QUERIES

Introduction

In this lesson, we will be learning different SQL commands to **query** a single table in a database.
One of the core purposes of the SQL language is to retrieve information stored in a database. This is commonly referred to as querying. Queries allow us to communicate with the database by asking questions and having the result set return data relevant to the question.
We will be querying a database with one table named `movies`.

Let's get started!
Fun fact: IBM started out SQL as SEQUEL (**S**tructured **E**nglish **QUE**ry **L**anguage) in the 1970's to query databases.

1. We should get acquainted with the `movies` table. In the editor, type the following:

```
SELECT * FROM movies;
```

What are the column names?

Select

Previously, we learned that `SELECT` is used every time you want to query data from a database and `*` means *all* columns. Suppose we are only interested in two of the columns. We can select individual columns by their names (separated by a comma):

```
SELECT column1, column2
FROM table_name;
```

To make it easier to read, we moved `FROM` to another line. Line breaks don't mean anything specific in SQL. We could write this entire query in one line, and it would run just fine.

1. Let's only select the `name` and `genre` columns of the table. In the code editor, type the following:

```
SELECT name, genre
FROM movies;
```

2. Now we want to include a third column. Edit your query so that it returns the name, genre, and year columns of the table.

```
SELECT name, genre, year
FROM movies;
```

As

Knowing how `SELECT` works, suppose we have the code below:

```
SELECT name AS 'Titles'
FROM movies;
```

Can you guess what `AS` does? `AS` is a keyword in SQL that allows you to *rename* a column or table using an alias. The new name can be anything you want as long as you put it inside of single quotes. Here we renamed the `name` column as `Titles`.

Some important things to note:

- Although it's not always necessary, it's best practice to surround your aliases with single quotes.
- When using `AS`, the columns are not being renamed in the table. The aliases only appear in the result.

1. To showcase what the `AS` keyword does, select the `name` column and rename it with an alias of your choosing. Place the alias inside single quotes, like so:

```
SELECT name AS '_____'
FROM movies;
```

Note in the result, that the name of the column is now your alias.

2. Edit the query so that instead of selecting and renaming the `name` column, select the `imdb_rating` column and rename it as `IMDb`.

```
SELECT imdb_rating AS 'IMDb'
FROM movies;
```

Distinct

When we are examining data in a table, it can be helpful to know what *distinct* values exist in a particular column. `DISTINCT` is used to return unique values in the output. It filters out all duplicate values in the specified column(s).

For instance,

```
SELECT tools
FROM inventory;
```

might produce:

tools
Hammer
Nails
Nails
Nails

By adding `DISTINCT` before the column name,

```
SELECT DISTINCT tools
FROM inventory;
```

the result would now be:

tools
Hammer
Nails

Filtering the results of a query is an important skill in SQL. It is easier to see the different possible `genre`s in the `movie` table after the data has been filtered than to scan every row in the table.

1. Let's try it out. In the code editor, type:

```
SELECT DISTINCT genre
FROM movies;
```

What are the unique genres?

3. Now, change the code so we return the unique values of the `year` column instead.

```
SELECT DISTINCT year
FROM movies;
```

Where

We can restrict our query results using the `WHERE` clause in order to obtain only the information we want. Following this format, the statement below filters the result set to only include top rated movies (IMDb ratings greater than 8):

```
SELECT *
FROM movies
WHERE imdb_rating > 8;
```

How does it work?

1. `WHERE` clause filters the result set to only include rows where the following *condition* is true.
2. `imdb_rating > 8` is the condition. Here, only rows with a value greater than 8 in the `imdb_rating` column will be returned.

The `>` is an *operator*. Operators create a condition that can be evaluated as either *true* or *false*.
Comparison operators used with the `WHERE` clause are:

- `=` equal to
- `!=` not equal to
- `>` greater than
- `<` less than

- `>=` greater than or equal to
- `<=` less than or equal to

There are also some special operators that we will learn more about in the upcoming exercises.

1. Suppose we want to take a peek at all the not-so-well-received movies in the database.
In the code editor, type:

```
SELECT *
FROM movies
WHERE imdb_rating < 5;
```

Ouch!

2. Edit the query so that it will now retrieve all the recent movies, specifically those that were released after 2014. Select all the columns using `*`.

```
SELECT *
FROM movies
WHERE year > 2014;
```

Like I

`LIKE` can be a useful operator when you want to compare similar values. The `movies` table contains two films with similar titles, 'Se7en' and 'Seven'. How could we select all movies that start with 'Se' and end with 'en' and have exactly one character in the middle?

```
SELECT *
FROM movies
WHERE name LIKE 'Se_en';
```

- `LIKE` is a special operator used with the `WHERE` clause to search for a specific pattern in a column.
- `name LIKE 'Se_en'` is a condition evaluating the `name` column for a specific pattern.
- `Se_en` represents a pattern with a *wildcard* character.

