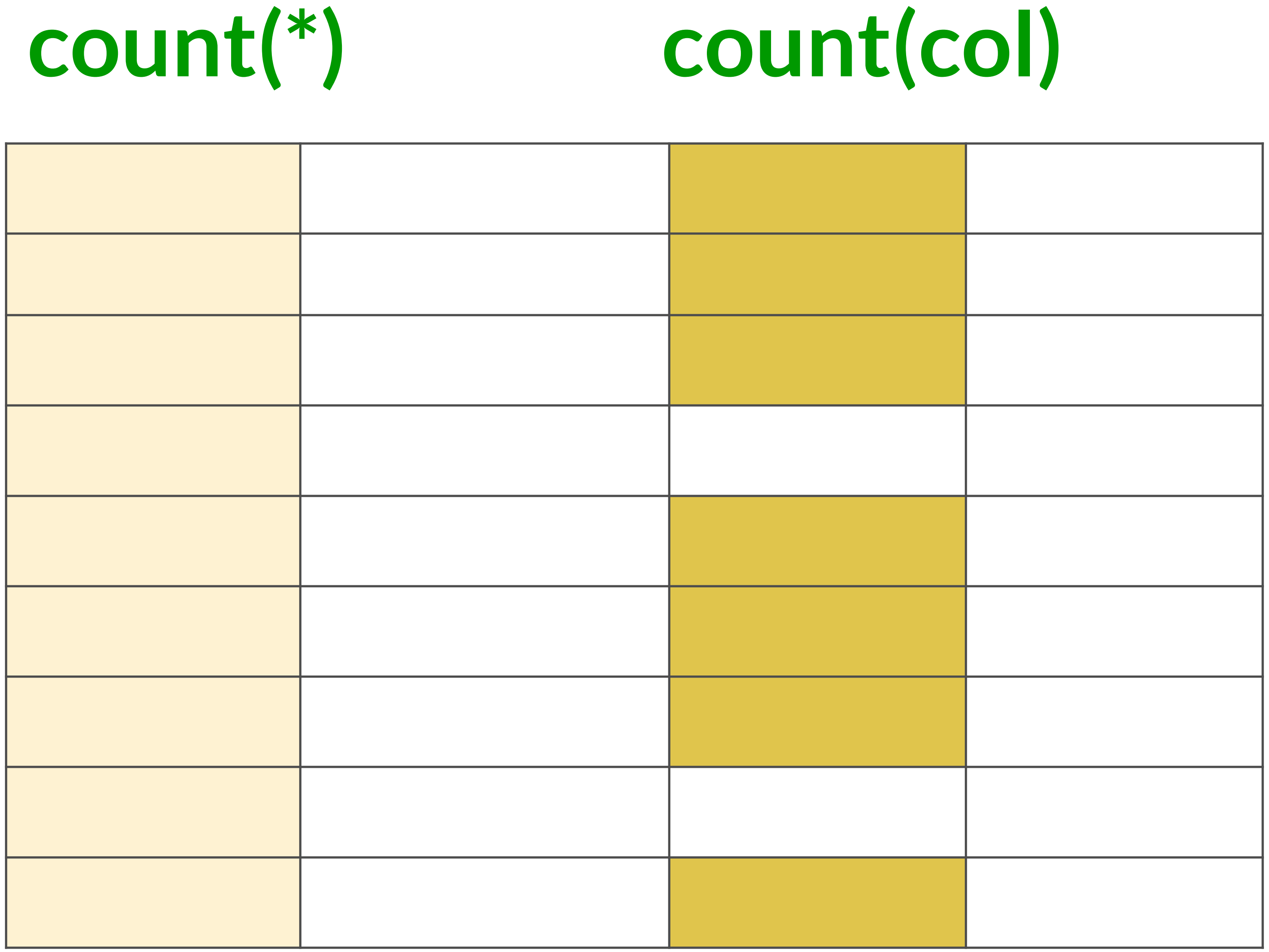
Aggregate Functions

- In this query, the `where` condition changes nothing to the result
  - Perhaps it makes more obvious that we are dealing with dead people only, but for the SQL engine it's implicit.



```sql
select max(died) most_recent_death
    from people
    where died is not null;
```

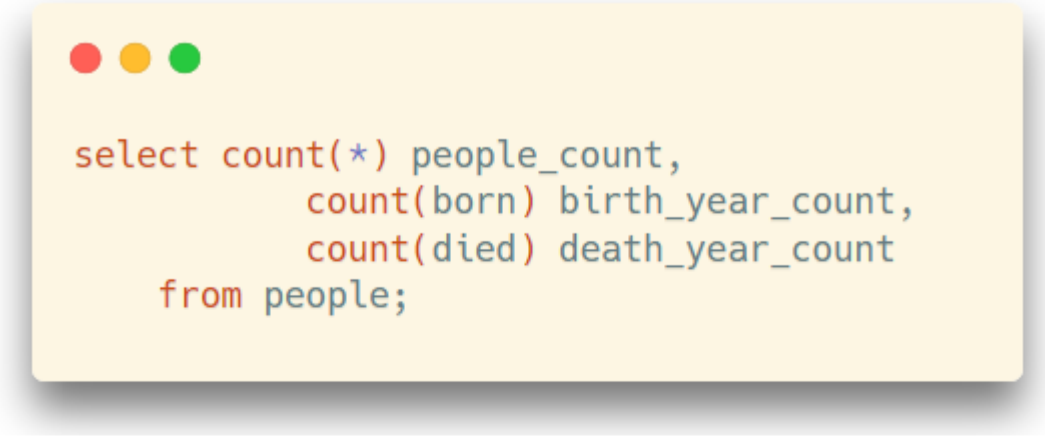
Aggregate Functions



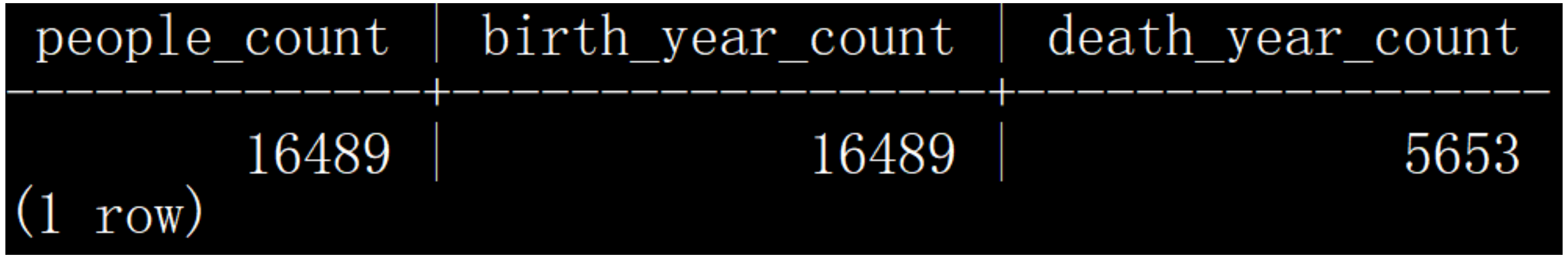
- Depending on the column you count, the function can therefore return different values. count(*) will always return the number of rows in the result set, because there is always one value that isn't null in a row (otherwise you wouldn't have a row in the first place)

Aggregate Functions

- Counting a mandatory column such as BORN will return the same value as COUNT(*)
  - The third count, though, will only return the number of dead people in the table.



```
select count(*) people_count,
           count(born) birth_year_count,
           count(died) death_year_count
    from people;
```



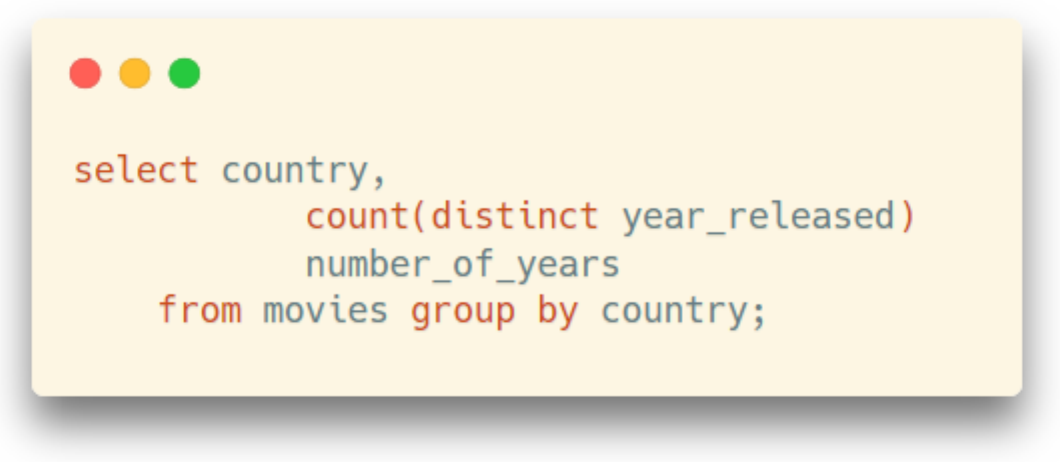
```
 people_count | birth_year_count | death_year_count
--------------+------------------+------------------
        16489 |            16489 |             5653
(1 row)
```

