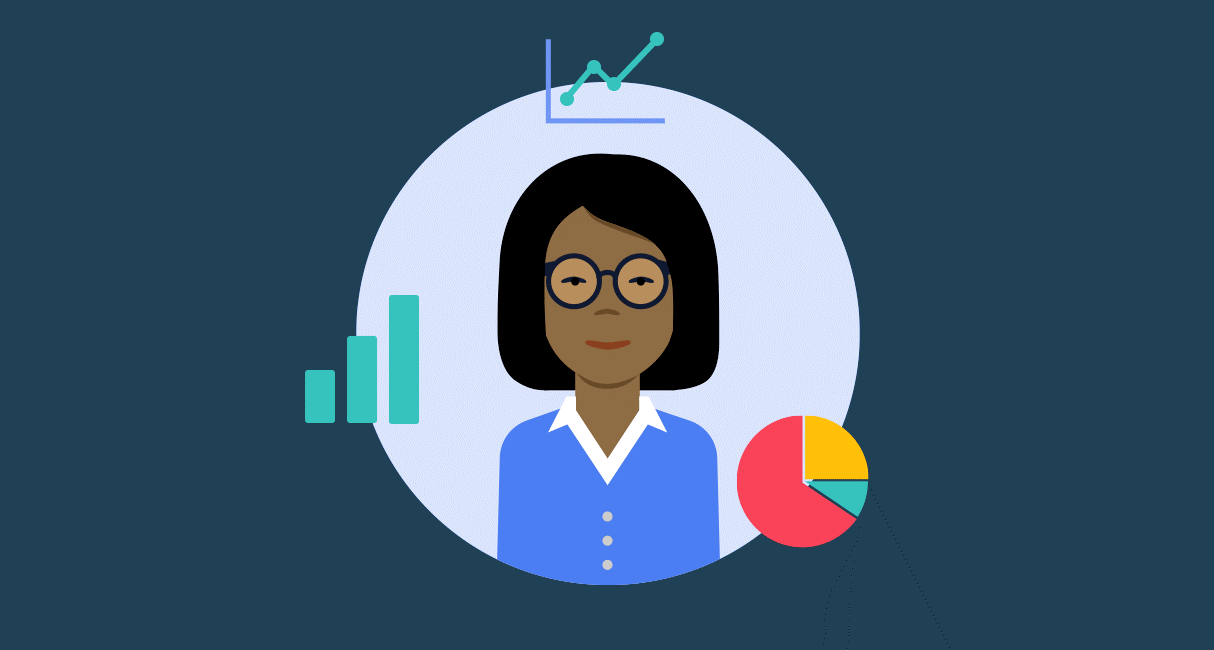
**A Day in the Life - Data Analyst**

It’s Catherine’s first day working as a Data Analyst for ACME. She’s excited to use some of the tools that she learned in **Learn SQL from Scratch**, such as:

* Writing basic queries
* Calculating aggregates
* Combining data from multiple tables
* Determining web traffic attribution
* Creating usage funnels
* Analyzing user churn



**Exploring Data with SQL**

A **database** is a set of data stored in a computer. This data is usually structured into *tables*. Tables can grow large and have a multitude of columns and records.

Spreadsheets, like Microsoft Excel and Google Sheets, allow you to view and manipulate data directly: with selecting, filtering, sorting, etc. By applying a number of these operations you can obtain the subset of data you are seeking.

SQL (pronounced “S-Q-L” or “sequel”) allows you to write *queries* which define the subset of data you are seeking. Unlike Excel and Sheets, your computer and SQL will handle *how* to get the data; you can focus on *what* data you would like. You can save these queries, refine them, share them, and run them on different databases.

Many databases use SQL (**S**tructured **Q**uery **L**anguage). It is a great way to access data and a great entry point to programming because its syntax (the specific vocabulary that gives instructions to the computer) is very human-readable. Without knowing any SQL, you might still be able to guess what each command will do.

On her first day at ACME, Catherine wants to become familiar with the company’s data, so she connects to the database and uses SQL to explore the database.

**Instructions**

**1.**

One of the tables in ACME’s database is called browse. It contains information on each time someone visited the ACME’s website. Paste the following code into the code editor (middle panel) and click Run.

SELECT \*

FROM browse

LIMIT 10;

This code will select all (\*) columns from browse for the first 10 records.

