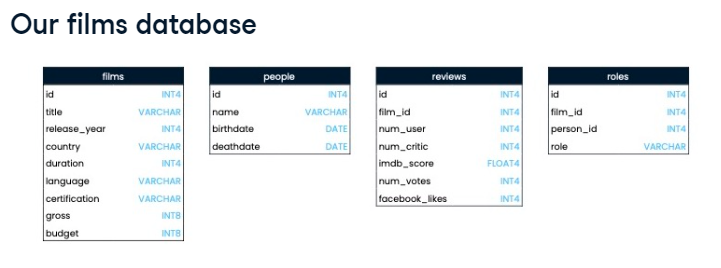
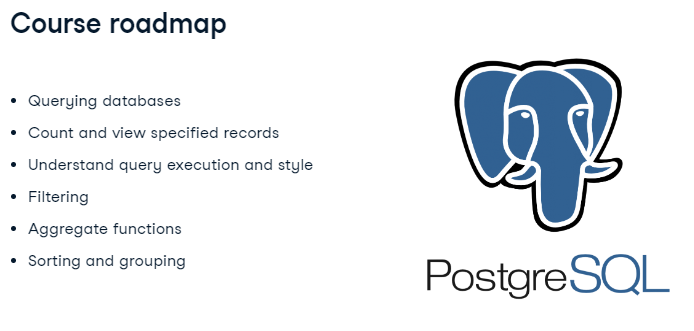
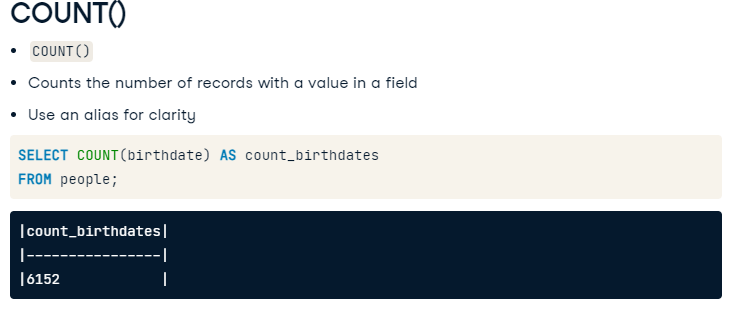
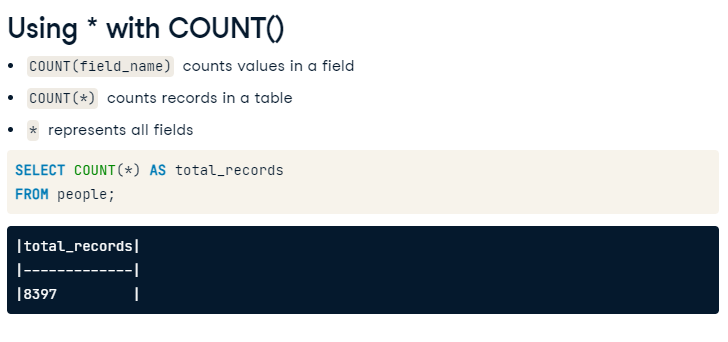
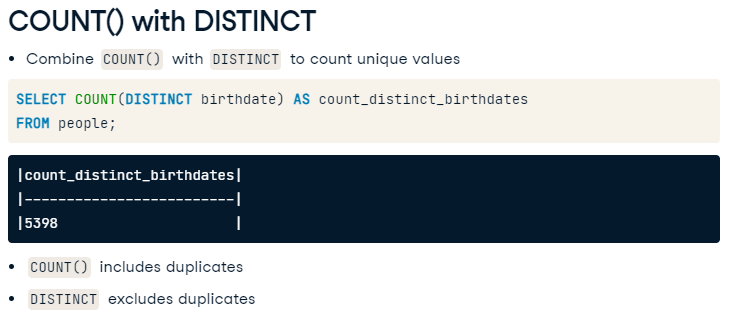


**Intermediate SQL**

****





-- Count the number of birthdates in the people table

select count(birthdate) as count\_birthdate from people;

-- Count the records for languages and countries represented in the films table

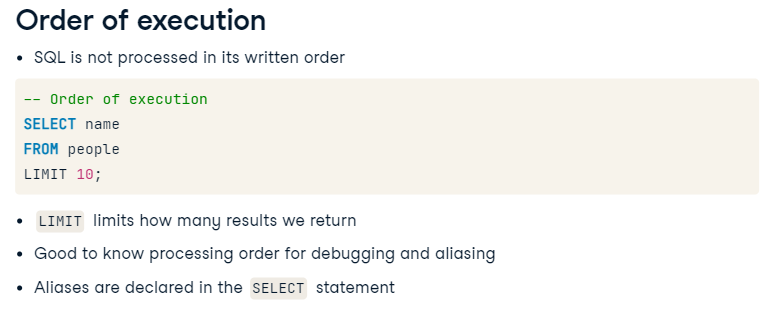
select count(language) as count\_languages , count(country) as count\_countries from films;

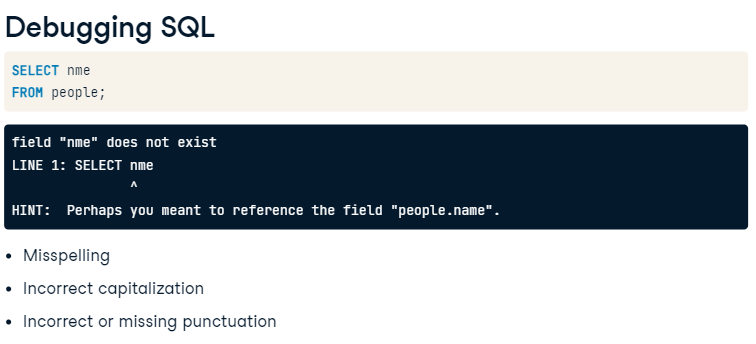
-- Return the unique countries from the films table

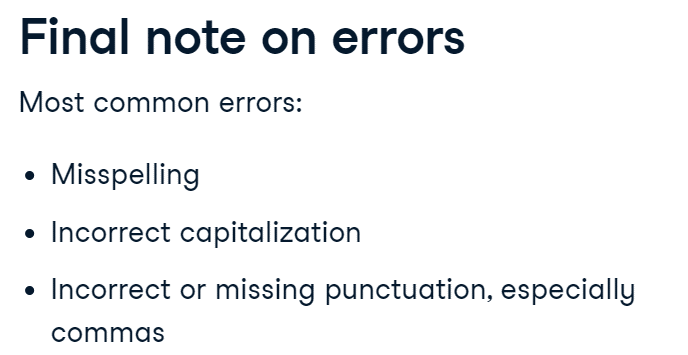
select count(distinct country) as unique\_countries from films;

