**Queries**

**Introduction**

2 min

In this lesson, we will be learning different SQL

Preview: Docs Loading link description

[commands](https://www.codecademy.com/resources/docs/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 returning a result set with 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.

**Instructions**

1. Checkpoint 1 Passed

**1.**

We should get acquainted with the movies table.

In the editor, type the following:

SELECT \* FROM movies;

Copy to Clipboard

What are the column names?

Hint

Suppose we want to see all values in a table. We will use the syntax:

SELECT \* FROM table\_name;

Copy to Clipboard

* 1. SELECT \* indicates that we want to *select* *all* columns.
  2. FROM table\_name indicates which table we are interested in.
  3. ; ends a SQL command.

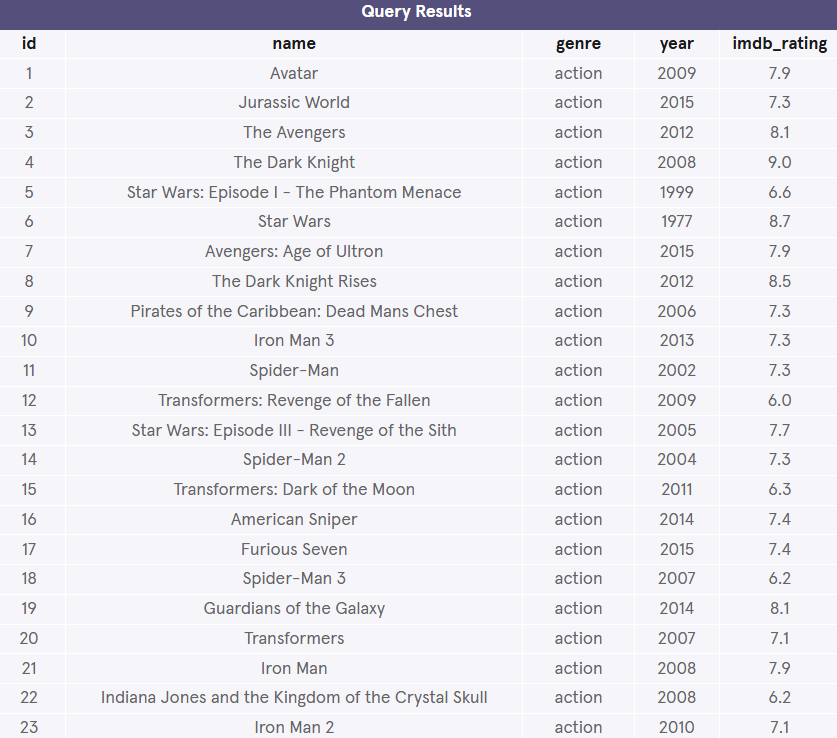
Take a look at the result table:

* 1. Scroll right to look at the columns.
  2. Scroll down to look at the rows.

The columns are id, name, genre, year, and imdb\_rating.

**query.sqlite**

SELECT \* FROM movies;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Select**

2 min

Previously, we learned that

Preview: Docs Loading link description

[SELECT](https://www.codecademy.com/resources/docs/sql/commands/select?page_ref=catalog)

 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;

Copy to Clipboard

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.

**Instructions**

1. Checkpoint 1 Passed

**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;

Copy to Clipboard

Hint

Instead of selecting all the columns:

SELECT \*  
FROM movies;

Copy to Clipboard

We are now just selecting the name and genre column:

SELECT name, genre  
FROM movies;

Copy to Clipboard

Double-check your query, character by character:

* + There is a comma between name and genre.
  + SQL commands end with a ;.

SELECT and FROM are clauses. Clauses perform specific tasks in SQL. By convention, clauses are written in capital letters.

**select.sqlite**

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.

Hint

The syntax for selecting three individual columns:

SELECT column1, column2, column3  
FROM table\_name;

Copy to Clipboard

Following this format, the query below selects name, genre, and year columns (in that order) from the movies table:

SELECT name, genre, year  
FROM movies;

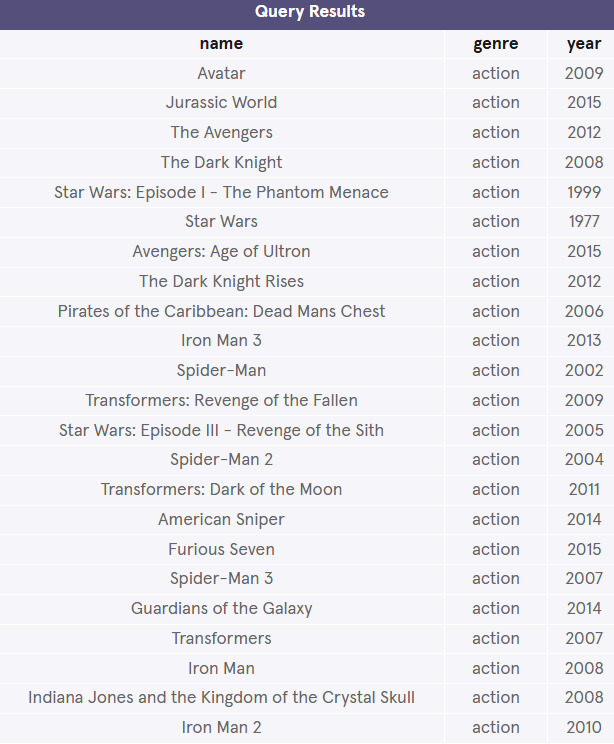
Copy to Clipboard

* We have to separate the column names with a comma.
* SQL commands end with a ;.

**select.sqlite**

SELECT name, genre, year

FROM movies;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**As**

4 min

Knowing how

Preview: Docs Loading link description

[SELECT](https://www.codecademy.com/resources/docs/sql/commands/select?page_ref=catalog)

 works, suppose we have the code below:

SELECT name AS 'Titles'  
FROM movies;

Copy to Clipboard

Can you guess what AS does?

