

Introduction to SQL

brought to you by:

Frank Burkholder (Neil Bezdek, Taryn Heilman, Jordan Hagan)

DSI

Motivation

2016 Data Science Salary Survey

Tools, Trends, What Pays (and What Doesn't) for Data Professionals

Key findings include:

- Python and Spark are among the tools that contribute most to salary.
- Among those who code, the highest earners are the ones who code the most.
- SQL, Excel, R and Python are the most commonly used tools.

Objectives

- Describe why we use Relational Database Management Systems (RDBMS)
- Understand primary keys, foreign keys, and table relationships
- Write simple SQL queries on a single table using SELECT, FROM, WHERE, GROUP BY, ORDER BY clauses as well as aggregation functions (COUNT, AVG, etc.)
- Write more complex SQL queries using joins and subqueries
- Interact with a Postgres database from the command line
- Learn how to say PostgreSQL ([link](#))

An inefficient way to store data...

A single table with records of customer purchases at an outdoor sports store.

id	cust_name	cust_state	item_purchased	price	date
1	Kayla	CO	skis	\$300	10/30
2	Kayla	CO	goggles	\$75	11/14
3	Erich	CO	snowboard	\$400	11/18
4	Adam	NY	skis	\$300	12/11
5	Frank	AZ	skis	\$300	12/19
6	Adam	NY	goggles	\$75	12/24

Relational Database Management Systems

A RDBMS is a type of database where **data is stored in multiple related tables**.

The tables are related through **primary** and **foreign keys**.

The same information in an RDBMS.

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

products

prod_id	description	price
1	skis	\$300
2	goggles	\$75
3	snowboard	\$400

purchases

cust_id	prod_id	date
1	1	10/30
1	2	11/14
2	3	11/18
3	1	12/11
4	1	12/19
3	2	12/24

Primary Keys

- Every table in a RDBMS has a **primary key** that uniquely identifies that row
- Each entry must have a primary key, and primary keys cannot repeat within a table
- Primary keys are usually integers but can take other forms

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

products

prod_id	description	price
1	skis	\$300
2	goggles	\$75
3	snowboard	\$400

purchases

purch_id	cust_id	prod_id	date
1	1	1	10/30
2	1	2	11/14
3	2	3	11/18
4	3	1	12/11
5	4	1	12/19
6	3	2	12/24

Foreign Keys and Table Relationships

- A **foreign key** is a column that uniquely identifies a column in another table
- Often, a foreign key in one table is a primary key in another table
- We can use foreign keys to join tables