Aggregate Functions

- `select count(colname)`
- `select count(distinct colname)`

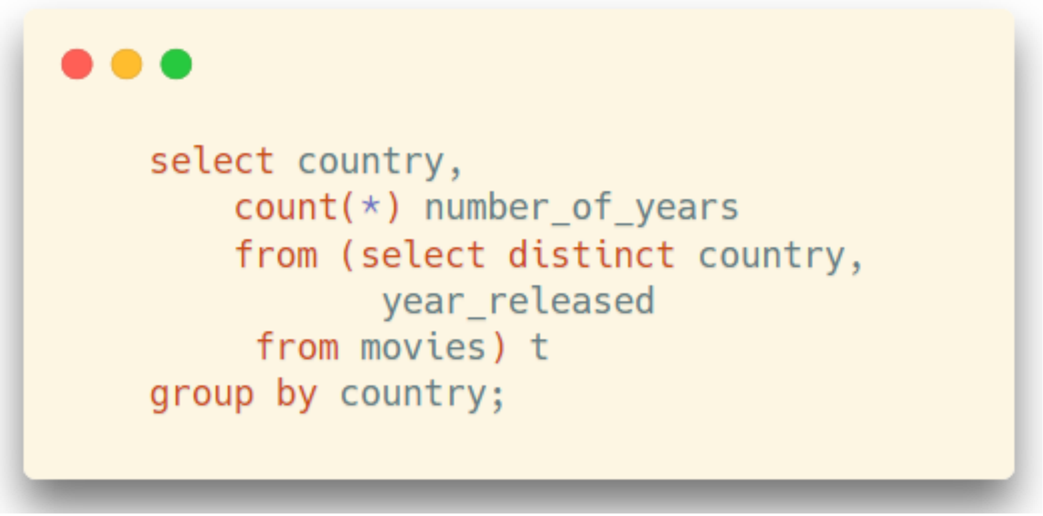
- In some cases, you only want to count distinct values
  - For instance, you may want to count how many different surnames start with a Q instead of how many people have a surname that starts with a Q.

Aggregate Functions



```
select country,
           count(distinct year_released)
           number_of_years
    from movies group by country;
```

- These two queries are equivalent



```
select country,
    count(*) number_of_years
    from (select distinct country,
           year_released
     from movies) t
group by country;
```

Here we'll only get one row per country and year

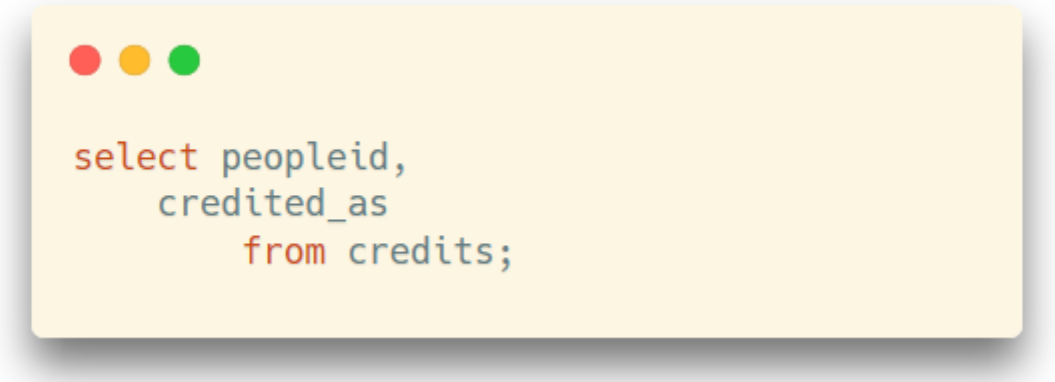
Aggregate Functions

- How many people are both actors and directors?

credits

Aggregate Functions

movie_id	people_id	credited_as
8	37	D
8	38	A
8	39	A
8	40	A
10	11	A
10	12	A
10	15	D
10	16	A
10	17	A

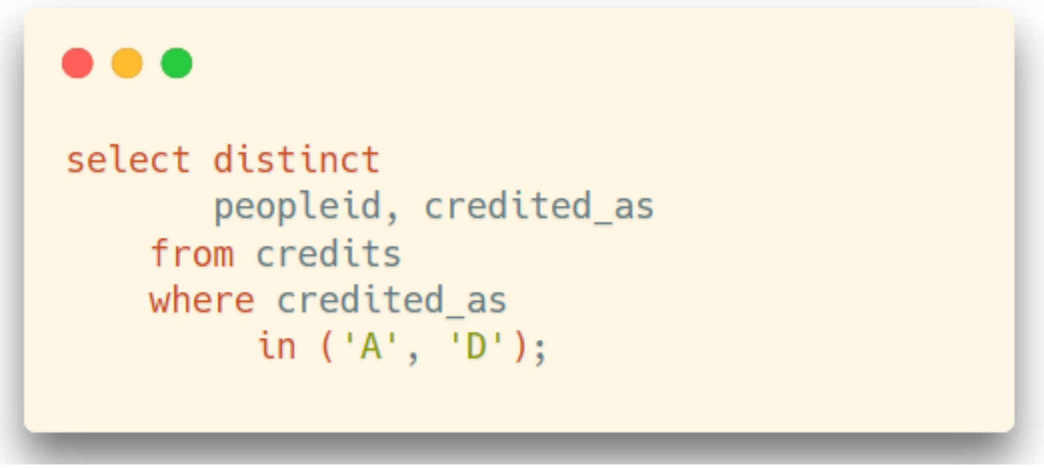


```
select peopleid,
    credited_as
        from credits;
```

- There is no restriction such as "that have played in a movie that they have directed", so the `movie_id` is irrelevant.
- But if we remove the `movie_id`, we have tons of duplicates. Not a relation!

Aggregate Functions

- People who appear twice are the ones we want.



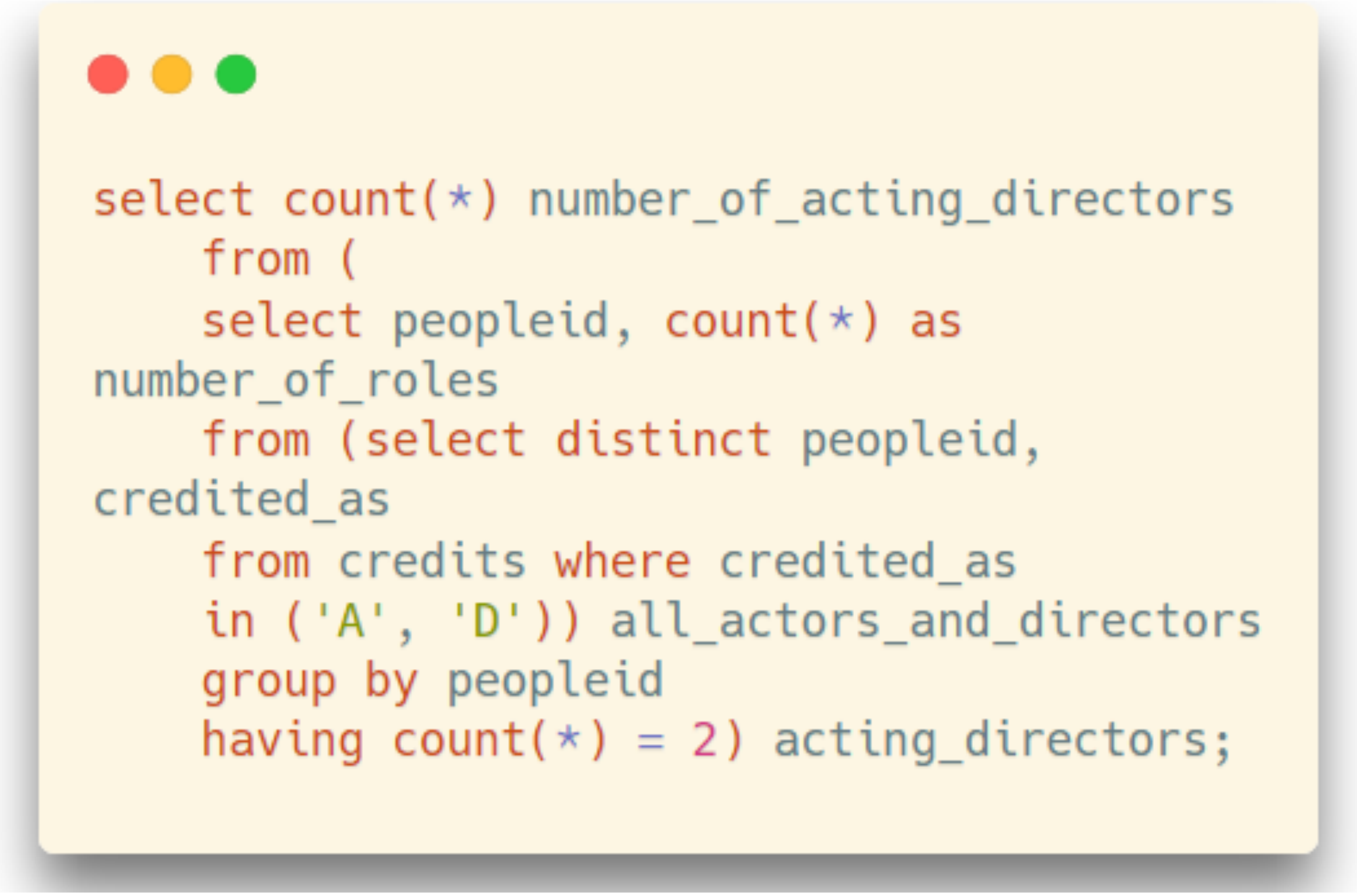
```
select distinct
       peopleid, credited_as
    from credits
    where credited_as
         in ('A', 'D');
```

people_id	credited_as
11	D
11	A
12	A
15	A
16	A
17	A
37	D
38	A
39	A

  - `distinct` will remove duplicates and provide a true relation.
  - We specify the values for `credited_as`
    - There are no other values now
    - but you can't predict the future. Someday there may be producers or directors of photography (cinematographer).

