GENERAL ASSEMBLY

SQL Bootcamp

- ❖ *Fundamentals of DB & SQL*
- ❖ *Filtering & Aggregating*
- ❖ *Combing Data Tables*
- ❖ *Correlation vs. Causation*

GENERAL ASSEMBLY

QUERYING LARGE DATABASES IN SQL

Celia Fryar

OPENING

LIGHTNING ROUND REVIEW

Name two ways to create comment code.

Describe the difference between the filters **`BETWEEN`** and **`IN`**?

What is the SQL syntax to create a “not equal to” filter?

What tool allows you to override logical operators order of execution?

Are boundary values included in **`BETWEEN`** filtered data output?

Describe the difference in filtering using **`WHERE`** versus **`HAVING`**.

LEARNING OBJECTIVES

Explore combining data together from different tables.

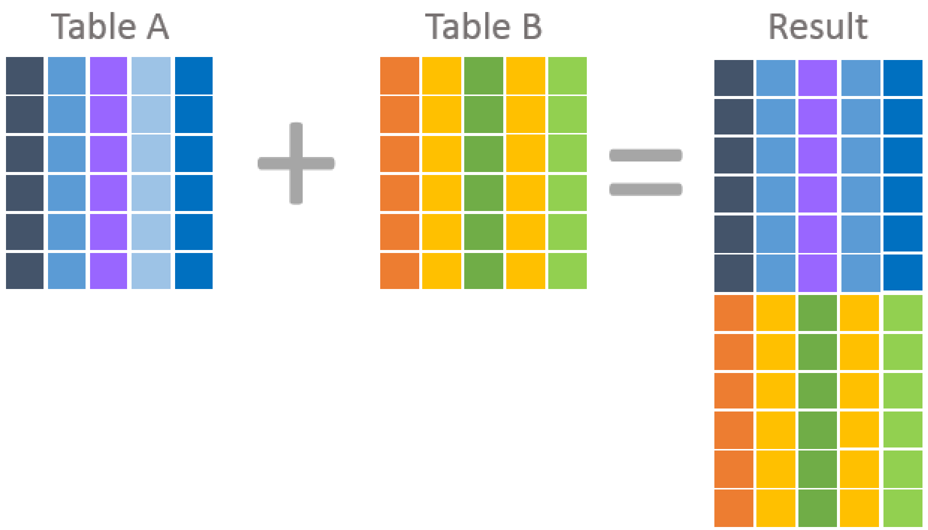
Use SQL commands `JOIN` and `UNION` for answering data questions.

Introduce the SQL structure to `JOIN` data from multiple sources.

QUERYING LARGE DATABASES IN SQL

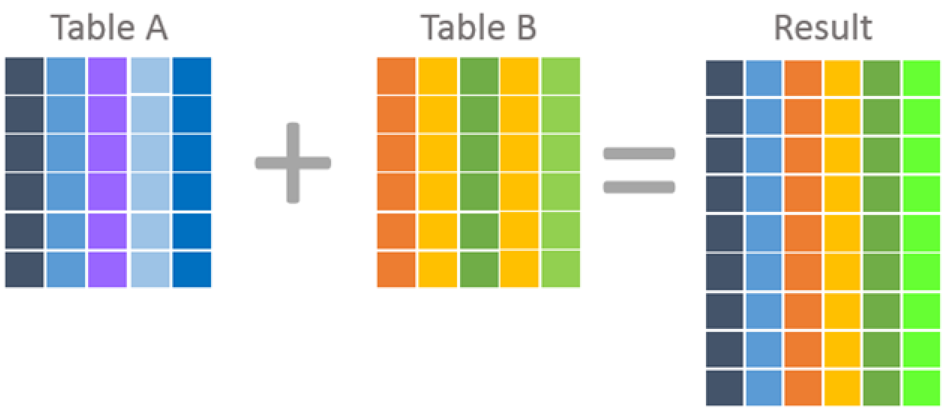
INTRODUCTION: UNIONS AND JOINS

UNIONS AND JOINS: ILLUSTRATION



UNION merges rows of *similar* data to create a new set.

JOIN combines columns from tables using common unique identifiers (keys).



Source: essentialSQL, https://goo.gl/FQXykj

UNIONS

UNIONS: MERGING 2 TABLES

- We will build three different examples of **`UNION`**`s`:
  - table **`UNION`**,
  - column **`UNION`**, or
  - combination of the two.

