GENERAL ASSEMBLY

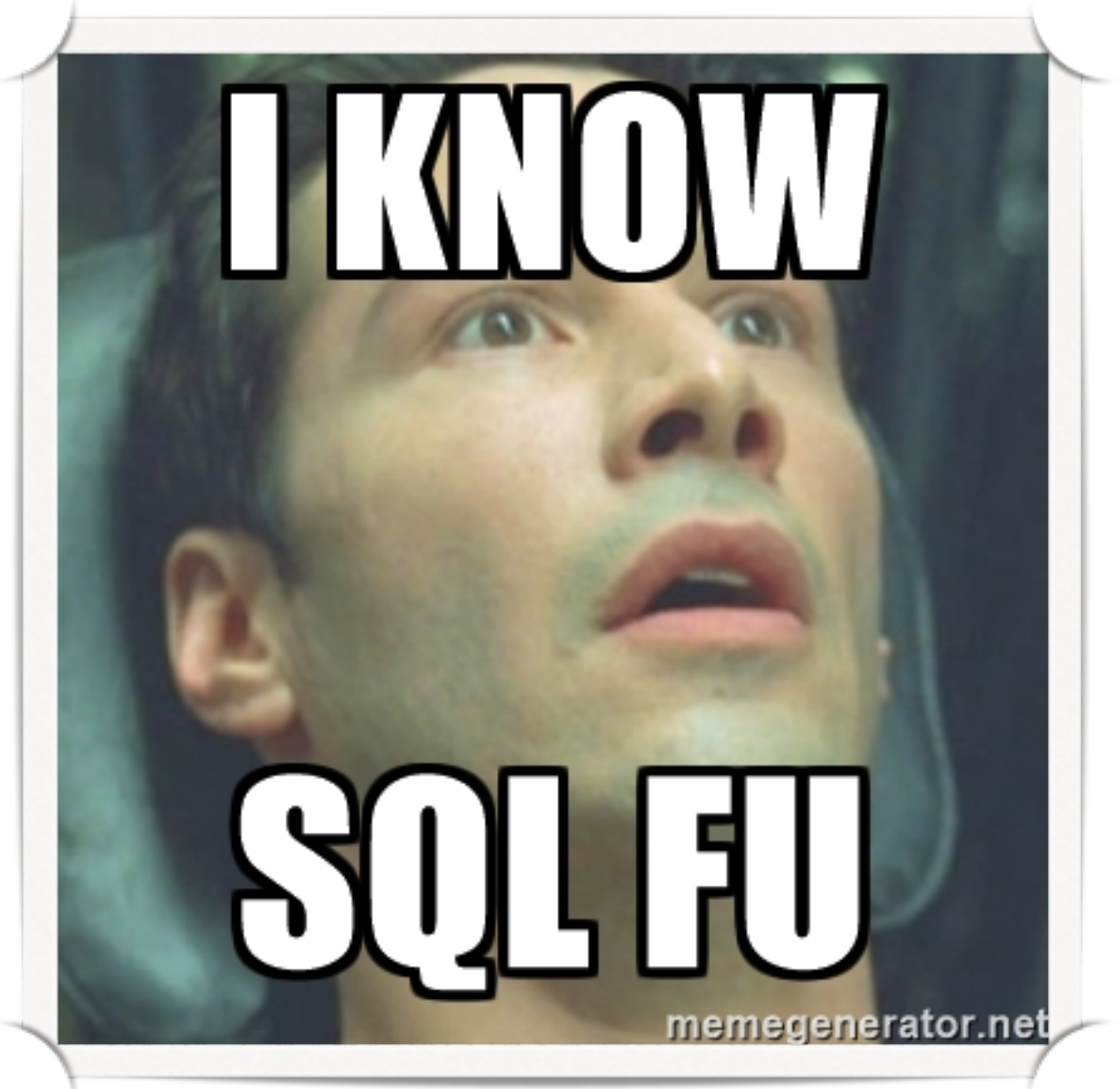
DO YOU KNOW SQL FU?

