SQL basics

LIMIT

- If you only want to return a certain number of results, you can use the LIMIT keyword to limit the number of rows returned:

```
SELECT *
FROM people
LIMIT 10;
```

DISTINCT

- If you want to select all the unique values from a column, you can use the DISTINCT keyword.

```
SELECT DISTINCT language
FROM films;
```

- Similar to .unique method of pandas.

COUNT()

- The COUNT() function lets you do this by returning the number of rows in one or more columns.

For example, this code gives the number of rows in the people table:

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

- If you want to count the number of *non-missing* values in a particular column, you can call COUNT() on just that column.

```
SELECT COUNT(birthdate)
FROM people;
SELECT COUNT(DISTINCT birthdate)
FROM people;
```

WHERE

- WHERE keyword allows you to filter based on both text and numeric values in a table. There are a few different comparison operators you can use:

- = equal
- **<> not equal**
- < less than
- > greater than
- <= less than or equal to
- >= greater than or equal to
- AND
- OR

- When combining AND and OR, be sure to enclose the individual clauses in parentheses, like so:

–

```
SELECT title
FROM films
WHERE (release_year = 1994 OR release_year = 1995)
AND (certification = 'PG' OR certification = 'R');
```

BETWEEN

- you can use the following query to get titles of all films released in and between 1994 and 2000:

```
SELECT title
FROM films
WHERE release_year >= 1994
AND release_year <= 2000;
```

Checking for ranges like this is very common, so in SQL the BETWEEN keyword provides a useful shorthand for filtering values within a specified range. This query is equivalent to the one above:

```
SELECT title
FROM films
WHERE release_year
BETWEEN 1994 AND 2000;
```

It's important to remember that BETWEEN is *inclusive*, meaning the beginning and end values are included in the results!

IN

- The IN operator allows you to specify multiple values in a WHERE clause, making it easier and quicker to specify multiple OR conditions! Neat, right?

```
SELECT name
FROM kids
WHERE age IN (2, 4, 6, 8, 10);
```

NULL

- In SQL, NULL represents a missing or unknown value. You can check for NULL values using the expression IS NULL. For example, to count the number of missing birth dates in the people table:

```
SELECT COUNT(*)
FROM people
WHERE birthdate IS NULL;
```

Sometimes, you'll want to filter out missing values so you only get results which are not NULL. To do this, you can use the IS NOT NULL operator.

LIKE
In SQL, the LIKE operator can be used in a WHERE clause to **search for a pattern in a column.** To accomplish this, you use something called a ***wildcard*** as a placeholder for some other values. There are two wildcards you can use with LIKE:
The **% wildcard** will match zero, one, or many characters in text. For example, the following query matches companies like 'Data', 'DataC' 'DataCamp', 'DataMind', and so on:
SELECT name
FROM companies
WHERE name LIKE 'Data%';

The **_ wildcard** will match a *single* character. For example, the following query matches companies like 'DataCamp', 'DataComp', and so on:
SELECT name
FROM companies
WHERE name LIKE 'DataC_mp';
You can also use the **NOT LIKE** operator to find records that *don't* match the pattern you specify.

select name
from people
where name not like 'A%'; - Selecting all rows in which name does not start with A

Aggregate functions
aggregate functions are not allowed in WHERE, instead we have to make use of nested queries

SELECT **AVG(budget)**
FROM films;
gives you the average value from the budget column of the films table.

MAX() function returns the highest budget:
SELECT MAX(budget)
FROM films;

The SUM() function returns the result of adding up the numeric values in a column:
SELECT SUM(budget)
FROM films;

SIMIALR to df[df['duration'] == df[duration].min()]
SELECT *

FROM films
WHERE duration = (Select min(duration) from films);

SELECT max(gross)
from films
where release_year between 2000 and 2012;

——————————————

A note on arithmetic

In addition to using aggregate functions, you can perform basic arithmetic with symbols like +, -, *, and /.
So, for example, this gives a result of 12:
SELECT (4 * 3);

SELECT (4 / 3); - Gives us 1.
SELECT (4.0 / 3.0); - Gives us 1.333.

——————————————————

Aliasing

SELECT MAX(budget), MAX(duration)
FROM films;
In the above query both of the columns will be named Max.
To solve this we can make use of Aliasing.
SELECT MAX(budget) AS max_budget,
    MAX(duration) AS max_duration
FROM films;

Select title, (gross - budget) as net_profit
from films;
——————————————————-

ORDER BY

- the ORDER BY keyword is used to sort results in ascending or descending order according to the values of one or more columns.
- By default ORDER BY will sort in ascending order. If you want to sort the results in descending order, you can use the **DESC** keyword.

SELECT title
FROM films
ORDER BY release_year DESC;

- Get the title of films ordered/sorted by the release year in descending order.

Sorting multiple columns

- ORDER BY can also be used to sort on multiple columns. It will sort by the first column specified, then sort by the next, then the next, and so on. For example,
- the second column you order on only steps in when the first column is not decisive to tell the order. The second column acts as a tie breaker.

```
SELECT birthdate, name
FROM people
ORDER BY birthdate, name;
```

GROUP BY

- We might want to count the number of male and female employees in your company. Here, what we would want to do is to group all the males together and count them, and group all the females together and count them.
- In SQL, GROUP BY allows you to group a result by one or more columns, like so:

```
SELECT sex, count(*)
FROM employees
GROUP BY sex;
```

Output:

sex	count
male	15
female	19

- Commonly, GROUP BY is used with *aggregate functions* like COUNT() or MAX(). Note that GROUP BY always goes after the FROM clause!

Get the country, release year, and lowest amount grossed per release year per country. Order your results by country and release year.

```
select country, release_year, min(gross)
from films
group by release_year, country
order by country, release_year
```

HAVING

- aggregate functions can't be used in WHERE clauses, to filter based on the where clause, we need another way and that way is the way of the having clause.

```
SELECT release_year
FROM films
```

GROUP BY release_year
HAVING COUNT(title) > 10;

- **Returns only those years in which there were more than 10 films released.**

Q: In how many different years were more than 200 movies released?
select release_year, count(*)
from films
group by release_year
having count(*) > 200;
Explanation: First break the question, in how many different years, here this implies making use of group by clause to group the films by their release_years, then add a count() aggregator function to count the number of movies made for each year, this will return a result where we have movie years and number of movies made for this movie year, to get the number of movies having more than 200 films we will make use of the having clause, this acts like a filter.

———————————————————

PostgreSQL EXTRACT Function

The PostgreSQL EXTRACT() function retrieves a field such as a year, month, and day from a date/time value.
source
The source is a value of type TIMESTAMP or INTERVAL. If you pass a DATE value, the function will cast it to a TIMESTAMP value.

Return value

The EXTRACT() function returns a double precision value.

———————————————————

Note: count when called on column containing null values, returns the number of non zero column
count(date) will return the number of non null date rows
count(*) will return the number of rows.

————————————————————————