Once you correctly enter the code and click Run, this instruction will turn green, letting you know that you completed this checkpoint.

Hint

Examine the data in the far right panel:

* What columns are there?
* What kinds of questions do you have about the ACME’s website?

The columns are:

* user\_id
* browse\_date
* item\_id

SELECT \* FROM browse LIMIT 10;

**Creating Usage Funnels**

Visitors to ACME’s website follow a simple workflow:

1. Browse items available for sale
2. Click an icon to begin the checkout process
3. Enter payment information to complete their purchase

Not all users who browse on the website will find something that they like enough to checkout, and not all users who begin the checkout process will finish entering their payment information to make a purchase.

This type of multi-step process where some users leave at each step is called a *funnel*.

Catherine wants to determine what percent of users make it through each step of the funnel so that she can recommend improvements to ACME’s website.

**Instructions**

**1.**

Catherine is going to combine data from three different tables:

* browse - gives the timestamps of users who visited different item description pages
* checkout - gives the timestamps of users who visited the checkout page
* purchase - gives the timestamps of when users complete their purchase

Using SQL, she finds that 24% of all users who browse move on to checkout. 89% of those who reach checkout purchase.

Click Run, to see Catherine’s analysis.

Hint

In the result, there should be two columns:

* browse\_to\_checkout\_percent
* checkout\_to\_purchase\_percent

 SELECT ROUND(100.0 \* COUNT(DISTINCT c.user\_id)/COUNT(DISTINCT b.user\_id)) AS browse\_to\_checkout\_percent,ROUND(100.0 \* COUNT(DISTINCT p.user\_id)/COUNT(DISTINCT c.user\_id)) AS checkout\_to\_purchase\_percent FROM browse b

 LEFT JOIN checkout c ON b.user\_id = c.user\_id

 LEFT JOIN purchase p ON c.user\_id = p.user\_id;

**Analyzing User Churn**

A *churn rate* is the percent of subscribers to a monthly service who have canceled. For example, in January, ACME has 1,000 customers. In February, 200 customers sign up, and 250 cancel.

The churn rate for February would be:

Catherine wants to analyze the churn rates for ACME for the past few months.

**Instructions**

**1.**

Click Run, to see Catherine’s analysis for the churn rate in March 2017.

What recommendations would you make to ACME based on Catherine’s analysis?

Hint

In the result, there should be three columns:

* enrollments
* march-cancellations
* churn\_rate

SELECT COUNT(DISTINCT user\_id) AS enrollments,

  COUNT(CASE

        WHEN strftime("%m", cancel\_date) = '03'

        THEN user\_id

  END) AS march\_cancellations,

  ROUND(100.0 \* COUNT(CASE

        WHEN strftime("%m", cancel\_date) = '03'

        THEN user\_id

  END) / COUNT(DISTINCT user\_id)) AS churn\_rate

FROM pro\_users

WHERE signup\_date < '2017-04-01'

  AND (

    (cancel\_date IS NULL) OR

    (cancel\_date > '2017-03-01')

  );

**Determining Web Traffic Attribution**

Catherine’s boss asks her to analyze how users are finding ACME’s websites using *UTM Parameters*. UTM Parameters are special tags that site owners add to their pages to track what website a user was on before they reach the website. For instance:

* If a user found ACME’s website through Google search, the table page\_visits might have utm\_source set to ‘google’.
* If a different user clicked a Facebook ad to get to ACME’s website, then their row in page\_visits might have utm\_source as ‘facebook’.

**Instructions**

**1.**

Catherine wants to know how many visits come from each utm\_source.

Click Run, to see Catherine’s analysis.

What is the most common source of traffic to ACME’s website?

Hint

The most common source is The New York Times!

