Principles of Database Systems (CS307)

Lecture 4: Intermediate and Advanced SQL

Zhong-Qiu Wang

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

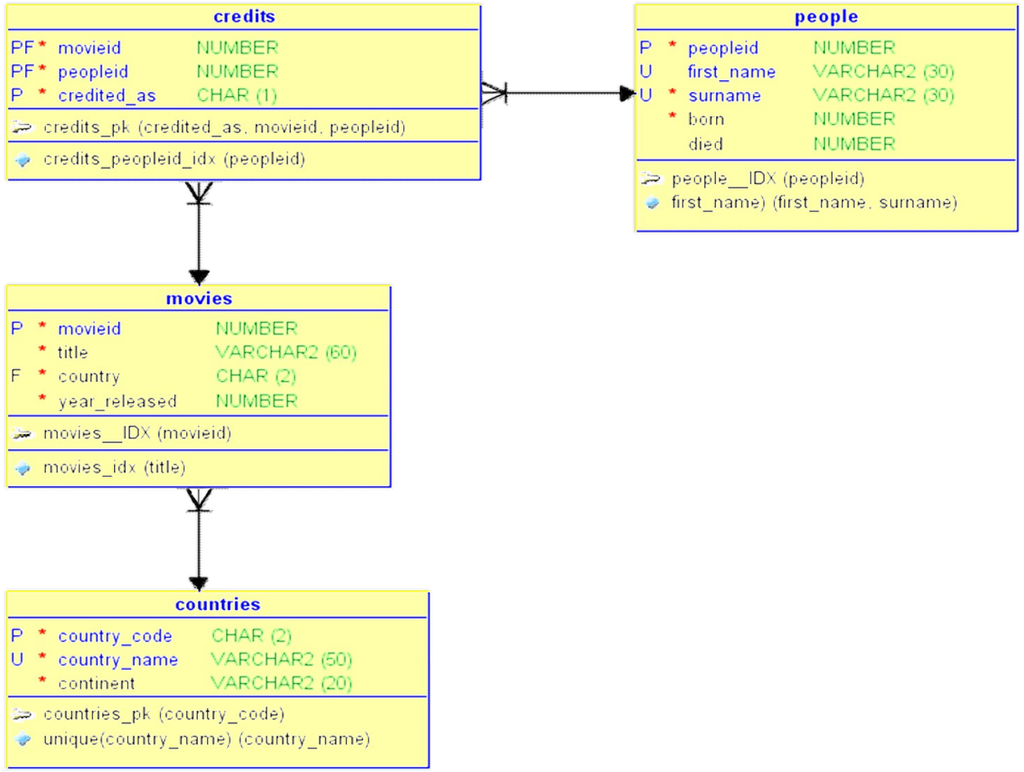
- Most contents are from slides made by Stéphane Faroult and the authors of Database System Concepts (7th Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.
- The slides are largely based on the slides provided by Dr. Yuxin Ma

Announcements

- First homework is out, due date: 9^{th} Oct, next Wednesday

Entity and Relationship

- Starring -> Actor table
- Country -> Country and Region table
  - You can also link the movies with corresponding actors, countries/regions, etc.
- Entity Relationship Diagram (E/R Diagram, ER Diagram, ERD)
  - A way of representing entity tables and their relationships (relationship tables)
  - Connect tables via foreign keys and relationship tables



Create Tables

- An SQL relation is defined using the create table command:

```
create table r (
    A1 D1, A2 D2, ..., An Dn,
    (integrity-constraint1),
    ...,
    (integrity-constraintk)
)
```

  - Relation schema: r is the name of the relation
  - Attribute: each Ai is an attribute name in the schema of relation r
  - Data type: Di is the data type of values in the domain of attribute Ai
  - Constraints: not null, unique, primary key, check function, referential integrity & foreign key

Select

- `select * from` [tablename]
  - The `select` clause lists the attributes desired in the result of a query
  - To display the full content of a table, you can use select *
    - * : all columns

```
select A1, A2, ..., An
 from r1, r2, ..., rm
       where P
```

Some Functions

- When to use functions
  - An example of showing a result that isn't stored as such is computing an age
    - You should never store an age; it changes all the time!
    - If you want to display the age of people who are alive, you must compute their age by subtracting the year when they were born from the current year.

  - In the table people:
    - Alive – died is `null`
    - Age: `<this year> - born`

```
select peopleid, surname,
       date_part('year', now()) - born as age
from people
where died is null;
```

7	7	Caroline	Aaron	1952	<null>	F
8	8	Quinton	Aaron	1984	<null>	M
9	9	Dodo	Abashidze	1924	1990	M

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;
```

23 rows

	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!

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;
```

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

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"

Caution: The table `movie` must be a relation (no duplicated movie records)
- … or, the counting result will not reflect the actual number of movies

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

```sql
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

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

- These aggregate functions are supported in almost every DBMS
  - Most DBMS implement other more advanced functions
  - Some functions work with any datatype, others only work with numerical columns

  - It is strongly recommended to refer to user manual for details
    - For example, SQLite doesn't have `stddev()`

Aggregate Functions

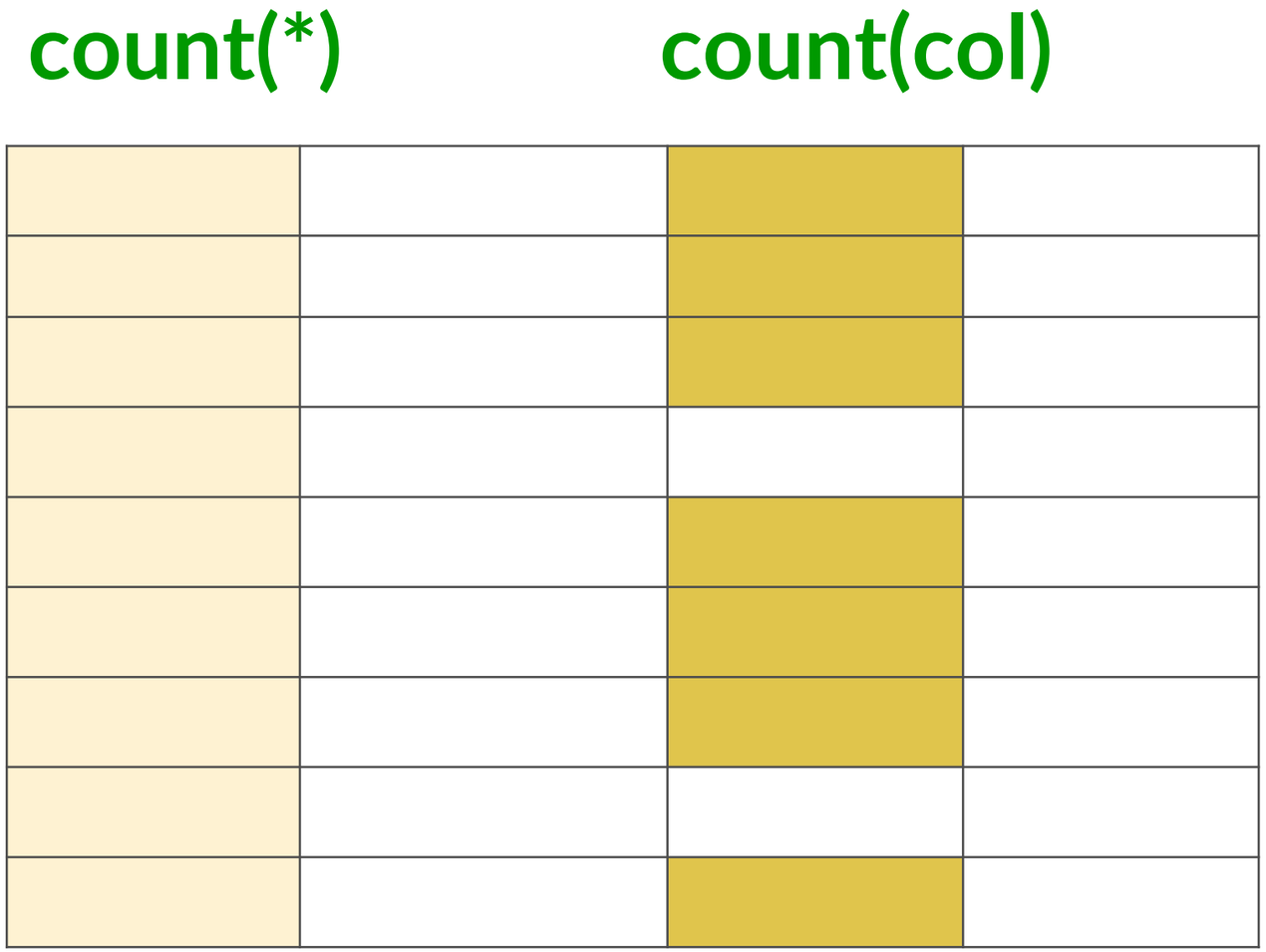
- Example: **Earliest release year by country?**

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

- Such a query answers the question
  - 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



- 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)
```