Preview: Docs Loading link description

[AS](https://www.codecademy.com/resources/docs/sql/commands/as?page_ref=catalog)

 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 is considered best practice to surround your aliases with single quotes.
  + Note that this practice is specific to SQLite, the RDBMS used in this exercise. When you work with other RDBMSs, notably PostgreSQL, [no quotes or double quotes](https://www.postgresql.org/docs/current/queries-select-lists.html#QUERIES-COLUMN-LABELS) may be required in place of single quotes.
* When using AS, the columns are not being renamed in the table. The aliases only appear in the result.

**Instructions**

1. Checkpoint 1 Passed

**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;

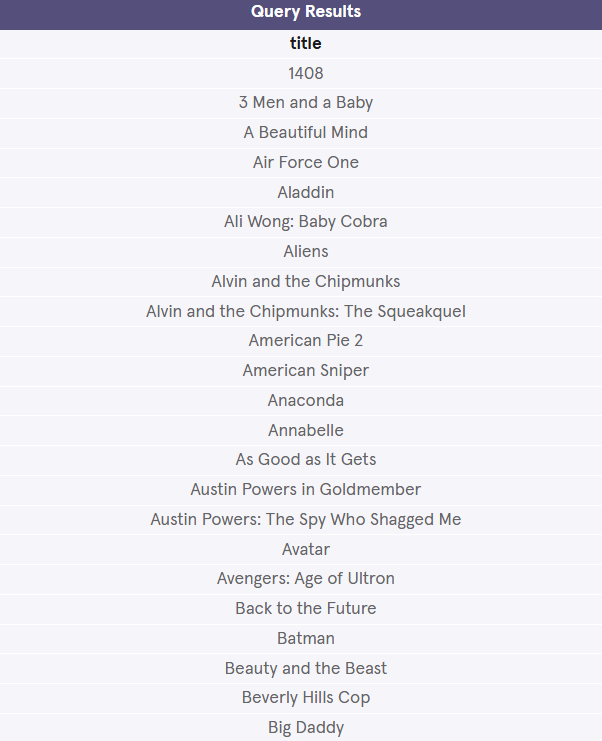
Copy to Clipboard

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

**alias.sqlite**

SELECT name AS 'title'

FROM movies;



**2.**

Edit the query so that instead of selecting and renaming the name column, select the imdb\_rating column and rename it as IMDb.

Hint

The AS syntax is as follows:

SELECT column AS 'Nickname'   
FROM table\_name;

Copy to Clipboard

To rename the imdb\_rating to IMDb:

SELECT imdb\_rating AS 'IMDb'   
FROM movies;

Copy to Clipboard

* Put single quotes around the alias.
* SQL commands end with a ;.

There should only be one column in the result and its name should now be IMDb.

**alias.sqlite**

SELECT imdb\_rating AS 'IMDb'

FROM movies;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Distinct**

3 min

When we are examining data in a table, it can be helpful to know what *distinct* values exist in a particular column.

Preview: Docs Loading link description

[DISTINCT](https://www.codecademy.com/resources/docs/sql/commands/select-distinct?page_ref=catalog)

 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;

Copy to Clipboard

might produce:

| **tools** |
| --- |
| Hammer |
| Nails |
| Nails |
| Nails |

By adding DISTINCT before the column name,

SELECT DISTINCT tools   
FROM inventory;

Copy to Clipboard

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 genres in the movie table after the data has been filtered than to scan every row in the table.

**Instructions**

1. Checkpoint 1 Passed

**1.**

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

SELECT DISTINCT genre   
FROM movies;

Copy to Clipboard

What are the unique genres?

Hint

The different genres are:

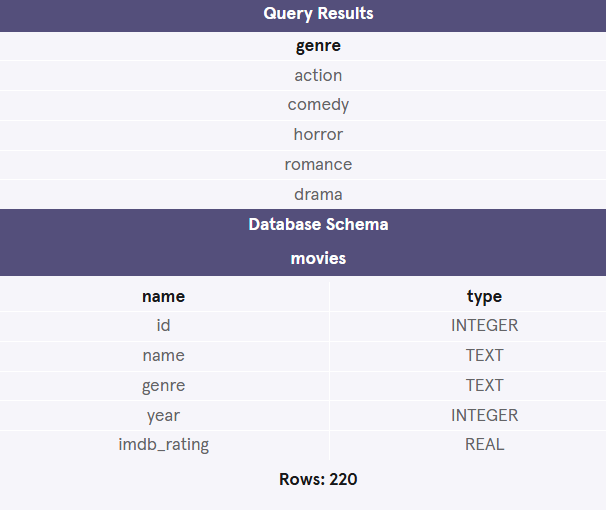
* + action
  + comedy
  + horror
  + romance
  + drama
  + ∅

The empty set symbol ∅ is just an empty value. DISTINCT recognize empty values, too.

