Principles of Database Systems (CS307)

Lecture 3: Basic and Intermediate SQL

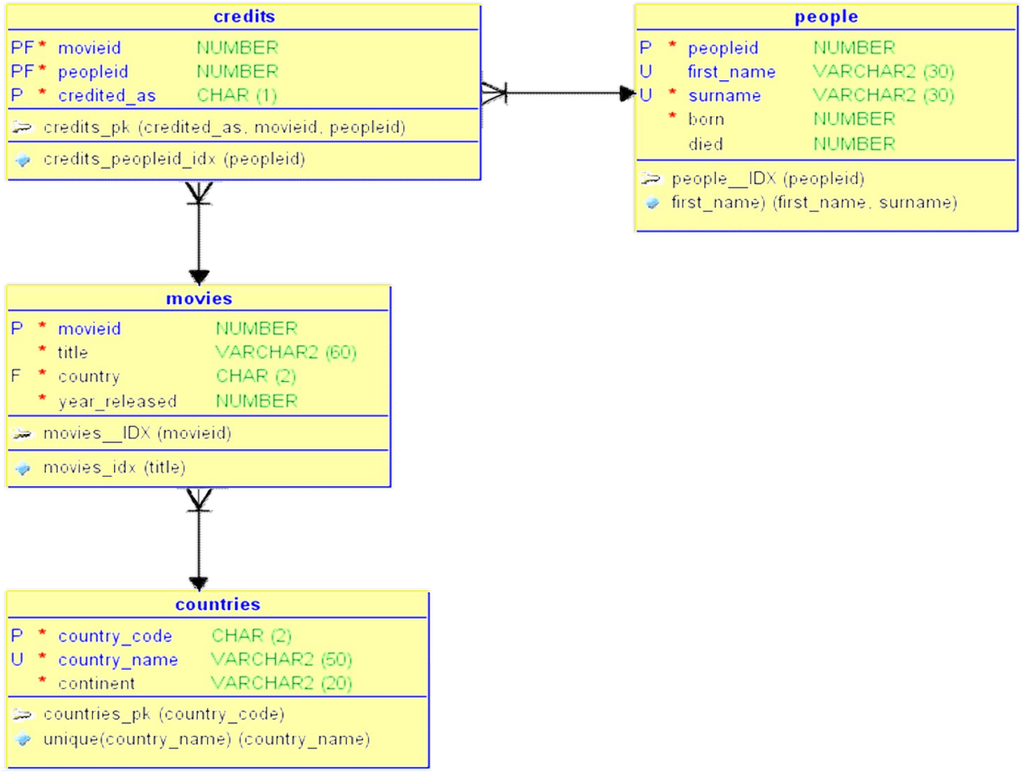
Zhong-Qiu Wang

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

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

- Show DDL of a table

```
desc movies;  -- Oracle, MySQL

describe table movies  -- IBM DB2

\d movies  -- PostgreSQL

.schema movies  -- SQLite
```

Some Functions – Compute and Derive

- One important feature of SQL is that you don't need to return data exactly as it was stored
  - Operators, and many (*mostly DBMS specific*) functions allow to return transformed data

Some Functions

- A simple transformation is concatenating two strings together
  - Most products use || (two vertical bars) to indicate string concatenation
  - SQL Server, though, uses +, and MySQL a special `concat()` function that also exists in some other products

```
select title
       || ' was released in '
       || year_released movie_release
from movies
where country = 'us';
```

	movie_release
1	Variety was released in 2008
2	Inglourious Basterds was released in 2009
3	La grande vadrouille was released in 1966
4	Pulp Fiction was released in 1994
5	Schindler's List was released in 1993
6	Star Wars was released in 1977
7	The Dark Knight was released in 2008
8	The Godfather was released in 1972
9	The Shawshank Redemption was released in 1994
10	Titanic was released in 1997
11	Charade was released in 1963
12	North by Northwest was released in 1959
13	Singin' in the Rain was released in 1952
14	Rear Window was released in 1954
15	City Lights was released in 1931

Some Functions

- A simple transformation is concatenating two strings together
  - Most products use || (two vertical bars) to indicate string concatenation
  - SQL Server, though, uses +, and MySQL a special `concat()` function that also exists in some other products

```
select title
       || ' was released in '
       || year_released movie_release
from movies
where country = 'us';
```

	movie_release
1	Variety was released in 2008
2	Inglourious Basterds was released in 2009
3	La grande vadrouille was released in 1966
4	Pulp Fiction was released in 1994
5	Schindler's List was released in 1993
6	Star Wars was released in 1977
7	The Dark Knight was released in 2008
8	The Godfather was released in 1972
9	The Shawshank Redemption was released in 1994
10	Titanic was released in 1997
11	Charade was released in 1963
12	North by Northwest was released in 1959
13	Singin' in the Rain was released in 1952
14	Rear Window was released in 1954
15	City Lights was released in 1931

Note that you can give a name to an expression

- This will be used as column header
- It also becomes a "virtual column" if you turn the query into a "virtual table"

Some Functions

- A simple transformation is concatenating two strings together
  - Most products use || (two vertical bars) to indicate string concatenation
  - SQL Server, though, uses +, and MySQL a special `concat()` function that also exists in some other products

```
select title
       || ' was released in '
       || year_released movie_release
from movies
where country = 'us';
```

Although YEAR_RELEASED is actually a number, it's implicitly turned into a string by the DBMS.

- In that case it's not a big issue, but it would be better to use a function to convert explicitly.

```
select title
       || ' was released in '
       || cast(year_released as varchar) movie_release
from movies
where country = 'us';
```

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.

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

Some Functions

- Numerical functions

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

- More string functions

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

Some Functions

- Type casting
  - `cast(column as type)`

```
select cast(born as char)||'abc' from people;
select cast(born as char(2)) ||'abc' from people;
select cast(born as char(10)) ||'abc' from people;
select cast(born as varchar) ||'abc' from people;
select cast(born as varchar(2)) ||'abc' from people;
```

Case

- A very useful construct is the CASE … END construct that is similar to IF or SWITCH statements in a program

```
CASE input_expression
    WHEN when_expression THEN result_expression
    [ ...n ]
    [ELSE else_result_expression]
END
```

```
CASE
    WHEN Boolean_expression THEN result_expression
    [ ...n ]
    [ELSE else_result_expression]
END
```

Case

- Example 1: Show the corresponding words of the gender abbreviations

```
select peopleid, surname,
    case gender
        when 'M' then 'male'
        when 'F' then 'female'
    end as gender_2
from people
where died is null;
```

*Similar to the switch-case statement in Java and C

Case

- Example 2: Decide whether someone’s age is older/younger than a pivot

Case

- Example 2: Decide whether someone's age is older/younger than a pivot

A horrible solution!

```
case age
    when 30 then 'younger than 44'
    when 31 then 'younger than 44'
    when 32 then 'younger than 44'
    when 33 then 'younger than 44'
    when 34 then 'younger than 44'
    when 35 then 'younger than 44'
    when 36 then 'younger than 44'
    ...
    when 43 then 'younger than 44'
    when 44 then '44 years old'
    when 45 then 'older than 44'
    ...
end as status
```

Case

- Example 2: Decide whether someone's age is older/younger than a pivot
  - CASE

```
select peopleid, surname,
    case (date_part('year', now()) - born > 44)
        when true then 'older than 44'
        when false then 'younger than 44'
        else '44 years old'
    end as status
from people
where died is null;
```

Case

- Example 2: Decide whether someone's age is older/younger than a pivot
  - CASE
  - CASE WHEN

```
select peopleid, surname,
    case
        when (date_part('year', now()) - born > 44) then 'older than 44'
        when (date_part('year', now()) - born < 44) then 'younger than 44'
        else '44 years old'
    end as status
from people
where died is null;
```

Case

- Example 2: Decide whether someone's age is older/younger than a pivot
  - CASE
  - CASE WHEN

```
select peopleid, surname,
    case
        when (date_part('year', now()) - born > 44) then 'older than 44'
        when (date_part('year', now()) - born < 44) then 'younger than 44'
        else '44 years old'
    end as status
from people
where died is null;
```

The ELSE branch

- Return a default value when all `when` criteria are not met
- If no else, `NULL` will be returned