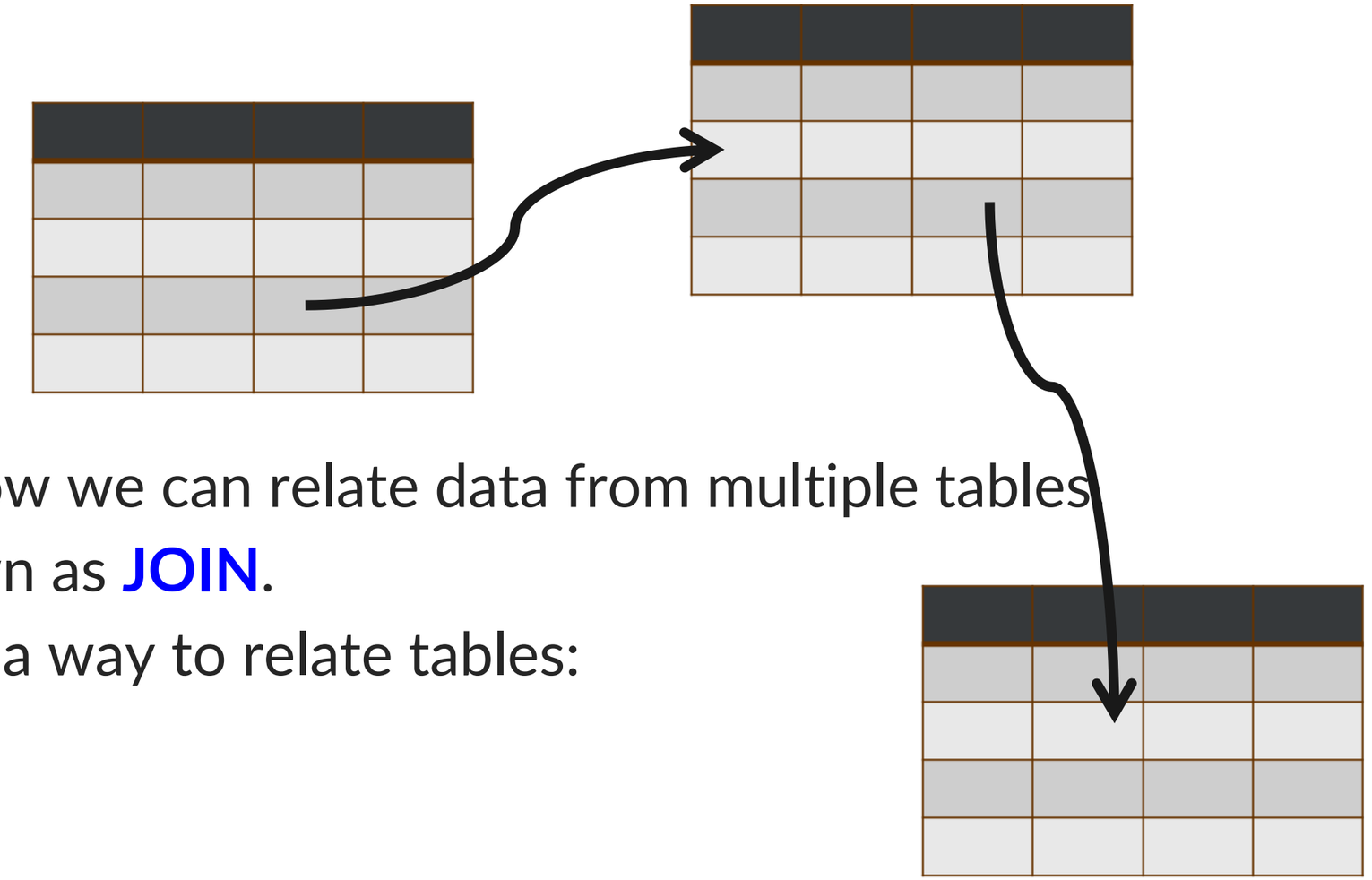
Aggregate Functions

- 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
```

Join

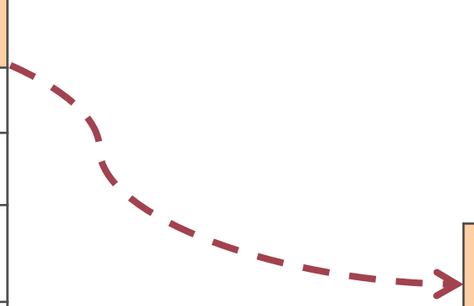
Retrieving Data from Multiple Tables



- 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

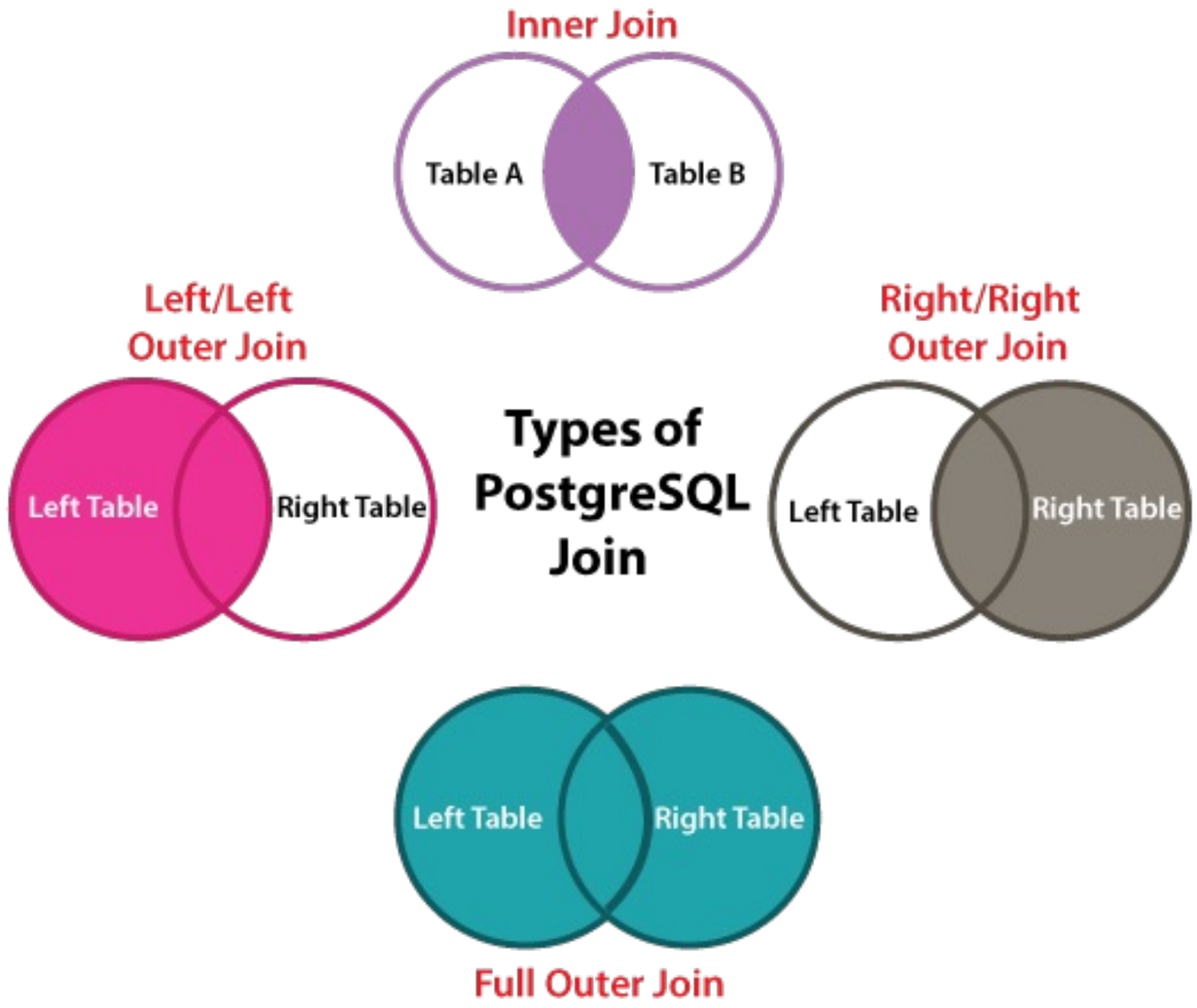
`movies join countries`

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.

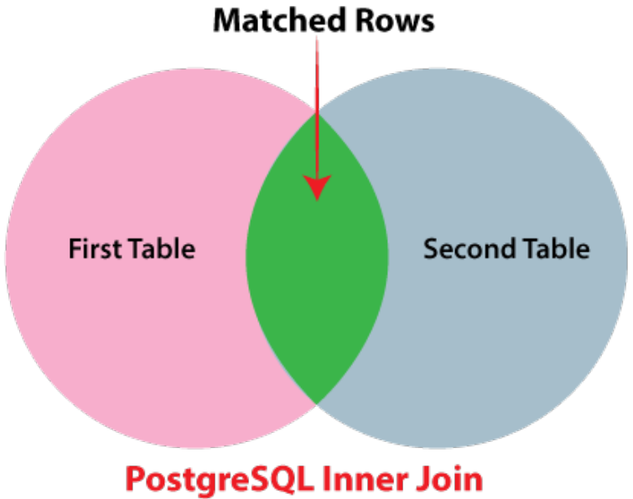
Inner and Outer Joins

- So far, we only join the rows with matching values on the corresponding columns
- There are more things we can do with join



Inner and Outer Joins

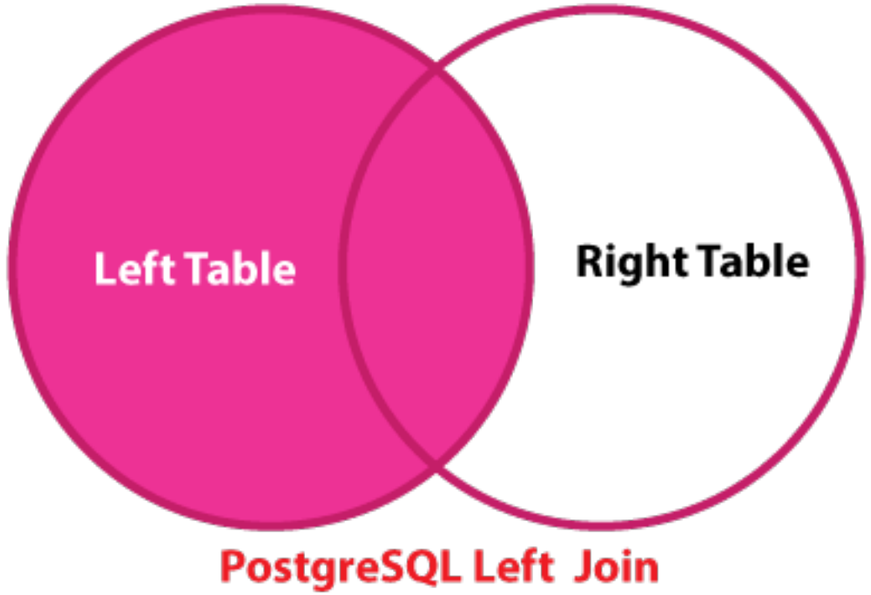
- Inner join
  - Join type in default
  - All examples so far are inner joins
  - Only select joined rows with matching values
    - Typically, a subset of rows in left table are joined with a subset of rows in right table
    - Some rows in left table may NOT be joined with any rows in right table
      - E.g., for movies released by a country that is not in the countries table




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

Inner and Outer Joins

- Left outer join
  - All the matching rows will be selected
  - … and the rows in **left table** with **no matches** will be **selected** as well