Aggregate Functions

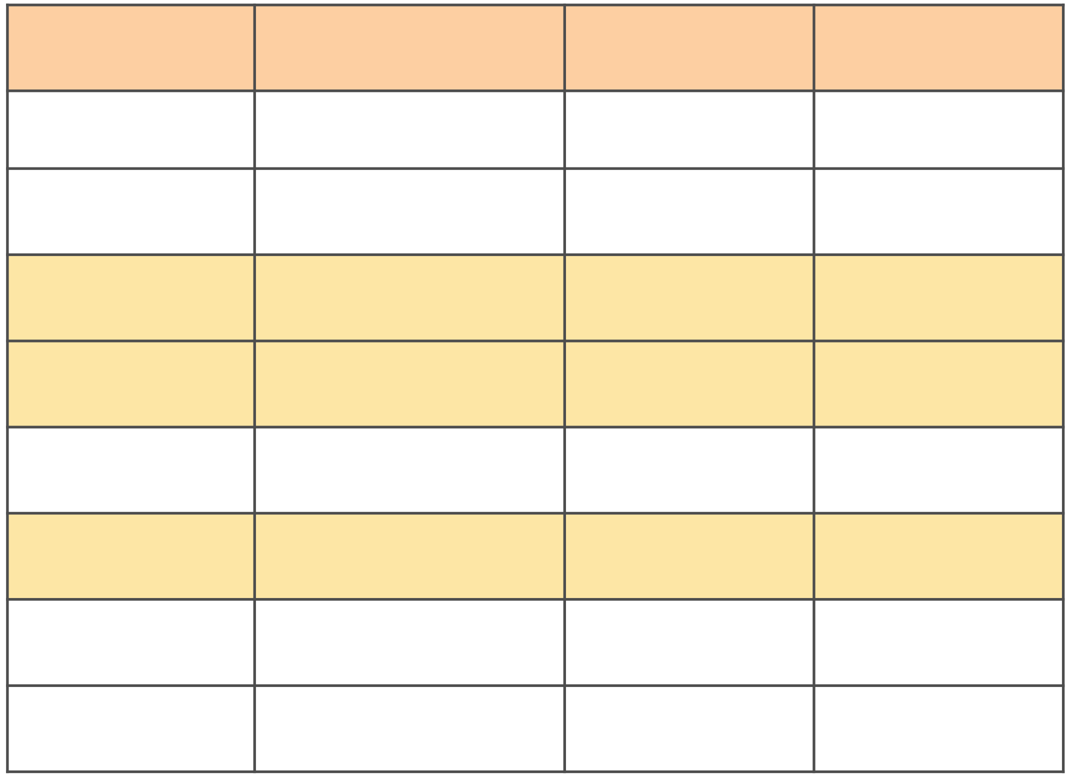
- The `having` selects only people who appear twice ... and we just have to count them. Mission accomplished.



```
select count(*) number_of_acting_directors
    from (
    select peopleid, count(*) as
number_of_roles
    from (select distinct peopleid,
credited_as
    from credits where credited_as
    in ('A', 'D')) all_actors_and_directors
    group by peopleid
    having count(*) = 2) acting_directors;
```

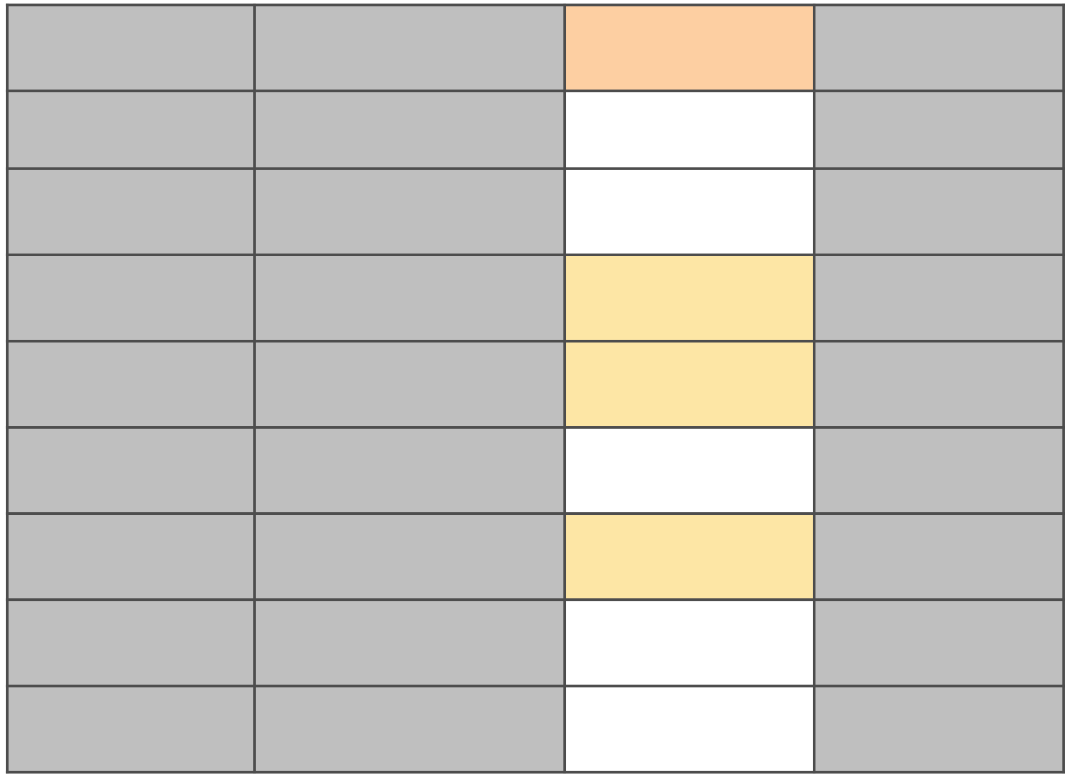
Join

Retrieving Data from Multiple Tables



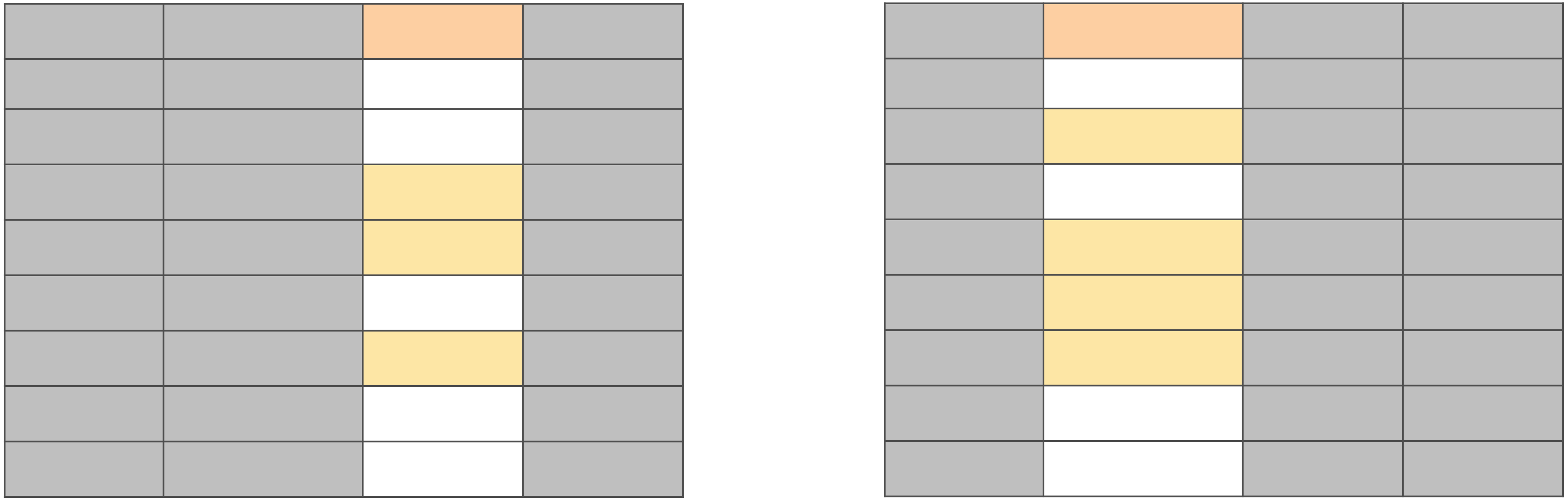
- We have seen the basic operation consisting in filtering rows (an operator called SELECT by Codd)

Retrieving Data from Multiple Tables



- We have seen how we can only return some columns (called PROJECT by Codd), and that we must be careful not to return duplicates when we aren't returning a full key.