 SELECT utm\_source,

  COUNT(DISTINCT user\_id) AS num\_users

FROM page\_visits

GROUP BY 1

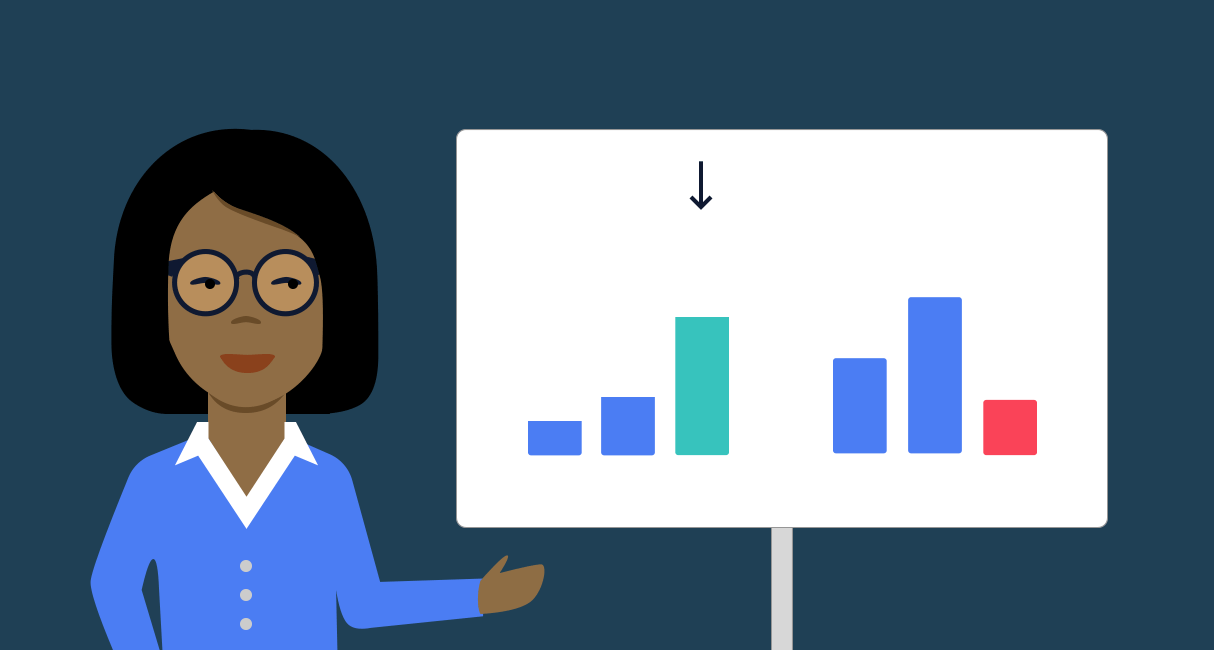
ORDER BY 2 DESC;

# Begin Your Journey

Catherine learned and applied a lot during the first two weeks of her job:

* Writing basic queries
* Calculating aggregates
* Combining data from multiple tables
* Creating usage funnels
* Analyzing user churn
* Determining web traffic attribution

Now it’s your turn. By the end of this Intensive, you’ll be able to do all of the things that Catherine can do.



**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 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.**

We should get acquainted with the movies table.

In the editor, type the following:

SELECT \* FROM movies;

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;

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:

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

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

SELECT \* FROM movies;

**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.

**Instructions**

**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;

Hint

Instead of selecting all the columns:

SELECT \*

FROM movies:

We are now just selecting the name and genre column:

SELECT name, genre

FROM movies;

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.

**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;

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;

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

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.

**Instructions**

**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.

Hint

Suppose you want to rename the name column as Binge, then the query would look like:

SELECT name AS 'Binge'

FROM movies;

If this is your query, then the name of the column in the result would be Binge.

The underscores are only there for directional purposes.

**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;

To rename the imdb\_rating to IMDb:

SELECT imdb\_rating AS 'IMDb'

FROM movies;

* 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.

**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.

**Instructions**

**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!

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.

**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 \*.

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;

**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.

**Instructions**

**1.**

Let’s test it out.

In the code editor, type:

SELECT \*

FROM movies

WHERE name LIKE 'Se\_en';

Hint

Double-check your query character by character:

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

**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.

**Instructions**

**1.**

In the text editor, type:

SELECT \*

FROM movies

WHERE name LIKE '%man%';

How many movie titles contain the word ‘man’?

Stuck? Get a hint

**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 %';

Notice how the % comes after The.

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

**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;

**Instructions**

**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;

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

# 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 years 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 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';

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.**

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';

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).

**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.

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;

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.

**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.

**Instructions**

**1.**

Let’s test this out:

SELECT \*

FROM movies

WHERE year > 2014

OR genre = 'action';

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.

**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';

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?

**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).

**Instructions**

**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;

Stuck? Get a hint

**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;

Next, let’s sort them.

SELECT name, year, imdb\_rating

FROM movies

ORDER BY imdb\_rating DESC;

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.