Case

- About the `NULL` value
  - Use the “`is null`” criteria

```
select surname,
    case
        when died is null then 'alive and kicking'
        else 'passed away'
    end as status
from people
```

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;
```

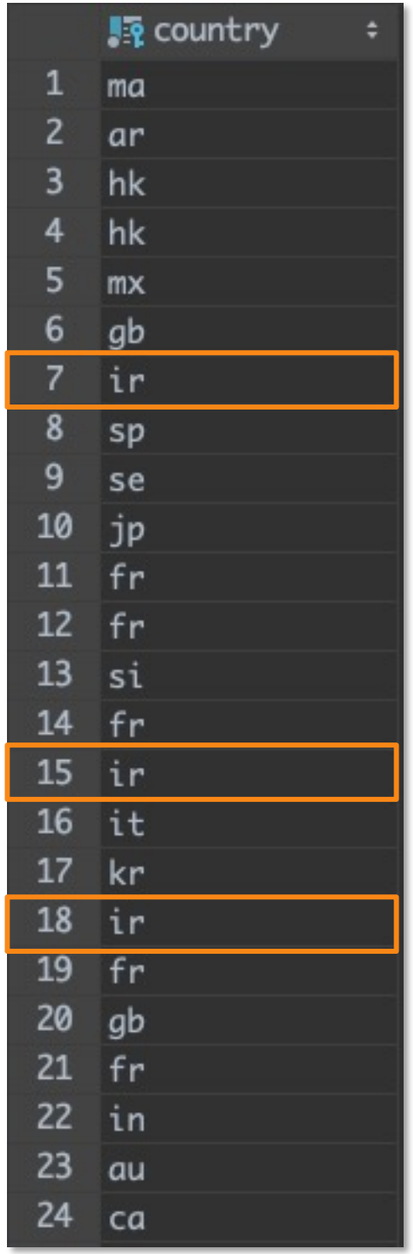
	country
1	ma
2	ar
3	hk
4	hk
5	mx
6	gb
7	ir
8	sp
9	se
10	jp
11	fr
12	fr
13	si
14	fr
15	ir
16	it
17	kr
18	ir
19	fr
20	gb
21	fr
22	in
23	au
24	ca

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;
```

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!

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);
```

44 rows

	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 are released in each country, 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

```
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"

```
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"

```
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

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

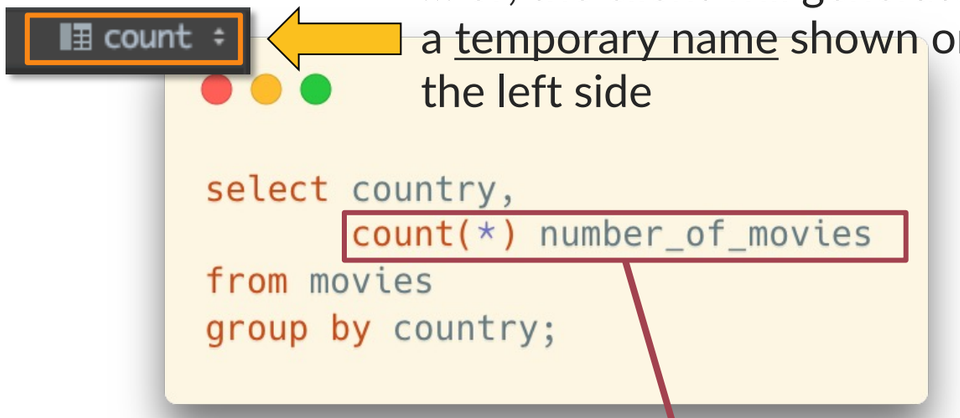
Aggregate Functions

- 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



	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"

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

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

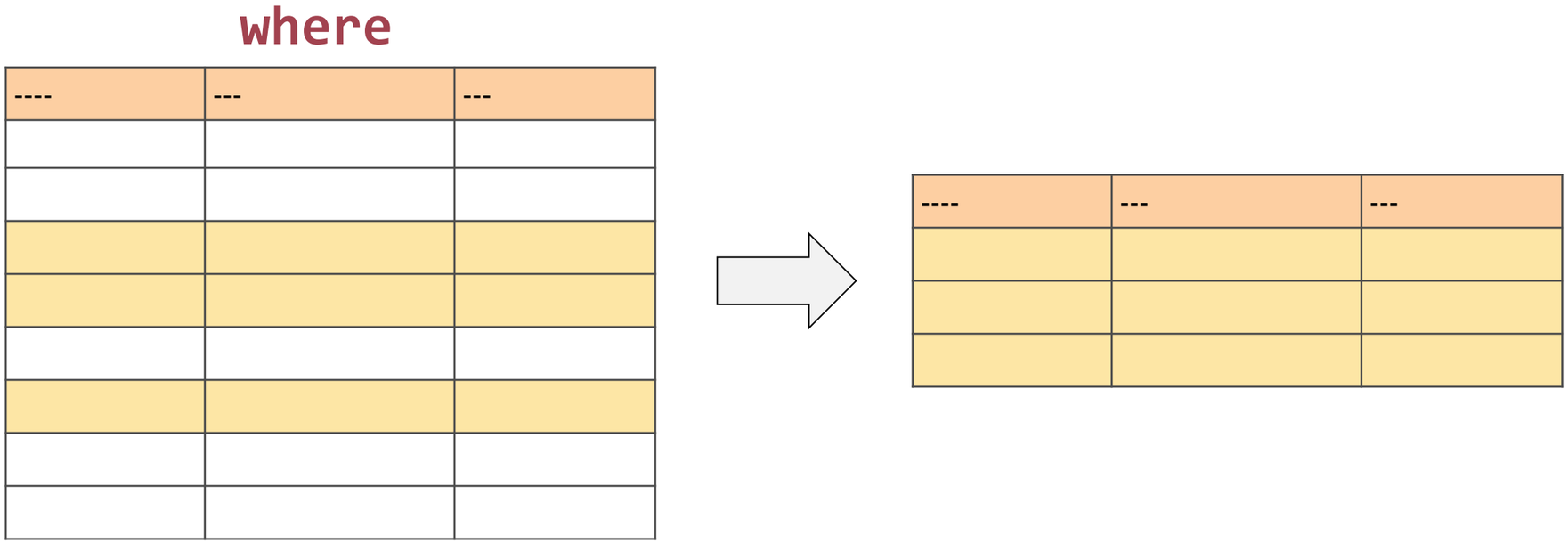
```
select country,
       year_released,
       count(*) number_of_movies
from movies
group by country, year_released
```