**distinct.sqlite**

SELECT DISTINCT genre

FROM movies;



**2.**

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

Hint

Suppose we only want to query the distinct results from a column. We will use the syntax:

SELECT DISTINCT column  
FROM table\_name;

Copy to Clipboard

Following this format, the code below returns the unique values of the year column:

SELECT DISTINCT year   
FROM movies;

Copy to Clipboard

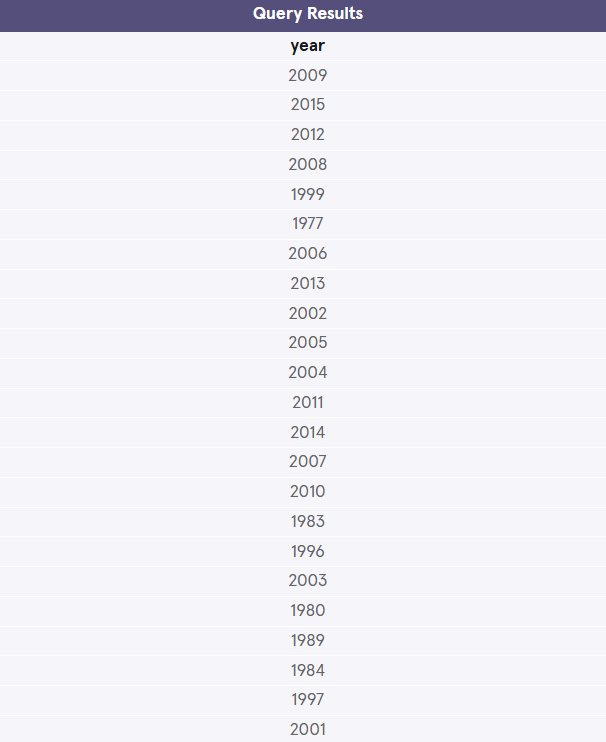
In the result, there should only be one column with all the distinct years.

Note: You might’ve noticed how there appears to be an empty set symbol ∅ near the bottom of the results (right below 1987 and above 2017). It is not a bug! DISTINCT recognize empty values, too.

**distinct.sqlite**

SELECT DISTINCT year

FROM movies;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Where**

4 min

We can restrict our query results using the

Preview: Docs Filters records (rows) that match a certain condition.

[WHERE](https://www.codecademy.com/resources/docs/sql/commands/where?page_ref=catalog)

 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;

Copy to Clipboard

How does it work?

1. The 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.

**Instructions**

1. Checkpoint 1 Passed

**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;

Copy to Clipboard

Ouch!

Hint

We are trying to retrieve all the movies with ratings lower than 5.

Common errors:

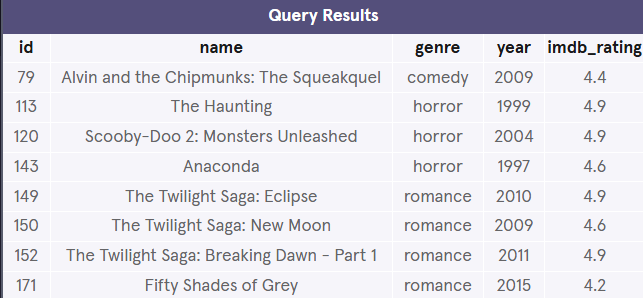
* + Missing underscore in the imdb\_rating column name.
  + Missing ; at the end.

**where.sqlite**

SELECT \*

FROM movies

WHERE imdb\_rating < 5



**2.**

Now retrieve all the recent movies, specifically those that were released after 2014.

Select all the columns using \*.

Hint

The condition here would be year > 2014

If you add the condition after the WHERE clause, it would look like:

SELECT \*  
FROM movies  
WHERE year > 2014;

**where.sqlite**

SELECT \*

FROM movies

WHERE year > 2014



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Like I**

2 min

Preview: Docs Loading link description

[LIKE](https://www.codecademy.com/resources/docs/sql/operators/like?page_ref=catalog)

 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';

Copy to Clipboard

* LIKE is a special operator used with the

Preview: Docs Loading link description

[WHERE](https://www.codecademy.com/resources/docs/sql/commands/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.

**Instructions**

1. Checkpoint 1 Passed

**1.**

Let’s test it out.

In the code editor, type:

SELECT \*   
FROM movies  
WHERE name LIKE 'Se\_en';

Copy to Clipboard

Hint

Double-check your query character by character:

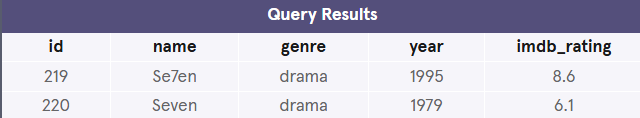
* + Note the single quotes around Se\_en.
  + Note the underscore in it.

**like-i.sqlite**

SELECT \*

FROM movies

WHERE name LIKE 'Se\_en';



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Like II**

4 min

The percentage sign % is another wildcard character that can be used with

Preview: Docs Loading link description

[LIKE](https://www.codecademy.com/resources/docs/sql/operators/like?page_ref=catalog)

.

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%';

Copy to Clipboard

% is a wildcard character that matches zero or more missing characters 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%';

Copy to Clipboard

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.

**Instructions**

1. Checkpoint 1 Passed

**1.**

In the text editor, type:

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

Copy to Clipboard

How many movie titles contain the word ‘man’?

Hint

There are 12 movie titles that contain ‘man’.

Secret level!

What are the different use cases of the LIKE operator?

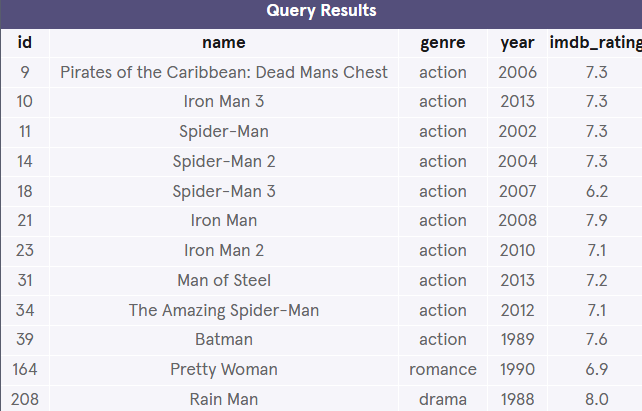
* + LIKE c% finds any values that start with the letter ‘c’
  + LIKE %c finds any values that end with the letter ‘c’
  + LIKE %re% finds values that have ‘re’ in any position
  + LIKE \_a% finds any values that have the letter ‘a’ in the second index
  + LIKE a\_%\_% finds any values that start with ‘a’ and are at least 3 characters in length.
  + LIKE a%r finds any values that start with ‘a’ and end with ‘r’.