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

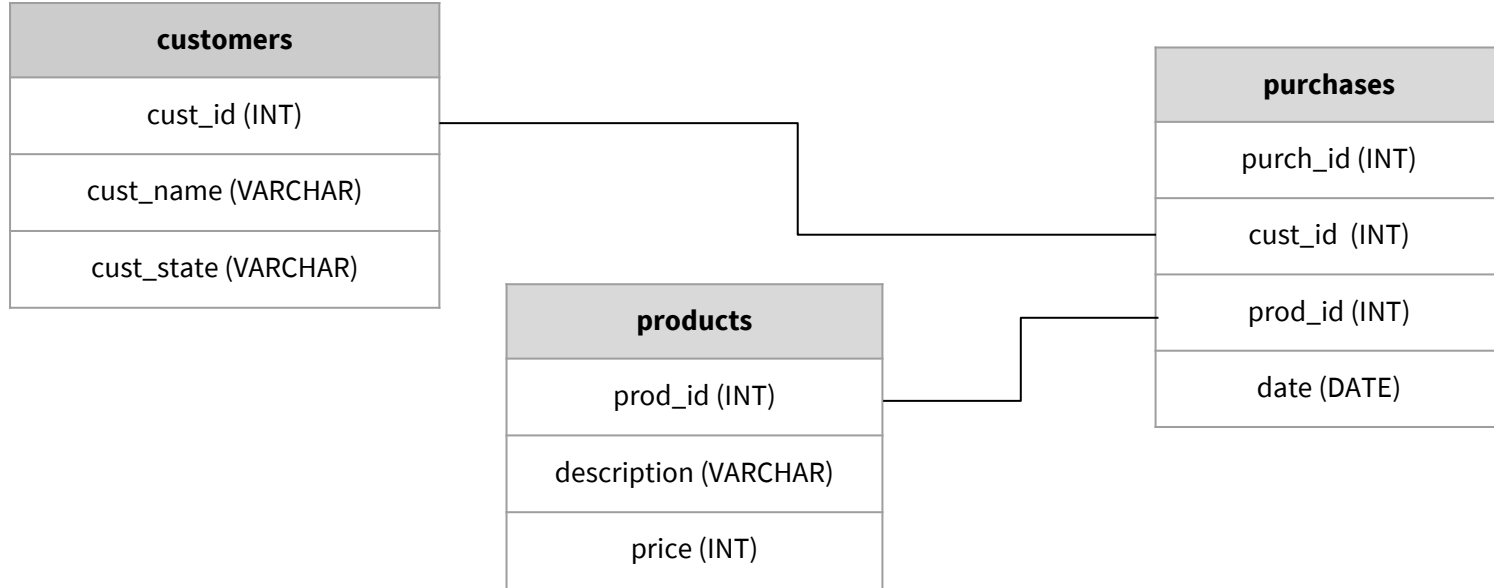
products

prod_id	description	price
1	skis	\$300
2	goggles	\$75
3	snowboard	\$400

purchases

purch_id	cust_id	prod_id	date
1	1	1	10/30
2	1	2	11/14
3	2	3	11/18
4	3	1	12/11
5	4	1	12/19
6	3	2	12/24

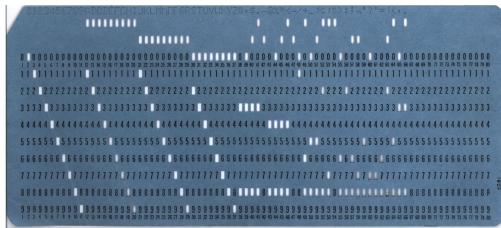
Entity Relationship Diagram (ERD)



Why RDBMS?

RDBMS provides one means of *persistent* data storage.

- Survives after the process in which it was created has ended
- Written to non-volatile storage (stored even if unpowered)



- Frequently accessed and unlikely to change in structure
- (e.g., a company database that contains records of customers and purchases)

Why RDBMS?

RDBMS provides to ability to:

- Model relations in data
- **Query data and their relations efficiently**
- Maintain data consistency and integrity

Why RDBMS?

For a long time, RDBMS was the *de facto* standard for storing data:

- Examples: Oracle, MySQL, SQL Server, Postgres
- In the era of “Big Data,” other options are emerging
 - however... ([link](#))
- There will always be legacy users
- The network effect
- To get a job (SQL is a desired skill on most DS job postings, and often a coding interview question or two on SQL)

RDBMS Terminology

- **Schema** defines the structure of a tables or a database
- Database is composed of a number of user-defined **tables**
- Each table has **columns** (or fields) and **rows** (or records)
- A column is of a certain **data type** such as an *integer*, *text*, or *date*

With a new data source, your first task is typically to understand the schema.

With a new data source, your first task is typically to understand the schema.

This will likely take time and conversations with those that gave you access to the database or its data.

Structured Query Language (SQL)

SQL is the tool we use to interact with RDBMS. We can use SQL commands to:

- Create tables
- Alter tables
- Insert records
- Update records
- Delete records
- **Query (`SELECT`) records within or across tables**

The most critical skill for a Data Scientist--as opposed to a Data Engineer or Database Administrator--is to extract information from databases.

We will focus on writing queries in PostgreSQL, but all of the commands use similar vocabulary and syntax. ([PostgreSQL syntax](#))

SQL Query Basics

SQL queries have three main components:

SELECT # What data (columns) do you want?

FROM # From what location (table) you want it?

WHERE # What data (rows) do you want?

Note: SQL queries always return tables.