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

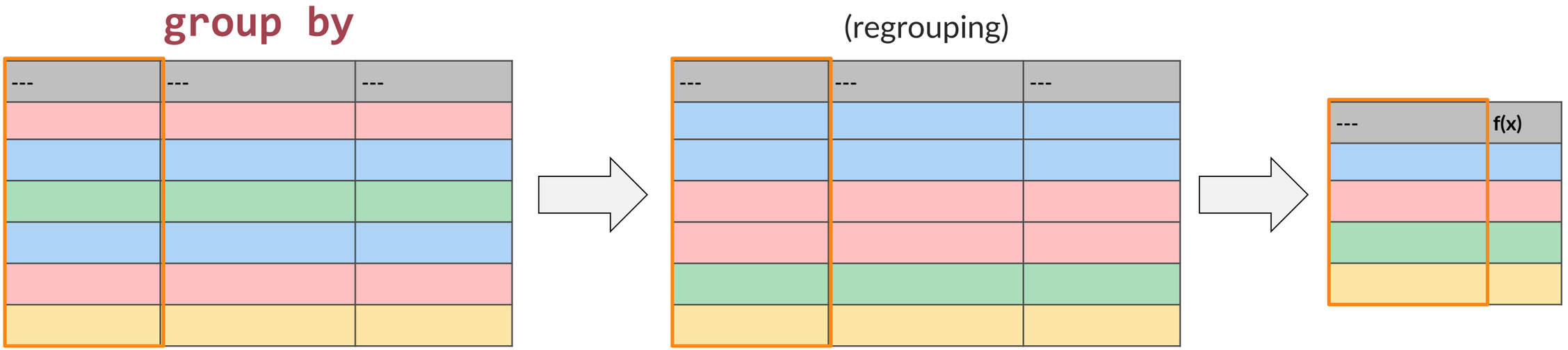
Aggregate Functions

- 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

- 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

- Therefore, we can apply another relational operation such as “select”, 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
```

Aggregate Functions

- Aggregating rows requires sorting them in a way or another
- Sorting is always costly, and does not scale well
  - cost increases faster than the number of rows sorted.

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

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

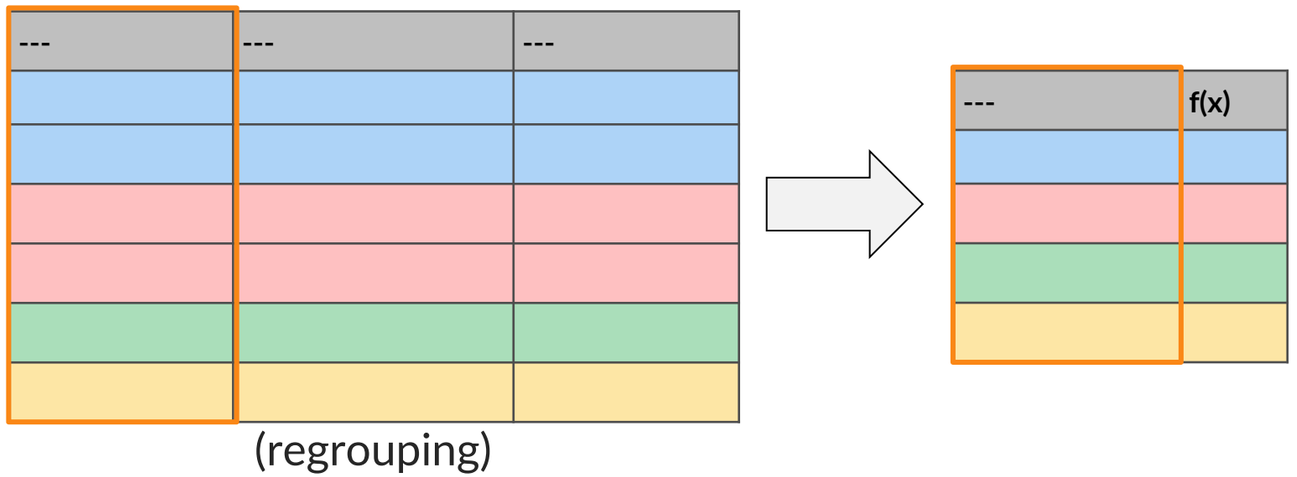
- All database management systems have a highly important component that we'll see again, called query optimizer
  - It takes your query, and tries to find the most efficient way to run it
  - Sometimes it tries to outsmart you, with from time to time unintended consequences
  - Sometimes it optimistically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

Aggregate Functions

- *Nulls?*
  - Some attributes could have missing values, when doing aggregation

- 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.
  - known + unknown = unknown

- What happens with aggregates?



Aggregate Functions

- ***Nulls?***

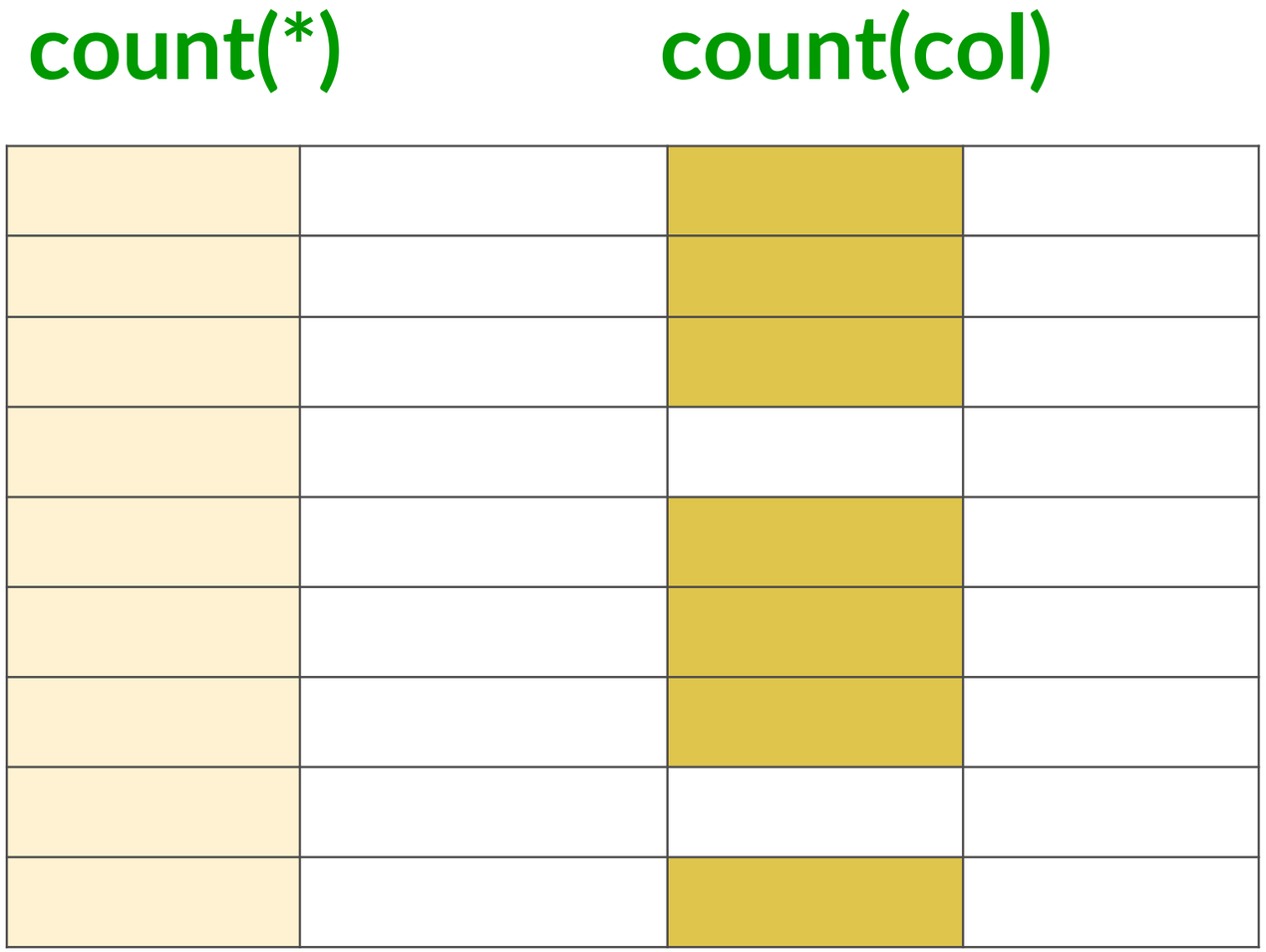
- Aggregate functions ***ignore*** Nulls

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.