John Sandall

Data Science Consultant

@john_sandall

LEARNING OBJECTIVES



RECAP

DATABASES

RECAP: RELATIONAL DATABASES

- Relational databases separate data out into tables, linked by common columns.
- A "normalised" database structure is shown below – the **users** table and **tweets** table are linked by **user_id**

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

RECAP: NORMALISED vs DENORMALISED

- A *denormalised* 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

ACCESSING DATABASES FROM PANDAS

ACTIVITY: KNOWLEDGE CHECK

ANSWER THE FOLLOWING QUESTIONS

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

SQL SYNTAX: SELECT, WHERE, GROUP BY, JOIN

SQL OPERATORS: SELECT

- Every query should start with `SELECT`. `SELECT` is 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

ANSWER THE FOLLOWING QUESTIONS

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

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:

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

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

ACTIVITY: KNOWLEDGE CHECK

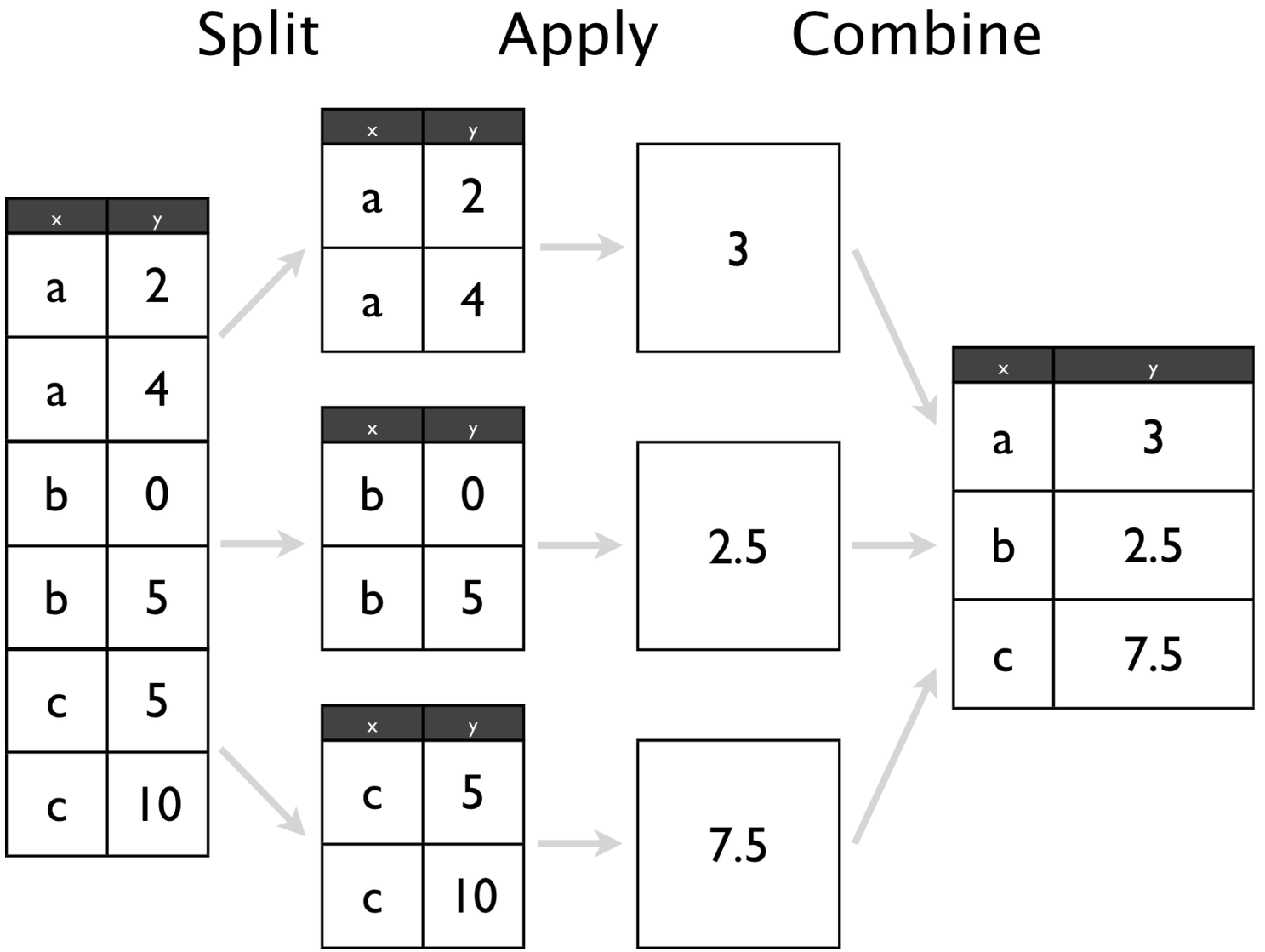
EXERCISE

ANSWER THE FOLLOWING QUESTIONS

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

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

ANSWER THE FOLLOWING QUESTIONS

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

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
WHERE Open = 1
GROUP BY Store
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
     sales.Store,
     sales.Sales,
     stores.CompetitionDistance
FROM rossmann_sales AS sales
INNER JOIN rossmann_stores AS stores ON sales.Store = stores.Store
```

- Here, INNER ON denotes an *inner* join.

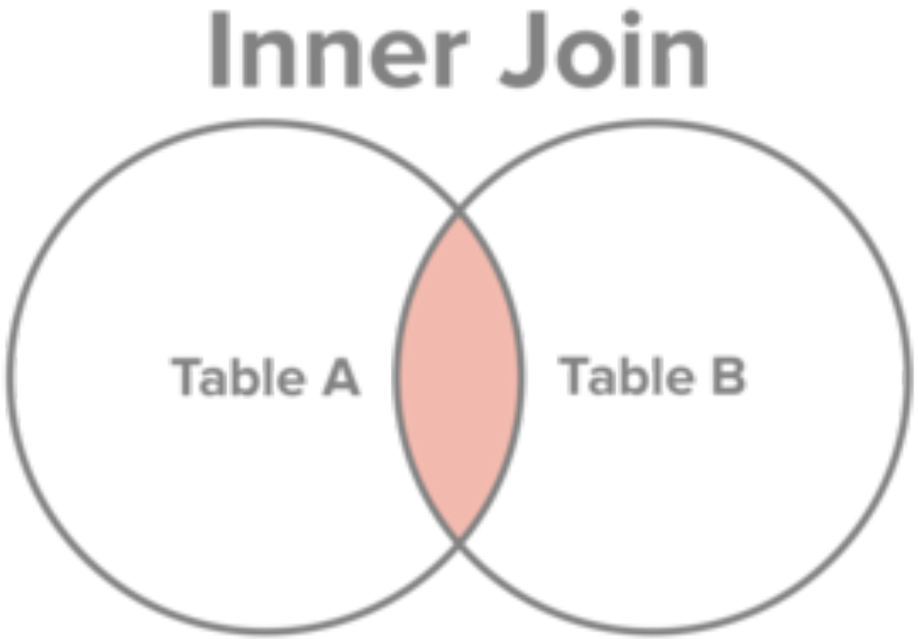