**like-ii.sqlite**

SELECT \*

FROM movies

WHERE name LIKE '%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!

Hint

The condition should be name LIKE 'The %':

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

Copy to Clipboard

Notice how the % comes after The.

There is also a space in between because we don’t want words like ‘There’, ‘They’, etc.

**like-ii.sqlite**

SELECT \*

FROM movies

WHERE name LIKE 'The %';



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Is Null**

2 min

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:

* Preview: Docs Loading link description

[IS NULL](https://www.codecademy.com/resources/docs/sql/operators/is-null?page_ref=catalog)

* Preview: Docs Loading link description

[IS NOT NULL](https://www.codecademy.com/resources/docs/sql/operators/is-not-null?page_ref=catalog)

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

SELECT name  
FROM movies   
WHERE imdb\_rating IS NOT NULL;

**Instructions**

1. Checkpoint 1 Passed

**1.**

Now let’s do the opposite.

Write a query to find all the movies *without* an IMDb rating.

Select only the name column!

Hint

We want to query for movies that have a missing value in their imdb\_rating field:

SELECT name  
FROM movies  
WHERE imdb\_rating IS NULL;

Copy to Clipboard

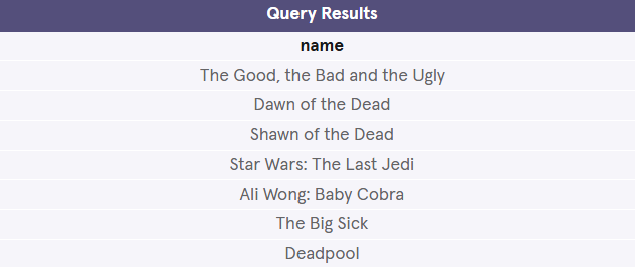
Notice how we used IS NULL instead of IS NOT NULL here.

**is-null.sqlite**

SELECT name

FROM movies

WHERE imdb\_rating IS NULL;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Between**

7 min

The

Preview: Docs Selects values, inclusively of beginning and end values, within a given range. BETWEEN works with numbers, text, or date data types.

[BETWEEN](https://www.codecademy.com/resources/docs/sql/operators/between?page_ref=catalog)

 operator is used in a

Preview: Docs Loading link description

[WHERE](https://www.codecademy.com/resources/docs/sql/commands/where?page_ref=catalog)

 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 years from 1990 up to, *and including* 1999.

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

Copy to Clipboard

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 names 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';

Copy to Clipboard

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’.

**Instructions**

1. Checkpoint 1 Passed

**1.**

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

Hint

This should be very similar to the second query in the narrative.

BETWEEN 'D' AND 'G' should be the condition:

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

Copy to Clipboard

This will return all the names that begin with ‘D’, ‘E’, and ‘F’.

BETWEEN 'D' AND 'F' should not be the condition because it would return names that begin with ‘D’ and ‘E’, but not ‘F’ (unless there was a movie with the single letter name of “F”).

And don’t forget to capitalize D and G!

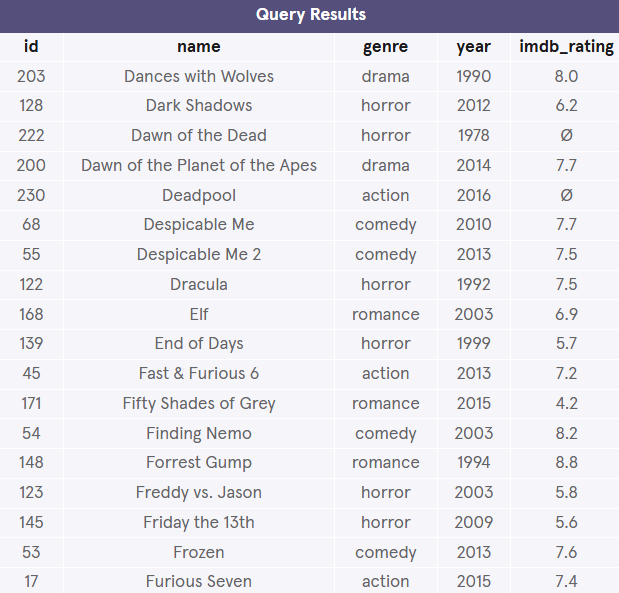
BETWEEN is case-sensitive. If the condition is BETWEEN 'a' AND 'z', it would only return lowercase (a-z) results and not uppercase (A-Z).

**between.sqlite**

SELECT \*

FROM movies

WHERE name BETWEEN 'D' AND 'G';



**2.**

Using the BETWEEN operator, write a new query that selects all information about movies that were released from the year 1970 up to and including 1979.

Hint

In this statement, the BETWEEN operator is being used to filter the result set to only include movies with years in 1970-1979:

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

Copy to Clipboard

Remember, BETWEEN two numbers *is* inclusive of the second number.

Notice that there is a movie from 1979 in the result.

Also, numeric values (*INTEGER* or *REAL* data types) don’t need to be wrapped with single quotes, whereas *TEXT* values do.

**between.sqlite**

/\*SELECT \*

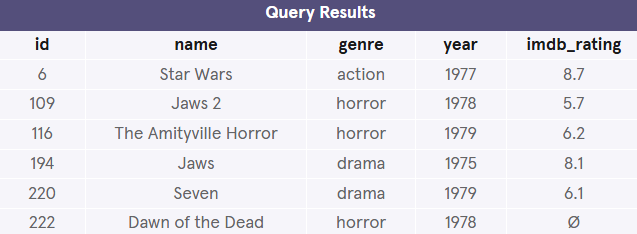
FROM movies

WHERE name BETWEEN 'D' AND 'G';\*/

SELECT \*

FROM movies

WHERE year BETWEEN 1970 and 1979;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**And**

5 min

Sometimes we want to *combine multiple conditions* in a

Preview: Docs Loading link description

[WHERE](https://www.codecademy.com/resources/docs/sql/commands/where?page_ref=catalog)

 clause to make the result set more specific and useful.