- Sample HR tables for illustration:

Employees1

id	first_name	last_name	current_salary
2	Gabe	Moore	50000
3	Doreen	Mandeville	60000
5	Simone	MacDonald	55000

Employees2

id	first_name	last_name	current_salary
7	Madisen	Flateman	75000
11	Ian	Paasche	120000
13	Mimi	St. Felix	70000

UNIONS: MERGING TWO TABLES

A **table** UNION includes selections from different tables:

```
SELECT * FROM Employees1 UNION SELECT * FROM Employees2
```

id	first_name	last_name	current_salary
2	Gabe	Moore	50000
3	Doreen	Mandeville	60000
5	Simone	MacDonald	55000

id	first_name	last_name	current_salary
7	Madisen	Flateman	75000
11	Ian	Paasche	120000
13	Mimi	St. Felix	70000

id	first_name	last_name	current_salary
2	Gabe	Moore	50000
3	Doreen	Mandeville	60000
5	Simone	MacDonald	55000
7	Madisen	Flateman	75000
11	Ian	Paasche	120000
13	Mimi	St. Felix	70000

QUERYING LARGE DATABASES IN SQL

GUIDED PRACTICE: UNIONS

UNIONS - MERGING 2 TABLES

- A Column **UNION** is when you append columns together:

```
SELECT id,first_name FROM Employees1
UNION
SELECT id,last_name FROM Employees1
```

id	first_name
2	Gabe
3	Doreen
5	Simone
2	Moore
3	Mandeville
5	MacDonald

UNIONS - MERGING 2 TABLES

- A combination of Column and Table UNIONS:

```
SELECT id,first_name FROM Employees1
UNION
SELECT id,last_name FROM Employees1
UNION
SELECT id,first_name FROM Employees2
UNION
SELECT id,last_name FROM Employees2
```

id	first_name
2	Gabe
3	Doreen
5	Simone
2	Moore
3	Mandeville
5	MacDonald
7	Madisen
11	Ian
13	Mimi
7	Flateman
11	Paasche
13	St. Felix

UNIONS - PRACTICE EXAMPLE

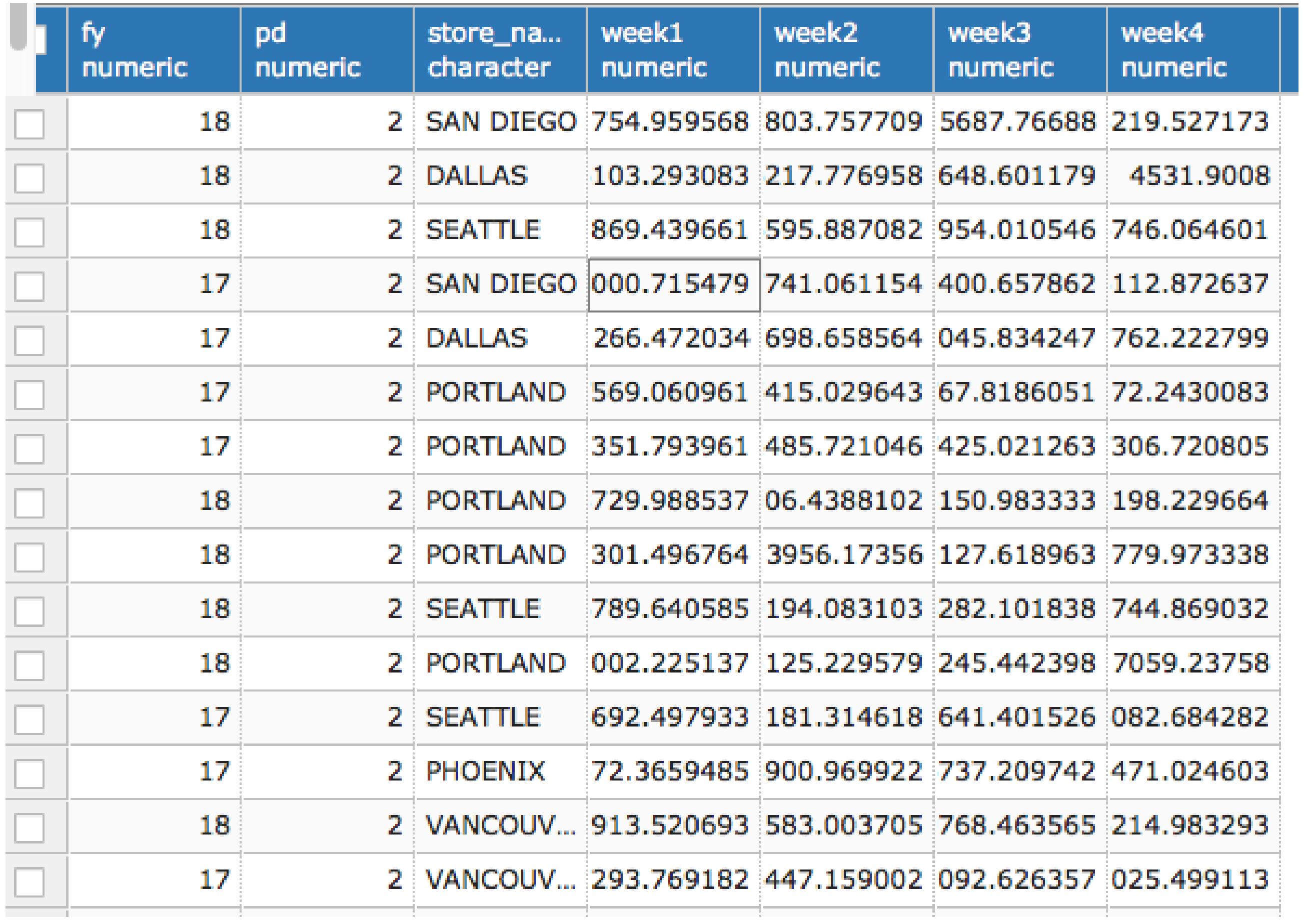
- Let’s try this with our Postgres database:

- Table Union:

```
SELECT *
FROM fy17

UNION

SELECT *
FROM fy18
```



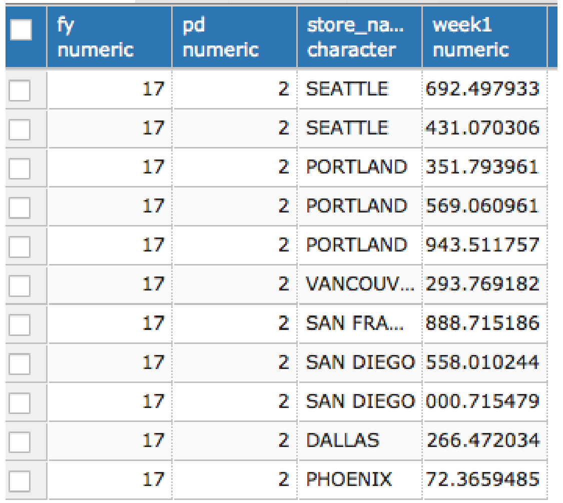
	fy numeric	pd numeric	store_na... character	week1 numeric	week2 numeric	week3 numeric	week4 numeric
□	18	2	SAN DIEGO	754.959568	803.757709	5687.76688	219.527173
□	18	2	DALLAS	103.293083	217.776958	648.601179	4531.9008
□	18	2	SEATTLE	869.439661	595.887082	954.010546	746.064601
□	17	2	SAN DIEGO	000.715479	741.061154	400.657862	112.872637
□	17	2	DALLAS	266.472034	698.658564	045.834247	762.222799
□	17	2	PORTLAND	569.060961	415.029643	67.8186051	72.2430083
□	17	2	PORTLAND	351.793961	485.721046	425.021263	306.720805
□	18	2	PORTLAND	729.988537	06.4388102	150.983333	198.229664
□	18	2	PORTLAND	301.496764	3956.17356	127.618963	779.973338
□	18	2	SEATTLE	789.640585	194.083103	282.101838	744.869032
□	18	2	PORTLAND	002.225137	125.229579	245.442398	7059.23758
□	17	2	SEATTLE	692.497933	181.314618	641.401526	082.684282
□	17	2	PHOENIX	72.3659485	900.969922	737.209742	471.024603
□	18	2	VANCOUV...	913.520693	583.003705	768.463565	214.983293
□	17	2	VANCOUV...	293.769182	447.159002	092.626357	025.499113

⇨ Add an **ORDER BY** statement to reorganize the output.

UNIONS - MERGING 2 TABLES

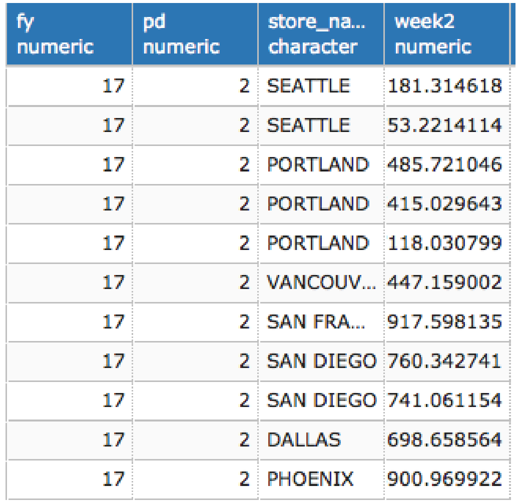
- Column Union:

```
SELECT fy, pd, store_name, week1
FROM fy17
UNION
SELECT fy, pd, store_name, week2
FROM fy17
```