SQL OPERATORS: JOIN

- 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*.

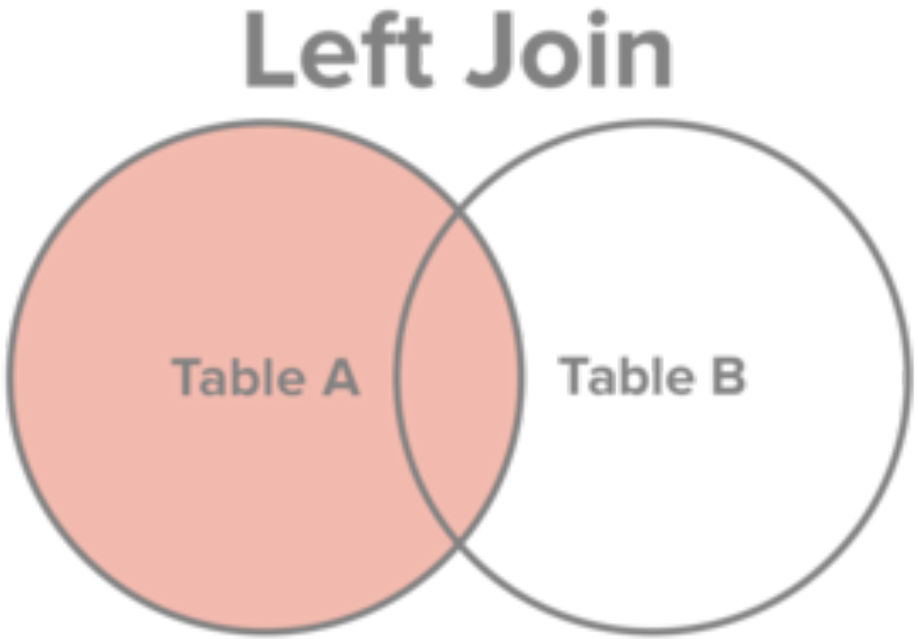
- Outer joins can be `LEFT`, `RIGHT`, or `FULL`, meaning keep all of the left rows, all the right rows, or all the rows, respectively.

JOINS: INNER JOIN



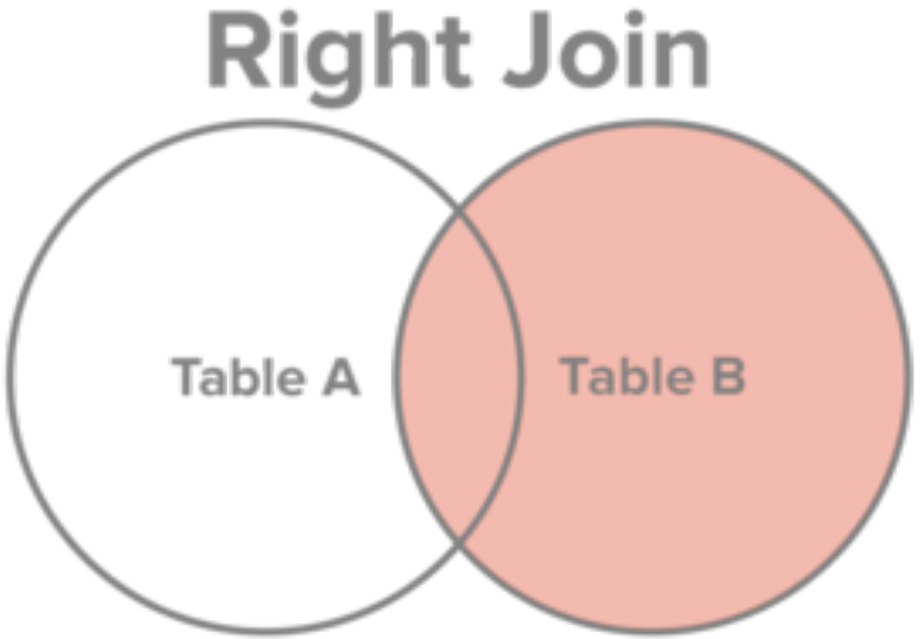
- All records in table A **and** in table B

JOINS: LEFT JOIN



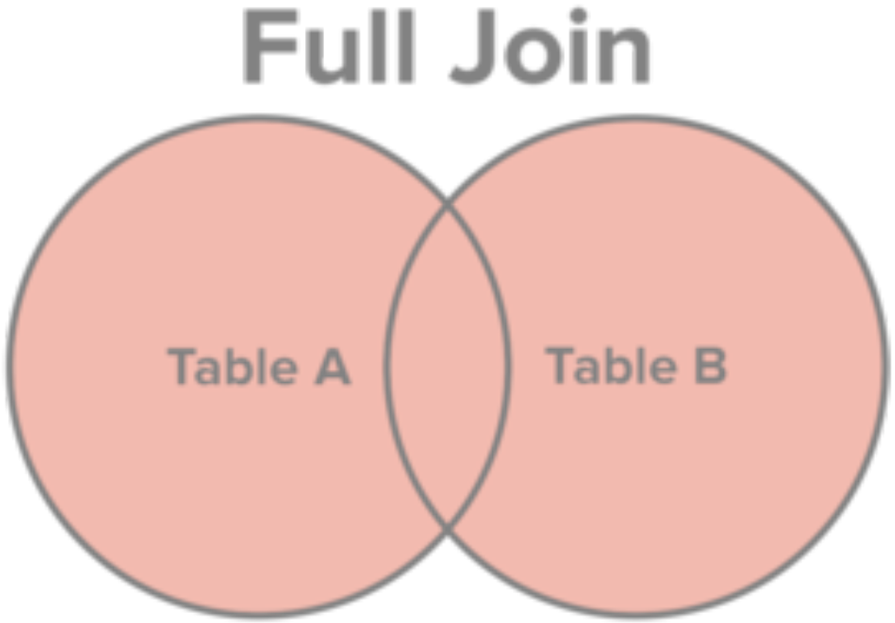
- All records in table A only
- Data from table B that matches records in table A

JOINS: RIGHT JOIN



- All records in table B only
- Data from table A that matches records in table B

JOINS: FULL JOIN



- All records in table A **and** all records in table B
- Data from table A that matches records in table B
- Data from table B that matches records in table A

SQL OPERATORS: AS

- `AS` allows us to rename tables for our own personal sanity