```
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 starting with a Q instead of how many people having a surname that starts with a Q

Aggregate Functions

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

Count number of years, in which each country released movies

- Group by country first
- Count distinct #years of each country

- These two queries are equivalent

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

- Select distinct country and year_released
- Group by country and count the number of unique years

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

Aggregate Functions

- **How many people are both actors and directors?**

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

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;
```

- `movie_id` is irrelevant
  - Do not require someone who is both the director and actor in a movie
- 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).

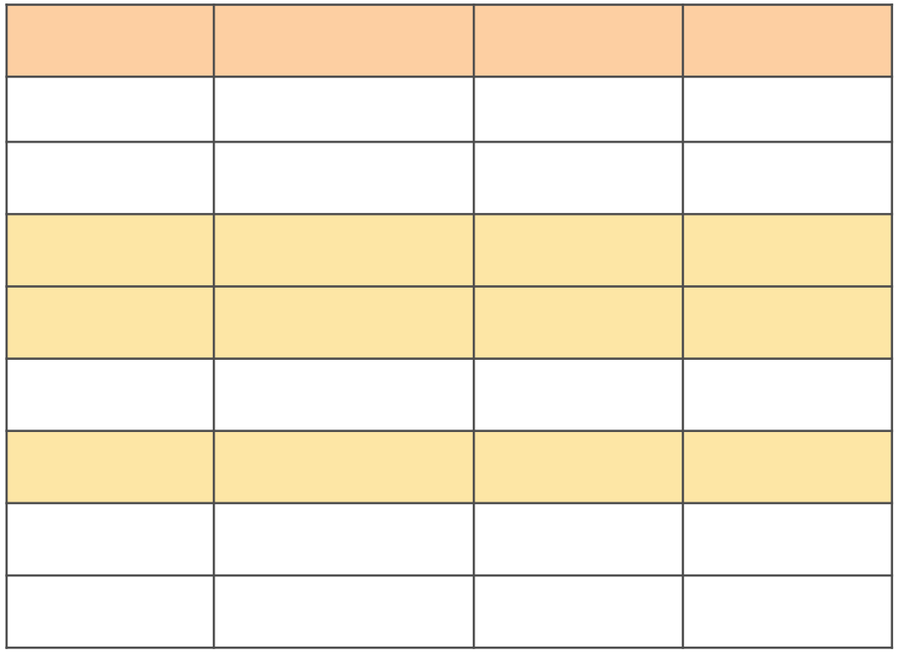
Aggregate Functions

- 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;
```

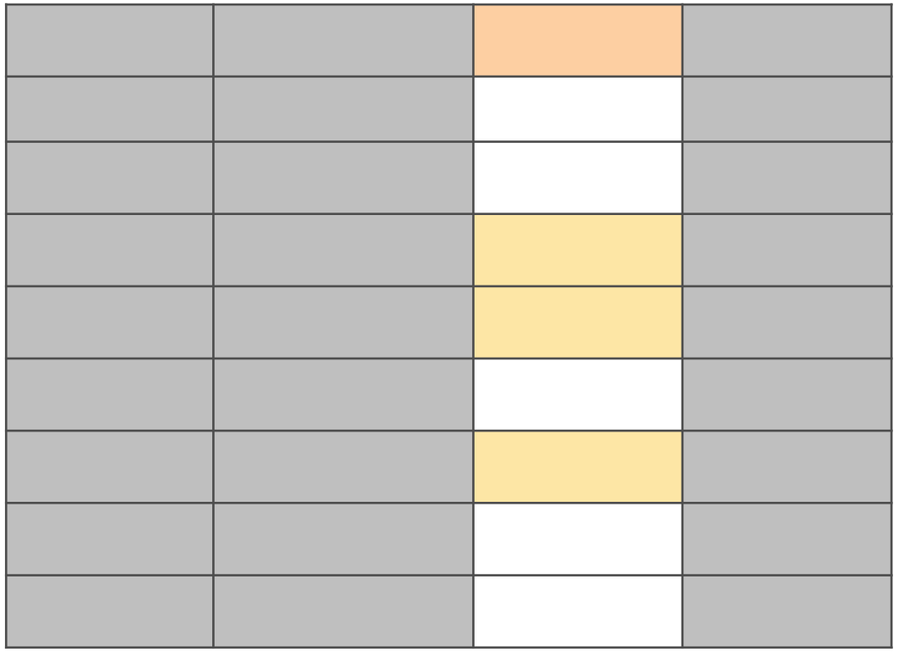
Join

Retrieving Data from Multiple Tables



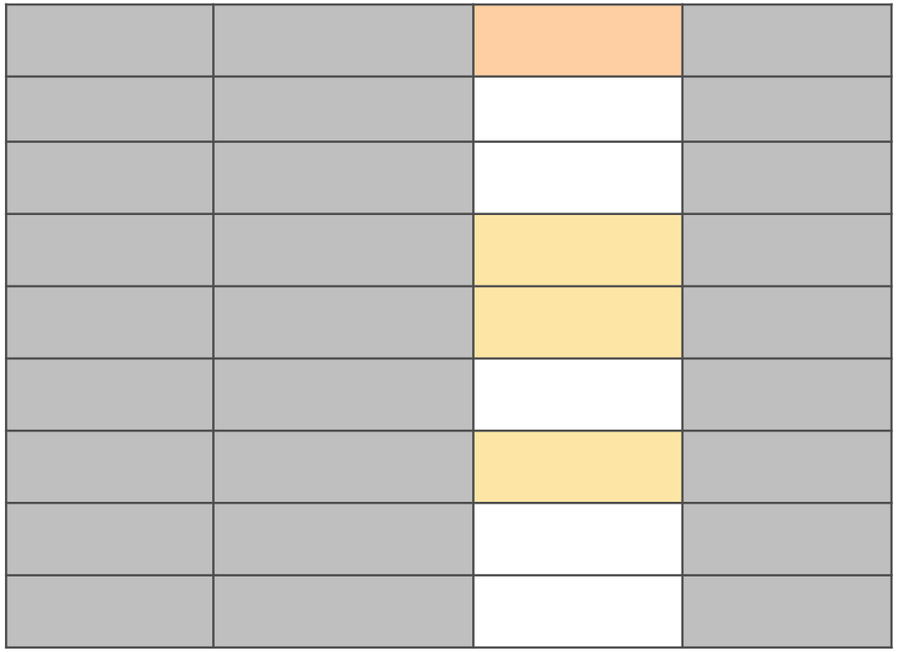
- We have seen the basic operation consisting in filtering rows (an operator called SELECT by Codd)

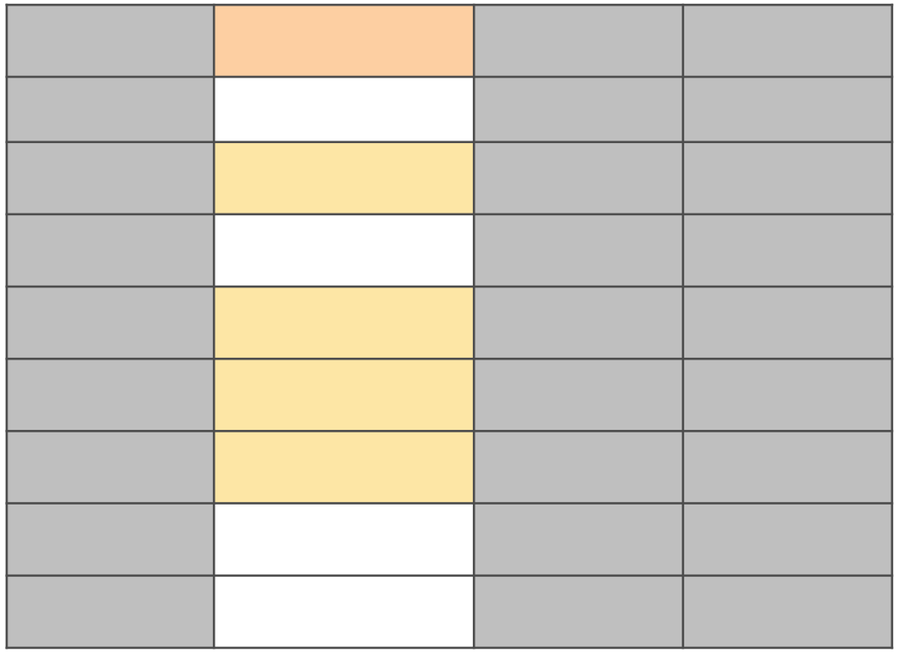
Retrieving Data from Multiple Tables



- We have seen how we can only return some columns (called PROJECT by Codd)
- Be careful not returning duplicates when we aren't returning a full key