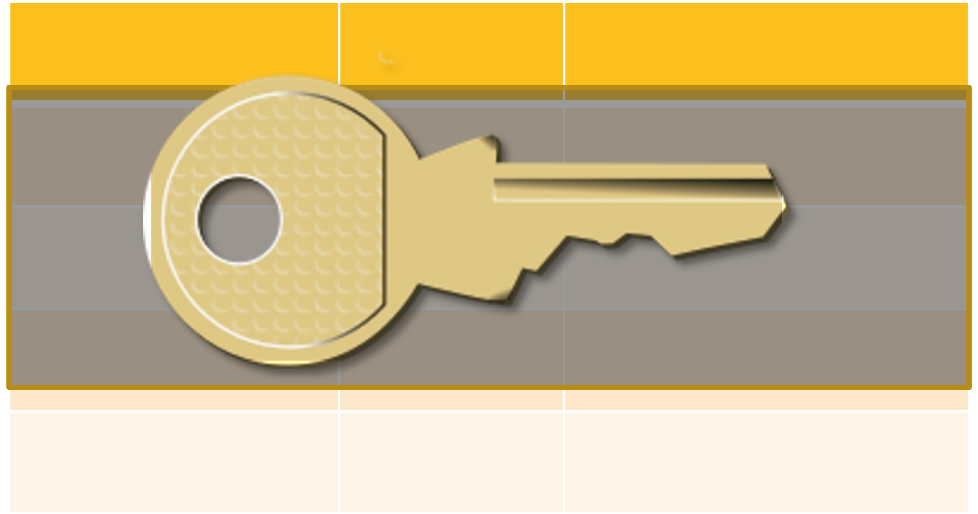
Retrieving Data from Multiple Tables



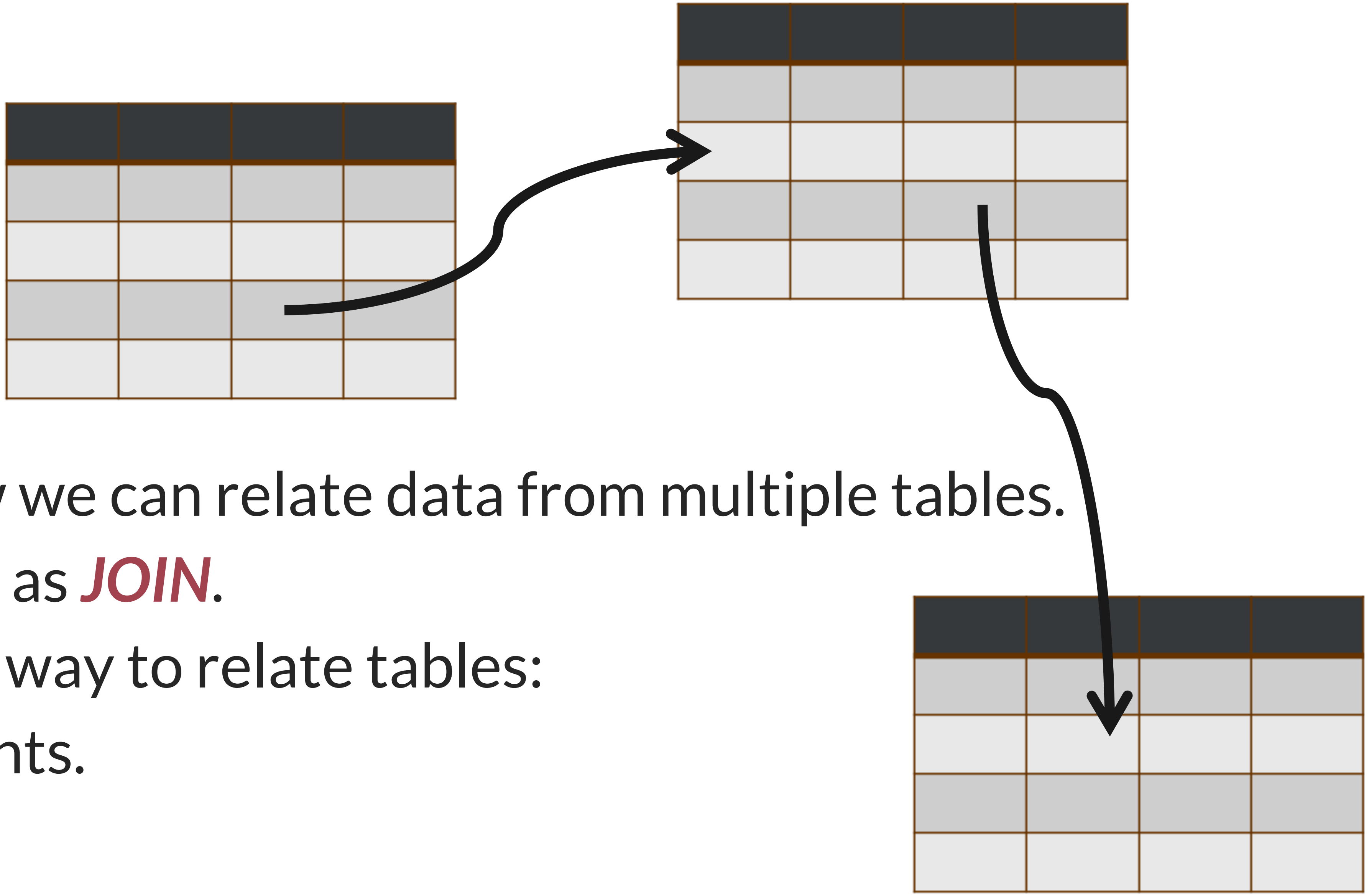
- We have also seen how we can return data that doesn't exist as such in tables by applying functions to columns.

Retrieving Data from Multiple Tables

- What is Important is that in all cases our result set looks like a clean table, with no duplicates and a column (or combination of columns) that could be used as a key
  - If this is the case, we are safe. This must be true at every stage in a complex query built by successive layers.



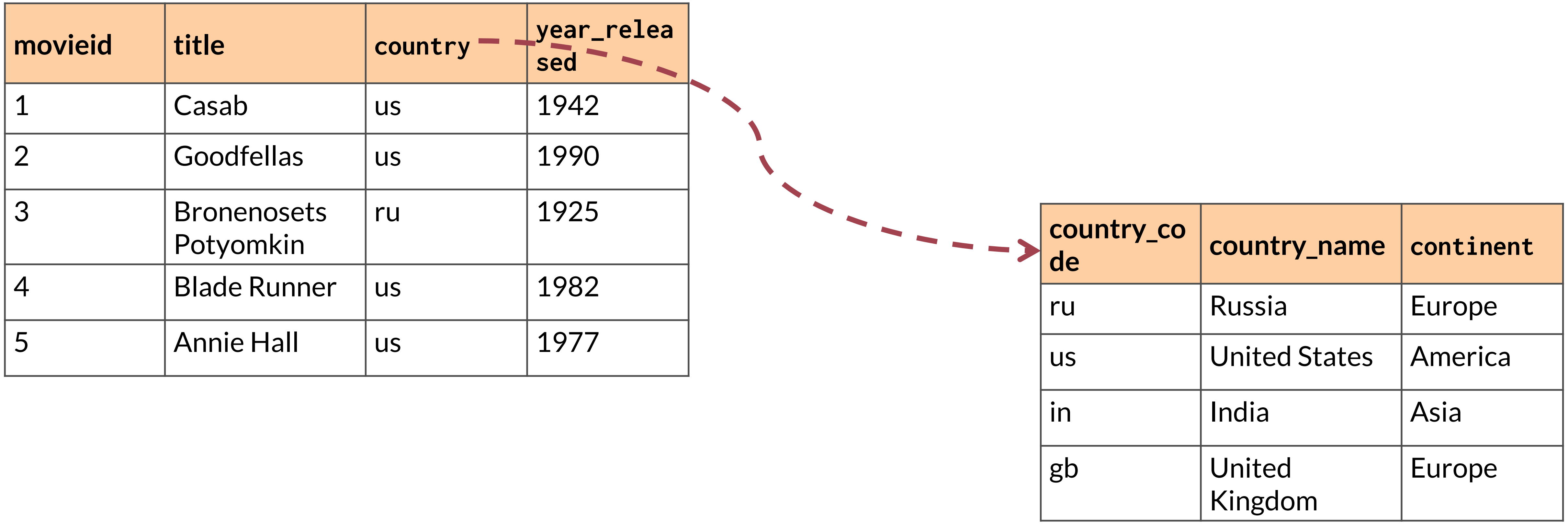
Retrieving Data from Multiple Tables



- It's time now to see how we can relate data from multiple tables.
- This operation is known as ***JOIN***.
- We have already seen a way to relate tables:

  foreign key constraints.