-- Count the distinct countries from the films table

select count(distinct country) as count\_distinct\_countries from films;





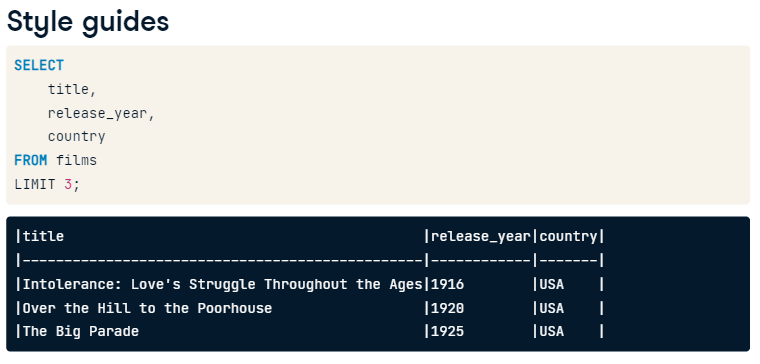


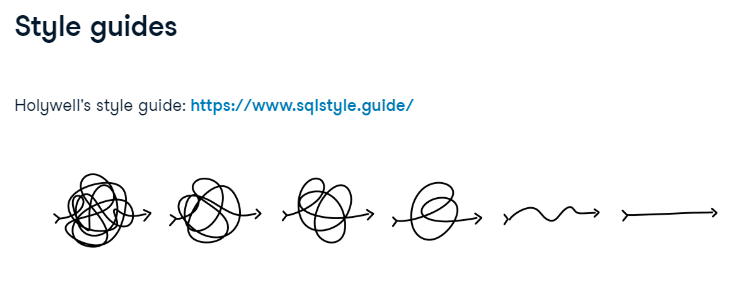
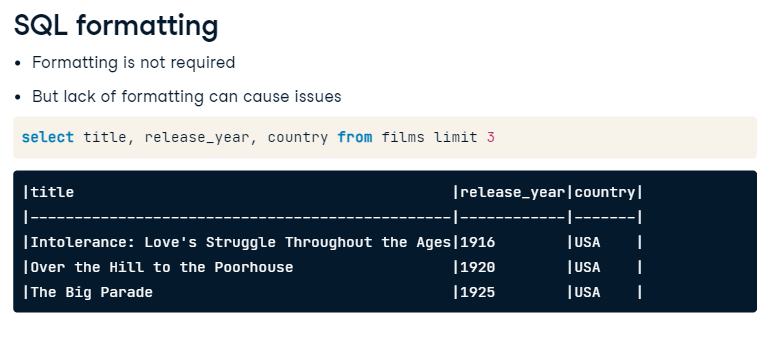
-- Debug this code

SELECT certfication

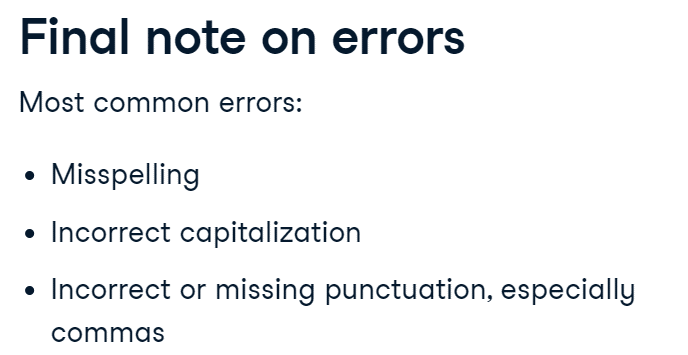
FROM films

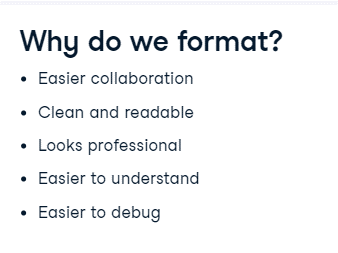
LIMIT 5;

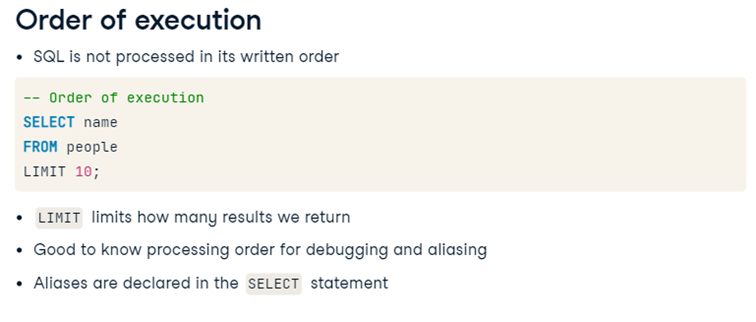


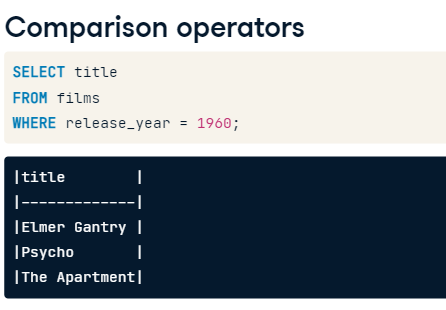
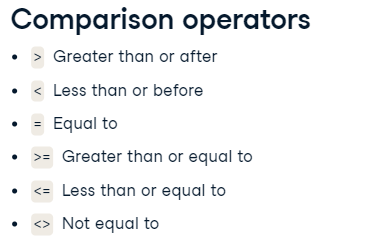
















-- Select film\_ids and facebook\_likes for ten records with less than 1000 likes