```
SELECT
     rossmann_sales.Store,
     rossmann_stores.CompetitionDistance
FROM rossmann_sales
INNER JOIN rossmann_stores stores
         ON rossmann_sales.Store = rossmann_stores.Store
```

SQL OPERATORS: AS

- AS allows us to rename tables for our own personal sanity

```
SELECT
    rossmann_sales.Store,
    rossmann_stores.CompetitionDistance
FROM rossmann_sales
INNER JOIN rossmann_stores stores
    ON rossmann_sales.Store = rossmann_stores.Store
```

Way too much typing!

Violates D-R-Y principles!

SQL OPERATORS: AS

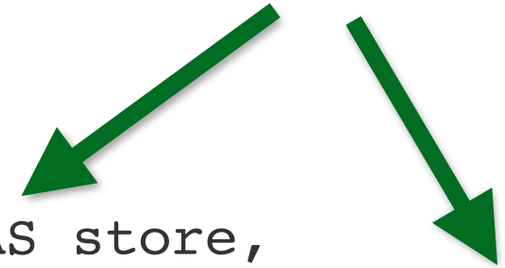
- `AS` allows us to rename tables for our own personal sanity

```
SELECT
    sales.Store,
    stores.CompetitionDistance
FROM rossmann_sales AS sales
INNER JOIN rossmann_stores AS stores ON sales.Store = stores.Store
```

Concise, easy to type, easy to read

SQL OPERATORS: AS

- AS allows us to rename columns for our own personal sanity



```
SELECT
     sales.Store AS store,
     stores.CompetitionDistance AS distance
FROM rossmann_sales AS sales
INNER JOIN rossmann_stores AS stores ON sales.Store = stores.Store
```

POSTGRES AND JSON

POSTGRESQL'S JSON SUPERPOWERS

- The following is an example of metadata for a tweet in JSON format


```
{
  "created_at": "Mon Sep 24 03:35:21 +0000 2012",
  "id_str": "250075927172759552",
  "entities": {
    "hashtags": [
      {
        "text": "datascience",
        "indices": [
          20,
          34
        ]
      }
    ],
    "text": "This is the text of a tweet #datascience"
  }
}
```

POSTGRESQL'S JSON SUPERPOWERS

- We might decide to pull out **created_at** and **id_str** and store them in their own columns in a database, but we may decided to leave **entities** as JSON "blob".

created_at	char	`Mon Sep 24 03:35:21 +0000 2012`
id_str	int	`250075927172759552`
entities	blob	`{ "hashtags": [{ "text": "datascience", "indices": [20, 34] }], "text": "This is the text of a tweet #datascience" }`

POSTGRESQL'S JSON SUPERPOWERS

- One of PostgreSQL's core strengths over MySQL is its ability to process JSON using SQL queries, allowing for a more flexible schema design:
    - you can store additional metadata without a schema migration (like a NoSQL database)
    - you still have all the power of a columnar relational database

POSTGRESQL'S JSON SUPERPOWERS

created_at	char	`Mon Sep 24 03:35:21 +0000 2012`
id_str	int	`250075927172759552`
entities	blob	`{ "hashtags": [{ "text": "datascience", "indices": [20, 34] }], "text": "This is the text of a tweet" }`

- `json_extract_path_text` allows us to extract and process JSON

```
SELECT
    id_str,
    json_extract_path(entities, "text") AS tweet_text
FROM tweets
```

INDEPENDENT PRACTICE

PANDAS AND SQL

ACTIVITY: PANDAS AND SQL

DIRECTIONS

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?

EXTRA SQL PRACTICE

ACTIVITY: EXTRA SQL PRACTICE

DIRECTIONS

PG-Exercises: The website pgexercises.com is a very good site for Postgres exercsises. Go here (pgexercises.com) to get started. Complete 3-5 questions in each of the following.

a. Simple SQL Queries
b. Aggregation
c. Joins and Subqueries