Note: SQL is a *declarative* language, unlike Python, which is *imperative*. With a declarative language, you tell the machine *what* you want, instead of *how*, and it figures out the best way to do it for you.

SELECT *

TABLE(S)

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

SELECT *

TABLE(S)

QUERY

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

```
SELECT
    *
FROM
    customers;
```

The asterisk means “everything.”

SELECT *

TABLE(S)

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

QUERY

```
SELECT
    *
FROM
    customers;
```

OUTPUT

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	MA
3	Adam	NY
4	Frank	AZ

The asterisk means “everything.”

Aliases

TABLE(S)

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

QUERY

```
SELECT
    cust_name AS name,
    cust_state state
FROM
    customers;
```

OUTPUT

name	state
Kayla	CO
Erich	CO
Adam	NY
Frank	AZ

- Aliasing can be used to rename columns and even tables (more on this later).
- “AS” makes code clearer but is not necessary.
- Be careful not to use keywords (e.g. count) as aliases!

WHERE

TABLE(S)

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

QUERY

```
SELECT
    cust_name AS name,
    cust_state AS state
FROM
    customers
WHERE
    cust_state = 'CO' ;
```

OUTPUT

name	state
Kayla	CO
Erich	CO

- WHERE specifies criterion for selecting specific rows (row filter)
- Note that the WHERE statement must reference the original column name, not the alias
- However, WHERE can reference a table column that is not in SELECT (e.g. cust_id)

Breakout #1

Using:

a text editor (to write and save your queries)

Terminal (to run PostgreSQL)

Schema and Data from

Introduction to SQL for Data Scientists by Ben Smith

Answer:

Queries 0 and 1 in `queries.sql`

(You'll first need to create the database using
`breakout_db_creation_guide.md`)

Formatting SQL statements

Unlike Python, whitespace and capitalization do not matter (except for strings)

```
select column1, column2 from my_table;
```

Convention is to use ALL CAPS for keywords

Line breaks and indentation help make queries more readable (especially complex ones)

```
SELECT
    column1,
    column2
FROM
    my_table;
```

Punctuation such as commas (between items under each clause) and semicolons (after each statement) are required for proper evaluation

WHERE (Multiple Criteria)

TABLE(S)

QUERY

OUTPUT

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

```
SELECT
    cust_name AS name,
    cust_state AS state
FROM
    customers
WHERE
    (cust_state = 'CO'
    AND cust_name = 'Kayla')
    OR cust_state = 'NY' ;
```

name	state
Kayla	CO
Adam	NY

- We can specify multiple conditions on the “WHERE” clause by using AND/OR
- Note that comparison operator uses a single equal sign (= instead of ==)

LIMIT and ORDER BY

TABLE(S)

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

QUERY

```
SELECT
    *
FROM
    customers
ORDER BY
    cust_name DESC
LIMIT 3;
```

OUTPUT

cust_id	cust_name	cust_state
1	Kayla	CO
4	Frank	AZ
2	Erich	CO

- ORDER BY is ascending by default; specify DESC for reverse sorting
- LIMIT specifies the number of records returned

SELECT DISTINCT

TABLE(S)

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

QUERY

```
SELECT DISTINCT
  cust_state
FROM
  customers;
```

OUTPUT

cust_state
CO
NY
AZ

- SELECT DISTINCT grabs all the unique records.
- If multiple columns are selected, then all unique combinations are returned.

ARITHMETIC OPERATORS (+, -, *, /, etc.)