# 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.

**Instructions**

**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;

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

SELECT \*

FROM movies

ORDER BY imdb\_rating DESC;

Lastly, add a LIMIT cap.

SELECT \*

FROM movies

ORDER BY imdb\_rating DESC

LIMIT 3;

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.

**Case**

A CASE statement allows us to create different outputs (usually in the 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;

* 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;

**Instructions**

**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;

* *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'

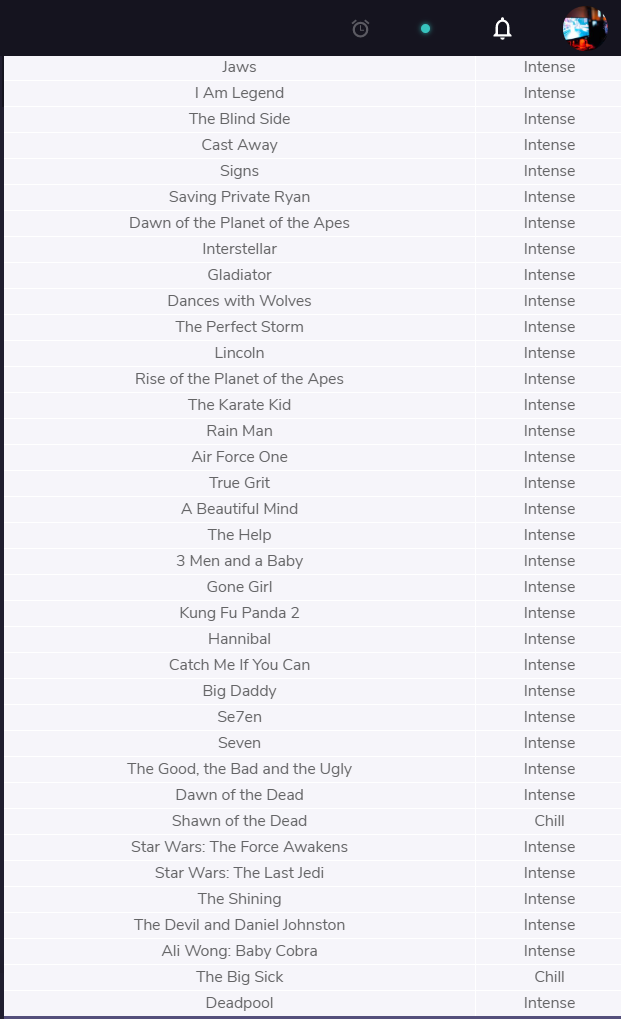
THEN 'Chill'

ELSE 'Intense'

END AS 'Mood'

FROM movies;

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



**Review**

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:

* SELECT is the clause we use every time we want to query information from a database.
* AS renames a column or table.
* DISTINCT return unique values.
* WHERE is a popular command that lets you filter the results of the query based on conditions that you specify.
* LIKE and BETWEEN are special operators.
* AND and OR combines multiple conditions.
* ORDER BY sorts the result.
* LIMIT specifies the maximum number of rows that the query will return.
* CASE creates different outputs.

**Instructions**

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

# Baby Names Introduction

Welcome to Code Challenge: Queries!