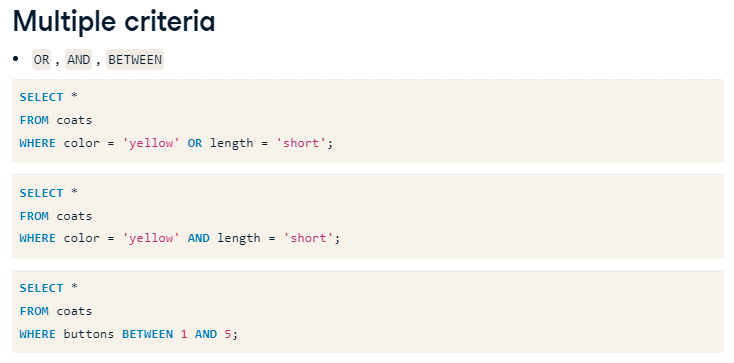
SELECT film\_id, facebook\_likes FROM reviews where facebook\_likes<1000 LIMIT 10;

-- Count the records with at least 100,000 votes

SELECT COUNT(\*) as films\_over\_100k\_votes FROM reviews where num\_votes>= 100000;

-- Count the Spanish-language films

SELECT Count(language) as count\_spanish FROM films where language = 'Spanish';





-- Select the title and release\_year for all German-language films released before 2000

SELECT title, release\_year FROM films WHERE language='German' AND release\_year<2000;

-- Select all records for German-language films released after 2000 and before 2010

SELECT \*

    FROM films

    WHERE release\_year>2000 AND release\_year<2010 AND language = 'German';

-- Find the title and year of films from the 1990 or 1999

SELECT title, release\_year

FROM films

    WHERE ( release\_year = 1990 OR release\_year =1999 ) AND

            (language='English' OR language='Spanish') AND

            (gross>2000000);

SELECT title, release\_year

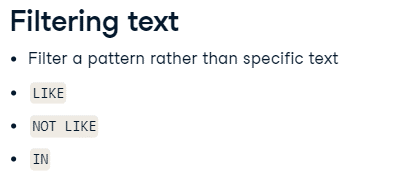
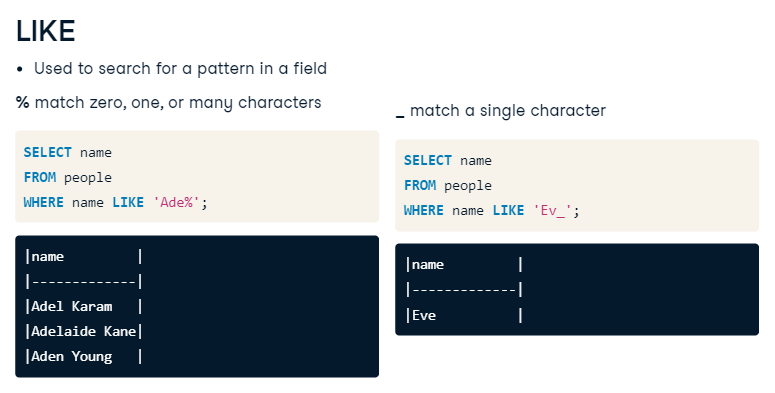
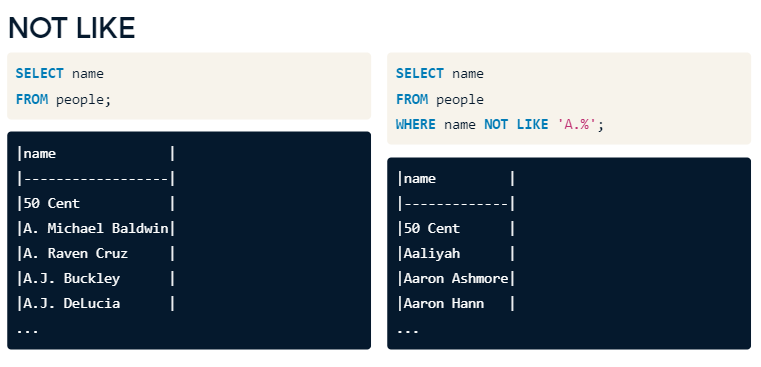
FROM films

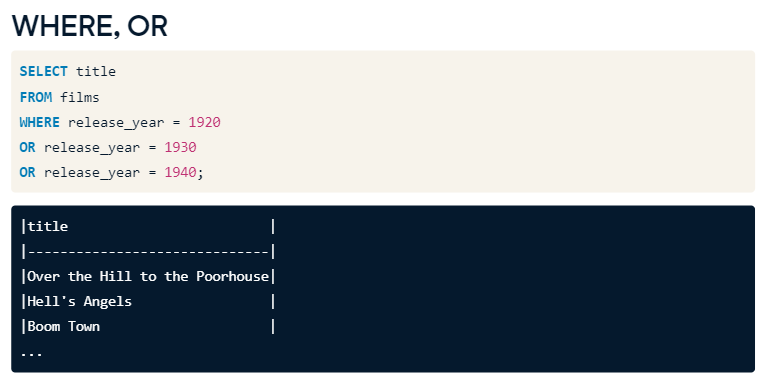
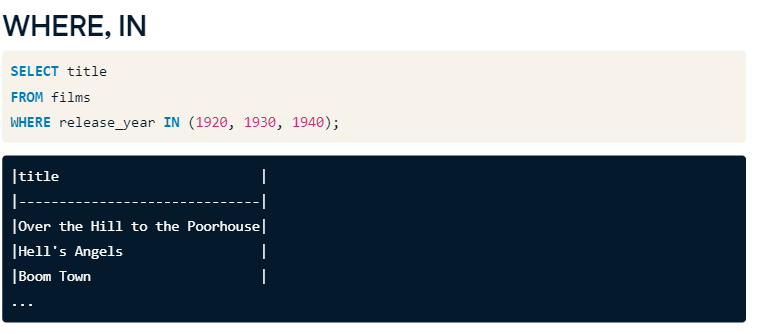
WHERE release\_year BETWEEN 1990 AND 2000

    AND budget > 100000000

-- Restrict the query to only Spanish-language films

    AND language = 'Spanish';





-- Select the names that start with B

SELECT name

    FROM people

    WHERE name LIKE 'B%';

-- Select the names that start with B

SELECT name

    FROM people

    WHERE name LIKE 'B%';

SELECT name

FROM people

-- Select the names that have r as the second letter

WHERE name LIKE '\_r';

SELECT name

FROM people

-- Select the names that have r as the second letter

WHERE name LIKE '\_r%';

SELECT name

FROM people

-- Select names that don't start with A

WHERE name NOT LIKE 'A%';

-- Find the title and release\_year for all films over two hours in length released in 1990 and 2000

