SQL, which stands for Structured Query Language, is a language for interacting with data stored in something called a relational database.

You can think of a relational database as a collection of tables. A table is just a set of rows and columns, like a spreadsheet, which represents exactly one type of entity. For example, a table might represent employees in a company or purchases made, but not both.

Each row, or record, of a table contains information about a single entity. For example, in a table representing employees, each row represents a single person. Each column, or field, of a table contains a single attribute for all rows in the table. For example, in a table representing employees, we might have a column containing first and last names for all employees

**LEARNING SQL FOR DATA SCIENCE**

SELECT occurred\_at, account\_id, channel

FROM web\_events

LIMIT 15;

\\*

#QUESTIONS

Practice

In order to gain some practice using ORDER BY:

Write a query to return the 10 earliest orders in the orders table.

Include the id, occurred\_at, and total\_amt\_usd.

#Solution

SELECT id, occurred\_at, total\_amt\_usd

FROM orders

ORDER BY occurred\_at

LIMIT 10;

Write a query to return the top 5 orders in terms of largest total\_amt\_usd.

Include the id, account\_id, and total\_amt\_usd.

# Solution

SELECT id, account\_id, total\_amt\_usd

FROM orders

order by total\_amt\_usd desc

LIMIT 5;

Write a query to return the bottom 20 orders in terms of least total.

Include the id, account\_id, and total.

# Solution

SELECT id, account\_id, total

FROM orders

ORDER BY total

LIMIT 20;

Questions

Write a query that returns the top 5 rows from orders ordered according to newest

to oldest, but with the largest total\_amt\_usd for each date listed first for each

date. You will notice each of these dates shows up as unique because of the time

element. When you learn about truncating dates in a later lesson, you will better

be able to tackle this question on a day, month, or yearly basis.

#Solution

SELECT \*

FROM orders

ORDER BY occurred\_at DESC, total\_amt\_usd DESC

LIMIT 5;

Write a query that returns the top 10 rows from orders ordered according to oldest

to newest, but with the smallest total\_amt\_usd for each date listed first for each

date. You will notice each of these dates shows up as unique because of the time

element. When you learn about truncating dates in a later lesson, you will better

be able to tackle this question on a day, month, or yearly basis.

#Solution

SELECT \*

FROM orders

ORDER BY occurred\_at, total\_amt\_usd

LIMIT 10;

Questions on where

Write a query that

Pull the first 5 rows and all columns from the orders table that have a dollar

amount of gloss\_amt\_usd greater than or equal to 1000.

#Solution

select \*

from orders

where gloss\_amt\_usd>= 1000

limit 5;

Pull the first 10 rows and all columns from the orders table that have a

total\_amt\_usd less than 500.

#Solution

select \*

from orders

where total\_amt\_usd< 500

limit 10;

Quiz: WHERE with Non-Numeric

Practice Question Using WHERE with Non-Numeric Data

Filter the accounts table to include the company name,

website, and the primary point of contact (primary\_poc) for Exxon Mobil in the

accounts table.

#Solution

select name, website, primary\_poc

from accounts

where name='Exxon Mobil';

Quiz: Arithmetic Operators

Questions using Arithmetic Operations

Using the orders table:

Create a column that divides the standard\_amt\_usd by the standard\_qty to find

the unit price for standard paper for each order. Limit the results to the first

10 orders, and include the id and account\_id fields.

#Solution

SELECT id, account\_id, standard\_amt\_usd/standard\_qty AS unit\_price

FROM orders

LIMIT 10;

Write a query that finds the percentage of revenue that comes from poster paper

for each order. You will need to use only the columns that end with \_usd.

(Try to do this without using the total column). Include the id and account\_id

fields. NOTE - you will be thrown an error with the correct solution to this question.

This is for a division by zero. You will learn how to get a solution without an error

to this query when you learn about CASE statements in a later section.

For now, you might just add some very small value to your denominator as a work around.

#Solution

SELECT id, account\_id,

poster\_amt\_usd/(standard\_amt\_usd + gloss\_amt\_usd + poster\_amt\_usd) AS post\_per

FROM orders;

Filter with LIKE

Questions using the LIKE operator

Use the accounts table to find

All the companies whose names start with 'C'.