One way of doing this is to use the

Preview: Docs Loading link description

[AND](https://www.codecademy.com/resources/docs/sql/operators/and?page_ref=catalog)

 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';

Copy to Clipboard

* 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.

**Instructions**

1. Checkpoint 1 Passed

**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;

Copy to Clipboard

Hint

We are putting AND imdb\_rating > 8 on another line and *indented* just so it is easier to read.

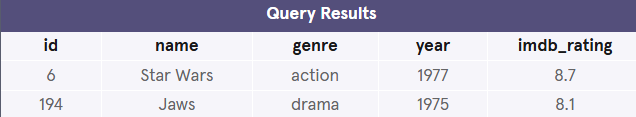
**and.sqlite**

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.

Hint

What are the two conditions?

* year < 1985
* genre = 'horror'

So your query should look like:

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

Copy to Clipboard

We indented and placed AND genre = 'horror' on another line just so it is easier to read.

Also, numeric values (1985) don’t need to be wrapped with single quotes, whereas string values do ('horror').

**and.sqlite**

/\*SELECT \*

FROM movies

WHERE year BETWEEN 1970 AND 1979

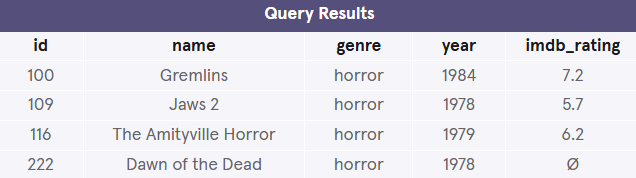
AND imdb\_rating > 8;\*/

SELECT \*

FROM movies

WHERE year < 1985

AND genre = "horror";



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Or**

3 min

Similar to AND, the OR operator can also be used to combine multiple conditions in

Preview: Docs Loading link description

[WHERE](https://www.codecademy.com/resources/docs/sql/commands/where?page_ref=catalog)

, but there is a fundamental difference:

* Preview: Docs Loading link description

[AND](https://www.codecademy.com/resources/docs/sql/operators/and?page_ref=catalog)

 operator displays a row if *all* the conditions are true.

* Preview: Docs Loading link description

[OR](https://www.codecademy.com/resources/docs/sql/operators/or?page_ref=catalog)

 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';

Copy to Clipboard

* 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.

**Instructions**

1. Checkpoint 1 Passed

**1.**

Let’s test this out:

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

Copy to Clipboard

Hint

This retrieves all the movies released after 2014 *or* in the action genre.

We are putting OR genre = 'action' on another line and indented just so it is easier to read.

**or.sqlite**

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.

Hint

What are the two conditions?

* genre = 'romance'
* genre = 'comedy'

So your query should look like:

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

Copy to Clipboard

We indented and placed OR genre = 'comedy' on another line just so it is easier to read.

Are there any good romantic comedies in the list?

**or.sqlite**

/\*SELECT \*

FROM movies

WHERE year > 2014

OR genre = 'action';\*/

SELECT \*

FROM movies

WHERE genre = "romance"

OR genre = "comedy";



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Order By**

5 min

That’s it with

Preview: Docs Loading link description

[WHERE](https://www.codecademy.com/resources/docs/sql/commands/where?page_ref=catalog)

 and its

Preview: Docs Loading link description

[operators](https://www.codecademy.com/resources/docs/sql/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

Preview: Docs Loading link description

[ORDER BY](https://www.codecademy.com/resources/docs/sql/commands/order-by?page_ref=catalog)

, 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;

Copy to Clipboard

* 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;

Copy to Clipboard

* 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).

**Instructions**

1. Checkpoint 1 Passed

**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;

Copy to Clipboard

Hint

We didn’t add ASC here because ascending order is the default.

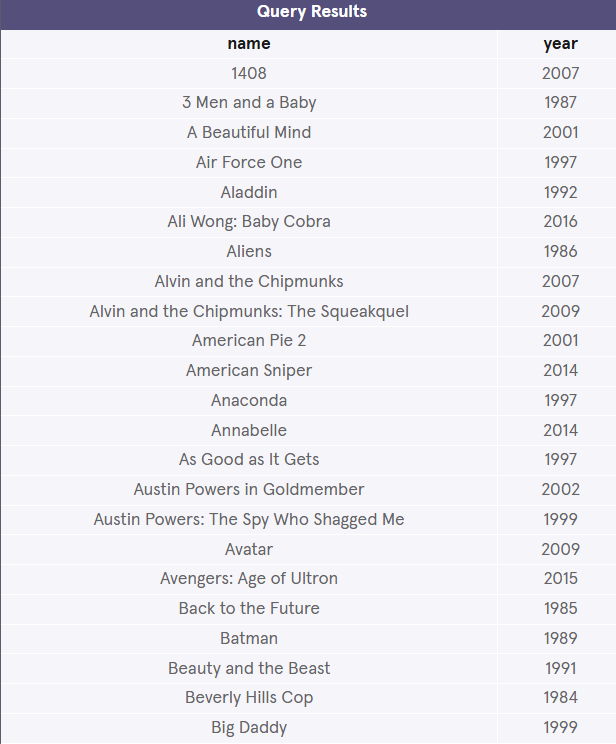
If you run this code, the result will start with ‘1408’, ‘3 Men and a Baby’ and then A-Z.