Retrieving Data from Multiple Tables



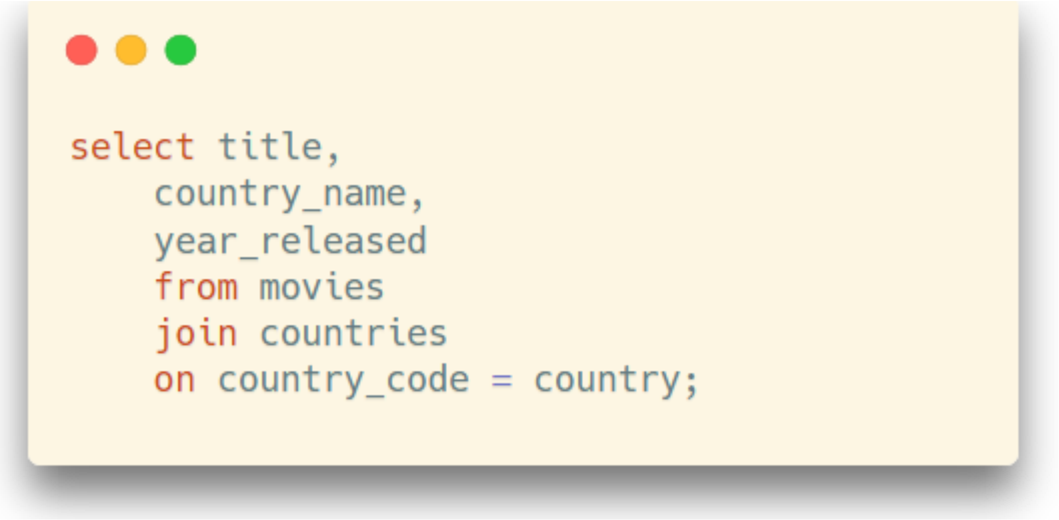
movieid	title	country	year_released
1	Casab	us	1942
2	Goodfellas	us	1990
3	Bronenosets Potyomkin	ru	1925
4	Blade Runner	us	1982
5	Annie Hall	us	1977

country_code	country_name	continent
ru	Russia	Europe
us	United States	America
in	India	Asia
gb	United Kingdom	Europe

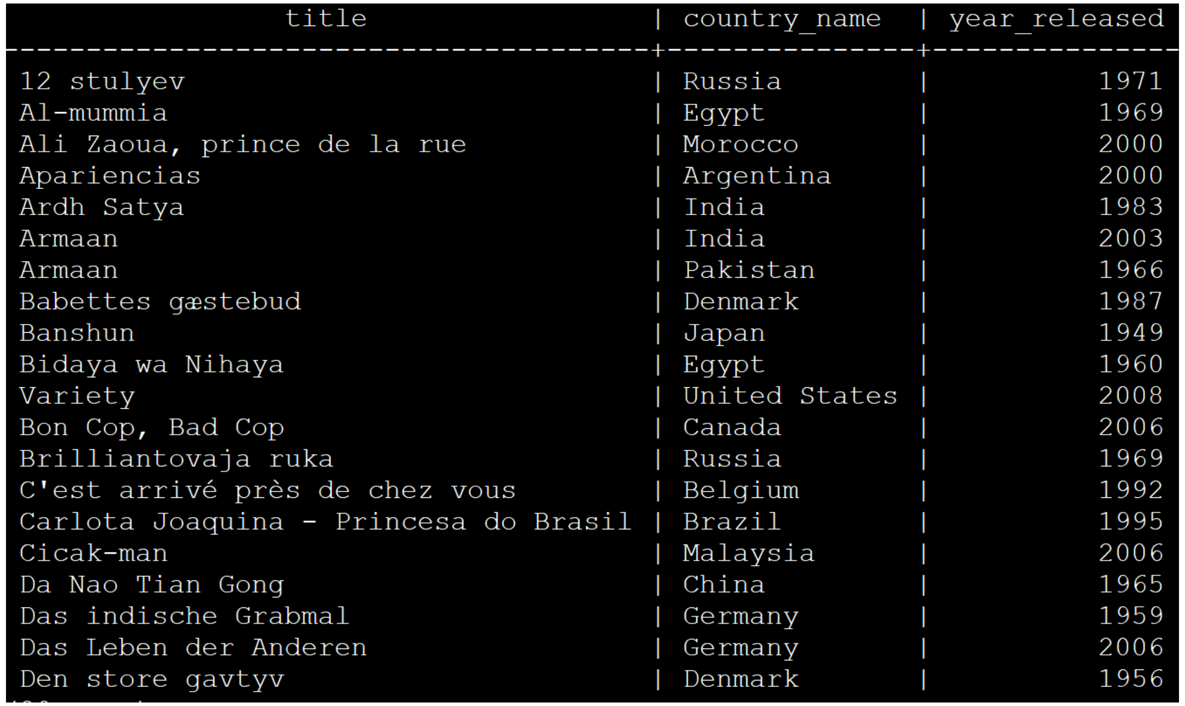
- The “country” column in “movies” can be used to retrieve the country name from “countries”.

Retrieving Data from Multiple Tables

- This is done with this type of query. We retrieve, and display as a single set, pieces of data coming from two different tables.



```
select title,
    country_name,
    year_released
    from movies
    join countries
    on country_code = country;
```



```
                 title                 | country_name  | year_released
---------------------------------------+---------------+---------------
 12 stulyev                            | Russia        |          1971
 Al-mummia                             | Egypt         |          1969
 Ali Zaoua, prince de la rue           | Morocco       |          2000
 Apariencias                           | Argentina     |          2000
 Ardh Satya                            | India         |          1983
 Armaan                                | India         |          2003
 Armaan                                | Pakistan      |          1966
 Babettes gæstebud                     | Denmark       |          1987
 Banshun                               | Japan         |          1949
 Bidaya wa Nihaya                      | Egypt         |          1960
 Variety                               | United States |          2008
 Bon Cop, Bad Cop                      | Canada        |          2006
 Brilliantovaja ruka                   | Russia        |          1969
 C'est arrivé près de chez vous        | Belgium       |          1992
 Carlota Joaquina - Princesa do Brasil | Brazil        |          1995
 Cicak-man                             | Malaysia      |          2006
 Da Nao Tian Gong                      | China         |          1965
 Das indische Grabmal                  | Germany       |          1959
 Das Leben der Anderen                 | Germany       |          2006
 Den store gavtyv                      | Denmark       |          1956
```

Retrieving Data from Multiple Tables

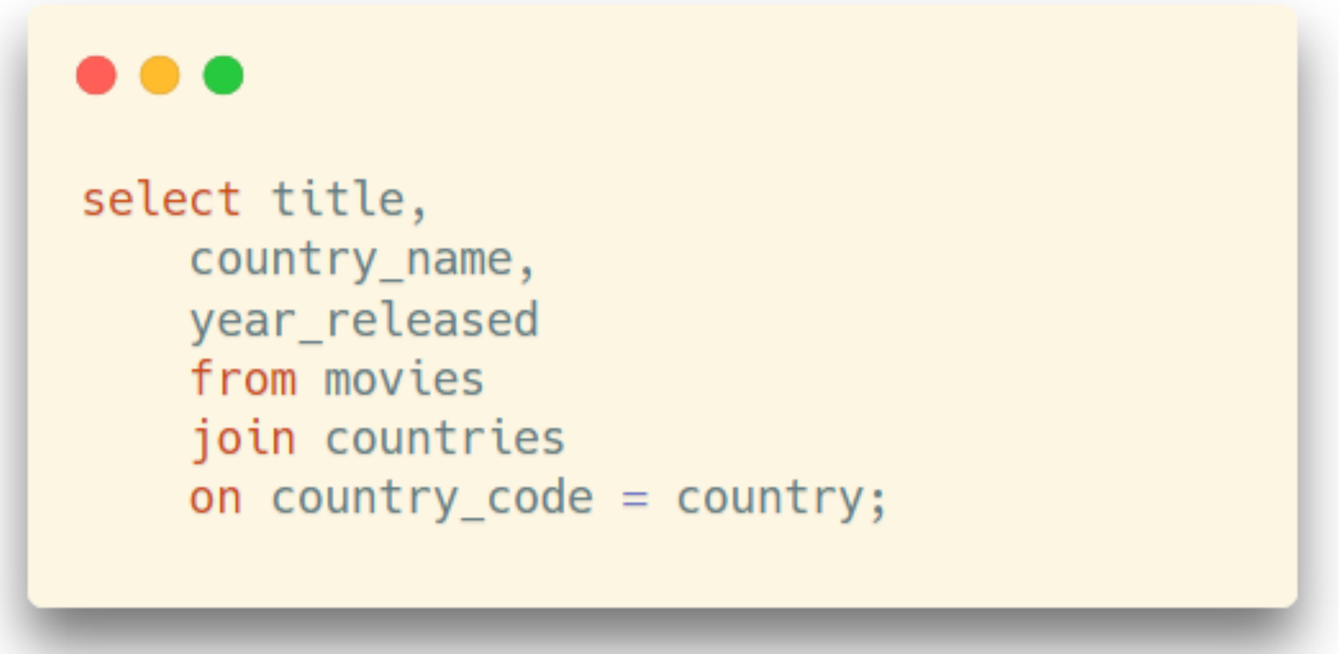
- The join operation will create a virtual table with all combinations between rows in Table1 and rows in Table2.
- If Table1 has R1 rows, and Table2 has R2, the huge virtual table has R1xR2 rows.

