GENERAL ASSEMBLY

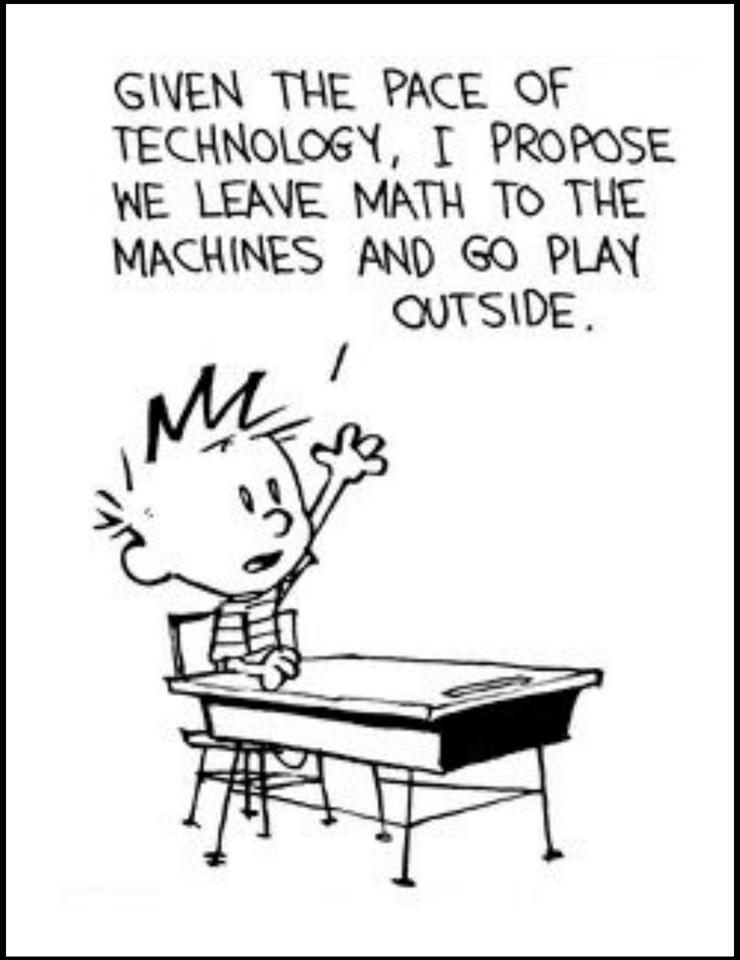
DATABASES

Adam Jones, PhD

Data Scientist @ Critical Juncture

GENERAL ASSEMBLY

DATABASES



Adam Jones, PhD

Data Scientist @ Critical Juncture

LEARNING OBJECTIVES

- Understanding of the uses and differences of databases
- Accessing databases from Pandas
- Navigate Postgres

COURSE

PRE-WORK

PRE-WORK REVIEW

- There will be multiple ways to run the exercises:
  - Install local Postgres
    - If `brew` is installed, this should be as simple as `brew install postgres`

OPENING

DATABASES

DATABASES

- Today's lesson will be on databases and the SQL query language

- Databases are the standard solution for data storage

    - They're far more robust than text and CSV files

- They come in many flavors, but we'll explore the most common: *relational databases*

DATABASES

- Relational databases also come in different varieties, but almost all use SQL as a basis for querying (i.e. retrieving) data
- Most analyses typically involve pulling data from a database

DATABASES

DATABASES

- Databases are computer systems that manage the storage and querying of datasets
- They provide a way to organize the data on disk (i.e. hard drive) and efficient methods to retrieve information
  - Databases allow a user to create rules that ensure proper data management and verification
- Typically, retrieval is performed using a query language, a mini programming language with a few basic operators for data transformation
- The most common query language is ***SQL*** (Structured Query Language)

DATABASES

- A *relational database* is based on links between data entities or concepts
- Typically, a relational databases is organized into *tables*
- Each table should correspond to one entity or concept
    - Each table is similar to a single CSV file or Pandas dataframe
- For example, consider an application like Twitter
    - Our two main entities are Users and Tweets
    - For each of these, we would have a separate table