SELECT title, release\_year

        FROM films

        WHERE release\_year in(1990,2000) AND duration>120;

-- Find the title and language of all films in English, Spanish, and French

SELECT title, language

    FROM films

    WHERE language in('English','Spanish','French')

-- Find the title, certification, and language all films certified NC-17 or R that are in English, Italian, or Greek

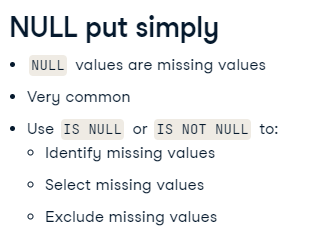
SELECT title, certification, language

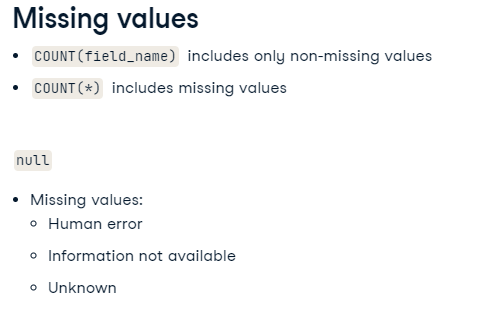
        FROM films

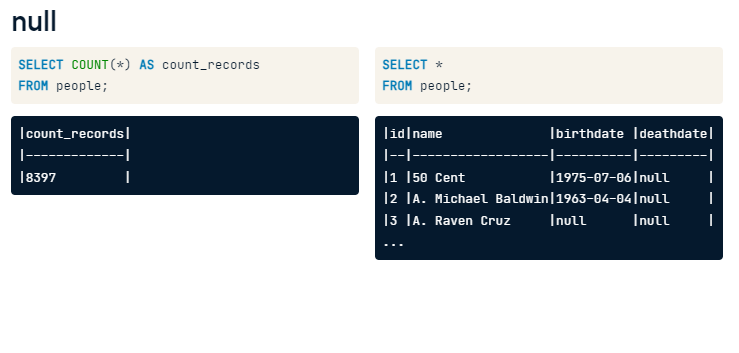
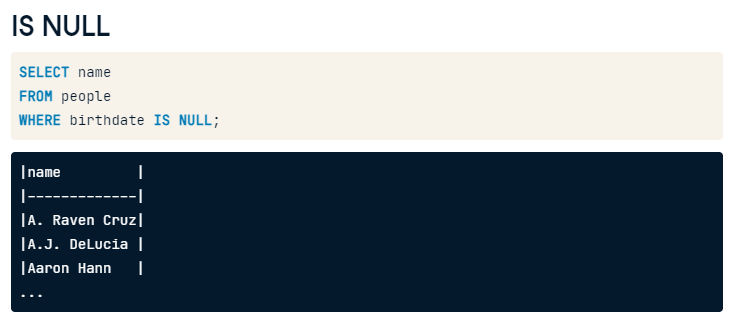
        WHERE language in('English','Italian','Greek')

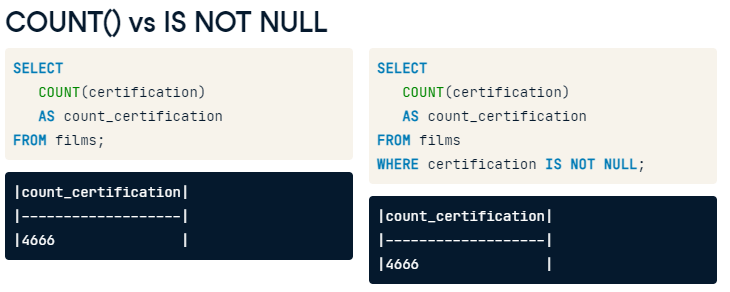
        AND certification in ('NC-17','R');

* Count the unique titles from the films database and use the alias provided.
* Filter to include only movies with a release\_year from 1990 to 1999, inclusive.
* Add another filter narrowing your query down to English-language films.
* Add a final filter to select only films with 'G', 'PG', 'PG-13' certifications.
* -- Count the unique titles
* SELECT Count(DISTINCT title) AS nineties\_english\_films\_for\_teens
* FROM films
* -- Filter to release\_years to between 1990 and 1999
* WHERE release\_year between 1990 and 1999
* -- Filter to English-language films
* AND language = 'English'
* -- Narrow it down to G, PG, and PG-13 certifications
* AND certification in ('G','PG','PG-13');

****







-- List all film titles with missing budgets

SELECT title as no\_budget\_info

    FROM films

    WHERE budget is null;

    -- List all film titles with missing budgets

-- Count the number of films we have language data for

--SELECT Count(title) as count\_language\_known1

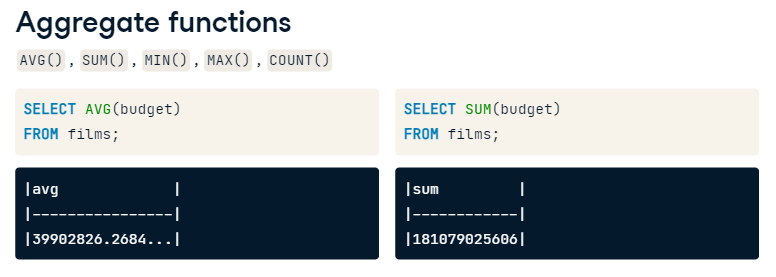
    --FROM films;

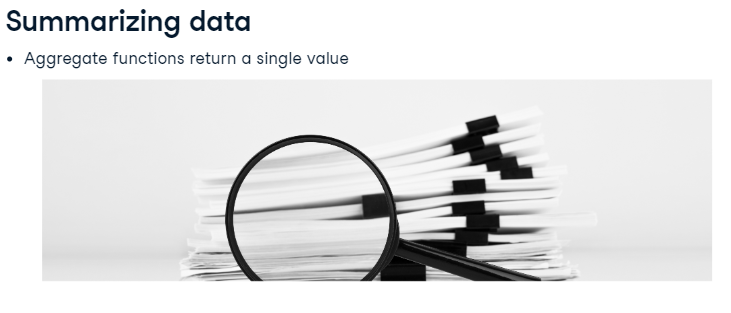
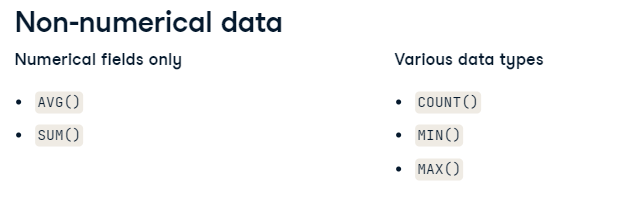
SELECT Count(title) as count\_language\_known2