```
select columns
from table1
LEFT [OUTER] join table2
on table1.column = table2.column;
```

Inner and Outer Joins

- Left outer join
  - Example: there is a movie in 2018 where there is no credit information
    - #9203 (A Wrinkle in Time)

```
select * from movies where movieid = 9203;
```

	movieid	title	country	year_released	runtime
1	9203	A Wrinkle in Time	us	2018	109

Inner and Outer Joins

- Left outer join
  - Example: there is a movie in 2018 where there is no credit information
    - #9203 (A Wrinkle in Time)
    - Inner join of all 2018 movies will not show any matching results for that movie

```
select *
from movies m join credits c
on m.movieid = c.movieid
where m.year_released = 2018;
```

	m.movieid	title	country	year_released	runtime	c.movieid	peopleid	credited_as
1	8987	Red Sparrow	us	2018	145	8987	4062	A
2	8987	Red Sparrow	us	2018	145	8987	6711	A
3	8987	Red Sparrow	us	2018	145	8987	8308	D
4	8987	Red Sparrow	us	2018	145	8987	8310	A
5	8987	Red Sparrow	us	2018	145	8987	11247	A
6	8987	Red Sparrow	us	2018	145	8987	12048	A
7	8987	Red Sparrow	us	2018	145	8987	13071	A
8	8988	Ready Player One	us	2018	0	8988	2934	A
9	8988	Ready Player One	us	2018	0	8988	9819	A
10	8988	Ready Player One	us	2018	0	8988	9971	A
11	8988	Ready Player One	us	2018	0	8988	11390	A
12	8988	Ready Player One	us	2018	0	8988	12758	A
13	8988	Ready Player One	us	2018	0	8988	13421	A
14	8988	Ready Player One	us	2018	0	8988	13850	D
15	8989	Guernsey	gb	2018	0	8989	1864	A
16	8989	Guernsey	gb	2018	0	8989	5280	A
17	8989	Guernsey	gb	2018	0	8989	6523	A
18	8989	Guernsey	gb	2018	0	8989	6836	A
19	8989	Guernsey	gb	2018	0	8989	10643	D
20	8989	Guernsey	gb	2018	0	8989	11261	A
21	8989	Guernsey	gb	2018	0	8989	11733	A
22	8989	Guernsey	gb	2018	0	8989	15708	A
23	8990	A Star Is Born	us	2018	0	8990	2431	A
24	8990	A Star Is Born	us	2018	0	8990	2759	A
25	8990	A Star Is Born	us	2018	0	8990	2939	A
26	8990	A Star Is Born	us	2018	0	8990	2939	D
27	8990	A Star Is Born	us	2018	0	8990	4158	A
28	8990	A Star Is Born	us	2018	0	8990	8105	A
29	8992	Mary Queen of Scots	us	2018	0	8992	272	A
30	8992	Mary Queen of Scots	us	2018	0	8992	2879	A
31	8992	Mary Queen of Scots	us	2018	0	8992	3056	A
32	8992	Mary Queen of Scots	us	2018	0	8992	3365	A
33	8992	Mary Queen of Scots	us	2018	0	8992	8892	A
34	8992	Mary Queen of Scots	us	2018	0	8992	11371	A
35	8992	Mary Queen of Scots	us	2018	0	8992	12435	A
36	8992	Mary Queen of Scots	us	2018	0	8992	12563	A
37	8992	Mary Queen of Scots	us	2018	0	8992	12636	D
38	8993	The Girl in the Spider's Web	se	2018	0	8993	4696	A
39	8993	The Girl in the Spider's Web	se	2018	0	8993	5543	A
40	8993	The Girl in the Spider's Web	se	2018	0	8993	16462	D
41	9202	Black Panther	us	2018	134	9202	3933	A
42	9202	Black Panther	us	2018	134	9202	5588	A
43	9202	Black Panther	us	2018	134	9202	15870	A

Inner and Outer Joins

- Left outer join
  - Example: there is a movie in 2018 where there is no credit information
    - #9203 (A Wrinkle in Time)
    - Inner join of all 2018 movies will not show any matching results for that movie
    - Left (outer) join can give you a row for the movie (in the left table) where all right-table columns are null

```
select * from movies m left join credits c on m.movieid = c.movieid
where m.year_released = 2018;
```

	m.movieid	title	country	year_released	runtime	c.movieid	peopleid	credited_as
41	9202	Black Panther	us	2018	134	9202	3933	A
42	9202	Black Panther	us	2018	134	9202	5588	A
43	9202	Black Panther	us	2018	134	9202	15870	A
44	9203	A Wrinkle in Time	us	2018	109	<null>	<null>	<null>

Inner and Outer Joins

- Left outer join
  - Why should we show the rows in the left table with no matches?
  - Scenario: Movie Website (e.g., douban)
    - We cannot just ignore the movies with no credit information
    - Instead, we should list them, and show that credit information is missing
    - We can distinguish them by checking the values in the right-table columns
      - Via a simple query

Inner and Outer Joins

- Left outer join
  - Another example: let's count how many movies we have per country (again)

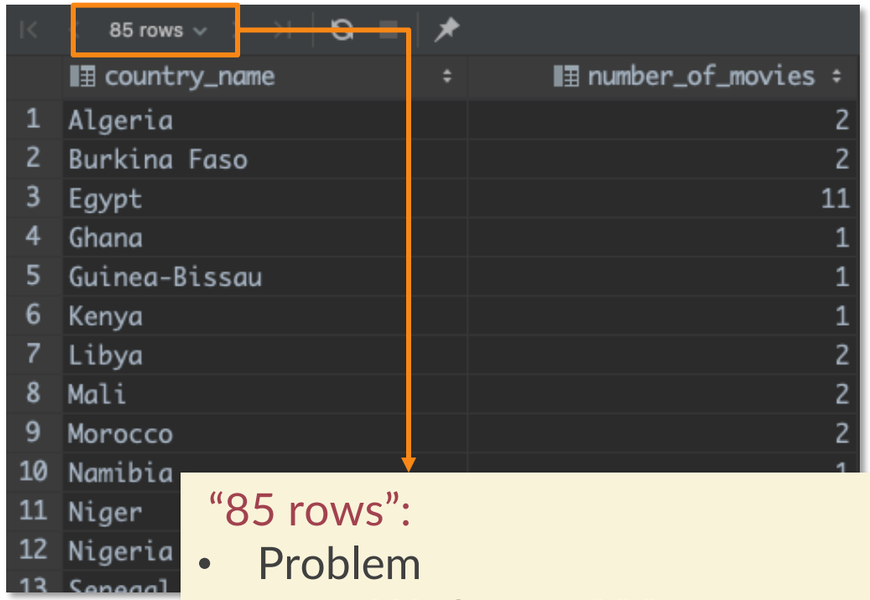
```sql
select country,
       count(*) number_of_movies
from movies
group by country;
```

	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

Inner and Outer Joins

- Left outer join
  - Another example: let's count how many movies we have per country (again)
    - Want to show country name instead of country code
      - Need to use join