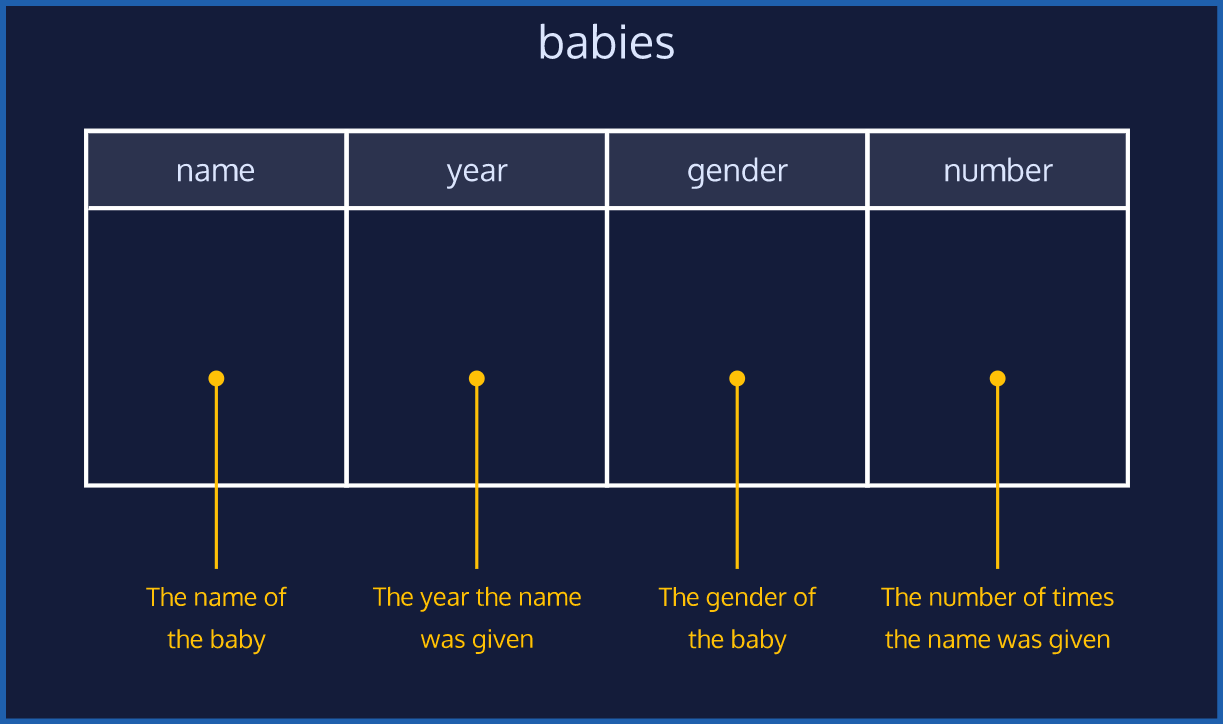
In this Code Challenge, you will be querying data from three different datasets:

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.



**Code Challenge 1**

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://s3.amazonaws.com/codecademy-content/courses/sql-intensive/babies.png) for the table diagram.

**Instructions**

**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';

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

SELECT year, number

FROM babies

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

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

**Code Challenge 2**

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://s3.amazonaws.com/codecademy-content/courses/sql-intensive/babies.png) for the table diagram.

**Instructions**

**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%';

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;

**Code Challenge 3**

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://s3.amazonaws.com/codecademy-content/courses/sql-intensive/babies.png) for the table diagram.

**Instructions**

**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;

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;

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;