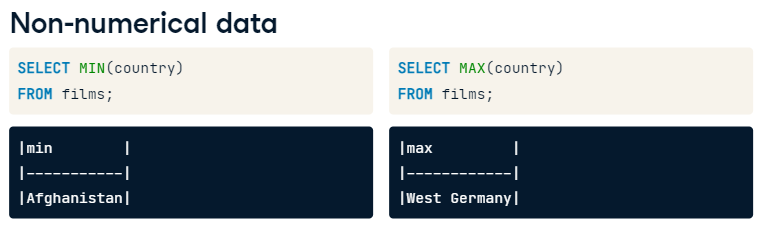
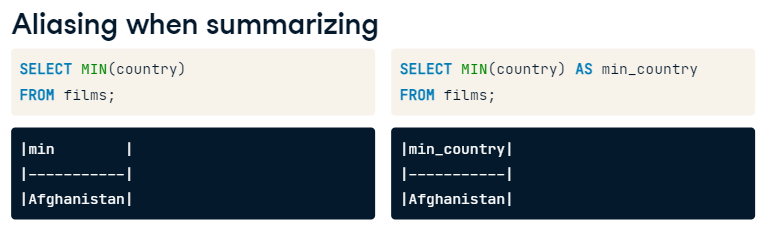
    FROM films

    WHERE language is not null;

**Aggregate functions are**

**Placed after the SELECT keyword.**



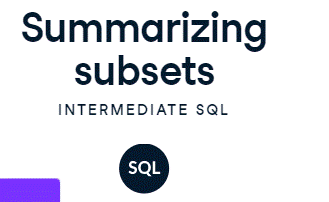


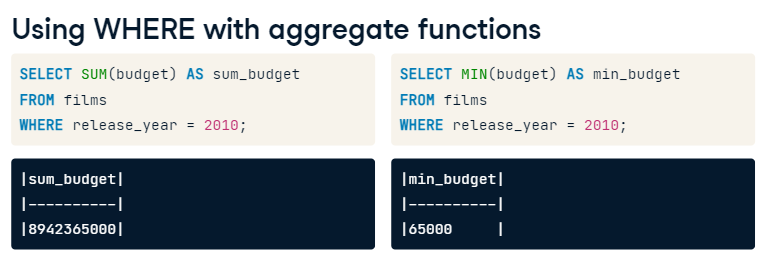
* **Use the SUM() function to calculate the total duration of all films and alias with total\_duration.**
* -- Query the sum of film durations
* SELECT sum(duration) as total\_duration
* FROM films;
* **Calculate the average duration of all films and alias with average\_duration.**
* -- Calculate the average duration of all films
* SELECT AVG(duration) as average\_duration
* FROM films;

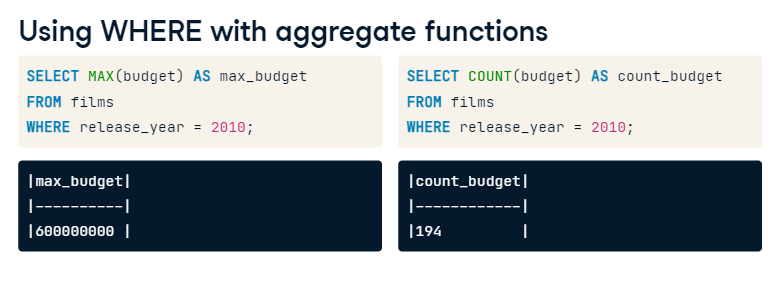
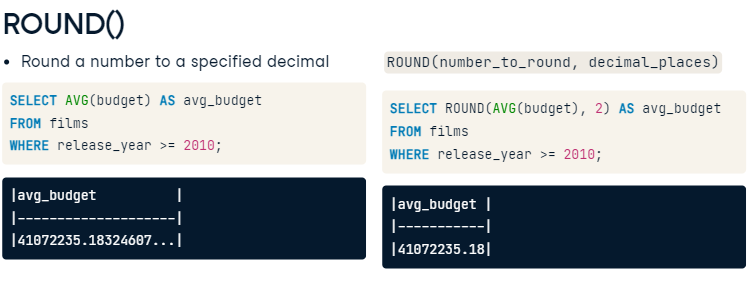
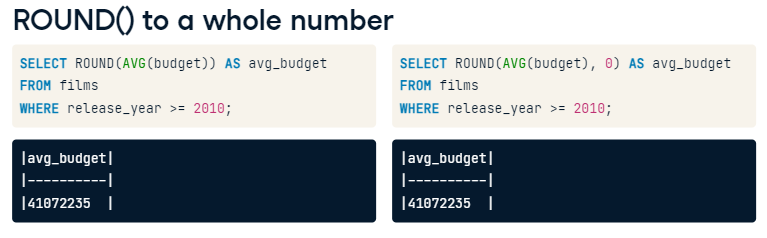
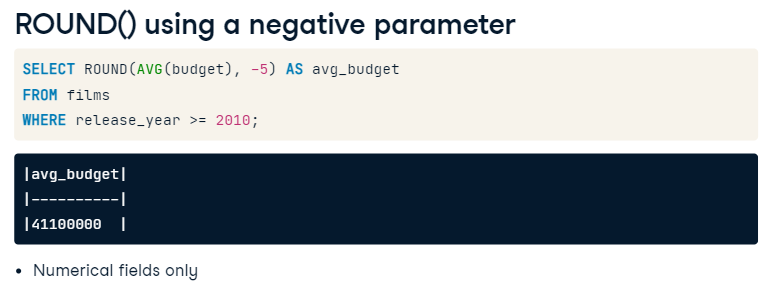
-- Find the latest release\_year

SELECT MAX(release\_year) as latest\_year

    FROM films;

* **Find the duration of the shortest film and use the alias shortest\_film.**
* -- Find the duration of the shortest film
* SELECT Min(duration) as shortest\_film
* FROM films;
* 





* Use SUM() to calculate the total gross for all films made in the year 2000 or later, and use the alias total\_gross.
* -- Calculate the sum of gross from the year 2000 or later
* SELECT SUM(gross) as total\_gross
* FROM films
* WHERE release\_year >=2000;
* Calculate the average amount grossed by all films whose titles start with the letter 'A' and alias with avg\_gross\_A.