```
select c.country_name, number_of_movies
from countries c join (
    select country as stat_country_code,
        count(*) as number_of_movies
    from movies
    group by country
) stat
on c.country_code = stat_country_code;
```



"85 rows":

- Problem
  - We have ~200 countries in total
  - How can we show the other countries?

Inner and Outer Joins

- Left outer join
  - All countries are here now
  - How to replace nulls?

```
select c.country_name, number_of_movies
from countries c left join (
    select country as stat_country_code,
        count(*) as number_of_movies
    from movies
    group by country
) stat
on c.country_code = stat_country_code;
```

185 rows

	country_name	number_of_movies
1	Algeria	2
2	Angola	<null>
3	Benin	<null>
4	Botswana	<null>
5	Burkina Faso	2
6	Burundi	<null>
7	Cameroon	<null>
8	Central African Republic	<null>
9	Chad	<null>
10	Comoros	<null>
11	Congo Brazzaville	<null>
12	Congo Kinshasa	<null>
13	Cote d'Ivoire	<null>
14	Djibouti	<null>
15	Egypt	11
16	Equatorial Guinea	<null>
17	Eritrea	<null>

Inner and Outer Joins

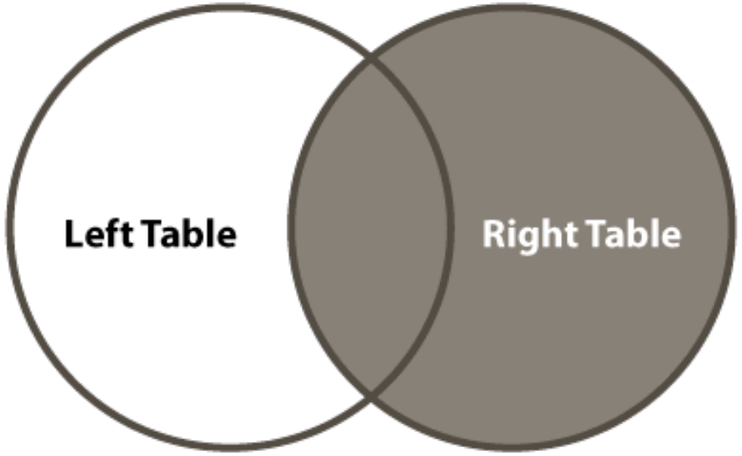
- Left outer join
  - All countries are here now
  - How to replace nulls?
    - Add another CASE condition

```
select c.country_name,
       case
           when stat.number_of_movies is null then 0
           else stat.number_of_movies
       end
from countries c left join (
    select country as stat_country_code,
        count(*) as number_of_movies
    from movies
    group by country
) stat
on c.country_code = stat_country_code;
```

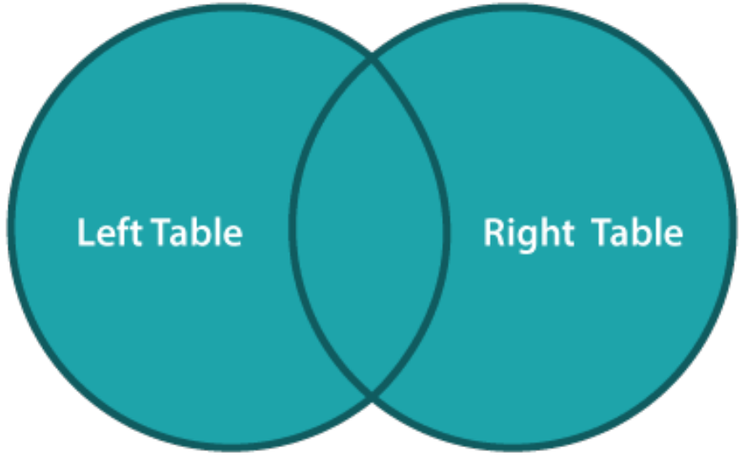
	country_name	number_of_movies
1	Algeria	2
2	Angola	0
3	Benin	0
4	Botswana	0
5	Burkina Faso	2
6	Burundi	0
7	Cameroon	0
8	Central African Republic	0
9	Chad	0
10	Comoros	0
11	Congo Brazzaville	0
12	Congo Kinshasa	0
13	Cote d'Ivoire	0
14	Djibouti	0
15	Egypt	11
16	Equatorial Guinea	0
17	Eritrea	0
18	Ethiopia	0

Inner and Outer Joins

- Right outer join, full outer join
  - Books always refer to three kinds of outer joins. Only one is useful and we can forget about anything but the LEFT OUTER JOIN
    - A right outer join can ALWAYS be rewritten as a left outer join
      - by swapping the order of tables in the join list
    - A full outer join is seldom used



PostgreSQL Right Join



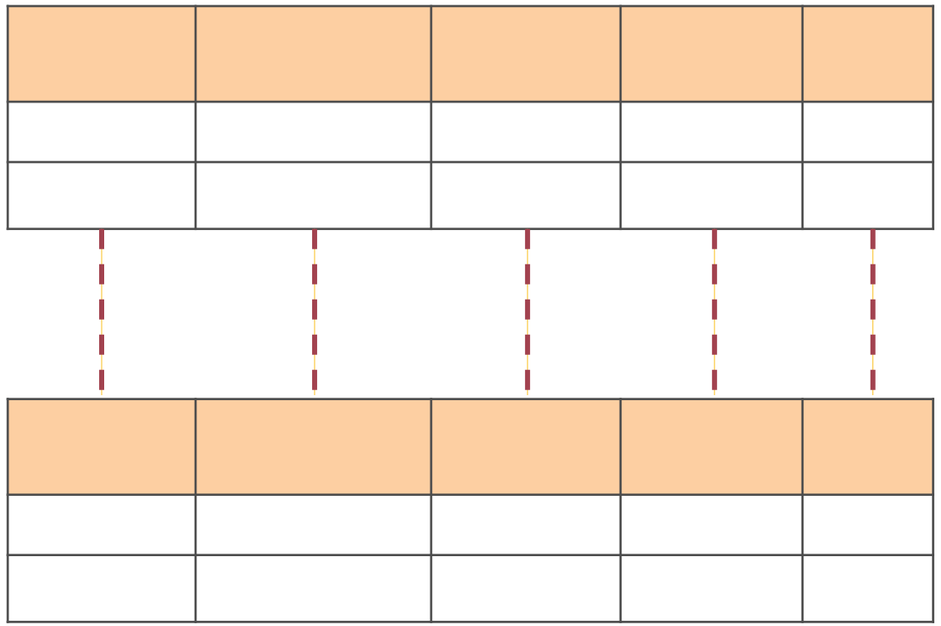
PostgreSQL Full Outer Join

Set Operators

Set Operators

- Union
  - Takes two result sets and combines them into a single result set

- Union requires two (commonsensical) conditions:
  - They must return the same number of columns
  - The data types of corresponding columns must match



Set Operators

- Union
  - Example: Stack US and GB movies together