TABLE(S)

QUERY

OUTPUT

products

prod_id	description	price
1	skis	300
2	goggles	75
3	snowboard	400

```
SELECT
    description,
    price,
    price * 2 AS ripoff
FROM
    products;
```

description	price	ripoff
skis	300	600
goggles	75	150
snowboard	400	800

- Arithmetic operators are similar to Python (except SQL uses ^ for exponents)
- Can be used with multiple columns (for example, adding one column value to another)

ARITHMETIC OPERATORS and DATA TYPES

TABLE(S)

QUERY

OUTPUT

products

prod_id	description	price
1	skis	300
2	goggles	75
3	snowboard	400

```
SELECT
    description,
    price,
    price/2 AS sale_int,
    price/2. AS sale_float
FROM
    products;
```

description	price	sale_int	sale_float
skis	300	150	150.0
goggles	75	37	37.5
snowboard	400	200	200.0

- Arithmetic operators are similar to Python (except SQL uses ^ for exponents)
- Can be used with multiple columns (for example, adding one column value to another)

CASE WHEN

TABLE(S)

QUERY

OUTPUT

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

```
SELECT
    cust_name AS name,
    CASE WHEN cust_state = 'CO' THEN 1
          ELSE 0 END AS in_state
FROM
    customers;
```

name	in_state
Kayla	1
Erich	1
Adam	0
Frank	0

- CASE WHEN statement is the SQL version of an if-then-else statement
- Used in the SELECT clause
- Can combine multiple WHEN statements and/or multiple conditionals

Aggregators

TABLE(S)

QUERY

OUTPUT

products

prod_id	description	price
1	skis	300
2	goggles	75
3	snowboard	400

```
SELECT
    COUNT ( * ) ,
    MAX (price)
FROM
    products;
```

COUNT	MAX
3	400

- Aggregators combine information from multiple rows into a single row.
- Other aggregators include MIN, MAX, SUM, COUNT, STDDEV, etc.

Breakout #2

Using:

a text editor (to write and save your queries)

Terminal (to run PostgreSQL)

Schema and Data from

Introduction to SQL for Data Scientists by Ben Smith

Answer:

Queries 2 and 3 in `queries.sql`

GROUP BY

TABLE(S)

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

QUERY

```
SELECT
    cust_state as state,
    count(*)
FROM
    customers
GROUP BY
    cust_state;
```

OUTPUT

state	count(*)
CO	2
NY	1
AZ	1

- The GROUP BY clause calculates aggregate statistics for groups of data
- **Any column that is not an aggregator *must* be in the GROUP BY clause** (for example, if we added `cust_name` to the SELECT clause only, SQL would not know whether to return Kayla or Erich in the CO row)
- Any column in the GROUP BY clause must also appear in the SELECT clause (true of Postgres but not MySQL)

GROUP BY and WHERE

TABLE(S)

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

QUERY

```
SELECT
    cust_state AS state,
    COUNT(*) AS total
FROM
    customers
WHERE
    cust_name != 'Adam'
GROUP BY
    cust_state;
```

OUTPUT

state	total
CO	2
AZ	1

GROUP BY and WHERE (cont'd)

TABLE(S)

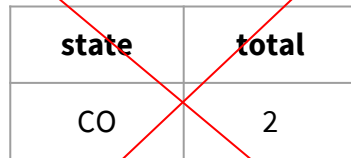
customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

QUERY

```
SELECT
    cust_state AS state,
    COUNT(*) AS total
FROM
    customers
WHERE
    COUNT(*) >= 2
GROUP BY
    cust_state;
```

OUTPUT



state	total
CO	2

ERROR

- Why does the query above not work?

GROUP BY and HAVING

TABLE(S)

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

QUERY

```
SELECT
    cust_state AS state,
    COUNT(*) AS total
FROM
    customers
WHERE
    count(*) >= 2
GROUP BY
    cust_state
HAVING
    COUNT(*) >= 2;
```

OUTPUT

state	total
CO	2

- Use HAVING instead of WHERE when filtering rows *after* aggregation
- WHERE clause filters rows in the root table *before* aggregation
- Like WHERE clause, HAVING clause cannot reference an alias (in Postgres, at least)

Joining Tables

The JOIN clause allows us to use a single query to extract information from multiple tables.