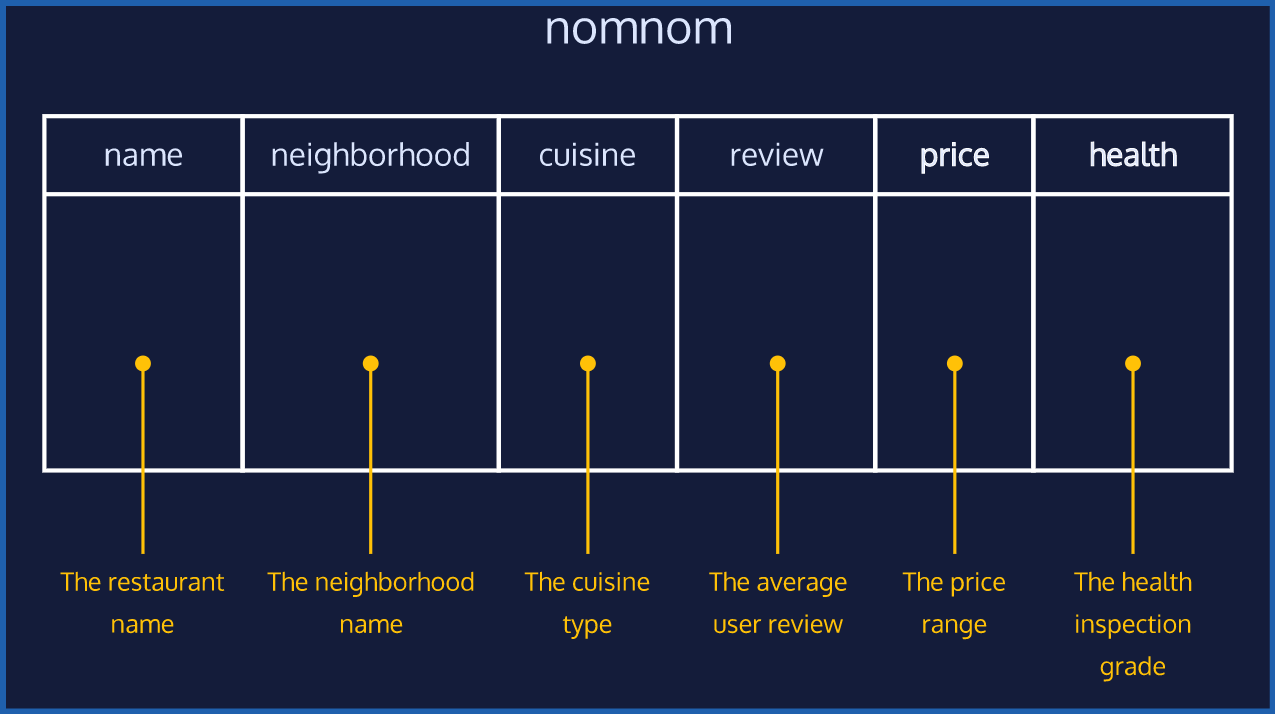
# Restaurants Introduction

Next up, you will be querying data on restaurants:

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

We need your help to answer some questions and find the best dinner spots in the city!

You’ll work with a table named nomnom with six columns.



**Code Challenge 4**

The nomnom table has the following columns:

* name - the restaurant name
* neighborhood - the neighborhood name
* cuisine - the cuisine type
* review - the average user review
* price - the price range
* health - the health inspection grade

Click [here](https://s3.amazonaws.com/codecademy-content/courses/sql-intensive/nomnom.png) for the table diagram.

**Instructions**

**1.**

Suppose your friend Jaime wants to go out to Japanese, but you’re on a budget.

Return all the restaurants that are Japanese and $$.

Select all the columns.

Hint

If you want to find Japanese restaurants with *exactly* two dollar signs:

SELECT \*

FROM nomnom

WHERE cuisine = 'Japanese'

AND price = '$$';

Remember, AND combines multiple conditions and makes the result set more specific:

**Code Challenge 5**

The nomnom table has the following columns:

* name - the restaurant name
* neighborhood - the neighborhood name
* cuisine - the cuisine type
* review - the average user review
* price - the price range
* health - the health inspection grade

Click [here](https://s3.amazonaws.com/codecademy-content/courses/sql-intensive/nomnom.png) for the table diagram.

**Instructions**

**1.**

Your roommate Bevers can’t remember the exact name of a restaurant he went to but he knows it *contains* the word ‘noodle’ in it.

Can you find it for him using a query?

Select all the columns.

Hint

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

To find the restaurant names that contain the word ‘noodle’:

SELECT \_\_\_

FROM \_\_\_\_\_

WHERE name LIKE '%noodle%';

**Code Challenge 6**

The nomnom table has the following columns:

* name - the restaurant name
* neighborhood - the neighborhood name
* cuisine - the cuisine type
* review - the average user review
* price - the price range
* health - the health inspection grade

Click [here](https://s3.amazonaws.com/codecademy-content/courses/sql-intensive/nomnom.png) for the table diagram.

**Instructions**

**1.**

Some of the restaurants have not been inspected yet or are currently appealing their health grade score.

Find the restaurants that have empty health values.

Select all the columns.

Hint

Empty values are indicated by NULL:

SELECT \_\_\_\_\_

FROM \_\_\_\_\_\_\_

WHERE health IS NULL;

 select \* from nomnom where health IS NULL;