Retrieving Data from Multiple Tables

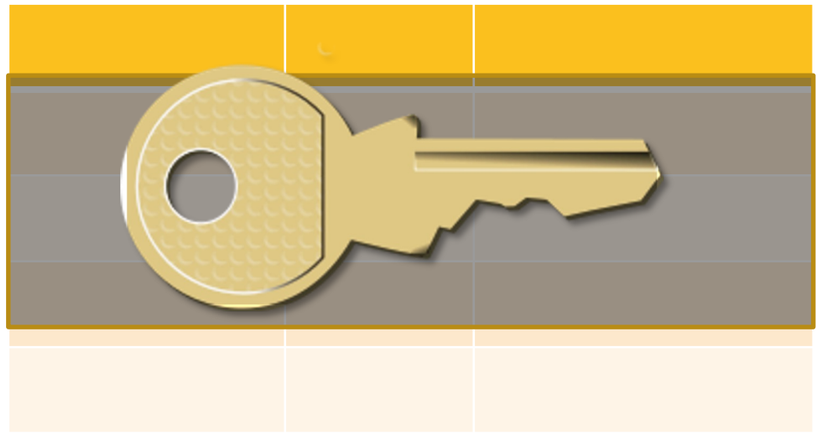




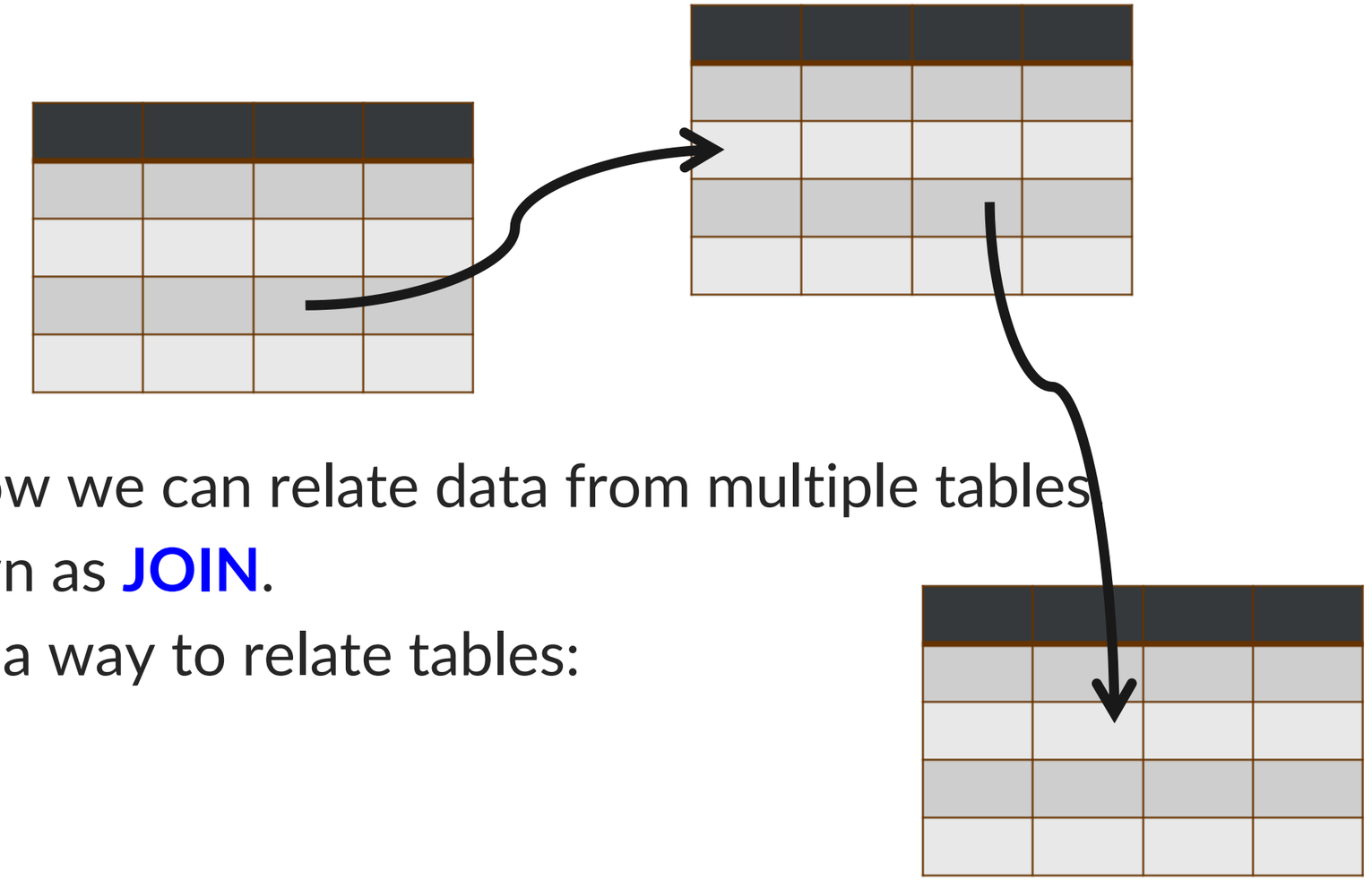
- We have also seen how we can return data that doesn't exist as such in tables by applying functions to columns

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



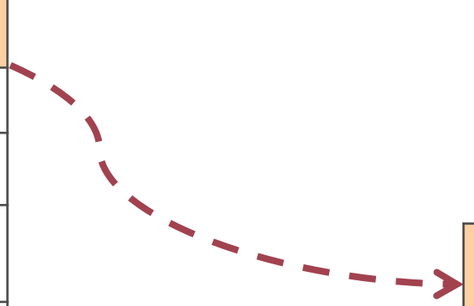
Retrieving Data from Multiple Tables



- 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

```sql
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?

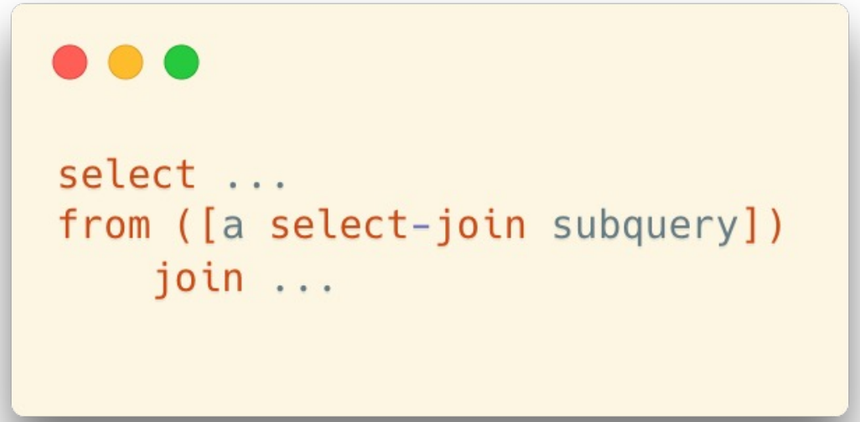
Self Join

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

```sql
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 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

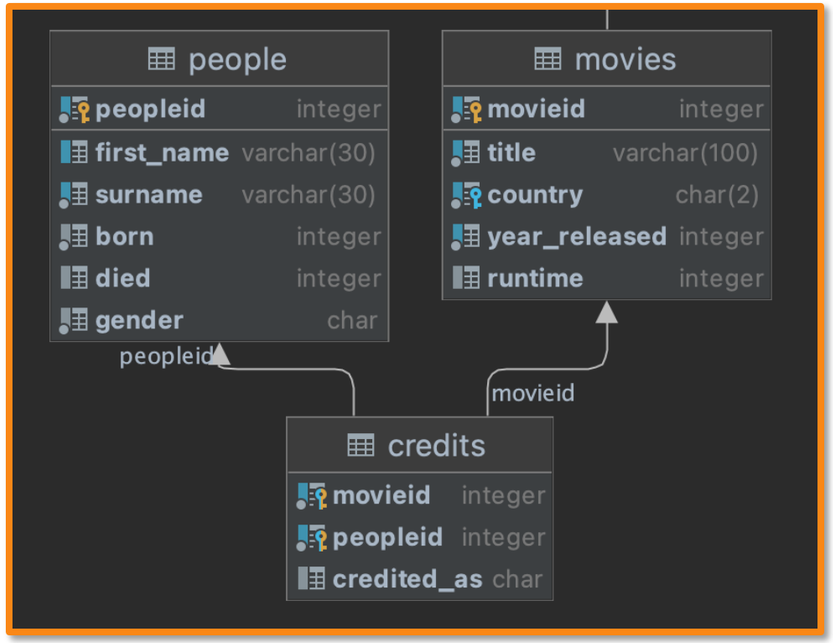
- We can chain joins the same way we chain filtering conditions with AND / OR / NOT
  - Joins between 10 or 15 tables aren't uncommon (i.e., common), and queries generated by programs often do much worse