Every JOIN statement has two parts:

1. Specifying the tables to be joined (JOIN)
2. Specifying the columns to join tables on (ON)

For example, we could learn the home state of every purchaser of an item:

1. JOIN the *purchases* table (history of purchase events) and the *customers* table (info about customers)
2. ON the *cust_id* column, which appears in both tables

purchases

purch_id	cust_id	prod_id	date
1	1	1	10/30
2	1	2	11/14
3	2	3	11/18
4	3	1	12/11
5	4	1	12/19
6	3	2	12/24

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

JOINS

QUERY

```
SELECT
    purchases.purch_id,
    customers.cust_id,
    customers.cust_state
FROM
    purchases
JOIN
    customers
ON
    purchases.cust_id =
    customers.cust_id;
```

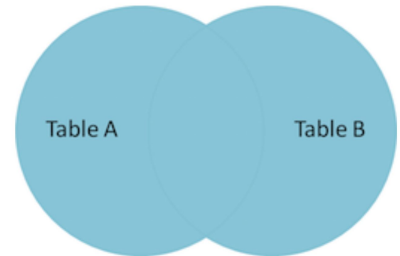
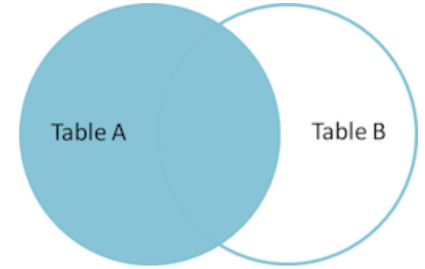
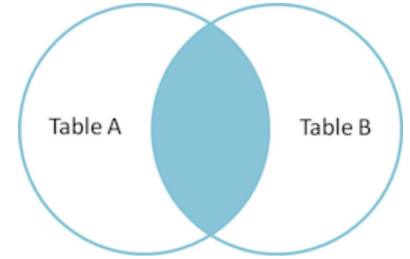
OUTPUT

purch_id	cust_id	cust_state
1	1	CO
2	1	CO
3	2	CO
4	3	NY
5	4	AZ
6	3	NY

JOIN Types

SELECT ... FROM TableA _____ JOIN TableB ON...

- **(INNER) JOIN:** Discards any entries that do not have match between the keys specified in the ON clause. No null/nan values.
- **LEFT (OUTER) JOIN:** Keeps all entries in the left (FROM) table, regardless of whether any matches are found in the right (JOIN) tables. Some null/nan values.
 - **RIGHT (OUTER) JOIN:** Is the same, except keeps all entries in the right (JOIN) table instead of the left (FROM) table); usually avoided because it does the same thing as a LEFT join
- **FULL (OUTER) JOIN:** Keeps the rows in both tables no matter what. More null/nan values.



purchases

purch_id	cust_id	prod_id	date
1	1	1	10/30
2	1	2	11/14
3	2	3	11/18
4	3	1	12/11
5	NULL	1	12/19
6	NULL	2	12/24

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ
5	Neil	NY

(INNER) JOIN

QUERY

```
SELECT
    purchases.purch_id,
    customers.cust_id,
    customers.cust_state
FROM
    purchases
INNER JOIN
    customers
ON
    purchases.cust_id =
    customers.cust_id;
```

OUTPUT

purch_id	cust_id	cust_state
1	1	CO
2	1	CO
3	2	CO
4	3	NY

INNER JOIN discards records that do not have a match in both tables

purchases

purch_id	cust_id	prod_id	date
1	1	1	10/30
2	1	2	11/14
3	2	3	11/18
4	3	1	12/11
5	NULL	1	12/19
6	NULL	2	12/24

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ
5	Neil	NY

LEFT (OUTER) JOIN

QUERY

```
SELECT
    purchases.purch_id,
    customers.cust_id,
    customers.cust_state
FROM
    purchases
LEFT OUTER JOIN
    customers
ON
    purchases.cust_id =
    customers.cust_id;
```

OUTPUT

purch_id	cust_id	cust_state
1	1	CO
2	1	CO
3	2	CO
4	3	NY
5	NULL	NULL
6	NULL	NULL

LEFT OUTER JOIN retains all records from the left (FROM) tables and includes records from the right (JOIN) table if they are available

purchases

purch_id	cust_id	prod_id	date
1	1	1	10/30
2	1	2	11/14
3	2	3	11/18
4	3	1	12/11
5	NULL	1	12/19
6	NULL	2	12/24