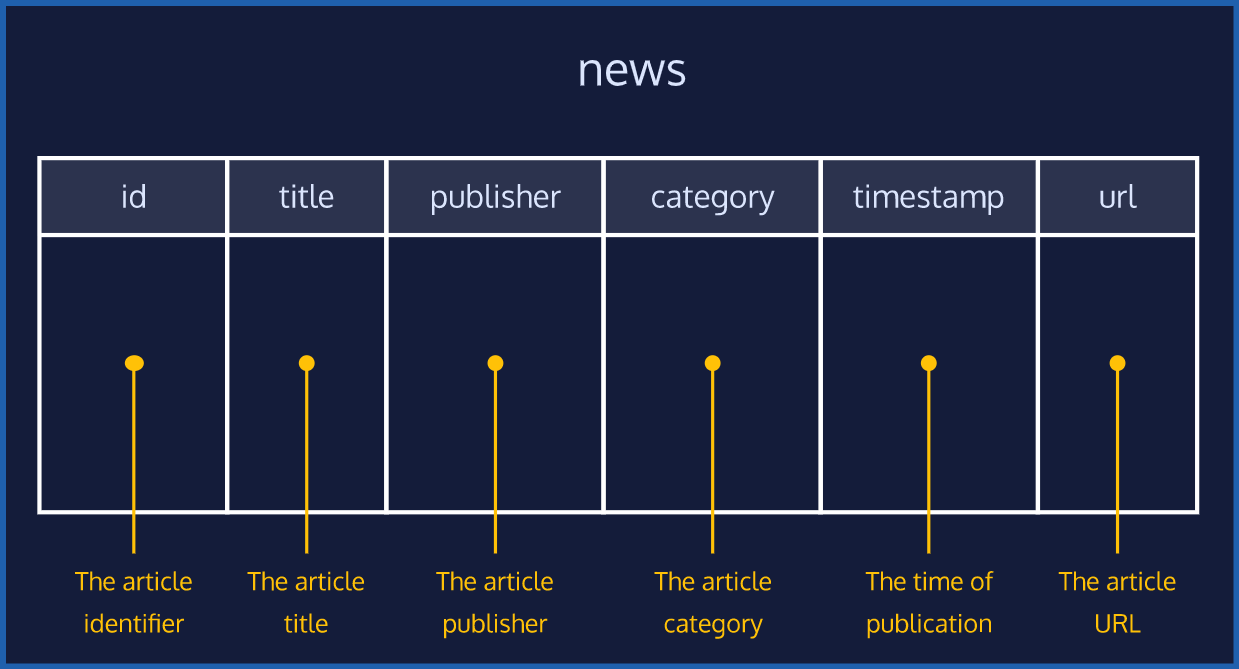
# News Headlines Introduction

Here is the last dataset of the Code Challenge #1:

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

There is a table called news with six columns.

It is full of news article headlines from different publishing companies!



**Code Challenge 7**

The news table has the following columns:

* id - the article identifier
* title - the article title
* publisher - the article publisher
* category - the article category
* timestamp - the time of publication
* url - the article web address

Click [here](https://s3.amazonaws.com/codecademy-content/courses/sql-intensive/news.png) for the table diagram.

**Instructions**

**1.**

Order the table by title (from A-Z).

Select only the title and publisher columns.

Hint

To sort the title in an ascending order:

SELECT title, publisher

FROM news

ORDER BY title;

If you want to be more explicit:

SELECT title, publisher

FROM news

ORDER BY title ASC;

**Code Challenge 8**

The news table has the following columns:

* id - the article identifier
* title - the article title
* publisher - the article publisher
* category - the article category
* timestamp - the time of publication
* url - the article web address

Click [here](https://s3.amazonaws.com/codecademy-content/courses/sql-intensive/news.png) for the table diagram.

**Instructions**

**1.**

Which article names have the word 'bitcoin' in it?

Select all the columns.

Hint

By default, the LIKE operator performs *case-insensitive* pattern match.

This means that article names that have words like ‘bitcoin’, ‘Bitcoin’, and ‘BITCOIN’ will all be selected.

WHERE title LIKE '%bitcoin%';

SELECT \* FROM news WHERE title LIKE '%bitcoin%';

**Code Challenge 9**

The news table has the following columns:

* id - the article identifier
* title - the article title
* publisher - the article publisher
* category - the article category
* timestamp - the time of publication
* url - the article web address

Click [here](https://s3.amazonaws.com/codecademy-content/courses/sql-intensive/news.png) for the table diagram.

**Instructions**

**1.**

The category column contains the article category:

* 'b' stands for Business
* 't' stands for Technology

What are the 20 *business* articles published most recently?

Select all the columns.

Hint

First, we filter our results to category = 'b' using a WHERE statement:

SELECT \*

FROM news

WHERE category = 'b';

Then, we sort the entries by the most recent timestamp. We use DESC so that the most recent comes first:

SELECT \*

FROM news

WHERE category = 'b'

ORDER BY timestamp DESC;

Finally, cap the results to only 20:

SELECT \*

FROM news

WHERE category = 'b'

ORDER BY timestamp DESC

LIMIT 20;