Chaining Joins Together

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

Chaining Joins Together

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



Chaining Joins Together

- We can chain joins the same way we chain filtering conditions with AND / OR / NOT
  - Joins between 10 or 15 tables aren't uncommon (i.e., common), 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';
```

More on Join

The Old Way of Writing Joins

- Use commas to separate the tables
  - Example: The solution for the same question in the previous slide
- A little bit history:
  - `join` was introduced in SQL-1999 (later than this original way)
- Relationship to the relational algebra
  - Filtering based on the Cartesian product
    - movies × credits × people

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

The Old Way of Writing Joins

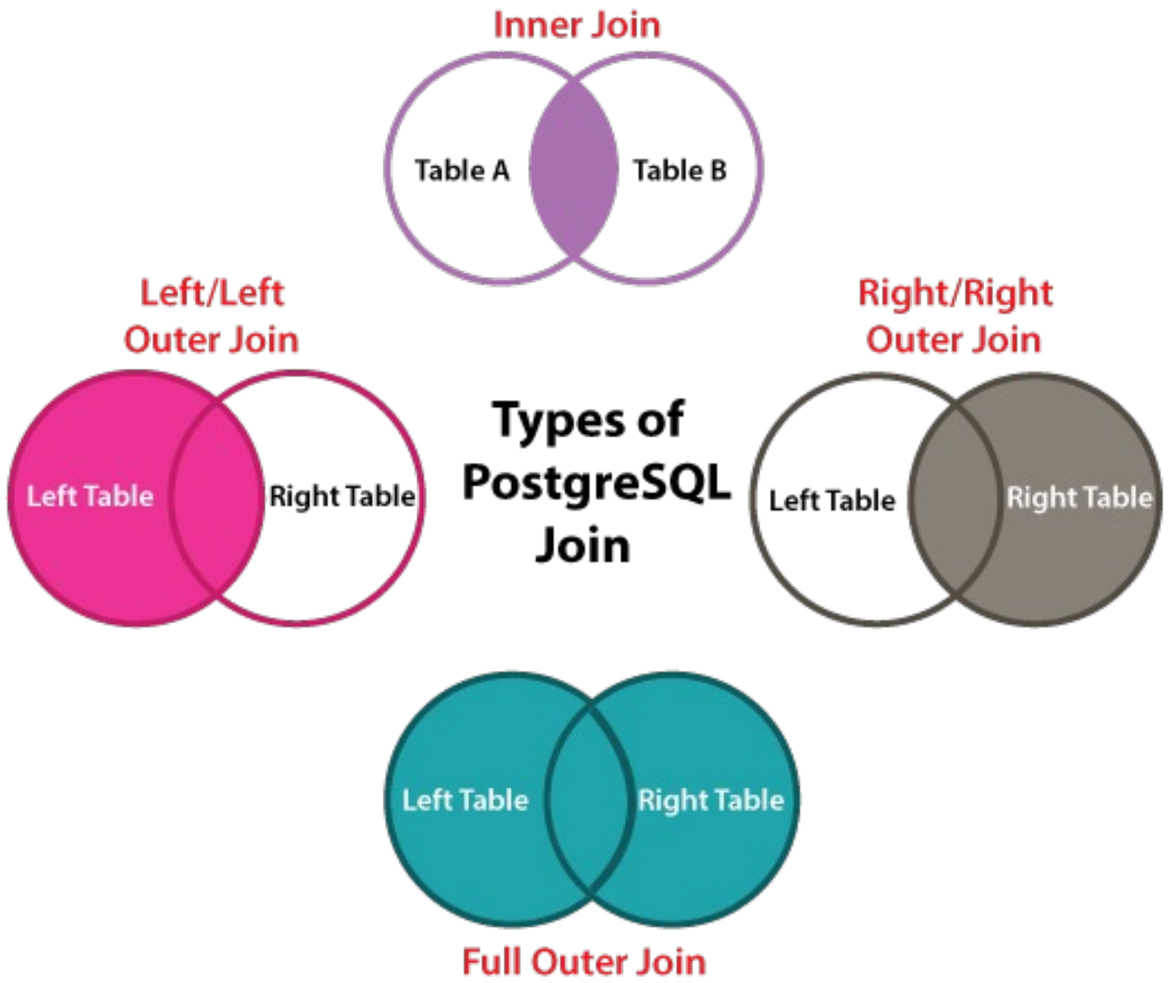
- Problems in the old way:
  - If you forget a comma, it will still work sometimes (interpreted as “renaming”)
  - The semantic meaning of the `where` clause here is a little bit different from the `where` we introduced before
    - Join key vs. filtering condition
    - If you forget `where`, the query will not return an error but to end up with **HUGE** amount of rows
      - `#movies * #credits * #people`

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

Inner and Outer Joins

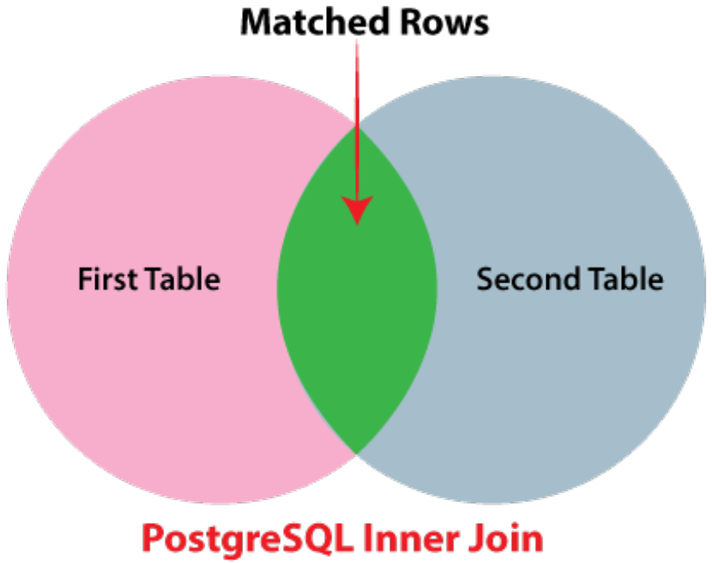
- So far, we only consider the rows with matching values on the corresponding columns
  - However, there are more things you can do with join

Inner and Outer Joins



Inner and Outer Joins

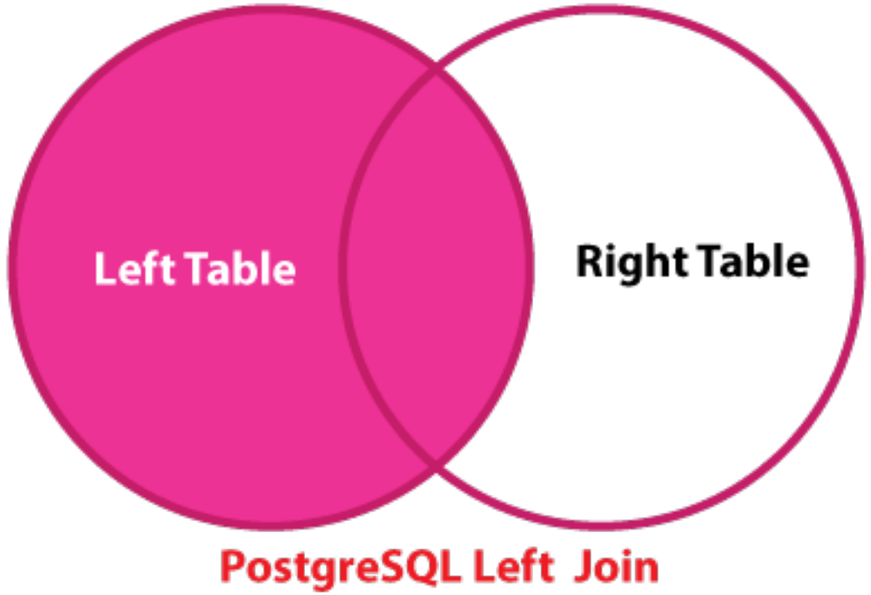
- Inner join
  - The default join type
  - Acturally, all examples before are considered inner joins
  - Only joined rows with matching values are selected




```
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 the **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
    - But, left (outer) join can give you a record for the movie (in the left table) where all right-table columns are null