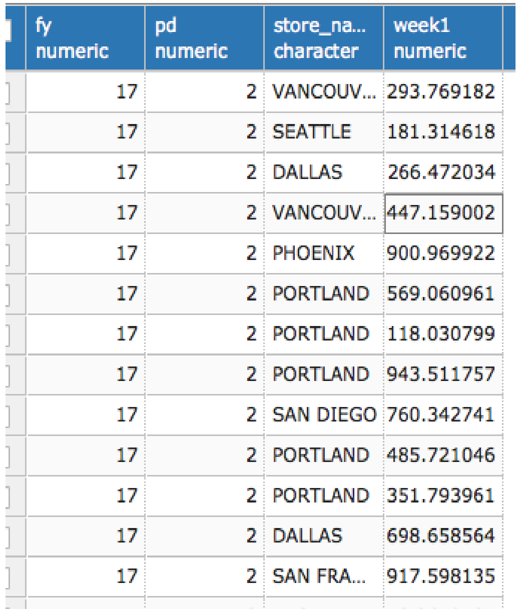


□	fy numeric	pd numeric	store_na... character	week1 numeric
□	17	2	SEATTLE	692.497933
□	17	2	SEATTLE	431.070306
□	17	2	PORTLAND	351.793961
□	17	2	PORTLAND	569.060961
□	17	2	PORTLAND	943.511757
□	17	2	VANCOUV...	293.769182
□	17	2	SAN FRA...	888.715186
□	17	2	SAN DIEGO	558.010244
□	17	2	SAN DIEGO	000.715479
□	17	2	DALLAS	266.472034
□	17	2	PHOENIX	72.3659485

UNION



fy numeric	pd numeric	store_na... character	week2 numeric
17	2	SEATTLE	181.314618
17	2	SEATTLE	53.2214114
17	2	PORTLAND	485.721046
17	2	PORTLAND	415.029643
17	2	PORTLAND	118.030799
17	2	VANCOUV...	447.159002
17	2	SAN FRA...	917.598135
17	2	SAN DIEGO	760.342741
17	2	SAN DIEGO	741.061154
17	2	DALLAS	698.658564
17	2	PHOENIX	900.969922

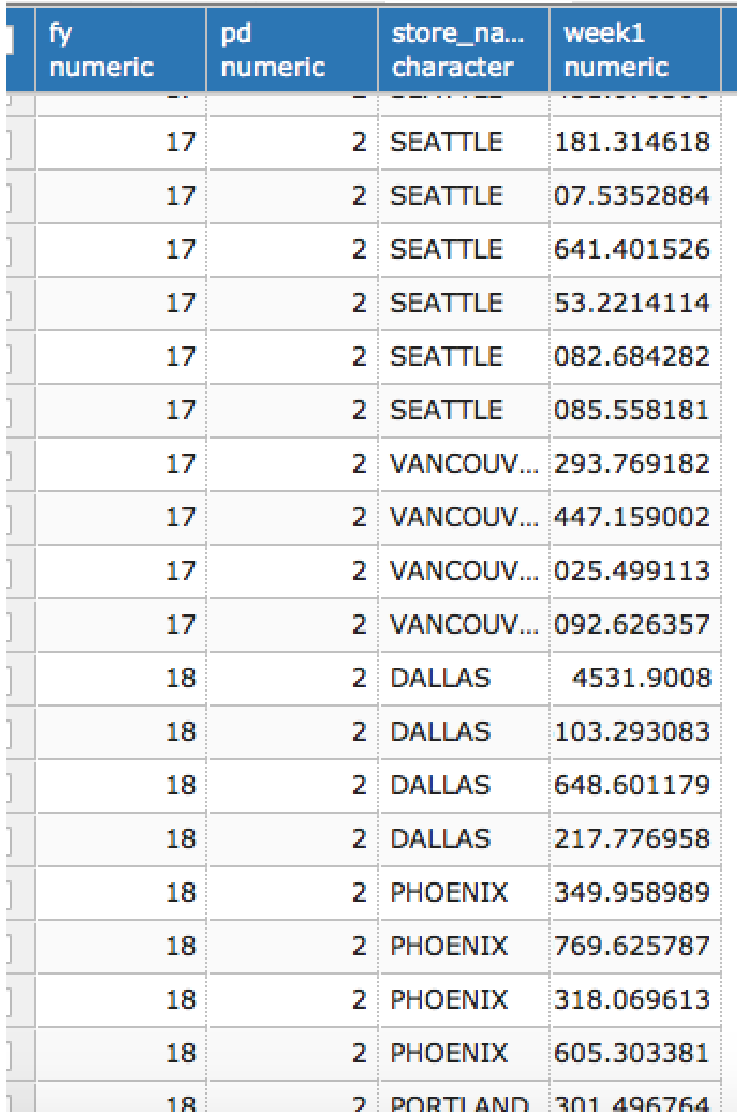


fy numeric	pd numeric	store_na... character	week1 numeric
17	2	VANCOUV...	293.769182
17	2	SEATTLE	181.314618
17	2	DALLAS	266.472034
17	2	VANCOUV...	447.159002
17	2	PHOENIX	900.969922
17	2	PORTLAND	569.060961
17	2	PORTLAND	118.030799
17	2	PORTLAND	943.511757
17	2	SAN DIEGO	760.342741
17	2	PORTLAND	485.721046
17	2	PORTLAND	351.793961
17	2	DALLAS	698.658564
17	2	SAN FRA...	917.598135