DATABASES

- A table is made up of rows and columns, similar to a Pandas dataframe or Excel spreadsheet

- Each table has a specific *schema*, a set of rules for what goes in each table
  - These specify which columns are contained in the table and what *type* of data is in each column (e.g. text, integers, decimals, etc)

Users Table Schema	
user_id	char
user_sign_up_date	date
user_follower_count	int

DATABASES

- This means you can’t add text data to an integer column in that database
- The additional *type* information make this constraint stronger than the header of a CSV file
- For this reason and many others, databases allow for stronger consistency of the data and are often a better solution for data storage

DATABASES

- Each table typically has a *primary key* column
    - This column has a unique value per row and serves as the identifier for the row
- A table can have many *foreign keys* as well
    - A *foreign key* is a column that contains values to link the table to the other tables
- These keys that link the table together define the relational database

DATABASES

- For example, the tweets table may have as columns:
  - tweet_id - the primary key tweet identifier
  - tweet_text
  - user_id - a foreign key to the users table

Users Table Schema	
user_id	char
user_sign_up_date	date
user_follower_count	int

Tweets Table Schema	
tweet_id	int
tweet_text	char
user_id	char

DATABASES

- MySQL and Postgres are popular variants of relational databases and are widely used
  - Both are open-source and available for free
- Alternatively, many companies use proprietary software such as Oracle or Microsoft SQL databases
- While these databases offer many of the same features and use the same SQL language, the latter two offer some maintenance features and support that large companies find useful

NORMALIZED VS DENORMALIZED DATA

- Once we start organizing our data into tables, we start to separate it into *normalized* and *denormalized* setups

- *Normalized* structures have a single table per entity and use many foreign keys or link tables to connect the entities

- *Denormalized* structures have fewer tables that combine different entities

NORMALIZED VS DENORMALIZED DATA

- With our Twitter example, a *normalized* structure would place users and tweets in different tables

Users Table Schema	
user_id	char
user_sign_up_date	date
user_follower_count	int

Tweets Table Schema	
tweet_id	int
tweet_text	char
user_id	int

NORMALIZED VS DENORMALIZED DATA

- A *denormalized* structure would put them both in one table

Twitter Table Schema	
tweet_id	int
tweet_text	char
user_id	int
user_sign_up_date	date
user_follower_count	int

NORMALIZED VS DENORMALIZED DATA

Denormalized structures:

- Duplicates a lot of information
- Makes data easy to access since it's all in one table

Normalized structures:

- Save storage space by separating information
- Requires joining of table to access information about two different entities, a slower operation

ALTERNATIVE DATABASES

- While relational databases are the most popular and broadly used, specific applications may require different data organization

- You don’t need to know every variety, but it’s good to know some overall themes

KEY-VALUE STORES

- Key-Value databases are nothing more than very large and very fast hashmaps or dictionaries
- These are useful for storing key based data
  - e.g. a count of things per user or customer, a last visit per customer
- Every entry in these databases has two values, a key and a value
  - We can retrieve any value based upon its key

KEY-VALUE STORES

- This is exactly like a python dictionary, but it can be larger than your memory (i.e. RAM)
  - So these systems use smart caching algorithms to ensure frequently or recently accessed items are quickly accessible

- Popular key-value stores include *Cassandra* and *MemcacheDB*

NOSQL OR DOCUMENT DATABASES

- “NoSQL” databases are those that don’t rely on a traditional relational table setup and more flexible in their data organization

- Typically they actually **do** have SQL querying abilities but model their data differently

NOSQL OR DOCUMENT DATABASES

- Relational Structure

user_id	user_name	user_hobby_1	user_hobby_2	user_age
13123	robby_g	guitar	cars	25
18423	jt1235	football		31