The `_` means you can substitute any individual character here without breaking the pattern. The names `Seven` and `Se7en` both match this pattern.

1. Let's test it out. In the code editor, type:

```
SELECT *
FROM movies
WHERE name LIKE 'Se_en';
```

Like II

The percentage sign `%` is another wildcard character that can be used with `LIKE`. This statement below filters the result set to only include movies with names that begin with the letter 'A':

```
SELECT *
FROM movies
WHERE name LIKE 'A%';
```

`%` is a wildcard character that matches zero or more missing letters in the pattern. For example:

- `A%` matches all movies with names that begin with letter 'A'
- `%a` matches all movies that end with 'a'

We can also use `%` both before and after a pattern:

```
SELECT *
FROM movies
WHERE name LIKE '%man%';
```

Here, any movie that *contains* the word 'man' in its name will be returned in the result.
`LIKE` is not case sensitive. 'Batman' and 'Man of Steel' will both appear in the result of the query above.

1. In the text editor, type:

```
SELECT *
FROM movies
WHERE name LIKE '%man%';
```

How many movie titles contain the word 'man'?

2. Let's try one more. Edit the query so that it selects all the information about the movie titles that *begin* with the word 'The'. You might need a space in there!

```
SELECT *
FROM movies
WHERE name LIKE 'The %';
```

Is Null

By this point of the lesson, you might have noticed that there are a few missing values in the `movies` table. More often than not, the data you encounter will have missing values.
Unknown values are indicated by `NULL`.

It is not possible to test for `NULL` values with comparison operators, such as `=` and `!=`.
Instead, we will have to use these operators:

- `IS NULL`
- `IS NOT NULL`

To filter for all movies *with* an IMDb rating:

```
SELECT name
FROM movies
WHERE imdb_rating IS NOT NULL;
```

1. Now let's do the opposite. Write a query to find all the movies *without* an IMDb rating. Select only the `name` column!

```
SELECT name
FROM movies
WHERE imdb_rating IS NULL;
```

Between

The `BETWEEN` operator is used in a `WHERE` clause to filter the result set within a certain *range*. It accepts two values that are either numbers, text or dates. For example, this statement filters the result set to only include movies with `year`s from 1990 up to, *and including* 1999.

```
SELECT *
FROM movies
WHERE year BETWEEN 1990 AND 1999;
```

When the values are text, `BETWEEN` filters the result set for within the alphabetical range. In this statement, `BETWEEN` filters the result set to only include movies with `name`s that begin with the letter 'A' up to, *but not including* ones that begin with 'J'.

```
SELECT *
FROM movies
WHERE name BETWEEN 'A' AND 'J';
```

However, if a movie has a name of simply 'J', it would actually match. This is because `BETWEEN` goes *up to* the second value — up to 'J'. So the movie named 'J' would be included in the result set but not 'Jaws'.

1. Using the `BETWEEN` operator, write a query that selects all information about movies whose `name` begins with the letters 'D', 'E', and 'F'.

```
SELECT *
FROM movies
WHERE name BETWEEN 'D' AND 'G';
```

2. Remove the previous query. Using the `BETWEEN` operator, write a new query that selects all information about movies that were released in the 1970's.

```
SELECT *
FROM movies
WHERE year BETWEEN 1970 AND 1979;
```

And

Sometimes we want to *combine multiple conditions* in a `WHERE` clause to make the result set more specific and useful. One way of doing this is to use the `AND` operator. Here, we use the `AND` operator to only return 90's romance movies.

```
SELECT *
FROM movies
WHERE year BETWEEN 1990 AND 1999
   AND genre = 'romance';
```

- `year BETWEEN 1990 AND 1999` is the 1st condition.
- `genre = 'romance'` is the 2nd condition.
- `AND` combines the two conditions.

With `AND`, *both* conditions must be true for the row to be included in the result.

1. In the previous exercise, we retrieved every movie released in the 1970's. Now, let's retrieve every movie released in the 70's, that's also well received. In the code editor, type:

```
SELECT *
FROM movies
WHERE year BETWEEN 1970 AND 1979
  AND imdb_rating > 8;
```

2. Remove the previous query. Suppose we have a picky friend who only wants to watch old horror films. Using `AND`, write a new query that selects all movies made prior to 1985 that are also in the `horror` genre.

```
SELECT *
FROM movies
WHERE year < 1985
AND genre = 'horror';
```

Or

Similar to `AND`, the `OR` operator can also be used to combine multiple conditions in `WHERE`, but there is a fundamental difference:

- `AND` operator displays a row if *all* the conditions are true.
- `OR` operator displays a row if *any* condition is true.