UNIONS - MERGING 2 TABLES

- A combination of Column and Table UNIONS:

```
SELECT fy, pd, store_name, week1 FROM fy17
UNION
SELECT fy, pd, store_name, week2 FROM fy17
UNION
SELECT fy, pd, store_name, week3 FROM fy17
UNION
SELECT fy, pd, store_name, week4 FROM fy17
UNION
SELECT fy, pd, store_name, week1 FROM fy18
UNION
SELECT fy, pd, store_name, week2 FROM fy18
UNION
SELECT fy, pd, store_name, week3 FROM fy18
UNION
SELECT fy, pd, store_name, week4 FROM fy18
ORDER BY 1,2,3
```



fy numeric	pd numeric	store_na... character	week1 numeric
17	2	SEATTLE	181.314618
17	2	SEATTLE	07.5352884
17	2	SEATTLE	641.401526
17	2	SEATTLE	53.2214114
17	2	SEATTLE	082.684282
17	2	SEATTLE	085.558181
17	2	VANCOUV...	293.769182
17	2	VANCOUV...	447.159002
17	2	VANCOUV...	025.499113
17	2	VANCOUV...	092.626357
18	2	DALLAS	4531.9008
18	2	DALLAS	103.293083
18	2	DALLAS	648.601179
18	2	DALLAS	217.776958
18	2	PHOENIX	349.958989
18	2	PHOENIX	769.625787
18	2	PHOENIX	318.069613
18	2	PHOENIX	605.303381
18	2	PORTLAND	301.496764

UNION RULES

- Remember these **three** rules when using `UNION`:

1. You must match the number of columns, of compatible data types.
2. You can only have one `ORDER BY` at the bottom `SELECT` statement.
3. Conditions between `UNION's` `SELECT` statements should match.

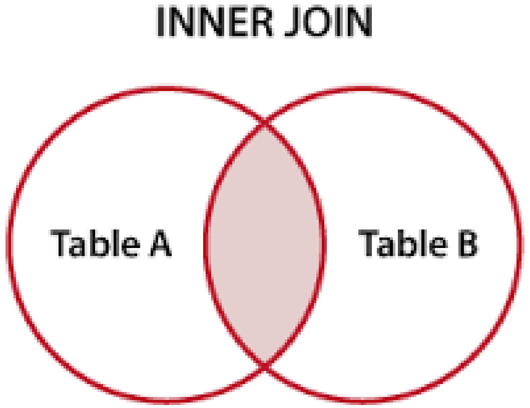
Pro Tip: While you may find yourself altering these rules in practice, keep in mind that this may damage your data integrity.

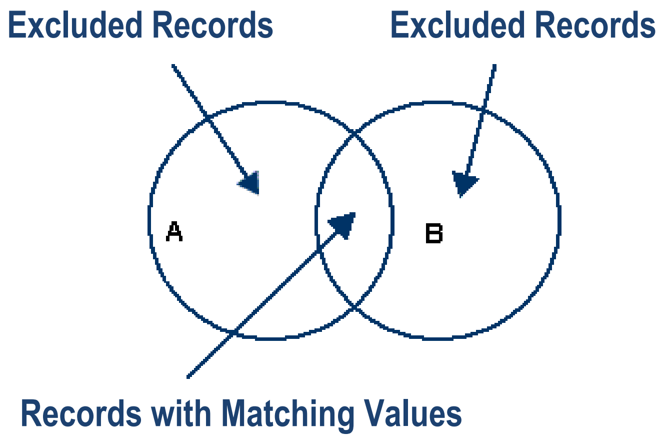
QUERYING LARGE DATABASES IN SQL

INTRODUCTION: JOINS

JOINS

- SQL **`JOINs`** connect data sources together in new combinations.
- For relational databases, tables are connected on common columns content, serving as **unique identifiers** (keys).
- Inner joins are the most common and are the default when the command **`JOIN`** is used alone.