- NoSQL Data Structure

```
{
    “user_id”: 13123,
    “user_name”: “robby_g”,
    “user_hobbies”: [“guitar”, “cars”],
    “user_age”: 25
}
```

```
{
    “user_id”: 19423,
    “user_name”: “jt1235”,
    “user_hobbies”: [“football”],
    “user_age”: 31
}
```

NOSQL OR DOCUMENT DATABASES

- They may organize data on an entity level, but often have denormalized and nested data setups
- This nested data layout is often similar to that in JSON documents
- Popular databases include *MongoDB* and *CouchDB*

NOSQL OR DOCUMENT DATABASES

Relational data model

Highly-structured table organization with rigidly-defined data formats and record structure.

Document data model

Collection of complex documents with arbitrary, nested data formats and varying "record" format.

NOSQL OR DOCUMENT DATABASES

- The following is an example of the storage document for a tweet


```
{
  "created_at": "Mon Sep 24 03:35:21 +0000 2012",
  "id_str": "250075927172759552",
  "entities": {
    "hashtags": [
      {
        "text": "freebandnames",
        "indices": [
          20,
          34
        ]
      }
    ],
    "user_mentions": [

    ]
  }
}
```

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS

In the following examples, which might be the best storage or database solution? Why?

1. An application where a user can create a profile
2. An online store
3. Storing the last visit date of a user

DELIVERABLE

Answers to the above questions

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS

Consider a dataset from Uber with the following fields:

- User ID
- User Name
- Driver ID
- Drive Name
- Ride ID
- Ride Time
- Pickup Latitude
- Pickup Longitude
- Pickup Location Entity
- Drop-off Longitude
- Drop-off Latitude
- Drop-off Location Entity
- Miles
- Travel Time
- Fare
- CC Number

How would you design a relational database to support this data? What tables would you create, what fields would they contain, and how would they link to other tables?

DELIVERABLE

Your database schema design

DEMO

ACCESSING DATABASES FROM PANDAS

ACCESSING DATABASES FROM PANDAS

- While databases provide many analytical capabilities, often it's useful to pull the data back into Python for more flexible programming

- Large, fixed operations would be more efficient in a database, but Pandas allows for interactive processing

- For example, if you just want to aggregate login or sales data to present a report or dashboard, this operation is operating on a large dataset and not often changing

- This would run very efficiently in a database vs connecting to Python

ACCESSING DATABASES FROM PANDAS

- However, if we want to investigate the login or sales data further and ask more interactive questions, then using Python would come in very handy

- Pandas can be used to connect to most relational databases

```python
import pandas as pd
from pandas.io import sql
```

ACCESSING DATABASES FROM PANDAS

- In this demonstration, we will create and connect to a SQLite database
    - SQLite creates portable relational databases saved in a single file

- These databases are stored in a very efficient manner and allow fast querying, making them ideal for small databases or databases that need to be moved across machines

- Additionally, SQLite databases can be created with the setup of MySQL or Postgres databases

ACCESSING DATABASES FROM PANDAS

- We can create a SQLite databases as follows

```
import sqlite3

conn = sqlite3.connect('dat-test.db')
```

- This creates a file, `dat-test.db`, which will act as a relational/SQL database

WRITING DATA INTO A DATABASE

- Data in Pandas can be loaded into a relational database
  - For the most part, Pandas can use the databases column information to infer the schema for the table it creates

- Let's return to the Rossmann sales data and load it into our database

```
import pandas as pd

data = pd.read_csv('../../lesson-15/code/datasets/rossmann.csv',
low_memory=False)

data.head()
```

WRITING DATA INTO A DATABASE

- Data is moved to the database with the `to_sql` command, similar to the `to_csv` command

- `to_sql` takes several arguments
  - `name` - the table name to create
  - `con` - a connection to a database
  - `index` - whether to input the index column
  - `schema` - if we want to write a custom schema for the new table
  - `if_exists` - what to do if the table already exists (overwrite it, add to it, or fail)