Pay attention to the syntax:

- `left join` or `left outer join`
- But some databases recognize the `outer` keyword, some do not. Refer to the database manual if you meet any error.

```
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? Why should we show the records in the left table with no matches?

  - Scenario: Movie Website (Douban, for example)
    - We cannot just ignore the movies with no credit information
    - Instead, we should list them, and also show that credit information is missing
    - All things can be done in a single query
      - And we can distinguish between them by checking the values in the right-table columns

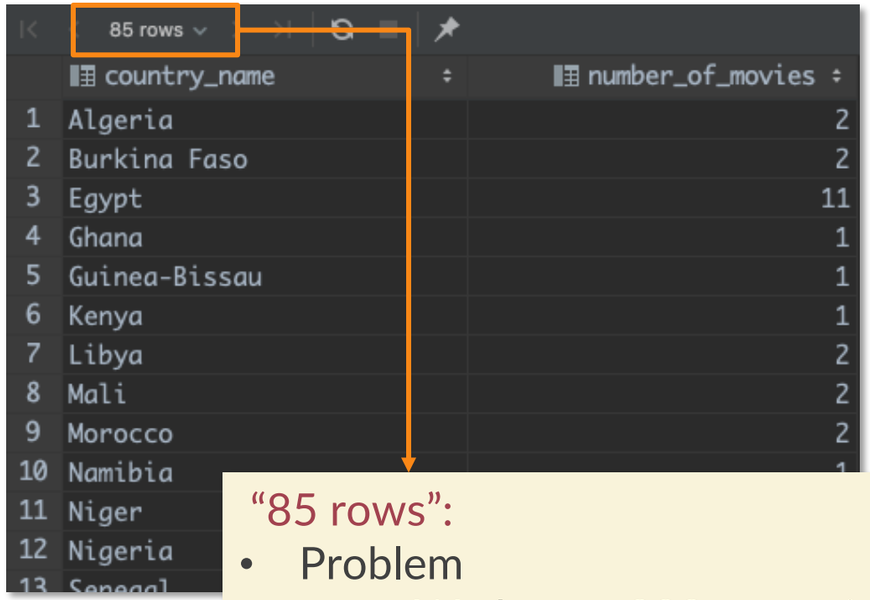
Inner and Outer Joins

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

Inner and Outer Joins

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

```
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
  - In addition, how can we 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

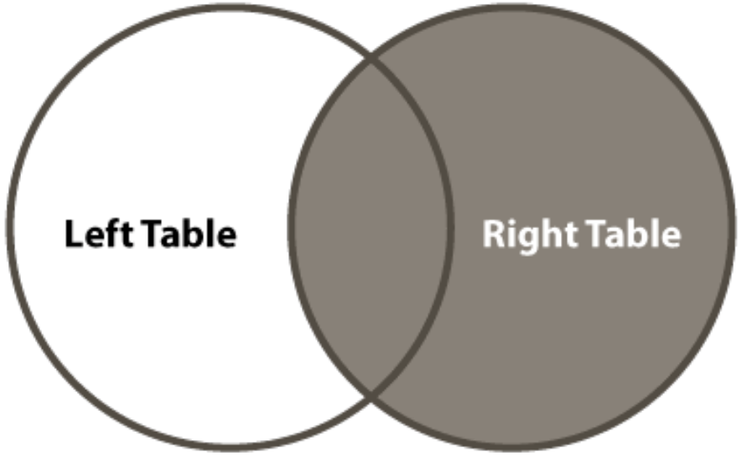
- Left outer join
  - All countries are here now
  - In addition, how can we replace nulls?
    - Add another CASE condition

```
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;
```

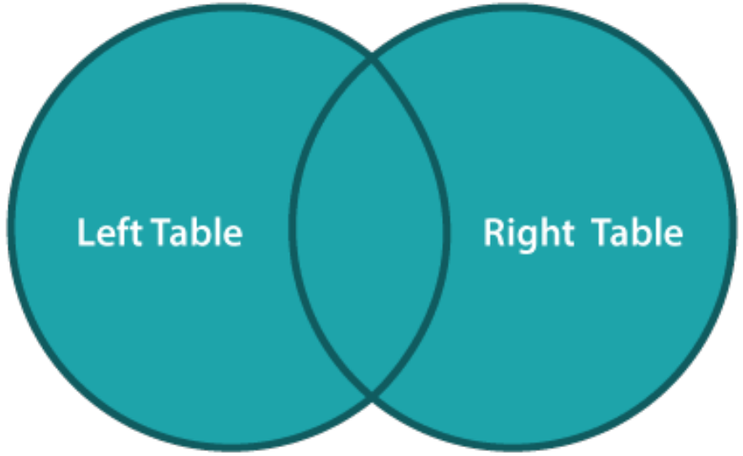
	country_name	number_of_movies
1	Algeria	2
2	Angola	0
3	Benin	0
4	Botswana	0
5	Burkina Faso	2
6	Burundi	0
7	Cameroon	0
8	Central African Republic	0
9	Chad	0
10	Comoros	0
11	Congo Brazzaville	0
12	Congo Kinshasa	0
13	Cote d'Ivoire	0
14	Djibouti	0
15	Egypt	11
16	Equatorial Guinea	0
17	Eritrea	0
18	Ethiopia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



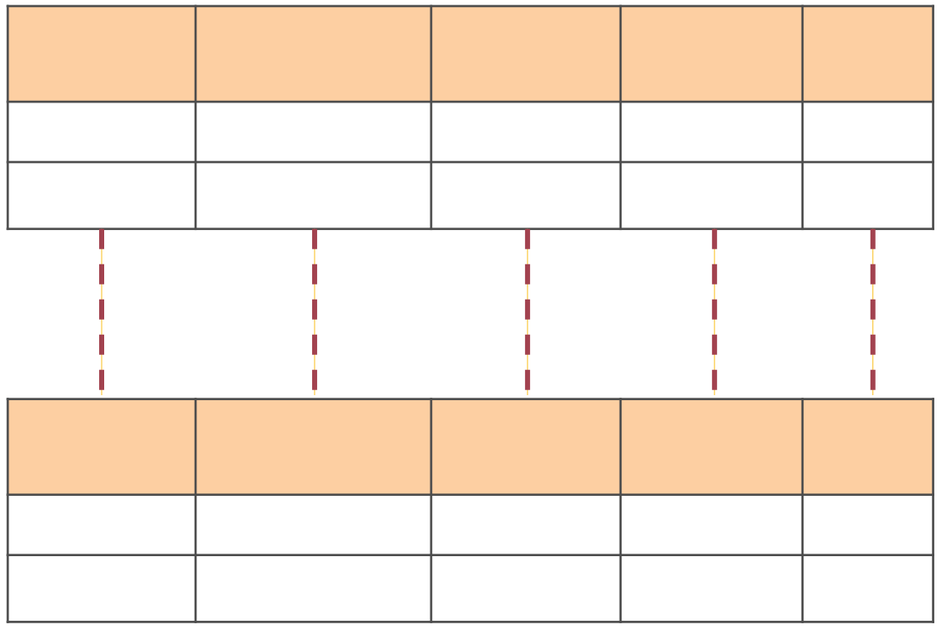
PostgreSQL Full Outer Join

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

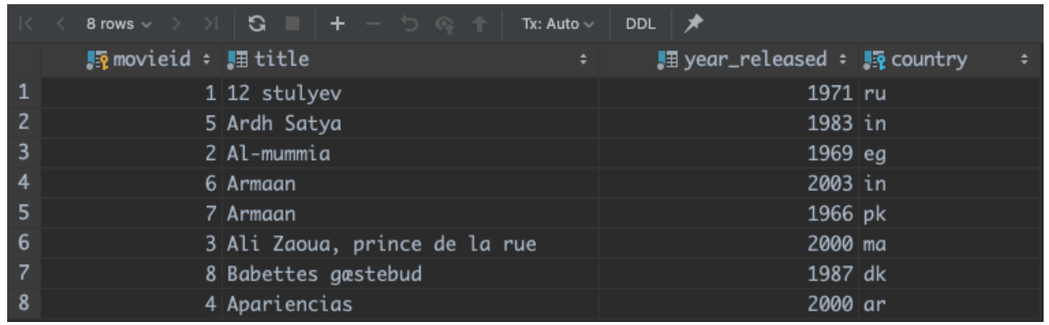
Set Operators

- Union
  - Usage scenario: 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