JOIN SYNTAX

JOINS SYNTAX

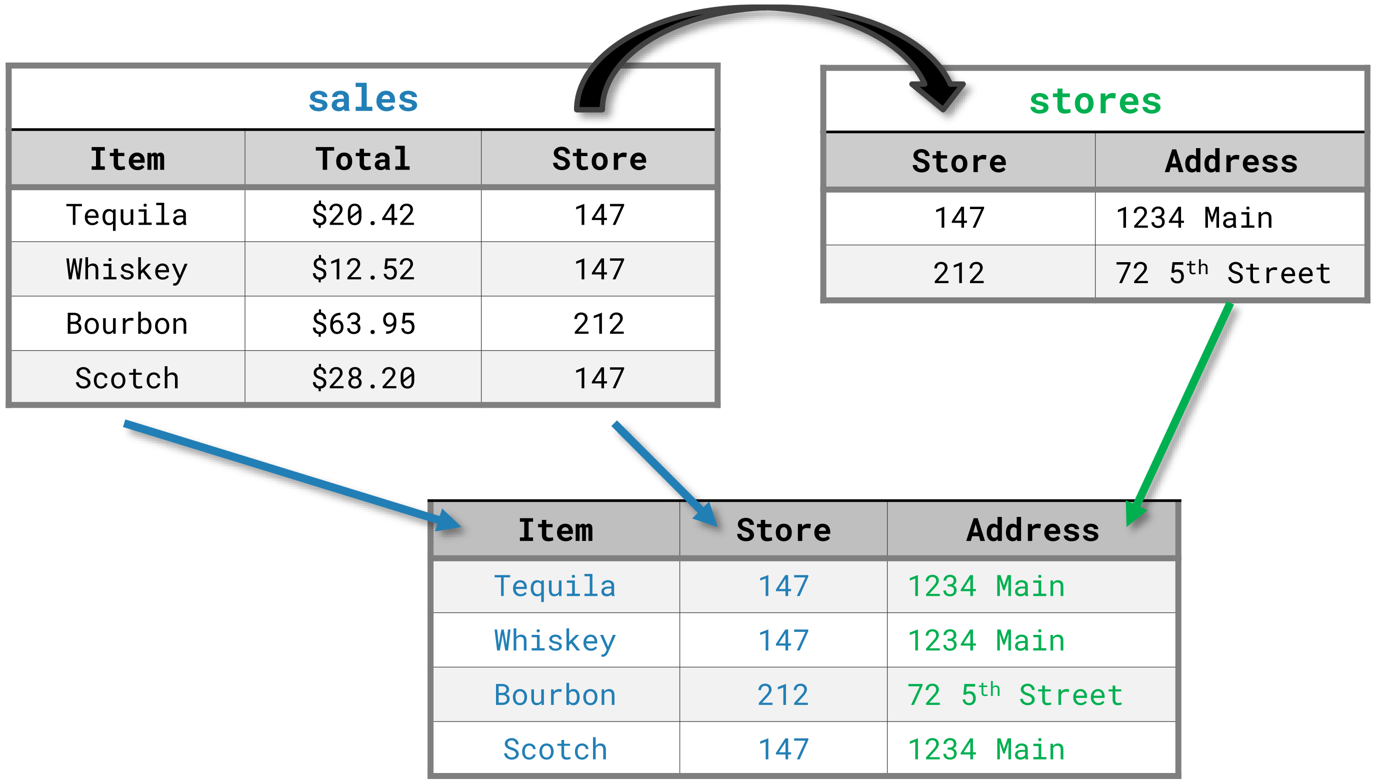
- Illustration of bringing data together.
  - **What** ⇨ columns of information
  - **Where** ⇨ identify the table to retrieve information
  - **How** ⇨ specify the connecting data (aka, “keys”)

```
SELECT
    sales.item,
    sales.store,
    stores.address

FROM
    sales

INNER JOIN stores

    ON sales.store = stores.store;
```



sales

Item	Total	Store
Tequila	$20.42	147
Whiskey	$12.52	147
Bourbon	$63.95	212
Scotch	$28.20	147

stores

Store	Address
147	1234 Main
212	72 5th Street

Item	Store	Address
Tequila	147	1234 Main
Whiskey	147	1234 Main
Bourbon	212	72 5th Street
Scotch	147	1234 Main

JOINS SYNTAX - TABLE ALIASES

- Each column name must have a **prefix that specifies its source**.
- Labeling is accomplished by writing out the source **table name** *or* with an **alias**.
- When using **aliases**, it is designated in the **`FROM`** statement.

```
SELECT
    stores.store,
    sales.item,
    sales.total

FROM
    sales

INNER JOIN stores

    ON sales.store = stores.store;
```

```
SELECT
    str.store,
    sls.item,
    sls.total

FROM
    sales sls

INNER JOIN stores str

    ON sls.store = str.store;
```

JOINS SYNTAX - HOW TO JOIN

After identifying the source tables and assigning any desired aliases,

- Select the type of **`JOIN`** `(`**`INNER JOIN`**, is default). Other common joins are **`LEFT`**, **`RIGHT`** and **`OUTER`**. The join type determines what information is brought to the new table and what is excluded.

- Specify the connection by column name on which to link tables.

    - **`ON`** `a.column_name1 = b.column_name4` → with alias for source table *otherwise,*
    - **`USING`**`(column_name)` → Only if the columns have **same** name in each table.

GUIDED PRACTICE: JOINING SALES TO PRODUCTS

JOINING SALES TO PRODUCTS

EXERCISE

- Starting with the Sales transaction table, let’s add into each row the proof value, from the Products table using an `INNER JOIN`. Let’s complete this starter code:

```
SELECT sls.item,sls.description,
       prd.proof,sls.total

FROM  the sales table and the products table,
     Assign the aliases designated above
     Add ON phrase with unique identifiers to connect tables

LIMIT 1000;
```

JOINING SALES TO PRODUCTS

EXERCISE

- Solution:

```
SELECT sls.item,
   sls.description,
   prd.proof,
   sls.total
FROM sales sls
INNER JOIN products prd
ON sls.item = prd.item_no
LIMIT 1000;
```

INDEPENDENT PRACTICE: JOINING SALES TO STORES

JOINING SALES TO STORES

EXERCISE

- Write a query to discover which distinct products were sold in Mason City, IA?
  - Bring back the product description, category and store address columns.
  - Qualify `DISTINCT ON(sales.description)`
  - Select them appropriately aliased `FROM` the sales and stores tables.
  - Connect the matching columns with the `USING` structure.