WRITING DATA INTO A DATABASE

- The following code loads the Rossmann sales data to our database

```
data.to_sql('rossmann_sales',
            con=conn,
            if_exists='replace',
            index=False)
```

READING FROM A DATABASE

- If we already have data in the database, we can use Pandas to query our database

- Querying is done through the `read_sql` command in the `sql` module

```
import pandas as pd
from pandas.io import sql
```

```
sql.read_sql('SELECT * FROM rossmann_sales LIMIT 10;', con=conn)
```

- This runs the query passed in and returns a dataframe with the results

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS

1. Load the Rossmann Store metadata in `rossmann-stores.csv` and create a table in the database with it

DELIVERABLE

Created table for store metadata

DEMO

SQL SYNTAX: SELECT, WHERE, GROUP BY, JOIN

SQL OPERATORS: SELECT

- Every query should start with `SELECT` followed by the names of the columns in the output

- `SELECT` is always paired with `FROM`, which identifies the table to retrieve data from

```
SELECT
<columns>
FROM
<table>;
```

- `SELECT *` denotes returning *all* of the columns

SQL OPERATORS: SELECT

- Rossmann Stores example:

```
SELECT
Store, Sales
FROM rossmann_sales;
```

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS

1. Write a query for the Rossmann Sales data that returns `Store`, `Date`, and `Customers`

DELIVERABLE

The requested query

SQL OPERATORS: WHERE

- `WHERE` is used to filter a table using a specific criteria
  - The `WHERE` clause follows the `FROM` clause

```
SELECT <columns>
FROM <table>
WHERE <condition>
```

- The condition is some filter applied to the rows, where rows that match the condition will be output

SQL OPERATORS: WHERE

- Rossmann Stores example:

```sql
SELECT Store, Sales
FROM rossmann_sales
WHERE Store = 1;
```

```sql
SELECT Store, Sales
FROM rossmann_sales
WHERE Store = 1 and Open = 1;
```