movies join countries

movieid	title	country	year_released	country_code	country_name	continent
1	Casablanca	us	1942	ru	Russia	Europe
1	Casablanca	us	1942	us	United States	America
1	Casablanca	us	1942	in	India	Asia
1	Casablanca	us	1942	gb	United Kingdom	Europe
1	Casablanca	us	1942	ru	Russia	Europe

Retrieving Data from Multiple Tables

- The join condition says which values in each table must match for our associating the other columns



```
select title,
    country_name,
    year_released
    from movies
    join countries
    on country_code = country;
```

Retrieving Data from Multiple Tables

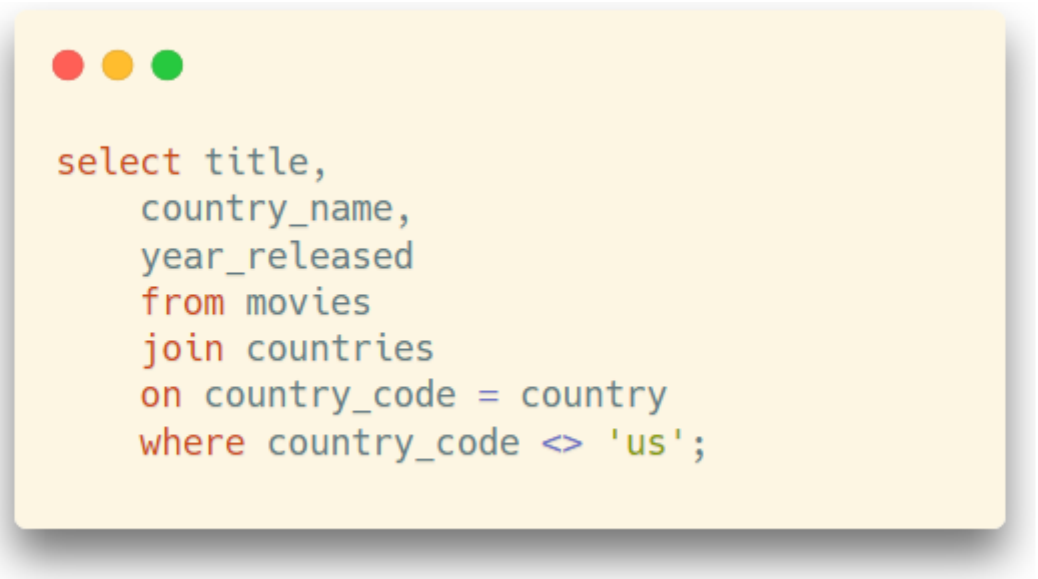
```
movies join countries
```

movieid	title	country	year_released	country_code	country_name	continent
1	Casablanca	us	1942	ru	Russia	Europe
1	Casablanca	us	1942	us	United States	America
1	Casablanca	us	1942	in	India	Asia
1	Casablanca	us	1942	gb	United Kingdom	Europe
1	Casablanca	us	1942	ru	Russia	Europe

- We use on country_code = country to filter out unrelated rows to make a much smaller virtual table.

Retrieving Data from Multiple Tables

- From this virtual table
  - Retrieve some columns and apply filtering conditions to any column

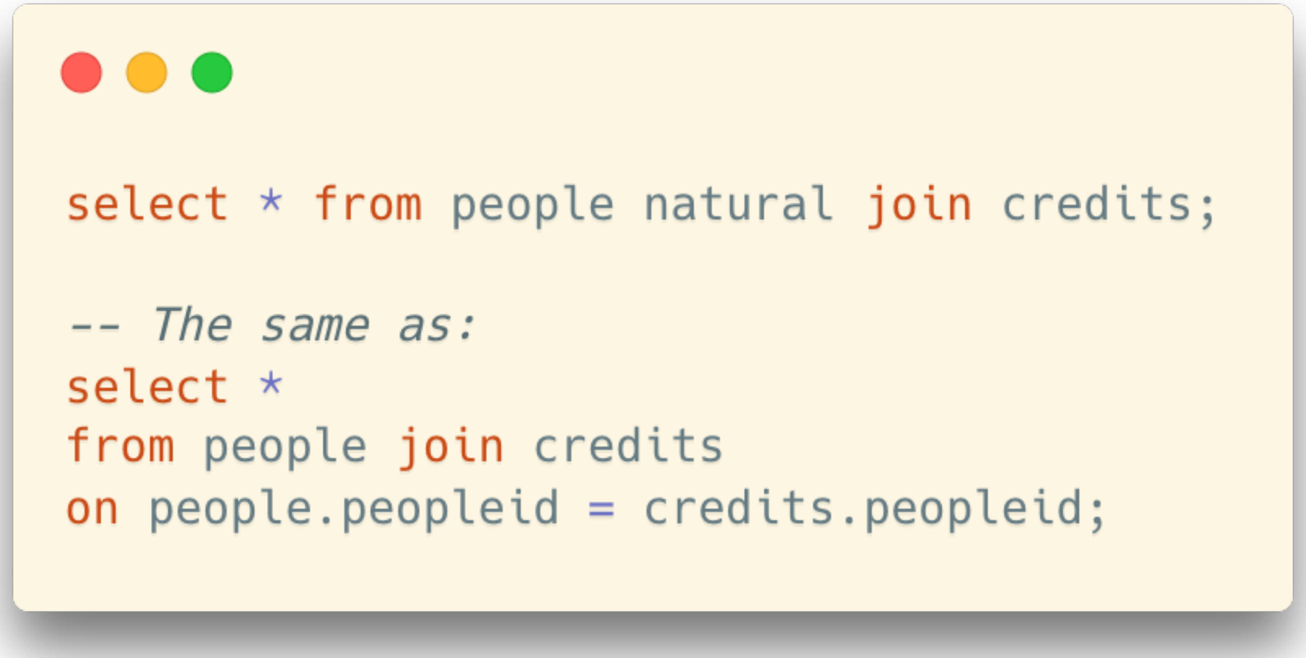


```
select title,
    country_name,
    year_released
    from movies
    join countries
    on country_code = country
    where country_code <> 'us';
```

movieid	title	country	year_released	country_code	country_name	continent
1	Casablanca	us	1942	us	United States	America
2	Goodfellas	us	1990	us	United States	America
3	Bronenosets Potyomkin	ru	1925	ru	Russia	Europe
4	Blade Runner	us	1982	us	United States	America

Natural Join

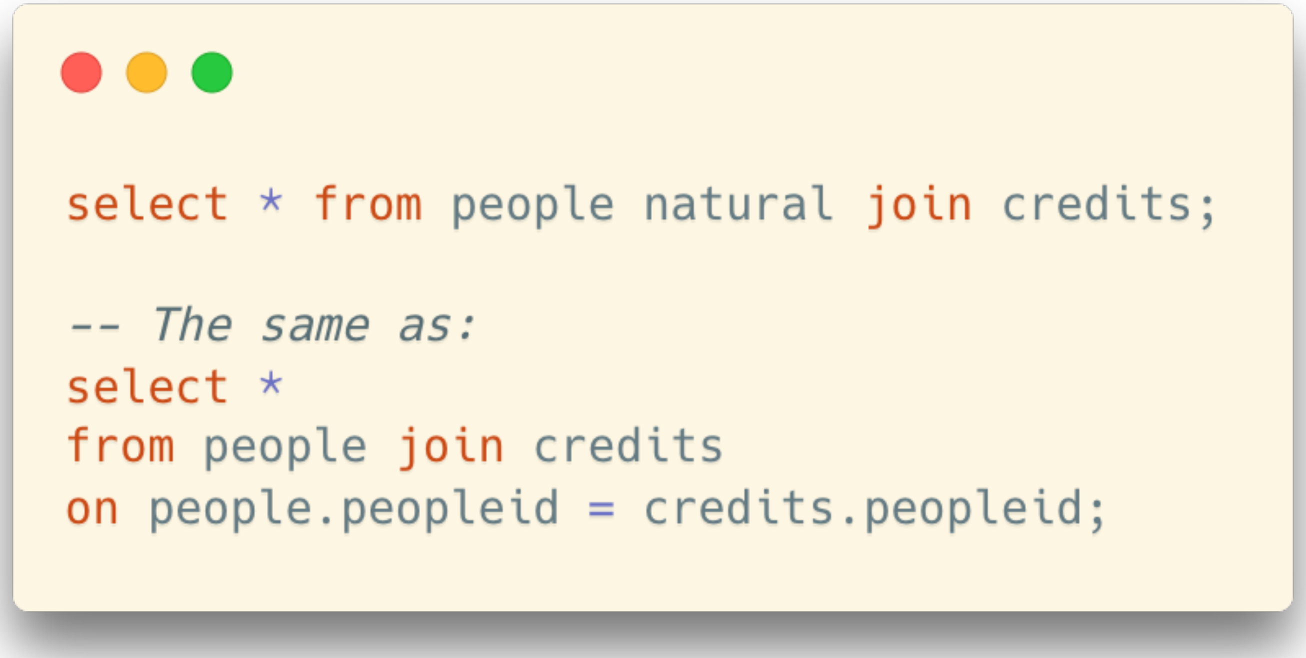
- What if we don’t specify the column?
  - Natural join



```
select * from people natural join credits;

-- The same as:
select *
from people join credits
on people.peopleid = credits.peopleid;
```