JOINING SALES TO STORES

EXERCISE

- Solution (Distinct products sold in Mason City):

```
SELECT DISTINCT ON(sales.description)
        sales.description, sales.category_name,
        stores.store_address

FROM sales
       INNER JOIN stores USING (store)

WHERE b.store_address ILIKE ‘%Mason City%’;
```

(1475 rows returned)

QUERYING LARGE DATABASES IN SQL

PREVIEW:
RIGHT & LEFT JOINS

WALK THROUGH SYNTAX

- Let’s look at some *sample* syntax for JOIN statements.

- **RIGHT OUTER JOIN**

```
SELECT b.location, b.address, b.status, a.location, a.sales
FROM table1 a
RIGHT JOIN table2 b
ON a.location = b.location;
```

- **LEFT OUTER JOIN**

```
SELECT a.location, a.sales, b.location, b.address, b.status
FROM table1 a
LEFT JOIN table2 b
ON a.location = b.location;
```

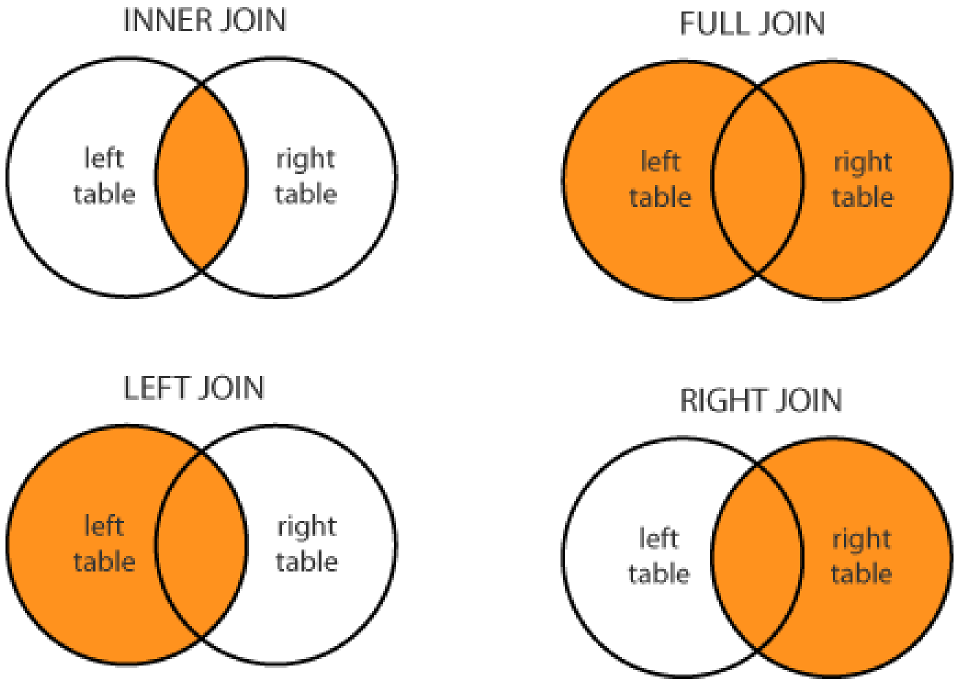
CONCLUSION

LIGHTNING ROUND REVIEW

- What is a `JOIN` and when would you use one?
- What information do you need to perform a `JOIN`?
- What's the difference between a `UNION` and a `JOIN`?
- What is an `ALIAS` used for?
- When connecting table "keys", how are `USING` and `ON` selected?

RECAP UNIONS AND JOINS

- **`UNIONS`** and **`JOINS`** `is` just the beginning of connecting tables.
- The next step with **`JOINS`** is showing priority to one of the tables and adding selected parts of the secondary table to create output.