**order-by.sqlite**

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.

Hint

What are the columns that are selected and the table we are interested in?

SELECT name, year, imdb\_rating  
FROM movies;

Copy to Clipboard

Next, let’s sort them.

SELECT name, year, imdb\_rating  
FROM movies  
ORDER BY imdb\_rating DESC;

Copy to Clipboard

We added DESC here because we want to sort it in a descending order.

If you run this query, the result will start with movies with an IMDb rating of 9.0 all the way down to 4.2.

**order-by.sqlite**

/\*SELECT name, year

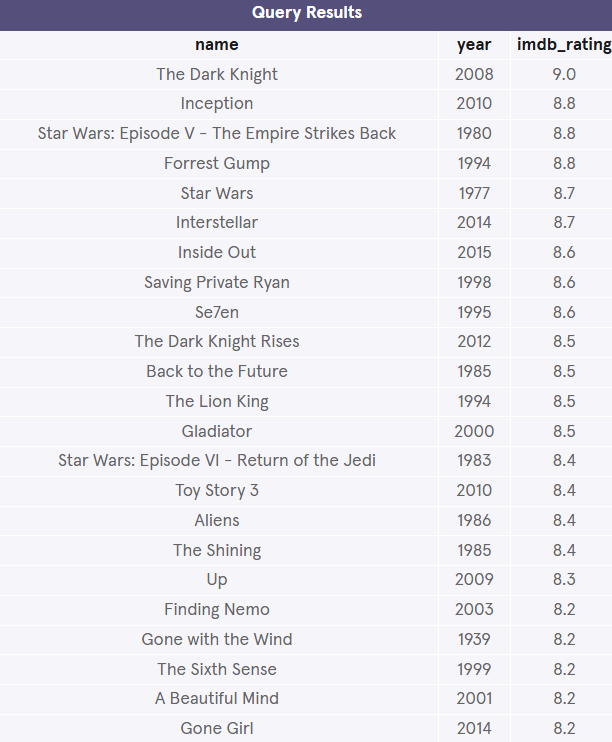
FROM movies

ORDER BY name;\*/

SELECT name, year, imdb\_rating

FROM movies

ORDER BY imdb\_rating DESC;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Limit**

3 min

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;

Copy to Clipboard

Preview: Docs Loading link description

[LIMIT](https://www.codecademy.com/resources/docs/sql/commands/limit?page_ref=catalog)

 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.

**Instructions**

1. Checkpoint 1 Passed

**1.**

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

Select all the columns.

Hint

First, what column(s) and table are we interested in?

SELECT \*  
FROM movies;

Copy to Clipboard

Next, sort them by rating (descending so we start from the highest).

SELECT \*  
FROM movies  
ORDER BY imdb\_rating DESC;

Copy to Clipboard

Lastly, add a LIMIT cap.

SELECT \*  
FROM movies  
ORDER BY imdb\_rating DESC  
LIMIT 3;

Copy to Clipboard

If you run this query, the result will be ‘The Dark Knight’ at an impressive 9.0, ‘Inception’ and ‘Star Wars: Episode V - The Empire Strikes Back’ tying for second with a rating of 8.8.

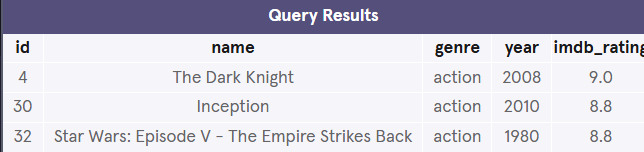
**limit.sqlite**

SELECT \*

FROM movies

ORDER BY imdb\_rating DESC

LIMIT 3;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Case**

8 min

A

Preview: Docs Loading link description

[CASE](https://www.codecademy.com/resources/docs/sql/commands/case?page_ref=catalog)

 statement allows us to create different outputs (usually in the

Preview: Docs Loading link description

[SELECT](https://www.codecademy.com/resources/docs/sql/commands/select)

 statement). It is SQL’s way of handling [if-then](https://en.wikipedia.org/wiki/Conditional_(computer_programming)) 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;

Copy to Clipboard

* 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

Preview: Docs Loading link description

[AS](https://www.codecademy.com/resources/docs/sql/commands/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;

Copy to Clipboard

**Instructions**

1. Checkpoint 1 Passed

**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.

Give it your best shot! Check hint for the answer.

Hint

This is the final boss!

Your query should look like:

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

Copy to Clipboard

* + *If the genre is romance, then it is Chill.*
  + *If the genre is comedy, then it is Chill.*
  + *Else, it is Intense.*

Don’t forget the comma after name.

Here is another query that will give us the same result:

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

Copy to Clipboard

* + *If the genre is romance or comedy, then it is Chill.*
  + *Else, it is Intense.*

**case.sqlite**

SELECT name,

  CASE

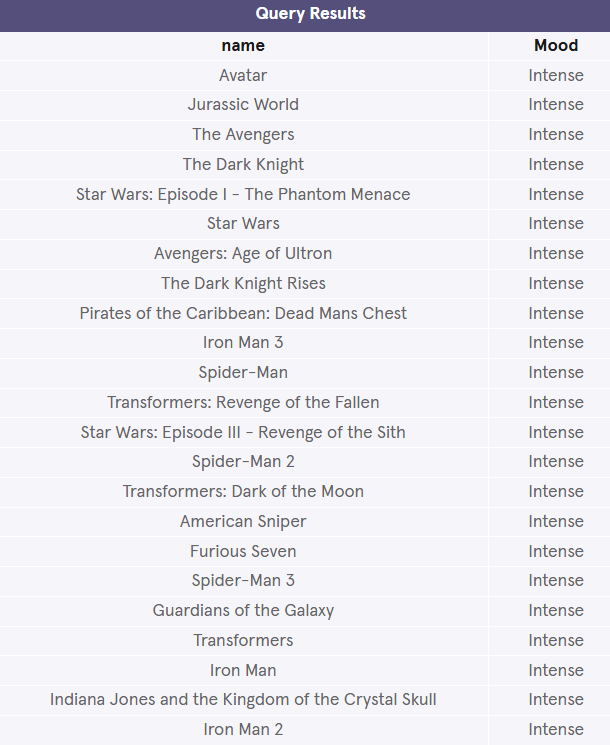
    WHEN genre = 'romance' THEN 'Chill'

    WHEN genre = 'comedy' THEN 'Chill'

    ELSE 'Intense'

  END AS 'Mood'

FROM movies;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Review**

<1 min

Congratulations!