Natural Join

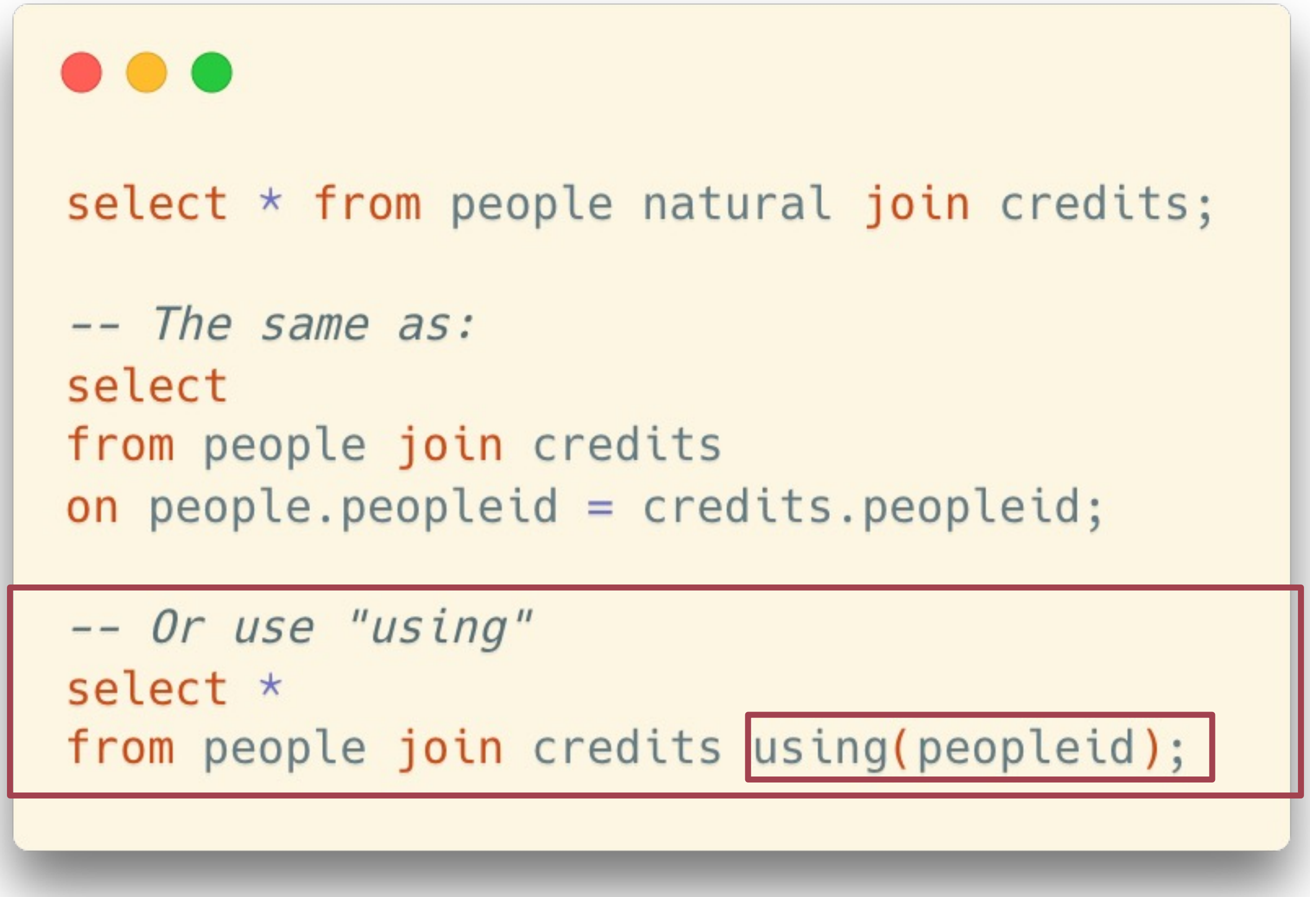
- What if we don’t specify the column?
  - Natural join
- *“If a column has the same name, then we should join on it”*
  - Bad idea!
  - Same name != Same meaning



```
select * from people natural join credits;

-- The same as:
select *
from people join credits
on people.peopleid = credits.peopleid;
```

Natural Join

- What if we don’t specify the column?
  - Natural join
- *“If a column has the same name, then we should join on it”*
  - Bad idea!
  - Same name != Same semantic
- In join (not natural join):
  - Use `using` to specify the column with the same name

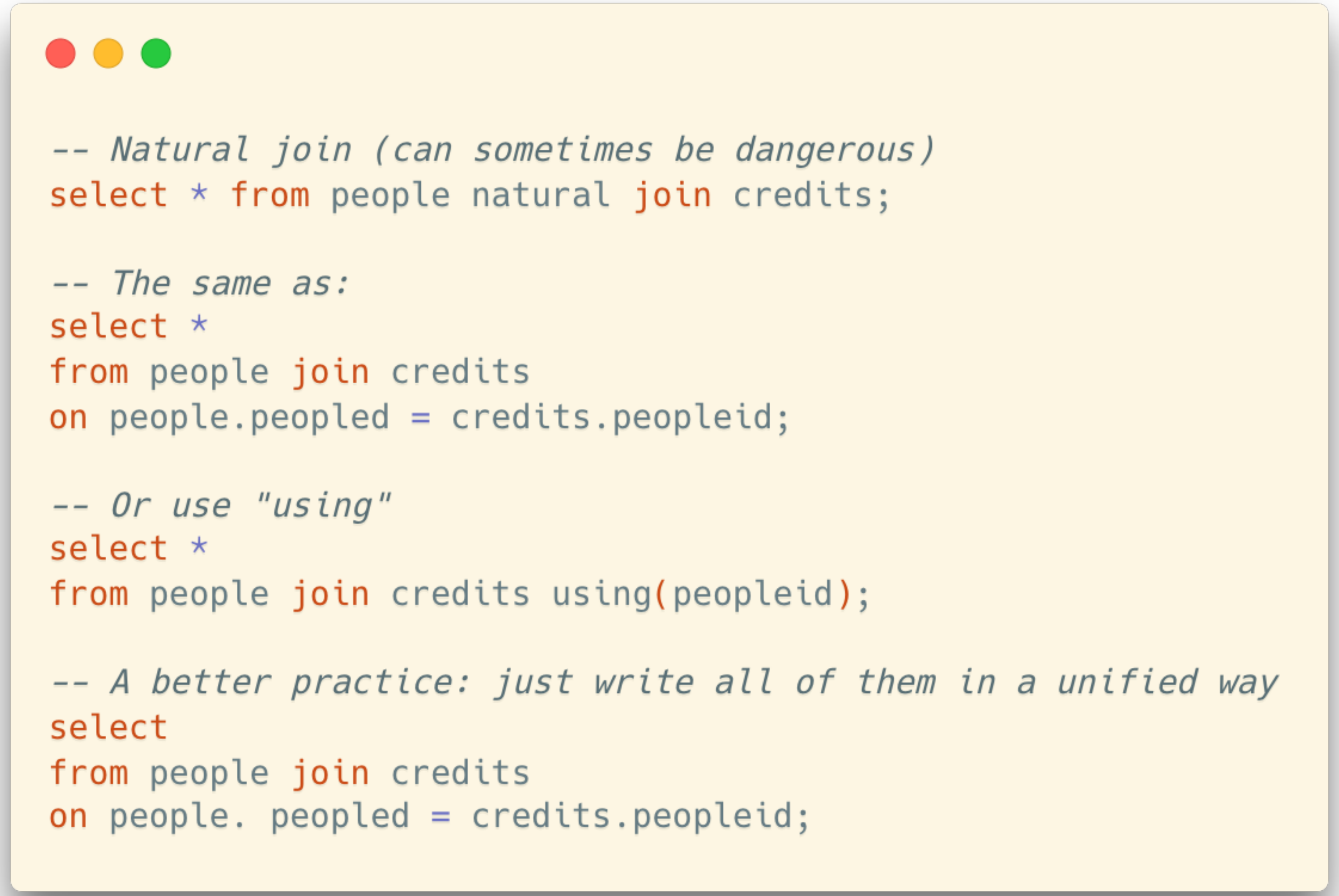


```sql
select * from people natural join credits;

-- The same as:
select
from people join credits
on people.peopleid = credits.peopleid;

-- Or use "using"
select *
from people join credits using(peopleid);
```

(Maybe) A Good Practice in Writing Queries

- It is preferred not to depend on how database designers name their columns
  - It can be a good practice to use a single (and sometimes straightforward) syntax that works all the time