-- Calculate the average gross of films that start with A

SELECT AVG(gross) as avg\_gross\_A

    FROM films

    WHERE title  like 'A%';

* Calculate the lowest gross film in 1994 and use the alias lowest\_gross.
* -- Calculate the lowest gross film in 1994
* SELECT MIN(gross) as lowest\_gross
* FROM films
* WHERE release\_year = 1994;
* Calculate the highest gross film between 2000 and 2012, inclusive, and use the alias highest\_gross.

-- Calculate the highest gross film released between 2000-2012

SELECT MAX(gross) as highest\_gross

    FROM films

    WHERE release\_year

    between 2000 AND 2012;

**Using ROUND()**

Aggregate functions work great with numerical values; however, these results can sometimes get unwieldy when dealing with long decimal values. Luckily, SQL provides you with the ROUND() function to tame these long decimals.

If asked to give the average budget of your films, ten decimal places is not necessary. Instead, you can round to two decimal places to create results that make more sense for currency.

Now you try!

* Calculate the average facebook\_likes to one decimal place and assign to the alias, avg\_facebook\_likes.
* -- Round the average number of facebook\_likes to one decimal place
* --WRONG ONE
* SELECT AVG(Round(facebook\_likes,1)) as avg\_facebook\_likes
* FROM reviews;

-- Round the average number of facebook\_likes to one decimal place

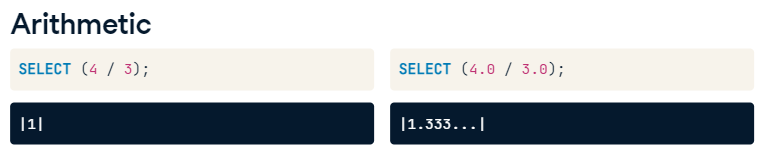
--RIGHT ONE

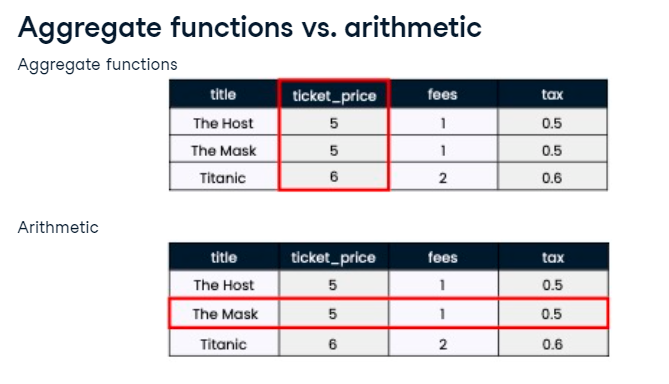
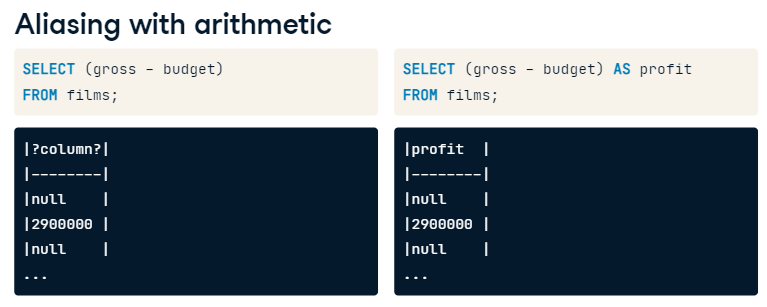
SELECT ROUND(AVG(facebook\_likes),1) as avg\_facebook\_likes

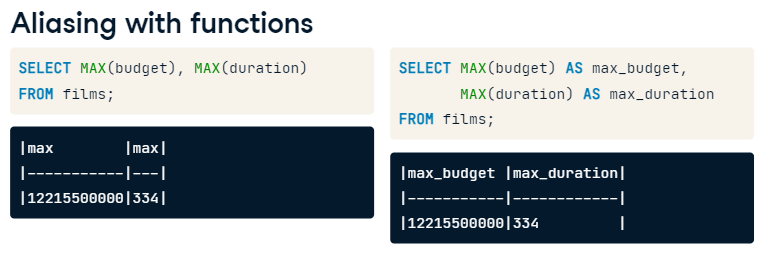
    FROM reviews;

* Calculate the average budget from the films table, aliased as avg\_budget\_thousands, and round to the nearest thousand.
* -- Calculate the average budget rounded to the thousands
* SELECT ROUND(AVG(budget)/1000)\*1000 as avg\_budget\_thousands
* FROM films;





 **Aggregate: Works Vertically, Arithmetic: Works horizontally.** 





# You have seen that SQL can act strangely when dividing integers. What is the result if you divide a discount of two dollars by the paid\_price of ten dollars to get the discount percentage?

# Aliasing with functions

Aliasing can be a lifesaver, especially as we start to do more complex SQL queries with multiple criteria. Aliases help you keep your code clean and readable. For example, if you want to find the MAX() value of several fields without aliasing, you'll end up with the result with several columns called max and no idea which is which. You can fix this with aliasing.

Now, it's over to you to clean up the following queries.

* **Select the title and duration in hours for all films and alias as duration\_hours; since the current durations are in minutes, you'll need to divide duration by 60.0.**

-- Calculate the title and duration\_hours from films

SELECT title, (duration/60) as duration\_hours

    FROM films;

Note the double and integer difference.