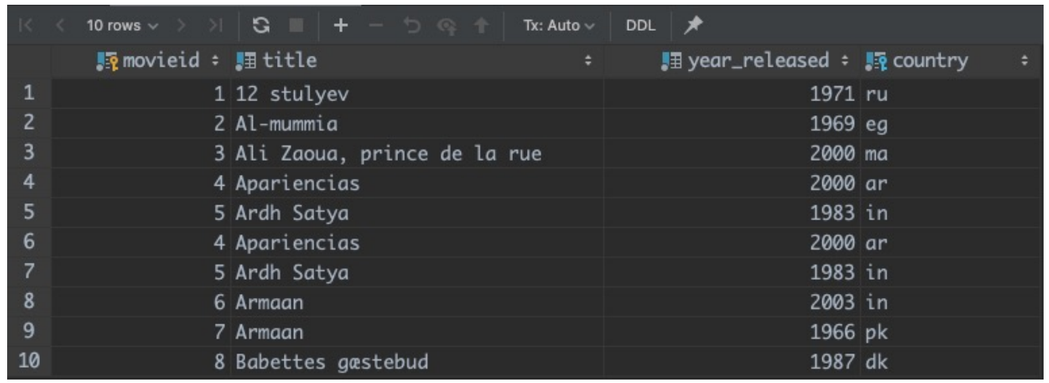
- Union
  - Warning: `union` will remove duplicated rows
    - Instead, you can use `union all`

```
(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 (`intersect`)
  - Return the rows that appears in both tables
- Except (`except`)
  - Return the rows that appear in the first table but not the second one
  - Sometimes written as `minus` in some database products

- However, they are not used as much as union
  - intersect -> inner join
  - except -> left outer join with an “is null” condition

Subquery

Subquery

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

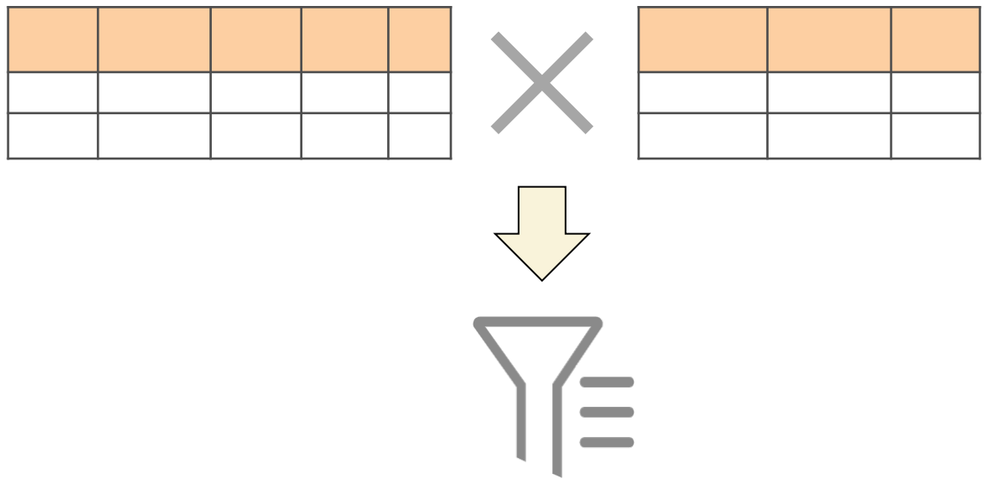
- And, we can add subqueries after `select` and `where` as well

```
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

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

```sql
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 has also other interesting uses)
  - It may make 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

- ... But `in()` is far more powerful than this
  - What is between parentheses may be, not only an explicit list, 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

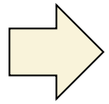
- Example: Select all European movies
  - A (slightly better) solution: list all European countries in an `in` operator

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

Subquery after Where

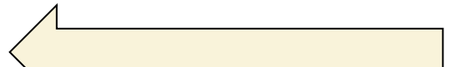
- Example: Select all European movies
  - A (slightly better) solution: list all European countries in an `in` operator

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



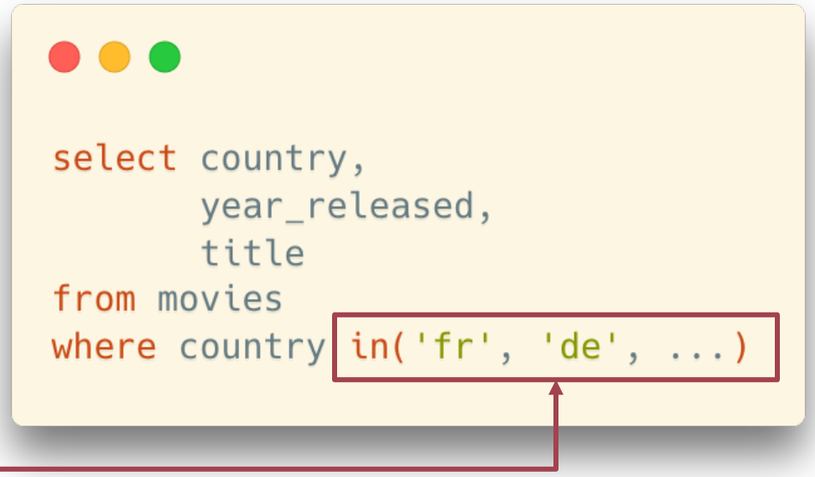
```
select * from countries where continent = 'EUROPE';
```

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 products (Oracle, DB2, PostgreSQL with some twisting) even allow comparing a set of column values (the correct word is "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()` means an implicit distinct in the subquery
    - `in('cn', 'us', 'cn', 'us', 'us')` is equal to `in('cn', 'us')`

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

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
```

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

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

Thus, the `not in()` expression always returns false, and hence no row will be selected and returned.

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:
  `TRUE and FALSE 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 the 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

- No matter how difficult the query is, you can apply order by to any result set

```
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)

```
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

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
  - For example:
    - The result set will be order by `col1` first
    - If there are rows with the same value on `col1`, these rows will be ordered by `col2`.

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

- Ascending or descending order
  - Add `desc` or `asc` after the column
    - However, `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

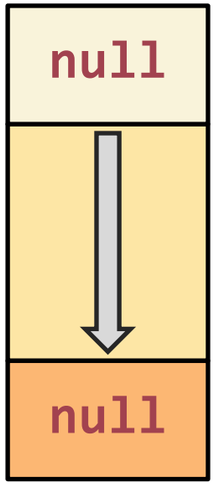
```
select * from credits
order by
    case credited_as
        when 'D' then 1
        when 'A' then 2
    end desc;
```

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”.
  - Collation

Self Study: Text Encoding

- Key Question: How does characters represented in a computer?
  - Wikipedia – Character encoding: `https://en.wikipedia.org/wiki/Character_encoding`
  - A video on Bilibili: `https://www.bilibili.com/video/BV1xP4y1J7CS`

Self Study: Text Encoding

手持两把锟斤拷，
口中疾呼烫烫烫。
脚踏千朵屯屯屯，
笑看万物锘锘锘。

- Try to answer the following questions:
  - What are ASCII, Unicode, UTF-8, and UTF-16? What are the relationships between them?
  - What are GB2312, GB18030, and GBK? What are “锟斤拷” and “烫烫烫”? How can you make it (not) happen?
  - Given a string with several characters, can you print the bitmap of this string?
  - Are emojis characters? How can you insert an emoji in a text editor?
  - What are the default character encodings in different platforms?
    - OS: Windows, MacOS, Linux
    - DBMS: PostgreSQL, etc.
    - Programming Languages: Java, C, C++, Python, etc.
  - How can we translate strings from one encoding to another?
    - E.g., with text editors (Windows Notepad, VSCode, Sublime Text, etc.); in programming languages; in DBMS

Limit and Offset

- Get a slice of the long query result
  - `limit k offset p`
    - Return the top-k rows in the result set and skip 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
    - Always refer to the software manual for specific features

```
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
```