Solution

select \*

from accounts

where name like 'C%';

All companies whose names contain the string 'one' somewhere in the name.

Solution

select \*

from accounts

where name like '%one%';

All companies whose names end with 's'.

Solution

select \*

from accounts

where name like '%s';

Filter using IN operator

Use the accounts table to find the account name, primary\_poc, and sales\_rep\_id

for Walmart, Target, and Nordstrom.

Solution

SELECT name, primary\_poc, sales\_rep\_id

FROM accounts

WHERE name IN ('Walmart', 'Target', 'Nordstrom');

Use the web\_events table to find all information regarding individuals who were

contacted via the channel of organic or adwords.

Solution

select \*

from web\_events

where channel in ('organic', 'adwords');

Filter with NOT IN

Questions using the NOT IN operator

We can pull all of the rows that were excluded from the queries in the previous

two concepts with our new operator.

Use the accounts table to find the account name, primary poc, and sales rep id

for all stores except Walmart, Target, and Nordstrom.

Solution

SELECT name, primary\_poc, sales\_rep\_id

FROM accounts

WHERE name NOT IN ('Walmart', 'Target', 'Nordstrom');

Use the web\_events table to find all information regarding individuals who were

contacted via any method except using organic or adwords methods.

Solution

select \*

from web\_events

where channel not in ('organic','adwords');

Filter with NOT LIKE

Use the accounts table to find:

All the companies whose names do not start with 'C'.

Solution

select name

from accounts

where name not like 'C%';

All companies whose names do not contain the string 'one' somewhere in the name.

Solution

select name

from accounts

where name not like '%one%';

All companies whose names do not end with 's'.

Solution

select name

from accounts

where name not like '%s';

Quiz: AND and BETWEEN

Questions using AND and BETWEEN operators

Write a query that returns all the orders where the standard\_qty is over 1000,

the poster\_qty is 0, and the gloss\_qty is 0.

Solution

select \*

from orders

where standard\_qty>1000 and poster\_qty=0 and gloss\_qty=0;

Using the accounts table find all the companies whose names do not start with 'C'

and end with 's'.

Solution

SELECT name

FROM accounts

WHERE name NOT LIKE 'C%' AND name LIKE '%s';

Use the web\_events table to find all information regarding individuals who were

contacted via organic or adwords and started their account at any point in 2016

sorted from newest to oldest.

Solution

SELECT \*

FROM web\_events

WHERE channel IN ('organic', 'adwords') AND occurred\_at BETWEEN '2016-01-01' AND '2017-01-01'

ORDER BY occurred\_at DESC;

Quiz: OR

Questions using the OR operator

Find list of orders ids where either gloss\_qty or poster\_qty is greater than 4000.

Only include the id field in the resulting table.

Solution

select id

from orders

where (gloss\_qty> 4000 or poster\_qty> 4000);

Write a query that returns a list of orders where the standard\_qty is zero and

either the gloss\_qty or poster\_qty is over 1000.

Solution

select \*

from orders

where (standard\_qty = 0) and (gloss\_qty> 1000 or poster\_qty>1000);

Find all the company names that start with a 'C' or 'W', and the primary contact

contains 'ana' or 'Ana', but it doesn't contain 'eana'.

Solution

SELECT \*

FROM accounts

WHERE (name LIKE 'C%' OR name LIKE 'W%')

AND ((primary\_poc LIKE '%ana%' OR primary\_poc LIKE '%Ana%')

AND primary\_poc NOT LIKE '%eana%');

Text + Quiz: Your First JOIN

Quiz Questions

Try pulling all the data from the accounts table, and all the data from the orders table.

Solution

SELECT orders.\*, accounts.\*

FROM accounts

JOIN orders

ON accounts.id = orders.account\_id;

Try pulling standard\_qty, gloss\_qty, and poster\_qty from the orders table,

and the website and the primary\_poc from the accounts table.