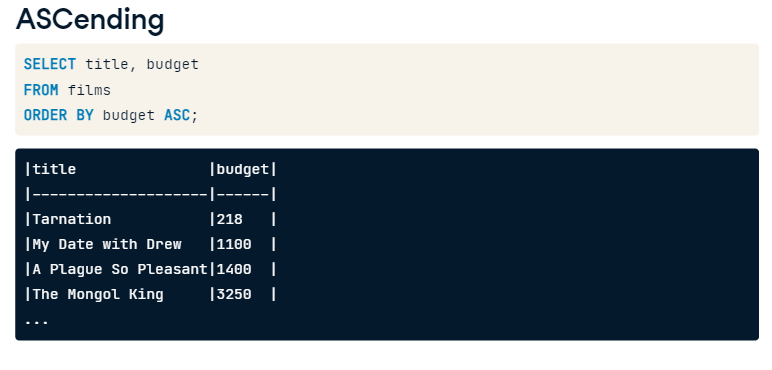
-- Calculate the title and duration\_hours from films

SELECT title, (duration/60.0) as duration\_hours

    FROM films;

* **Calculate the percentage of people who are no longer alive and alias the result as percentage\_dead.**
* -- Calculate the percentage of people who are no longer alive
* SELECT COUNT(deathdate) \* 100.0 /COUNT(\*) AS percentage\_dead
* FROM people;
* Find how many decades (period of ten years) the films table covers by using MIN() and MAX(); alias as number\_of\_decades.
* -- Find the number of decades in the films table
* SELECT (MAX(release\_year) - MIN(release\_year)) / 10 AS number\_of\_decades
* FROM films;









# Sorting single fields

Now that you understand how ORDER BY works, you'll put it into practice. In this exercise, you'll work on sorting single fields only. This can be helpful to extract quick insights such as the top-grossing or top-scoring film.

The following exercises will help you gain further insights into the film database.

* Select the name of each person in the people table, sorted alphabetically.
* -- Select name from people and sort alphabetically
* SELECT name
* FROM people
* ORDER BY name ;
* Select the title and duration for every film, from longest duration to shortest.
* -- Select the title and duration from longest to shortest film
* SELECT title, duration
* FROM films
* ORDER BY duration DESC;

# Sorting multiple fields

ORDER BY can also be used to sort on multiple fields. It will sort by the first field specified, then sort by the next, and so on. As an example, you may want to sort the people data by age and keep the names in alphabetical order.

Try using ORDER BY to sort multiple columns.

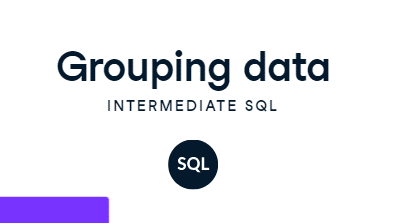
* Select the release\_year, duration, and title of films ordered by their release year and duration, in that order.
* -- Select the release year, duration, and title sorted by release year and duration
* SELECT release\_year, duration, title
* FROM films
* ORDER BY release\_year,duration;
* Select the certification, release\_year, and title from films ordered first by certification (alphabetically) and second by release year, starting with the most recent year.

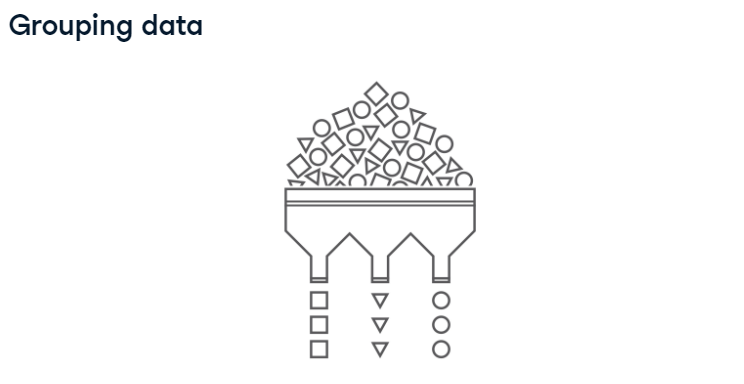
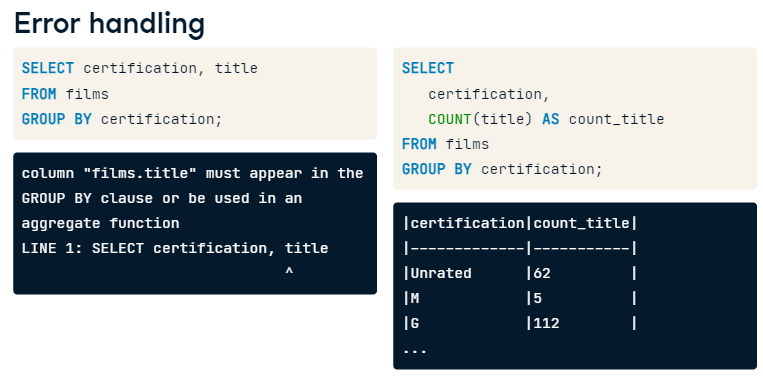
-- Select the certification, release year, and title sorted by certification and release year

Select certification, release\_year, title

    FROM films

    ORDER BY certification, release\_year DESC;







# GROUP BY single fields

GROUP BY is a SQL keyword that allows you to group and summarize results with the additional use of aggregate functions. For example, films can be grouped by the certification and language before counting the film titles in each group. This allows you to see how many films had a particular certification and language grouping.

In the following steps, you'll summarize other groups of films to learn more about the films in your database.

* Select the release\_year and count of films released in each year aliased as film\_count.
* -- Find the release\_year and film\_count of each year
* SELECT release\_year, count(title) as film\_count
* FROM films
* GROUP BY release\_year;
* Select the release\_year and average duration aliased as avg\_duration of all films, grouped by release\_year.

-- Find the release\_year and average duration of films for each year

SELECT release\_year, Round(AVG(duration),4) as avg\_duration

    FROM films

    GROUP BY release\_year;

# GROUP BY multiple fields

GROUP BY becomes more powerful when used across multiple fields or combined with ORDER BY and LIMIT.

Perhaps you're interested in learning about budget changes throughout the years in individual countries. You'll use grouping in this exercise to look at the maximum budget for each country in each year there is data available.

* Select the release\_year, country, and the maximum budget aliased as max\_budget for each year and each country; sort your results by release\_year and country.

-- Find the release\_year, country, and max\_budget, then group and order by release\_year and country

SELECT release\_year as release\_year, country as country, MAX(budget) as max\_budget

    FROM films

    group by release\_year, country

    order by release\_year, country;

# Answering business questions

In the real world, every SQL query starts with a business question. Then it is up to you to decide how to write the query that answers the question. Let's try this out.

Which release\_year had the most language diversity?

Take your time to translate this question into code. We'll get you started then it's up to you to test your queries in the console.

"Most language diversity" can be interpreted as COUNT(DISTINCT \_\_\_). Now over to you.