We just learned how to query data from a database using SQL. We also learned how to filter queries to make the information more specific and useful.

Let’s summarize:

* Preview: Docs Loading link description

[SELECT](https://www.codecademy.com/resources/docs/sql/commands/select?page_ref=catalog" \t "_blank)

 is the clause we use every time we want to query information from a database.

* Preview: Docs Loading link description

[AS](https://www.codecademy.com/resources/docs/sql/commands/as?page_ref=catalog)

 renames a column or table.

* Preview: Docs Loading link description

[DISTINCT](https://www.codecademy.com/resources/docs/sql/commands/select-distinct?page_ref=catalog)

 return unique values.

* Preview: Docs Loading link description

[WHERE](https://www.codecademy.com/resources/docs/sql/commands/where?page_ref=catalog)

 is a popular command that lets you filter the results of the query based on conditions that you specify.

* Preview: Docs Loading link description

[LIKE](https://www.codecademy.com/resources/docs/sql/operators/like?page_ref=catalog)

 and

Preview: Docs Loading link description

[BETWEEN](https://www.codecademy.com/resources/docs/sql/operators/between?page_ref=catalog)

 are special operators.

* Preview: Docs Loading link description

[AND](https://www.codecademy.com/resources/docs/sql/operators/and?page_ref=catalog)

 and

Preview: Docs Loading link description

[OR](https://www.codecademy.com/resources/docs/sql/operators/or?page_ref=catalog)

 combines multiple conditions.

* Preview: Docs Loading link description

[ORDER BY](https://www.codecademy.com/resources/docs/sql/commands/order-by?page_ref=catalog)

 sorts the result.

* Preview: Docs Loading link description

[LIMIT](https://www.codecademy.com/resources/docs/sql/commands/limit?page_ref=catalog)

 specifies the maximum number of rows that the query will return.

* Preview: Docs Loading link description

[CASE](https://www.codecademy.com/resources/docs/sql/commands/case?page_ref=catalog)

 creates different outputs.

**Instructions**

Feel free to experiment a bit more with the movies table before moving on!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**QUERIES CHALLENGE**

**Baby Names Introduction**

<1 min

Welcome to Code Challenge: Queries!

In this Code Challenge, you will be querying data from three different datasets:

1. **Baby Names**
2. Restaurants
3. News Headlines

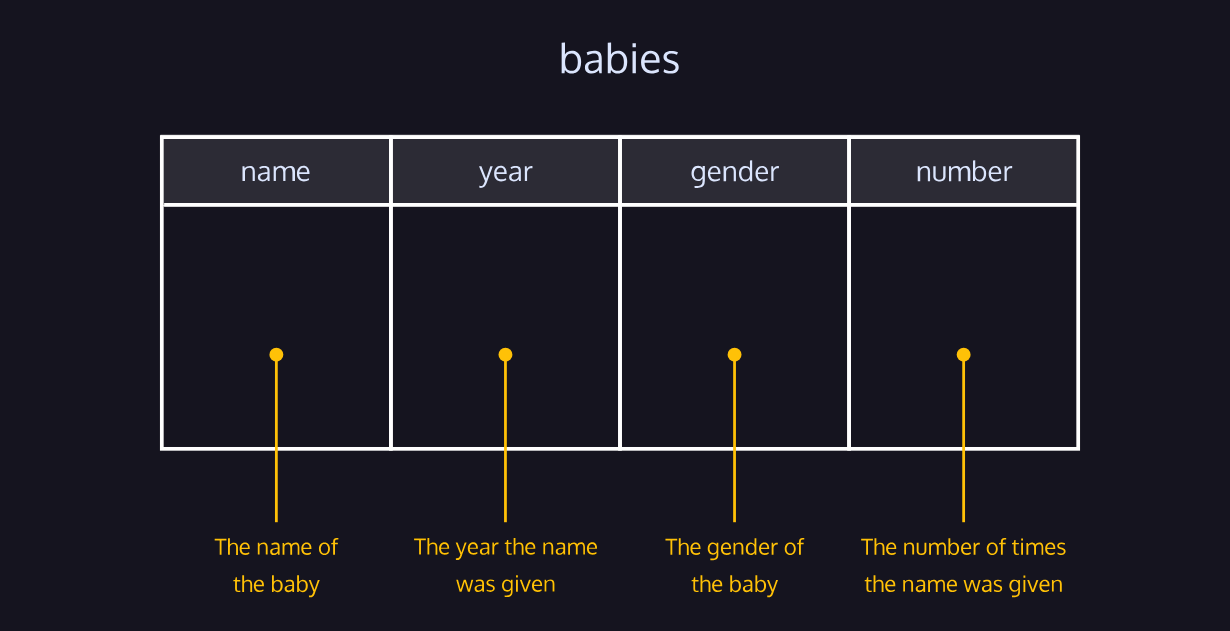
In the first one, you will be performing analysis on U.S. baby names.

You’ll start with a table named babies with four columns.

Thank you [SSA](https://www.ssa.gov/oact/babynames/) for the data.

**Instructions**

When you’re ready, let’s begin!