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS

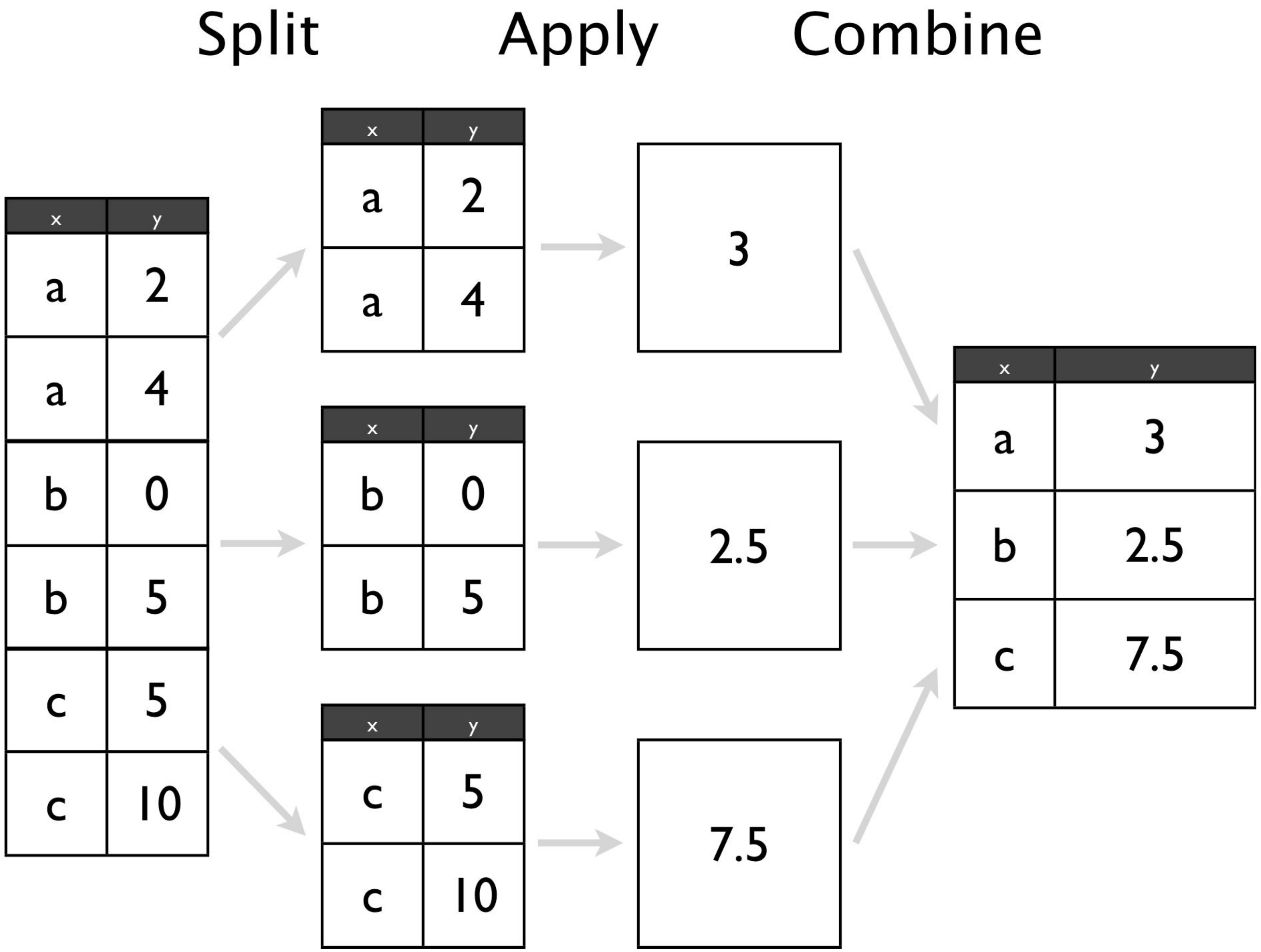
1. Write a query for the Rossmann Sales data that returns `Store`, `Date`, and `Customers` for stores that were open and running a promotion

DELIVERABLE

The requested query

SQL OPERATORS: GROUP BY

- GROUP BY allows us to aggregate over any field in the table by applying the concept of 'Split, Apply, Combine'
- We identify some key with which we want to segment the rows
- Then, we roll up or compute some statistics over all of the rows that match that key



SQL OPERATORS: GROUP BY

- `GROUP BY` *must* be paired with an aggregate function, the statistic we want to compute in the rows, in the `SELECT` statement

- `COUNT(*)` denotes counting up all of the rows
  - Other aggregate functions commonly available are `AVG` (average), `MAX`, `MIN`, and `SUM`

- If we want to aggregate over the entire table, without results specific to any key, we can use an aggregate function in the `SELECT` clause and ignore the `GROUP BY` clause

SQL OPERATORS: GROUP BY

- Rossmann Stores example:

```
SELECT Store, SUM(Sales), AVG(Customers)
FROM rossmann_sales
WHERE Open = 1
GROUP BY Store;
```

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS

1. Write a query that returns the total sales on the promotion and non-promotion days

DELIVERABLE

The requested query

SQL OPERATORS: ORDER BY

- `ORDER BY` is used to sort the results of a query

```
SELECT <columns>
FROM <table>
WHERE <condition>
ORDER BY <columns>;
```

- You can order by multiple columns in ascending (`ASC`) or descending (`DESC`) order

SQL OPERATORS: ORDER BY

- Rossmann Stores example:

```
SELECT Store, SUM(Sales) as total_sales, AVG(Customers)
FROM rossmann_sales
GROUP BY Store
WHERE Open = 1
ORDER BY total_sales desc;
```

- `SUM(Sales) as total_sales` renames the `SUM(Sales)` value to `total_sales` so we can refer to it later in the `ORDER BY` clause