```sql
select movieid, title, year_released, country
from movies
where country = 'us'
  and year_released between 1940 and 1949

union

select movieid, title, year_released, country
from movies
where country = 'gb'
  and year_released between 1940 and 1949;
```

	movieid	title	year_released	country
1	3840	The Secret Life of Walter Mitty	1947	us
2	678	The Ox-Bow Incident	1943	us
3	3174	The Red House	1947	us
4	5152	Minesweeper	1943	us
5	1487	Kiss of Death	1947	us
6	3408	Ministry of Fear	1944	us
7	2543	The Way to the Stars	1945	gb
8	5341	All Through the Night	1942	us
9	1435	They Live by Night	1948	us
10	2644	Criminal Court	1946	us
11	7250	The Seventh Veil	1945	gb
12	7341	Mr. Lucky	1943	us

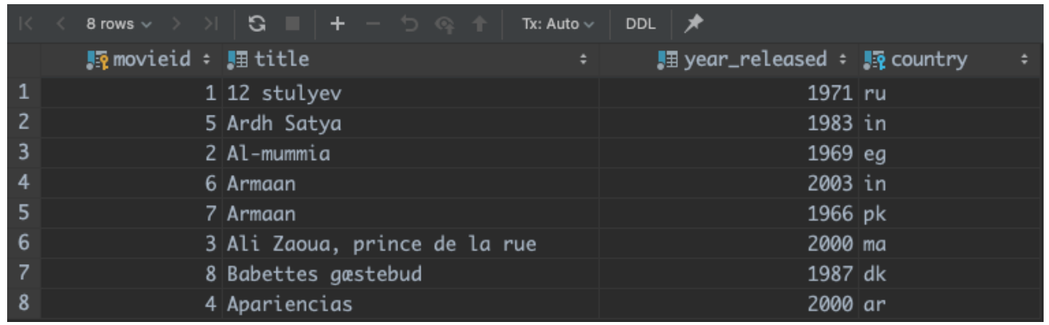
Set Operators

- Union
  - Example: combine movies from two tables, one for standard accounts and one for VIP accounts
    - We don’t want to miss the “standard movies” for the VIP accounts

Set Operators

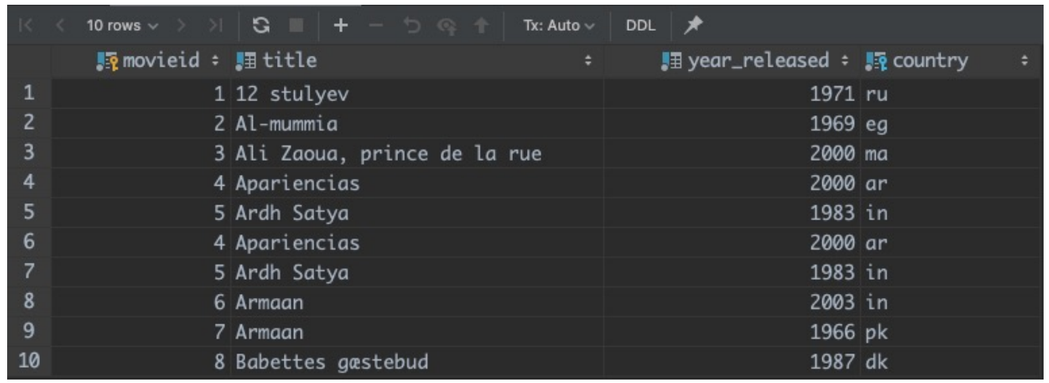
- Union
  - Warning: `union` removes duplicated rows
    - Instead, you can use `union all`

```
(select movieid, title, year_released, country
from movies limit 5 offset 0)
union
(select movieid, title, year_released, country
from movies limit 5 offset 3);
```



	movieid	title	year_released	country
1	1	12 stulyev	1971	ru
2	5	Ardh Satya	1983	in
3	2	Al-mummia	1969	eg
4	6	Armaan	2003	in
5	7	Armaan	1966	pk
6	3	Ali Zaoua, prince de la rue	2000	ma
7	8	Babettes gæstebud	1987	dk
8	4	Apariencias	2000	ar

```
(select movieid, title, year_released, country
from movies limit 5 offset 0)
union all
(select movieid, title, year_released, country
from movies limit 5 offset 3);
```



	movieid	title	year_released	country
1	1	12 stulyev	1971	ru
2	2	Al-mummia	1969	eg
3	3	Ali Zaoua, prince de la rue	2000	ma
4	4	Apariencias	2000	ar
5	5	Ardh Satya	1983	in
6	4	Apariencias	2000	ar
7	5	Ardh Satya	1983	in
8	6	Armaan	2003	in
9	7	Armaan	1966	pk
10	8	Babettes gæstebud	1987	dk

Set Operators

- Intersect
  - A `intersect` B
  - Return the rows that appears in both table A and B
- Except
  - A `except` B
  - Return the rows that appear in table A but not in B
  - Sometimes written as `minus` in some database products
- However, they are not used as much as union

Subquery

Subquery

- We have used subqueries after the keyword `from` before
  - … in order to build queries upon a query result

```
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;
```

Subquery

- We have used subqueries after the keyword `from` before
  - … in order to build queries upon a query result

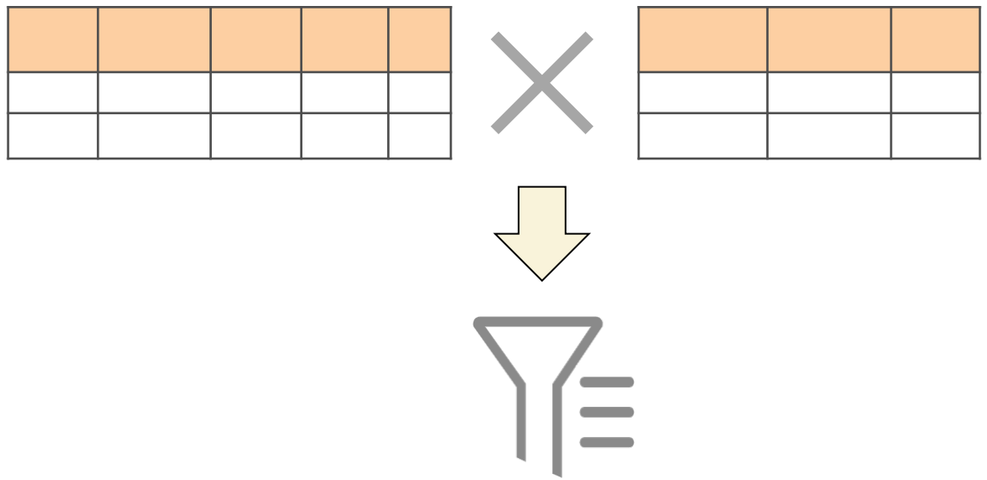
- And, we can add subqueries after `select` and `where` as well

```
select A1, A2, ..., An
 from r1, r2, ..., rm
      where P
```

Subquery after Select

- Example: show titles, released years, and country names for non-US movies
  - Solution 1: Join

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



Subquery after Select

- Example: show titles, released years, and country names for non-US movies
  - Solution 2: Nested selection

```
select m.title,
       m.year_released,
       m.country
from movies m
where m.country <> 'us';
```

... still a country code though

- How can we replace it with the country name?

Subquery after Select

- Example: show titles, released years, and country names for non-US movies
  - Solution 2: Nested selection

```
select m.title,
       m.year_released,
       m.country
from movies m
where m.country <> 'us';
```

```
select m.title,
       m.year_released,
       (
           select c.country_name
           from countries c
           where c.country_code = m.country
       ) country_name
from movies m
where m.country <> 'us';
```

A subquery after select:

- For each selected row in the outer query, find the corresponding country name in the `countries` table

Subquery after Where

- Recall: the `in ()` operator
  - It can be used as the equivalent for a series of equalities with OR
  - It makes a comparison clearer than a parenthesized expression

```
where (country = 'us' or country = 'gb')
  and year_released between 1940 and 1949

where country in ('us', 'gb')
  and year_released between 1940 and 1949
```

Subquery after Where

- `in()` is far more powerful than this
  - Between the parentheses, we can put, not only an explicit list of values, but also an implicit list of values generated by a query

```
in (select col
    from ...
    where ...)
```

Subquery after Where

- Example: Select all European movies
  - How can we specify the filtering condition?