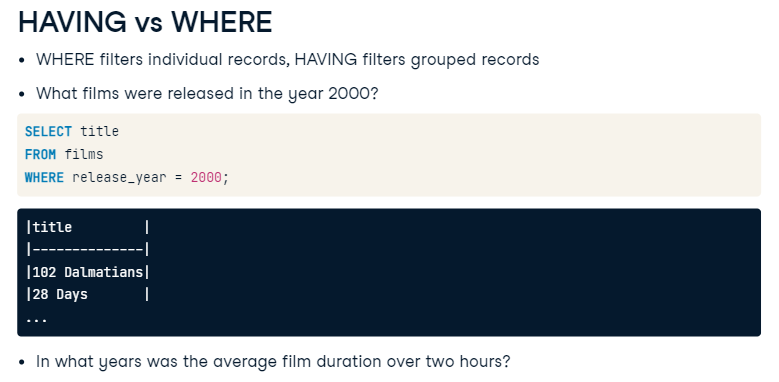
SELECT distinct release\_year, count(distinct language) as language\_diversity

    FROM films

    group by release\_year

    order by language\_diversity

    limit 1;





# Filter with HAVING

Your final keyword is HAVING. It works similarly to WHERE in that it is a filtering clause, with the difference that HAVING filters grouped data.

Filtering grouped data can be especially handy when working with a large dataset. When working with thousands or even millions of rows, HAVING will allow you to filter for just the group of data you want, such as films over two hours in length!

Practice using HAVING to find out which countries (or country) have the most varied film certifications.

* Select country from the films table, and get the distinct count of certification aliased as certification\_count.
* -- Select the country and distinct count of certification as certification\_count
* SELECT country, count(distinct certification) as certification\_count
* FROM films
* -- Group by country
* Group by country
* -- Filter results to countries with more than 10 different certifications
* HAVING count(certification) > 10 ;

State the difference

-- Select the country and distinct count of certification as certification\_count

SELECT country, count(distinct certification) as certification\_count

    FROM films

-- Group by country

    Group by country

-- Filter results to countries with more than 10 different certifications

    HAVING count(distinct certification) > 10 ;

# HAVING and sorting

Filtering and sorting go hand in hand and gives you greater interpretability by ordering our results.

Let's see this magic at work by writing a query showing what countries have the highest average film budgets.

* Select the country and the average budget as average\_budget, rounded to two decimal, from films.
* Group the results by country.
* Filter the results to countries with an average budget of more than one billion (1000000000).
* Sort by descending order of the average\_budget.
* -- Select the country and average\_budget from films
* SELECT country, Round(avg(budget),2) as average\_budget
* FROM films
* -- Group by country
* Group by country
* -- Filter to countries with an average\_budget of more than one billion
* Having AVG(average\_budget) > 1000000000
* -- Order by descending order of the aggregated budget
* ORDER by average\_budget desc ;

-- Select the country and average\_budget from films

SELECT country, Round(avg(budget),2) as average\_budget

        FROM films

-- Group by country

    Group by country

-- Filter to countries with an average\_budget of more than one billion

    Having AVG(budget) > 1000000000

-- Order by descending order of the aggregated budget

    ORDER by average\_budget desc ;

# All together now

It's time to use much of what you've learned in one query! This is good preparation for using SQL in the real world where you'll often be asked to write more complex queries since some of the basic queries can be answered by playing around in spreadsheet applications.

In this exercise, you'll write a query that returns the average budget and gross earnings for films each year after 1990 if the average budget is greater than 60 million.

This will be a big query, but you can handle it!

* Select the release\_year for each film in the films table, filter for records released after 1990, and group by release\_year.

-- Select the release\_year for films released after 1990 grouped by year

SELECT release\_year

    FROM films

    Group by release\_year

    having release\_year>1990;

* Modify the query to include the average budget aliased as avg\_budget and average gross aliased as avg\_gross for the results we have so far.
* -- Modify the query to also list the average budget and average gross
* SELECT release\_year, AVG(budget) as average\_budget, AVG(gross) as avg\_gross
* FROM films
* WHERE release\_year > 1990
* GROUP BY release\_year;

-- Modify the query to also list the average budget and average gross

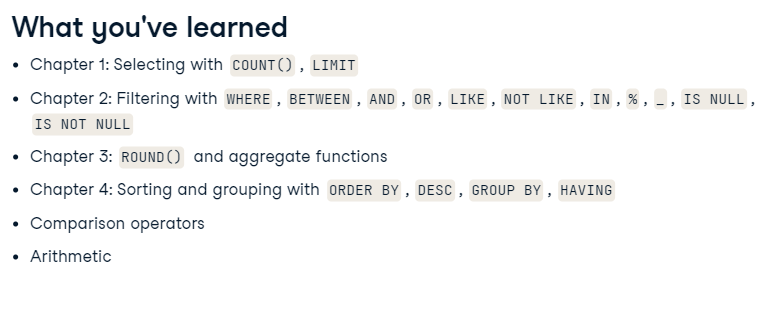
SELECT release\_year, AVG(budget) as avg\_budget, AVG(gross) as avg\_gross

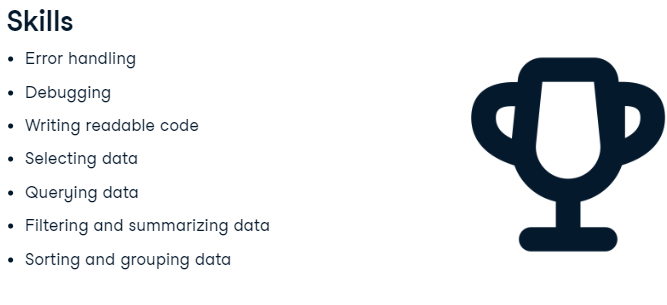
FROM films

WHERE release\_year > 1990

GROUP BY release\_year;

* Modify the query once more so that only years with an average budget of greater than 60 million are included.
* SELECT release\_year, AVG(budget) AS avg\_budget, AVG(gross) AS avg\_gross
* FROM films
* WHERE release\_year > 1990
* GROUP BY release\_year
* -- Modify the query to see only years with an avg\_budget of more than 60 million
* Having Avg(budget)>=60000000;
* Finally, order the results from the highest average gross and limit to one.
* SELECT release\_year, AVG(budget) AS avg\_budget, AVG(gross) AS avg\_gross
* FROM films
* WHERE release\_year > 1990
* GROUP BY release\_year
* HAVING AVG(budget) > 60000000
* -- Order the results from highest to lowest average gross and limit to one
* ORDER BY avg\_gross desc
* limit 1;







Joining Data In SQL(09/09/23)