SQL OPERATORS: JOIN

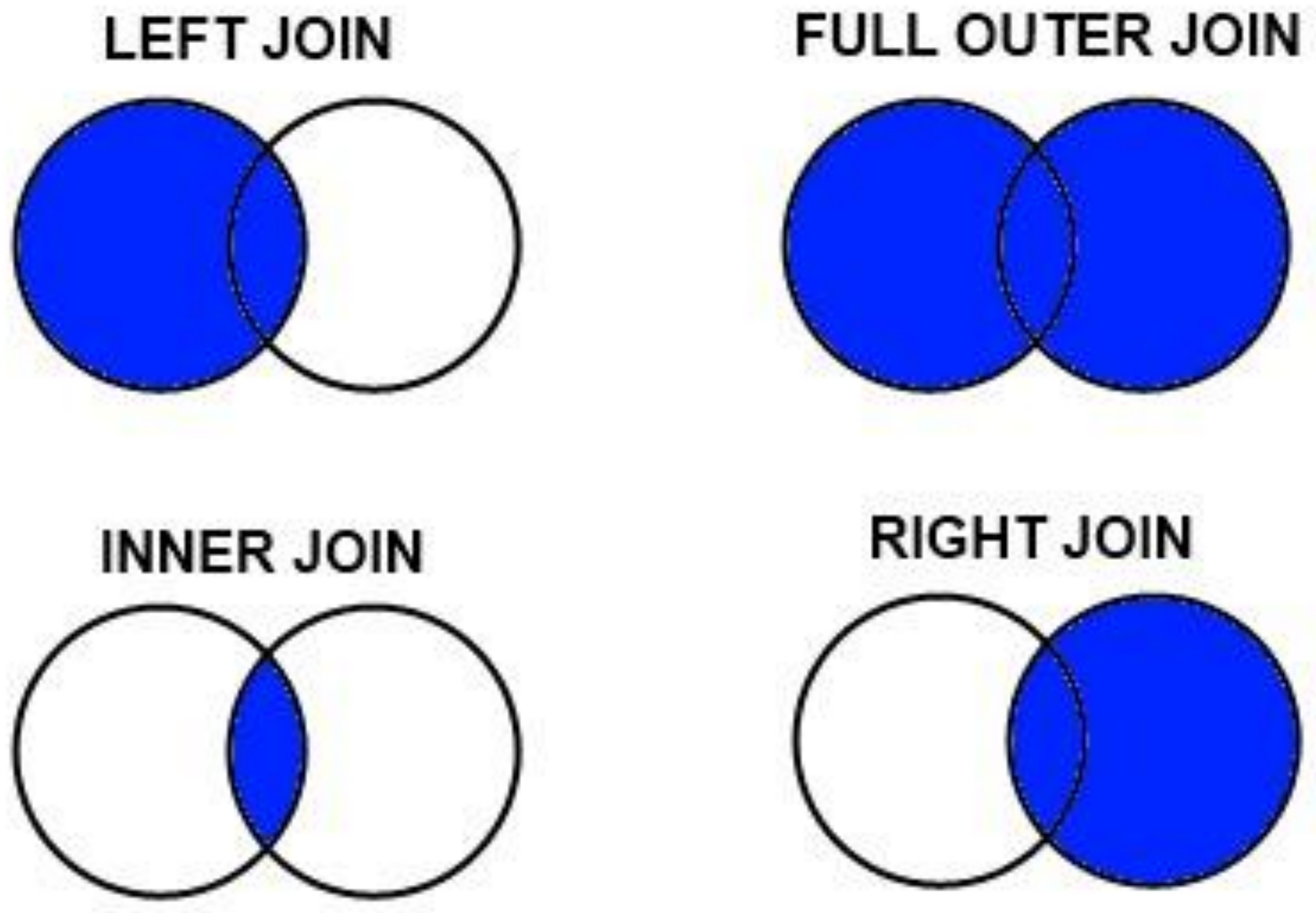
- `JOIN` allows us to access data across many tables