Solution

select orders.standard\_qty,orders.gloss\_qty, orders.poster\_qty, accounts.website, accounts.primary\_poc

from orders

join accounts

on orders.account\_id=accounts.id

Quiz: JOIN Questions Part I

Provide a table for all web\_events associated with account name of Walmart.

There should be three columns. Be sure to include the primary\_poc, time of the event,

and the channel for each event. Additionally, you might choose to add a fourth column

to assure only Walmart events were chosen.

Solution

select w.occurred\_at, w.channel ,a.primary\_poc, a.name

from web\_events w

join accounts a

on a.id=w.account\_id

where a.name='Walmart';

Provide a table that provides the region for each sales\_rep along with their

associated accounts. Your final table should include three columns: the region name,

the sales rep name, and the account name. Sort the accounts alphabetically (A-Z)

according to account name.

Solution

SELECT r.name region, s.name rep, a.name account

FROM sales\_reps s

JOIN region r

ON s.region\_id = r.id

JOIN accounts a

ON a.sales\_rep\_id = s.id

ORDER BY a.name;

Provide the name for each region for every order, as well as the account name

and the unit price they paid (total\_amt\_usd/total) for the order. Your final

table should have 3 columns: region name, account name, and unit price.

A few accounts have 0 for total, so I divided by (total + 0.01) to assure

not dividing by zero.

Solution

SELECT r.name region, a.name account,

o.total\_amt\_usd/(o.total + 0.01) unit\_price

FROM region r

JOIN sales\_reps s

ON s.region\_id = r.id

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id;

Quiz: Last Check

Questions

Provide a table that provides the region for each sales\_rep along with their

associated accounts. This time only for the Midwest region. Your final table

should include three columns: the region name, the sales rep name, and the

account name. Sort the accounts alphabetically (A-Z) according to account

name.

Solution

\*\

While SQL can be used to create and modify databases, the focus of this course will be querying databases. A query is a request for data from a database table (or combination of tables). Querying is an essential skill for a data scientist, since the data you need for your analyses will often live in databases.

**Recap**

**Primary and Foreign Keys**

You learned a key element for **JOIN**ing tables in a database has to do with primary and foreign keys:

* **primary keys** - are unique for every row in a table. These are generally the first column in our database (like you saw with the **id** column for every table in the Parch & Posey database).
* **foreign keys** - are the **primary key** appearing in another table, which allows the rows to be non-unique.

Choosing the set up of data in our database is very important, but not usually the job of a data analyst. This process is known as **Database Normalization**.

**JOINs**

In this lesson, you learned how to combine data from multiple tables using **JOIN**s. The three **JOIN**statements you are most likely to use are:

1. **JOIN** - an **INNER JOIN** that only pulls data that exists in both tables.
2. **LEFT JOIN** - a way to pull all of the rows from the table in the **FROM** even if they do not exist in the **JOIN** statement.
3. **RIGHT JOIN** - a way to pull all of the rows from the table in the **JOIN** even if they do not exist in the **FROM** statement.

There are a few more advanced **JOIN**s that we did not cover here, and they are used in very specific use cases. [**UNION and UNION ALL**](https://www.w3schools.com/sql/sql_union.asp), [**CROSS JOIN**](http://www.w3resource.com/sql/joins/cross-join.php), and the tricky [**SELF JOIN**](https://www.w3schools.com/sql/sql_join_self.asp). These are more advanced than this course will cover, but it is useful to be aware that they exist, as they are useful in special cases.

**Alias**

You learned that you can alias tables and columns using **AS** or not using it. This allows you to be more efficient in the number of characters you need to write, while at the same time you can assure that your column headings are informative of the data in your table.

**Looking Ahead**

The next lesson is aimed at **aggregating** data. You have already learned a ton, but **SQL** might still feel a bit disconnected from **statistics** and using **Excel** like platforms. Aggregations will allow you to write **SQL** code that will allow for more complex queries, which assist in answering questions like:

* Which **channel** generated more revenue?
* Which **account** had an order with the most items?
* Which **sales\_rep** had the most orders? or least orders? How many orders did they have?