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code Challenge 1**

3 min

The babies table has the following columns:

* name - the name of the baby
* year - the year the name was given
* gender - the gender of the baby
* number - the number of times the name was given

Click [here](https://content.codecademy.com/courses/sql-intensive/babies.png) for the table diagram.

**Instructions**

1. Checkpoint 1 Passed

**1.**

Find the number of girls who were named Lillian for the full span of time of the database.

Select only the year and number columns.

Hint

One way is to:

SELECT year, number  
FROM babies  
WHERE name = 'Lillian';

Copy to Clipboard

If you want to be more specific and make the gender is ‘F’:

SELECT year, number  
FROM babies  
WHERE name = 'Lillian' AND gender = 'F';

Copy to Clipboard

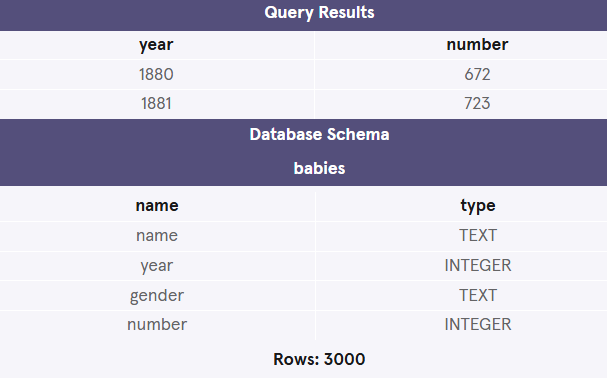
Remember, AND combine multiple conditions and make the result set more specific:

**challenge.sqlite**

SELECT year, number

FROM babies

WHERE name="Lillian" AND gender="F";



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code Challenge 2**

2 min

The babies table has the following columns:

* name - the name of the baby
* year - the year the name was given
* gender - the gender of the baby
* number - the number of times the name was given

Click [here](https://content.codecademy.com/courses/sql-intensive/babies.png) for the table diagram.

**Instructions**

1. Checkpoint 1 Passed

**1.**

Find 20 distinct names that start with ‘S’.

Select only the name column.

Hint

First, find all the distinct names that start with ‘S’:

SELECT DISTINCT name  
FROM babies  
WHERE name LIKE 'S%';

Copy to Clipboard

Notice how the % is placed after ‘S’.

Next, use LIMIT to cap the result to 20:

SELECT DISTINCT name  
FROM babies  
WHERE name LIKE 'S%'  
LIMIT 20;

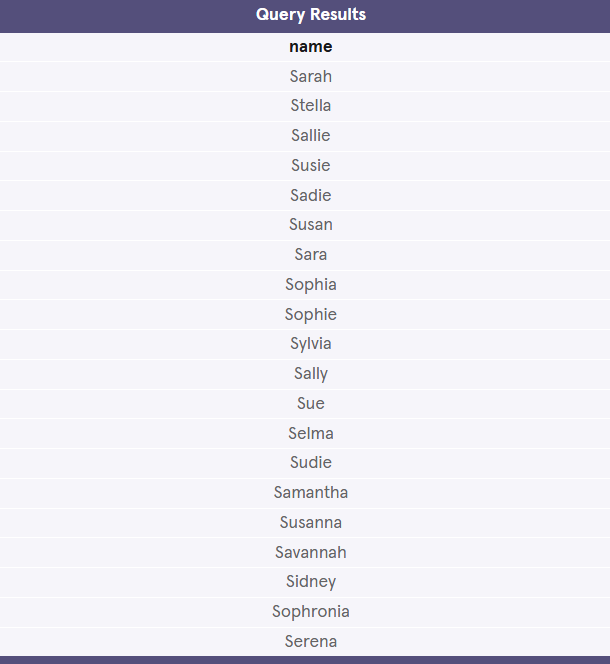
**challenge.sqlite**

SELECT name

FROM babies

WHERE name LIKE 'S%'

LIMIT 20;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code Challenge 3**

3 min

The babies table has the following columns:

* name - the name of the baby
* year - the year the name was given
* gender - the gender of the baby
* number - the number of times the name was given

Click [here](https://content.codecademy.com/courses/sql-intensive/babies.png) for the table diagram.

**Instructions**

1. Checkpoint 1 Passed

**1.**

What are the top 10 names in 1880?

Select the name, gender, and number columns.

Hint

First, we need to specify the year using a WHERE clause:

SELECT name, gender, number  
FROM babies  
WHERE year = 1880;

Copy to Clipboard

Then, we need to ORDER BY the number of times that name was given. We also need to add DESC so that we start with the most popular names:

SELECT name, gender, number  
FROM babies  
WHERE year = 1880  
ORDER BY number DESC;

Copy to Clipboard

Finally, we need to limit our results to the first 10 results:

SELECT name, gender, number  
FROM babies  
WHERE year = 1880  
ORDER BY number DESC  
LIMIT 10;

**challenge.sqlite**

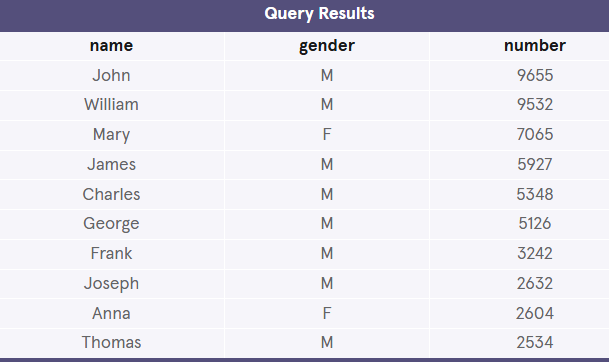
SELECT name, gender, number

FROM babies

WHERE year = 1880

ORDER BY number DESC

LIMIT 10;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_