```
select country,
       year_released,
       title
from movies
where [?]
```

Subquery after Where

- Example: Select all European movies
  - A horrible solution: list all European countries with `or`

```
select country,
       year_released,
       title
from movies
where country = 'fr' or country = 'de' or ...
```

Subquery after Where

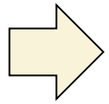
- Example: Select all European movies
  - A (slightly better) solution: list all European countries in an `in` operator

```
select country,
       year_released,
       title
from movies
where country in('fr', 'de', ...)
```

Subquery after Where

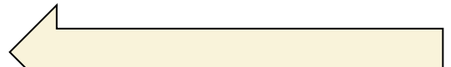
- Example: Select all European movies
  - A (slightly better) solution: list all European countries in an `in` operator

```
select country,
       year_released,
       title
from movies
where country in('fr', 'de', ...)
```



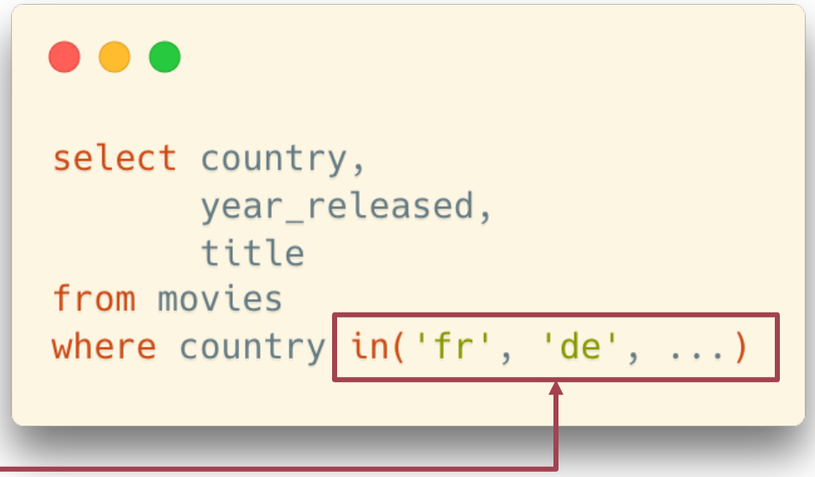
```
select * from countries where continent = 'EUROPE';
```

40 rows



Subquery after Where

- Example: Select all European movies
  - A proper solution: (dynamically) fill in the list of country codes in an `in` operator



The same results (if you fill in all European country codes on the right side)

- But you can automatically generate this list
- Especially useful when the table in the subquery changes often

Subquery after Where

- Some DBMS (Oracle, DB2, PostgreSQL with some twisting) even allow comparing a set of column values (i.e., tuple) to the result of a subquery

```
(col1, col2) in
        (select col3, col4
         from t
         where …)
```

Subquery after Where

- Some important points for `in()`
  - `in()` test membership in an implicit set
    - `in('cn', 'us', 'cn', 'us', 'us')` is equal to `in('cn', 'us')`

Subquery after Where

- Some important points for `in()`
  - `in()` test membership in an implicit set
    - `in('cn', 'us', 'cn', 'us', 'us')` is equal to `in('cn', 'us')`
  - null values in `in()`
    - Be extremely cautious if you are using `not in(...)` with a null value in it

Subquery after Where

- Some important points for `in()`
  - `in()` test membership in an implicit set
    - `in('cn', 'us', 'cn', 'us', 'us')` is equal to `in('cn', 'us')`
  - null values in `in()`
    - Be extremely cautious if you are using `not in(...)` with a null value in it

```
value not in(2, 3, null)
⇒not (value=2 or value=3 or value=null)
⇒value<>2 and value<>3 and value<>null
⇒false  -- always false or null, never true
```

... however, `value=null` and `value<>null` are always not true:

- We should use `is [not] null` instead

The `not in()` expression always returns false or null, and hence no row will be selected

```
not(A or B or C) -> (not A) and (not B) and (not C)
```

NULL

Expressions with NULL Values

- Most expressions with `NULL` will be evaluated into `NULL`
  - Arithmetic operations:

```
col+NULL -> NULL
col-NULL -> NULL
col*NULL -> NULL
col/NULL -> NULL

...
```

  - Comparison operations:

```
(col > NULL) -> NULL
(col = NULL) -> NULL

...
```

Expressions with NULL Values

- Most expressions with `NULL` will be evaluated into `NULL`
  - But, there are some conditions where the values are not `NULL`

```
TRUE and NULL -> NULL
FALSE and NULL -> FALSE

TRUE or NULL -> TRUE
FALSE or NULL -> NULL
```

Logical operators (or, and):

- Three-valued logic (true, false, and unknown)

More on this: Three-valued logic and its application in SQL
`https://en.wikipedia.org/wiki/Three-valued_logic#SQL`

```
col is NULL -> True or False
```

The way we use to check a `NULL` value: use `is`, not `=`

Recall: Subquery after Where

- Some important points for `in()`
  - `in()` means an implicit distinct in the subquery
    - `in('cn', 'us', 'cn', 'us', 'us')` is equal to `in('cn', 'us')`
  - null values in `in()`
    - Be extremely cautious if you are using `not in(...)` with a null value in it

```
value not in(2, 3, null)
⇒not (value=2 or value=3 or value=null)
⇒value<>2 and value<>3 and value<>null
⇒false  -- always false or null, never true
```

- If value is 2, the result is:
  `FALSE and TRUE and NULL -> FALSE`
- if value is 5, the result is:
  `TRUE and TRUE and NULL -> NULL`
- if value is NULL, the result is :
  `NULL and NULL and NULL -> NULL`

Ordering

Ordering in SQL

- `order by`
  - A simple expression in SQL to order a result set
  - It comes at the end of a query
    - ... and, you can have it in subqueries, definitely
  - Followed by a list of columns used as sort columns

```
select title, year_released
from movies
where country = 'us'
order by year_released;
```

	title	year_released
1	Ben Hur	1907
2	The Lonely Villa	1909
3	From the Manger to the Cross	1912
4	Falling Leaves	1912
5	Traffic in Souls	1913
6	At Midnight	1913
7	Lime Kiln Field Day	1913
8	The Sisters	1914
9	The Only Son	1914
10	Tess of the Storm Country	1914
11	Under the Gaslight	1914
12	Brute Force	1914
13	The Wishing Ring: An Idyll of Old England	1914

Ordering in SQL

- We can apply “order by” to any result set, no matter how difficult the query is

```
select m.title,
       m.year_released
from movies m
where m.movieid in
  (select distinct c.movieid
   from credits c
        inner join people p
        on p.peopleid = c.peopleid
   where c.credited_as = 'A'
     and p.born >= 1970)
order by m.year_released
```

	title	year_released
1	Snehaseema	1954
2	Nairu Pidicha Pulivalu	1958
3	Mudiyanaya Puthran	1961
4	Puthiya Akasam Puthiya Bhoomi	1962
5	Doctor	1963
6	Aadyakiranangal	1964
7	Odayil Ninnu	1965
8	Adimakal	1969
9	Karakanakadal	1971
10	Ghatashraddha	1977
11	Kramer vs. Kramer	1979
12	The Champ	1979
13	The Shining	1980

Ordering in SQL

- Ordering with joins
  - We can sort by any column of any table in the join
    - Remember the super wide table with all the columns from all tables involved

```sql
select c.country_name,
       m.title,
       m.year_released
from movies m
     inner join countries c
     on c.country_code = m.country
where m.movieid in
  (select distinct c.movieid
   from credits c
        inner join people p
        on p.peopleid = c.peopleid
   where c.credited_as = 'A'
     and p.born >= 1970)
order by m.year_released
```

	country_name	title	year_released
1	India	Snehaseema	1954
2	India	Nairu Pidicha Pulivalu	1958
3	India	Mudiyanaya Puthran	1961
4	India	Puthiya Akasam Puthiya Bhoomi	1962
5	India	Doctor	1963
6	India	Aadyakiranangal	1964
7	India	Odayil Ninnu	1965
8	India	Adimakal	1969
9	India	Karakanakadal	1971
10	India	Ghatashraddha	1977
11	United States	Kramer vs. Kramer	1979
12	United States	The Champ	1979
13	United States	The Shining	1980

Advanced Ordering

- Multiple columns
  - The result set will be order by `col1` first
  - Rows with the same value on `col1` will be ordered by `col2`

```
order by col1, col2, ...
```

- Ascending or descending order
  - Add `desc` or `asc` after the column
    - `asc` is the default option and thus always omitted

```
-- Order col1 descendingly
order by col1 desc

-- Order based on col1 first, then col2.
-- col1 will be in the descending order, col2 ascending.
order by col1 desc, col2 asc, ...
```

Advanced Ordering

- Self-defined ordering
  - Use “`case` … `when`” in `order by` to define criteria on how to order the rows