Suppose we want to check out a new movie or something action-packed:

```
SELECT *
FROM movies
WHERE year > 2014
   OR genre = 'action';
```

- `year > 2014` is the 1st condition.
- `genre = 'action'` is the 2nd condition.
- `OR` combines the two conditions.

With `OR`, if *any* of the conditions are true, then the row is added to the result.

1. Let's test this out:

```
SELECT *
FROM movies
WHERE year > 2014
   OR genre = 'action';
```

2. Suppose we are in the mood for a good laugh or a good cry. Using `OR`, write a query that returns all movies that are either a romance or a comedy.

```
SELECT *
FROM movies
WHERE genre = 'romance'
   OR genre = 'comedy';
```

Order By

That's it with `WHERE` and its operators. Moving on! It is often useful to list the data in our result set in a particular order. We can *sort* the results using `ORDER BY`, either alphabetically or numerically. Sorting the results often makes the data more useful and easier to analyze. For example, if we want to sort everything by the movie's title from A through Z:

```
SELECT *
FROM movies
ORDER BY name;
```

- `ORDER BY` is a clause that indicates you want to sort the result set by a particular column.
- `name` is the specified column.

Sometimes we want to sort things in a decreasing order. For example, if we want to select all of the well-received movies, sorted from highest to lowest by their year:

```
SELECT *
FROM movies
WHERE imdb_rating > 8
ORDER BY year DESC;
```

- `DESC` is a keyword used in `ORDER BY` to sort the results in *descending order* (high to low or Z-A).
- `ASC` is a keyword used in `ORDER BY` to sort the results in *ascending* order (low to high or A-Z).

The column that we `ORDER BY` doesn't even have to be one of the columns that we're displaying.
Note: `ORDER BY` always goes after `WHERE` (if `WHERE` is present).

1. Suppose we want to retrieve the `name` and `year` columns of all the movies, ordered by their name alphabetically. Type the following code:

```
SELECT name, year
FROM movies
ORDER BY name;
```

2. Your turn! Remove the previous query. Write a new query that retrieves the `name`, `year`, and `imdb_rating` columns of all the movies, ordered highest to lowest by their ratings.

Limit

We've been working with a fairly small table (fewer than 250 rows), but most SQL tables contain hundreds of thousands of records. In those situations, it becomes important to cap the number of rows in the result. For instance, imagine that we just want to see a few examples of records.

```
SELECT *
FROM movies
LIMIT 10;
```

`LIMIT` is a clause that lets you specify the maximum number of rows the result set will have. This saves space on our screen and makes our queries run faster. Here, we specify that the result set can't have more than 10 rows. `LIMIT` always goes at the very end of the query. Also, it is not supported in all SQL databases.

1. Combining your knowledge of `LIMIT` and `ORDER BY`, write a query that returns the top 3 highest rated movies. Select all the columns.

```
SELECT *
FROM movies
ORDER BY imdb_rating DESC
LIMIT 3;
```

Case

A `CASE` statement allows us to create different outputs (usually in the `SELECT` statement). It is SQL's way of handling if-then logic.

Suppose we want to condense the ratings in `movies` to three levels:

- *If the rating is above 8, then it is Fantastic.*
- *If the rating is above 6, then it is Poorly Received.*
- *Else, Avoid at All Costs.*

```
SELECT name,
 CASE
  WHEN imdb_rating > 8 THEN 'Fantastic'
  WHEN imdb_rating > 6 THEN 'Poorly Received'
  ELSE 'Avoid at All Costs'
 END
FROM movies;
```

- Each `WHEN` tests a condition and the following `THEN` gives us the string if the condition is true.
- The `ELSE` gives us the string if *all* the above conditions are false.
- The `CASE` statement must end with `END`.

In the result, you have to scroll right because the column name is very long. To shorten it, we can rename the column to 'Review' using `AS`:

```
SELECT name,
 CASE
  WHEN imdb_rating > 8 THEN 'Fantastic'
  WHEN imdb_rating > 6 THEN 'Poorly Received'
  ELSE 'Avoid at All Costs'
 END AS 'Review'
FROM movies;
```

1. Let's try one on your own. Select the `name` column and use a `CASE` statement to create the second column that is:

   - 'Chill' if `genre = 'romance'`
   - 'Chill' if `genre = 'comedy'`
   - 'Intense' in all other cases

Optional: Rename the whole `CASE` statement to 'Mood' using `AS`.

```
SELECT name,
 CASE
  WHEN genre = 'romance' THEN 'Chill'
  WHEN genre = 'comedy' THEN 'Chill'
  ELSE 'Intense'
 END AS 'Mood'
FROM movies;
```