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ
5	Neil	NY

FULL (OUTER) JOIN

QUERY

```
SELECT
    purchases.purch_id,
    customers.cust_id,
    customers.cust_state
FROM
    purchases
FULL OUTER JOIN
    customers
ON
    purchases.cust_id =
    customers.cust_id;
```

OUTPUT

purch_id	cust_id	cust_state
1	1	CO
2	1	CO
3	2	CO
4	3	NY
5	NULL	NULL
6	NULL	NULL
NULL	4	AZ
NULL	5	NY

FULL OUTER JOIN retains all records from both tables regardless of matches

Query Components vs. Order of Evaluation

Order of Components	Order of Evaluation
SELECT	5 - Targeted list of columns evaluated and returned
FROM	1 - Product of all tables is formed
JOIN / ON	
WHERE	2 - Rows filtered out that do not meet condition
GROUP BY	3 - Rows combined according to GROUP BY clause and aggregations applied
HAVING	4 - Aggregations that do not meet that HAVING criteria are removed
ORDER BY	6 - Rows sorted by column(s)
LIMIT	7 - Final table truncated based on limit size
;	8 - Semicolon included as reminder

Breakout #3

Using:

a text editor (to write and save your queries)

Terminal (to run PostgreSQL)

Schema and Data from

Introduction to SQL for Data Scientists by Ben Smith

Answer:

Queries 4, 5, and 6 in `queries.sql`

Subqueries

- In general, you can replace any table name with a subquery:

```
SELECT ... FROM (SELECT ...)
```

- If a query returns a single value, you can use it as such:

```
...WHERE column1 = (SELECT ...)
```

- If a query returns a single column, you can treat it like a vector:

```
...WHERE column1 IN (SELECT ...)
```

Breakout #4

Using:

a text editor (to write and save your queries)

Terminal (to run PostgreSQL)

Schema and Data from

Introduction to SQL for Data Scientists by Ben Smith

Answer:

Queries 7, 8, and 9 in `queries.sql`

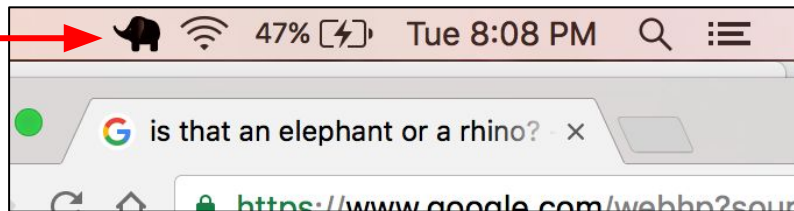
Objectives

- Describe why we use Relational Database Management Systems (RDBMS)
- Understand primary keys, foreign keys, and table relationships
- Write simple SQL queries on a single table using SELECT, FROM, WHERE, GROUP BY, ORDER BY clauses as well as aggregation functions (COUNT, AVG, etc.)
- Write more complex SQL queries using joins and subqueries
- Interact with a Postgres database from the command line
- Learn how to say PostgreSQL ([link](#))

Appendix

- Instructions on Postgres installation and set-up are in the *individual.md* file
- Postgres must be running in order to use it from the command line:

Look for
ambiguous
ungulate icon



- Instructions on loading the database and entering postgres prompt from the command line are also in the *individual.md* file

Load .sql file into a DB and run queries

One-time step to create a database and load .sql file. From the command line:

```
$ psql
# CREATE DATABASE MyDatabase;
# \q
$ psql MyDatabase < file.sql
```

Now you can access this database any time:

```
psql MyDataBase
```

Some Postgres commands

Useful commands from the psql interactive shell prompt:

- `\l` - list all databases
- `\d` - list all tables
- `\d <table name>` - describe a table's schema
- `\h <clause>` - Help for SQL clause help
- `q` - exit current view and return to command line
- `\q` - quit psql
- `\i script.sql` - run script (or query)