```
select * from credits
order by
    case credited_as
        when 'D' then 1
        when 'A' then 2
    end desc;
```

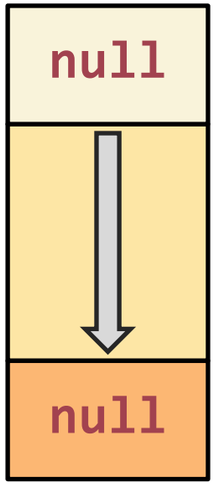
- In ASIC code table, ‘A’ is smaller than ‘D’
- Use case command after order by to assign larger values to ‘A’

Data Types in Ordering

- Ordering depends on the data type
  - Strings: alphabetically
  - Numbers: numerically
  - Dates and times: chronologically

Data Types in Ordering

- What about `NULL`?
  - It is implementation-dependent
  - SQL Server, MySQL and SQLite:
    - “nothing” is smaller than everything
  - Oracle and PostgreSQL:
    - “nothing” is greater than anything



Ordering in Text Data

- Remember, we have many different languages other than English
  - "Alphabetical order" in different languages means different things
    - Mandarin: Pinyin? Number of strokes?
    - Swedish and German
      - "ö" is considered the last letter in Swedish, while in German it is ordered after "o"

Limit and Offset

- Get a slice of the long query result
  - `limit k offset p`
    - Return the top-k rows in the result set after skipping the first p rows
    - `offset` is optional (which means "`offset 0`")
  - Always used together with order by
    - e.g., get the top-k query results under a certain ordering criteria
  - * In some DBMS, the syntax can be different
    - Refer to the user manual for specifics

```
select * from movies
where country = 'us'
order by year_released
limit 10 offset 5
```

```
select * from movies
where country = 'us'
order by year_released
limit 10
```

Window Function

Scalar Functions and Aggregation Functions

- Scalar function
  - Functions that operate on values in the current row

```
round(3.141592, 3)  -- 3.142
trunc(3.141592, 3)  -- 3.141
```

```
upper('Citizen Kane')
lower('Citizen Kane')
substr('Citizen Kane', 5, 3)  -- 'zen'
trim('   Oops  ')  -- 'Oops'
replace('Sheep', 'ee', 'i')  -- 'Ship'
```

- Aggregation function
  - Functions that operate on sets of rows and return an aggregated value

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

Issues with Aggregate Functions

- A Problem: In aggregated functions, the details of the rows are vanished
  - For example: if we ask for the year of the oldest movie per country
    - We get a country, a year, and nothing else.

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

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

Issues with Aggregate Functions

- A Problem: In aggregated functions, the details of the rows are vanished
  - For example: if we ask for the year of the oldest movie per country
    - We get a country, a year, and nothing else

If we want some more details, like the title of the oldest movies for each country, we can only use self-join to keep the columns

```
select m1.country,
       m1.title,
       m1.year_released
from movies m1
     inner join
     (select country,
        min(year_released) minyear
        from movies
        group by country) m2
on m2.country = m1.country and m2.minyear = m1.year_released
order by m1.country
```

Issues with Aggregate Functions

- A Problem: In aggregated functions, the details of the rows are vanished
  - For example: if we ask for the year of the oldest movie per country
    - We get a country, a year, and nothing else

If we want some more details, like the title of the oldest movies for each country, we can only use self-join to keep the columns

- What if the title of the second oldest movie is what we want?

```
select m1.country,
       m1.title,
       m1.year_released
from movies m1
     inner join
     (select country,
        min(year_released) minyear
        from movies
        group by country) m2
on m2.country = m1.country and m2.minyear = m1.year_released
order by m1.country
```

Issues with Aggregate Functions

- A Problem
  - In aggregated functions, the details of the rows are vanished
  - Another example
    - How can we rank the movies in each country separately based on the released year?
    - “order by” for subgroups
  - One more example
    - Get the top-3 oldest movies for each country
    - How can we implement it?

Window Function

- Syntax:

```
<function> over (partition by <col_p> order by <col_o1, col_o2, …>)
```

  - `<function>`
    - Ranking window functions
    - Aggregation functions
  - `partition by`
    - Specify the column for grouping
  - `order by`
    - Specify the column(s) for ordering in each group

Ranking Window Function

- Example
  - How can we rank the movies in each country separately based on the released year?
    - “order by” for subgroups

```
select country,
       title,
       year_released,
       rank() over (
           partition by country order by year_released
       ) oldest_movie_per_country
from movies;
```

	country	title	year_released	oldest_movie_per_country
1	am	Sayat Nova	1969	1
2	ar	Pampa bárbara	1945	1
3	ar	Albéniz	1947	2
4	ar	Madame Bovary	1947	2
5	ar	La bestia debe morir	1952	4
6	ar	Las aguas bajan turbias	1952	4
7	ar	Intermezzo criminal	1953	6
8	ar	La casa del ángel	1957	7
9	ar	Bajo un mismo rostro	1962	8
10	ar	Las aventuras del Capitán Piluso	1963	9
11	ar	Savage Pampas	1966	10
12	ar	La hora de los hornos	1968	11
13	ar	Waiting for the Hearse	1985	12
14	ar	La historia oficial	1985	12
15	ar	Hombre mirando al sudeste	1986	14

Ranking Window Function

- Example
  - How can we rank the movies in each country separately based on the released year?
    - “order by” for subgroups

```
select country,
       title,
       year_released,
       rank() over (
           partition by country order by year_released
       ) oldest_movie_per_country
from movies;
```

You can also add “desc” here, similar to the “order by” we introduced before

country	title	year_released	oldest_movie_per_country
ar	some title	1948	1
ar	some title	1959	2
ar	some title	1980	3
cn	some title	1987	1
cn	some title	2002	2
uk	some title	1985	1
uk	some title	1992	2
uk	some title	2010	3

`partition by country`

- the selected rows will be grouped (partitioned) according to the values in the column `country`

`rank()`

- A function to say that “I want to order the rows in each partition”
- No parameters in the parentheses

`order by year_released`

- In each group (partition), the rows will be ordered by the column “`year_released`”

Ranking Window Function

- Example
  - How can we rank the movies in each country separately based on the released year?
    - “order by” for subgroups

```
select country,
       title,
       year_released,
       rank() over (
           partition by country order by year_released
       ) oldest_movie_per_country
from movies;
```

country	title	year_released	oldest_movie_per_country
ar	some title	1948	1
ar	some title	1959	2
ar	some title	1980	3
cn	some title	1987	1
cn	some title	2002	2
uk	some title	1985	1
uk	some title	1992	2
uk	some title	2010	3

Note: partition functions can only be used in the `select` clause

- … since it is designed to work on the query result

Ranking Window Function

- Example
  - How can we rank the movies in each country separately based on the released year?
    - “order by” for subgroups

```
select country,
       title,
       year_released,
       rank() over (
           partition by country order by year_released
       ) oldest_movie_per_country
from movies;
```

country	title	year_released	oldest_movie_per_country
ar	some title	1948	1
ar	some title	1959	2
ar	some title	1980	3
cn	some title	1987	1
cn	some title	2002	2
uk	some title	1985	1
uk	some title	1992	2
uk	some title	2010	3

Partitioned by country
- i.e., a country in a group

An order value is computed for each row in a partition.
- Only inside the partition, not across the entire result set

Ranking Window Function

- Why window function, not group by?
  - “Group by” reduces the rows in a group (partition) into one result, which is the meaning of “aggregation”
    - Then, the values in non-aggregating columns are vanished
  - Window functions do not reduce the rows
    - Instead, they attach computed values next to the rows in a group (partition) and keep the details
    - Actually, the partition here means “window”: an affective range for statistics

Ranking Window Function

- Some more ranking window functions
  - Besides `rank()`, we also have `dense_rank()` and `row_number()`
  - The difference is about how they treat rows with the same rank

```
select country,
       title,
       year_released,

       rank() over (
           partition by country order by year_released
       ) rank_result,

       dense_rank() over (
           partition by country order by year_released
       ) dense_rank_result,

       row_number() over (
           partition by country order by year_released
       ) row_number_result
from movies;
```

country	title	year_released	rank_result	dense_rank_result	row_number_result
cn	some title	1948	1	1	1
cn	some title	1959	2	2	2
cn	some title	1959	2	2	3
cn	some title	1987	4	3	4
cn	some title	2002	5	4	5
uk	some title	1985	1	1	1
uk	some title	1992	2	2	2
uk	some title	2010	3	3	3

Aggregation Functions as Window Functions

- `min/max(col)` , `sum(col)`, `count(col)`, `avg(col)`, `stddev(col)`, etc.
  - For these aggregation functions, it means the **aggregation value** from the first row to the current row in its partition when `order by` is specified

```
select country,
       title,
       year_released,
       sum(runtime) over (
           partition by country order by year_released
       ) total_runtime_till_this_row
from movies;
```

Need to specify a column in the parameter list

		title	year_released	total_runtime_till_this_row
1	am	Sayat Nova	1969	78
2	ar	Pampa bárbara	1945	98
3	ar	Albéniz	1947	308
4	ar	Madame Bovary	1947	308
5	ar	La bestia debe morir	1952	494
6	ar	Las aguas bajan turbias	1952	494
7	ar	Intermezzo criminal	1953	494
8	ar	La casa del ángel	1957	570
9	ar	Bajo un mismo rostro	1962	695
10	ar	Las aventuras del Capitán Piluso	1963	785
11	ar	Savage Pampas	1966	897
12	ar	La hora de los hornos	1968	1157
13	ar	Waiting for the Hearse	1985	1354
14	ar	La historia oficial	1985	1354

However, if there is no `order by`, the behavior will be to fill all rows with a single aggregation result computed on all rows in the same group

- One result for all rows

Pay attention to the behavior on rows with the same rank:

- They are “treated like the same row” here

Aggregation Functions as Window Functions

- `min/max(col)` , `sum(col)`, `count(col)`, `avg(col)`, `stddev(col)`, etc.
  - For these aggregation functions, it means the **aggregation value** from the first row to the current row in its partition when `order by` is specified