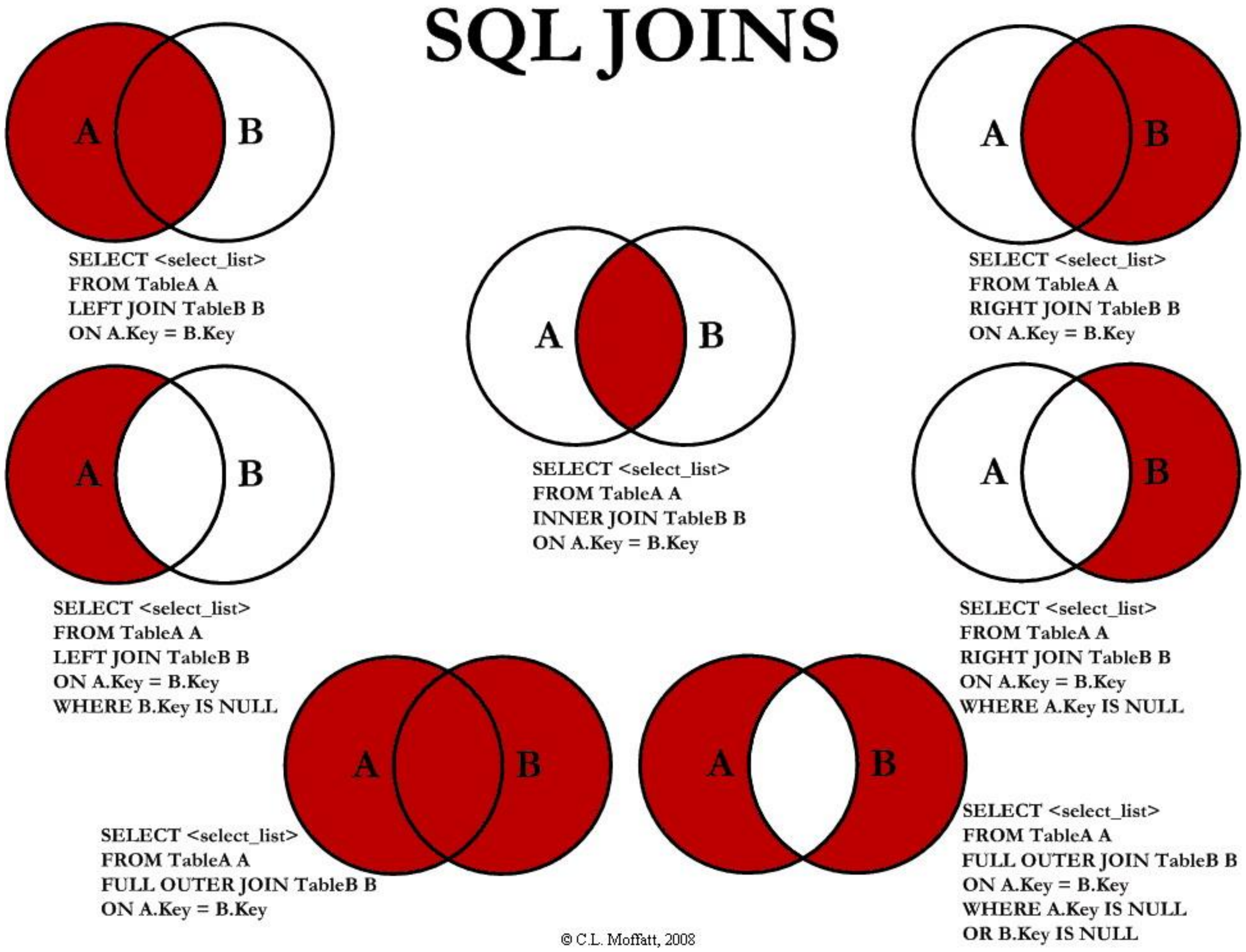
WHAT HAVE WE LEARNED SO FAR?

In this Bootcamp we have learned how to:

1. Explain SQL database structures and connect to our class databases.
2. Define SQL grammar, syntax, and punctuation.
3. Use SQL's **`SELECT`** statement with **`WHERE`** clauses.
4. Practiced with query commands including **`DISTINCT`**, **`COUNT`**, **`AND`**, **`OR`**, and **`CAST`**.
5. Apply SQL commenting using **`--`** and **`/* comment */.`**
6. Apply SQL conditional operators **`=`**, **`!=`**, **`>`**, **`<`**, **`IN`**, **`NOT IN`**, and **`BETWEEN`**.
7. Use the SQL Boolean operator **`OR`** to include only the desired data.
8. Filter data with the commands **`GROUP BY`** and **`HAVING`**.
9. Use the aggregate functions **`MIN`**, **`MAX`**, **`SUM`**, **`AVG`**, and **`COUNT`**.
10. Apply calculations to fields using the order of operations (PEMDAS).

RESOURCES

SQL JOIN BIBLE



SQL JOINS

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

© C.L. Moffatt, 2008

RESOURCES

- Microsoft reference material on unions: https://docs.microsoft.com/en-us/sql/t-sql/language-elements/set-operators-union-transact-sql
- Inner Join tutorial: http://www.sqltutorial.org/sql-inner-join/
- “What is the Difference Between a Join and a Union?” https://www.essentialsql.com/what-is-the-difference-between-a-join-and-a-union/
- “What is the difference between a primary and unique key?” https://www.essentialsql.com/primary-and-unique-key/

CREDITS

Union and Join graphical illustration from EssentialSQL.com, https://goo.gl/FQXykj.