Keep it simple stupid



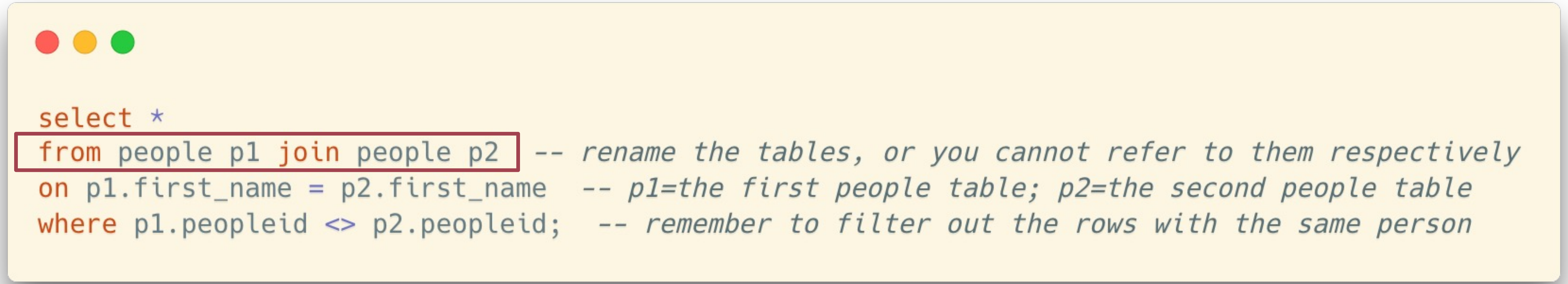
```sql
-- Natural join (can sometimes be dangerous)
select * from people natural join credits;

-- The same as:
select *
from people join credits
on people.peopled = credits.peopleid;

-- Or use "using"
select *
from people join credits using(peopleid);

-- A better practice: just write all of them in a unified way
select
from people join credits
on people. peopled = credits.peopleid;
```

Self Join

- Join the same table together
  - For example: How can we find all the pairs of people with the same first name?

Self Join

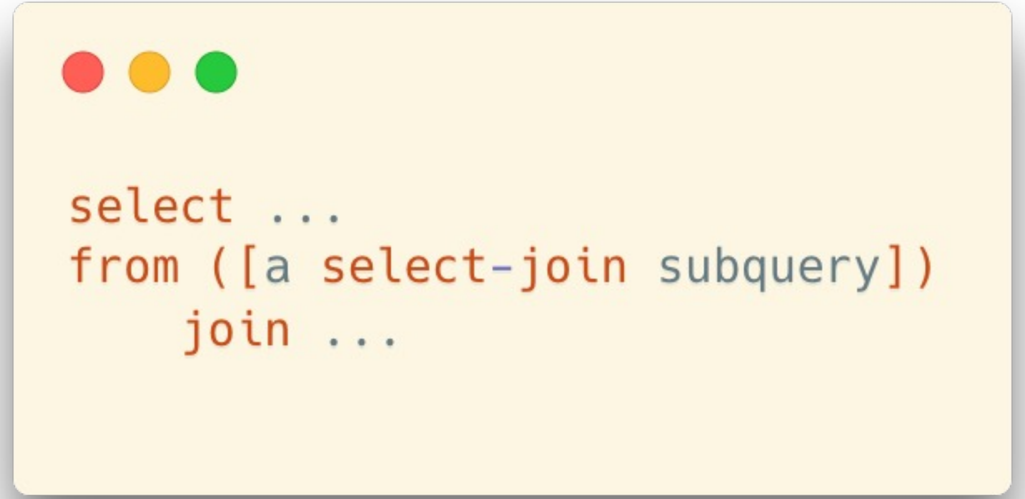
- Join the same table together
  - For example: How can we find all the pairs of people with the same first name?



```
select *
from people p1 join people p2 -- rename the tables, or you cannot refer to them respectively
on p1.first_name = p2.first_name  -- p1=the first people table; p2=the second people table
where p1.peopleid <> p2.peopleid;  -- remember to filter out the rows with the same person
```

Join in a Subquery

- A join can as well be applied to a subquery seen as a virtual table
  - … as long as the result of this subquery is a valid relation in Codd's sense



```
select ...
from ([a select-join subquery])
    join ...
```

Chaining Joins Together

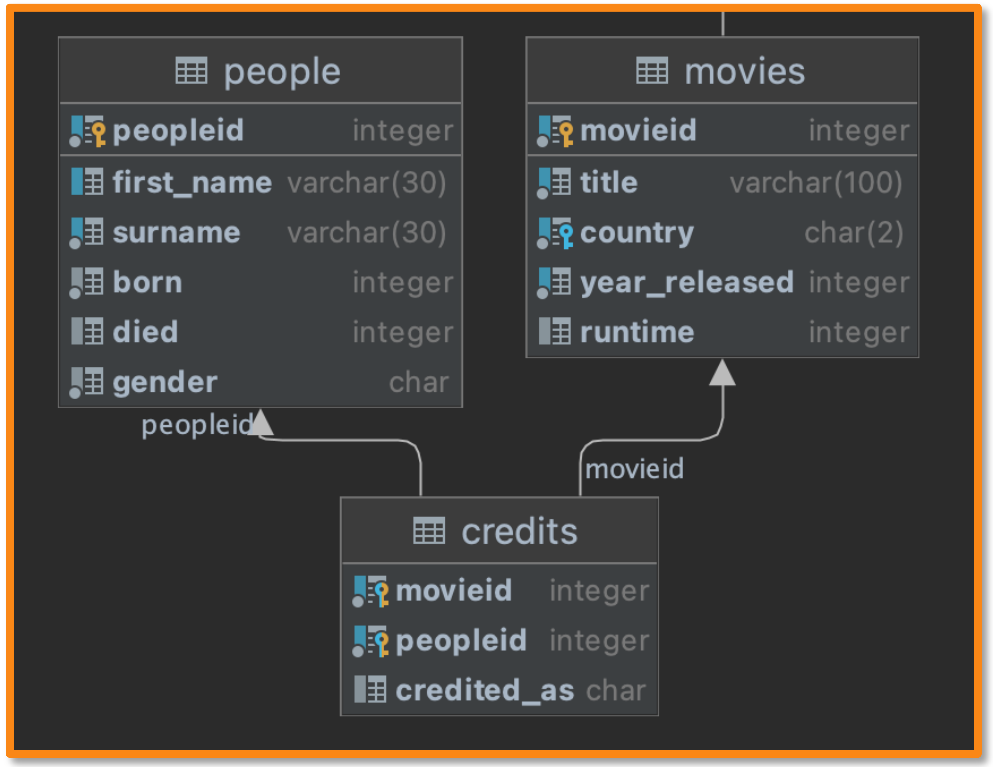
- We can also chain joins the same way we chain filtering conditions with AND.
  - Joins between 10 or 15 tables aren’t uncommon, and queries generated by programs often do much worse.

Chaining Joins Together

- We can also chain joins the same way we chain filtering conditions with AND.
  - Joins between 10 or 15 tables aren’t uncommon, and queries generated by programs often do much worse.
  - Example: Show names of actors and directors for Chinese movies

Chaining Joins Together

- We can also chain joins the same way we chain filtering conditions with AND.
  - Joins between 10 or 15 tables aren't uncommon, and queries generated by programs often do much worse.
  - Example: Show names of actors and directors for Chinese movies



people

peopleid	integer
first_name	varchar(30)
surname	varchar(30)
born	integer
died	integer
gender	char

movies

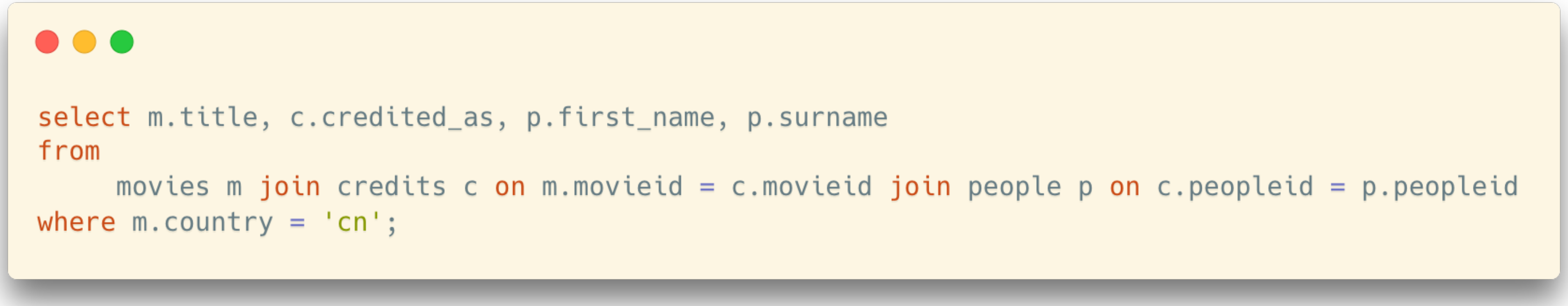
movieid	integer
title	varchar(100)
country	char(2)
year_released	integer
runtime	integer

credits

movieid	integer
peopleid	integer
credited_as	char

Chaining Joins Together

- We can also chain joins the same way we chain filtering conditions with AND.
  - Joins between 10 or 15 tables aren’t uncommon, and queries generated by programs often do much worse.
  - Example: Show names of actors and directors for Chinese movies



```
select m.title, c.credited_as, p.first_name, p.surname
from
     movies m join credits c on m.movieid = c.movieid join people p on c.peopleid = p.peopleid
where m.country = 'cn';
```