```
select country,
       title,
       year_released,
       min(year_released) over (
           partition by country order by year_released
       ) oldest_movie_per_country
from movies;
```

	cou...	title	year_released	oldest_movie_per_country
1	am	Sayat Nova	1969	1969
2	ar	Pampa bárbara	1945	1945
3	ar	Albéniz	1947	1945
4	ar	Madame Bovary	1947	1945
5	ar	La bestia debe morir	1952	1945
6	ar	Las aguas bajan turbias	1952	1945
7	ar	Intermezzo criminal	1953	1945
8	ar	La casa del ángel	1957	1945
9	ar	Bajo un mismo rostro	1962	1945
10	ar	Las aventuras del Capitán Piluso	1963	1945
11	ar	Savage Pampas	1966	1945
12	ar	La hora de los hornos	1968	1945
13	ar	Waiting for the Hearse	1985	1945
14	ar	La historia oficial	1985	1945

However, if there is no `order by`, the behavior will be to fill all rows with a single aggregation result computed on all rows in the same group

- One result for all rows

Pay attention to the behavior on rows with the same rank:

- They are “treated like the same row” here

Exercise

- Question: How can we get the top-5 most recent movies for each country?
  - Hint: Use a subquery in the “`from`” clause

Exercise

- Question: How can we get the top-5 most recent movies for each country?
  - Hint: Use a subquery in the "`from`" clause

```
select x.country,
       x.title,
       x.year_released
from (
 select country,
        title,
        year_released,
        row_number()
        over (partition by country
              order by year_released desc) rn
 from movies) x
where x.rn <= 5
```

Update and Delete

So Far…

- We have learned:
  - How to access existing data in tables (select)
  - How to create new rows (insert)

- CRUD/CURD
  - create, read, update, delete
    - In SQL: insert, select, update, delete
    - In RESTful API: Post, Get, Put, Delete
  - Necessary operations for persistent storage

Update

- Make changes to the existing rows in a table
- `update` is the command that changes column values
  - You can even set a non-mandatory column to NULL
  - The change is applied to all rows selected by the `where`

```
update table_name
set column_name = new_value,
    other_col = other_new_val,
    ...
where ...
```

Update

- Remember
  - When you are doing any experiments with writing operations (update, delete), **backup the data first**
    - E.g., copy the tables

Update

- Example: A nobiliary particle is used in a surname or family name in many Western cultures to signal the nobility of a family
  - We may want to modify some names in such a way as they sort as they should
    - `von Neumann -> Neumann (von)`

23 rows | Tx: Auto | DDL

	peopleid	first_name	surname	born	died	gender
1	16439	Axel	von Ambesser	1910	1988	M
2	16440	Daniel	von Bargen	1950	2015	M
3	16441	Eduard	von Borsody	1898	1970	M
4	16442	Suzanne	von Borsody	1957	<null>	F
5	16443	Tomas	von Brömssen	1943	<null>	M
6	16444	Erik	von Detten	1982	<null>	M
7	16445	Theodore	von Eltz	1893	1964	M
8	16446	Gunther	von Fritsch	1906	1988	M
9	16447	Katja	von Garnier	1966	<null>	F
10	16448	Harry	von Meter	1871	1956	M
11	16449	Jenna	von Oÿ	1977	<null>	F
12	16450	Alicia	von Rittberg	1993	<null>	F
13	16451	Daisy	von Scherler Mayer	1966	<null>	F
14	16452	Gustav	von Seyffertitz	1862	1943	M
15	16453	Josef	von Sternberg	1894	1969	M

John von Neumann

Update

- Example: A nobiliary particle is used in a surname or family name in many Western cultures to signal the nobility of a family
  - We may want to modify some names in such a way as they sort as they should
    - `von Neumann -> Neumann (von)`

- First, how can we find these names?

Update

- Example: A nobiliary particle is used in a surname or family name in many Western cultures to signal the nobility of a family.
  - We may want to modify some names in such a way as they sort as they should
    - **`von Neumann -> Neumann (von)`**

- First, how can we find these names?
  - Wildcards
  - Strings starting with “von ”

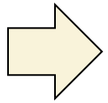
```
select * from people_1 where surname like 'von %';
```

Update

- Example: A nobiliary particle is used in a surname or family name in many Western cultures to signal the nobility of a family.
  - We may want to modify some names in such a way as they sort as they should.

- Then, how should we update the names?

```
(first_name) John
(surname) von Neumann
```



```
(first_name) John
(surname) Neumann (von)
```

  - Try the transformation with select:

```
select replace('von Neumann', 'von ', '') || ' (von)';
```

	?column?
1	Neumann (von)

Update

- Example: A nobiliary particle is used in a surname or family name in many Western cultures to signal the nobility of a family.
  - We may want to modify some names in such a way as they sort as they should.

- Finally, the `update` statement:

```
-- Specify the table
update people

-- Set the update rule
set surname = replace(surname, 'von ', '') || ' (von)'

-- Find the rows that need to be updated
where surname like 'von %';
```

This could be used to postfix all surnames starting by 'von' with '(von)'

- Turn, e.g., 'von Neumann' into Nuemann (von)'

Update

- The `where` clause specifies the affected rows
  - However, you can use update without `where`, where the updates will be applied to all rows in the table
    - Use with caution!
    - Sometimes, there will be a warning in IDEs such as DataGrip

Update

- The update operation may not be successful when constraints are violated
  - For example, update the primary key but with duplicated values

```
update people set peopleid = 1 where peopleid < 10;
```

```
[23505] ERROR: duplicate key value violates unique constraint "people_pkey"
Detail: Key (peopleid)=(1) already exists.
```

  - This is why we need constraints when creating tables
    - Avoid unacceptable writing operations that break the integrity of the tables

Update

- Subqueries in update
  - Complex update operations where values are based on a query result

- Example: Add a column in people table to record the number of movies one has joined (either directed or played a role in)

Update

- Example: Add a column in people table to record the number of movies one has joined (either directed or played a role in)
  - First, how do we count the movies for a person?
    - (Used as the subquery part in the update statement)

```
select count(*) from credits c where c.peopleid = [some peopleid];
```

Update

- Example: Add a column in people table to record the number of movies one has joined (either directed or played a role in)
  - First, how do we count the movies for a person?
    - (Used as the subquery part in the update statement)
  - Then, let's update the data

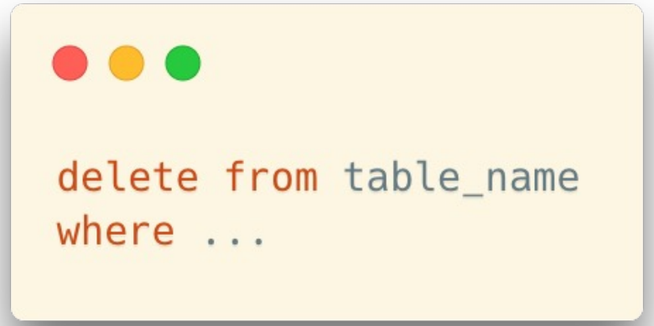
```
update people p

set num_movies = (
    select count(*) from credits c where c.peopleid = p.peopleid
)

where peopleid < 500;
-- This where is only for testing purpose;
-- You should change it (or remove it) when in actual use.
```

Delete

- As the name shows, `delete` removes rows from tables



  - If you omit the WHERE clause, then (as with UPDATE) the statement affects all rows and you **end up with an empty table!**

  - Well,
    - many database products provide a roll-back mechanism when deleting rows
    - Transactions can also protect you (to some extent)

Delete

- One important point with constraints (foreign keys in particular) is that they guarantee that data remains consistent
  - They not only work with `insert`, but with `update` and `delete` as well.

  - Example: Try to delete some rows in the `country` table

```
delete from countries where country_code = 'us';
```

```
[23503] ERROR: update or delete on table "countries" violates foreign key constraint "movies_country_fkey" on table "movies"
Detail: Key (country_code)=(us) is still referenced from table "movies".
```

  - Foreign-key constraints are especially useful in controlling `delete` operations

Constraints

- This is why constraints are so important:
  - They ensure that whatever happens, you'll always be able to make sense of ALL pieces of data in your database
  - All declared constraints need to be satisfied for any changed made to the tables