- We specify how a row in one table links to another

```
SELECT a.Store, a.Sales, s.CompetitionDistance
FROM rossmann_sales a
JOIN rossmann_stores s
ON a.Store = s.Store;
```

- Here, ON denotes an *inner* join

SQL OPERATORS: JOIN



https://www.shanelynn.ie/wp-content/uploads/2017/03/join-types-merge-names.jpg

SQL OPERATORS: JOIN

- By default, most joins are an *Inner Join*, which means only when there is a match in both tables does a row appear in the results
- If we want to keep the rows of one table *even if there is no matching counterpart*, we can perform an *Outer Join*
- Outer joins can be `LEFT`, `RIGHT`, or `FULL`, meaning keep all of the left rows, all the right rows, or all the rows, respectively

INDEPENDENT PRACTICE

PANDAS AND SQL

ACTIVITY: PANDAS AND SQL

EXERCISE

DIRECTIONS (40 minutes)

1. Load the Walmart sales and store features data
2. Create a table for each of those datasets
3. Select the store, date and fuel price on days it was over 90 degrees
4. Select the store, date and weekly sales and temperature
5. What were average sales on holiday vs. non-holiday sales?
6. What were average sales on holiday vs. non-holiday sales when the temperature was below 32 degrees?

DELIVERABLE

Answers to the above questions

DEMO

INSTALLING POSTGRES

INSTALLING POSTGRES

- On a Mac, `brew install postgres`
- `brew` will provide a few commands to make sure postgres runs on startup
- If this is done, you can use the Postgres command line tool

POSTGRES SHELL

- Starting Postgres: `psql`
- Listing tables : `\dt`
- Creating a table:

```
CREATE TABLE example(
    id int,
    name varchar(140),
    value float
);
```

POSTGRES SHELL

- Inserting a row:

```
INSERT INTO example VALUES(1, 'general assembly', 3.14);
```

- Querying the table:

```
SELECT *
FROM example;
```

INDEPENDENT PRACTICE

EXTRA SQL PRACTICE

ACTIVITY: EXTRA SQL PRACTICE

EXERCISE

DIRECTIONS

There are many options for extra SQL practice

1. **PG-Exercises**: The website pgexercises.com is a very good site for Postgres exercises. Go here to get started. Complete 3-5 questions in each of the following.
   a. Simple SQL Queries
   b. Aggregation
   c. Joins and Subqueries

ACTIVITY: EXTRA SQL PRACTICE

EXERCISE

DIRECTIONS

There are many options for extra SQL practice.

2. **Wagon**: This requires signing up for the Wagon service and downloading their application. It gives access to some sample databases.
   a. Display all tracks on which Jimmy Page was the composer.
   b. Who were the top five composers by number of tracks?
   c. Who were the top five composers by length of tracks?
   d. Select all of the albums from Led Zeppelin.
   e. Count the number of albums per artist, and display the top 10.
   f. Display the track name and album name from all Led Zeppelin albums.
   g. Compute how many songs and how long (in minutes) each Led Zeppelin album was.

CONCLUSION

TOPIC REVIEW

CONCLUSION

- While this was a brief introduction, databases are often at the core of any data analysis
  - Most analysis starts with retrieving data from a database
- SQL is a key language that any data scientist should understand
  - `SELECT`: Used in every query to define the resulting columns
  - `WHERE`: Filters rows based on a given condition
  - `GROUP BY`: Groups rows for aggregation
  - `JOIN`: Combines two tables based upon a given condition

CONCLUSION

- Pandas can be used to access data from databases as well
  - The result of the queries will end up in a Pandas dataframe
- There is *much* more to learn about query optimization if one dives further!

COURSE

BEFORE NEXT CLASS

DUE DATE

- Final Project, Part 4 due: Thurs (5/17)

LESSON

Q & A

LESSON

EXIT TICKET

DON’T FORGET TO FILL OUT YOUR EXIT TICKET