GENERAL ASSEMBLY

SQL Bootcamp

- *Introduction to Database & Queries*
- *Building SELECT statements*
- *Filtering & Aggregating with WHERE*
- *Combining Data Tables*

GENERAL ASSEMBLY

FILTERING AND AGGREGATING IN SQL

Celia Fryar

OPENING

LIGHTNING ROUND REVIEW

- Name two ways to access a quick view of first 100 rows.
- What is the SQL command used to designate data source?
- Does SQL see upper and lower case letters as same or distinct pieces of information?
- How do you solve comparing a number to text value?
- What punctuation is used to encapsulate a text string in SQL?
- Name three ways to see how many rows a query returns.

LEARNING OBJECTIVES

- Apply commenting to code using `--` and `/* */`
- Use SQL conditional operators `=,!=,>,<, IN, NOT,IN,` and `BETWEEN`.
- Use SQL Boolean operators `OR` to include only data desired to be included.
- Introduce advanced SQL commands, `GROUP BY` and `HAVING` to filter data.
- Use aggregate functions `MIN`, `MAX`, `SUM`, `AVG`, `COUNT`.
- Apply calculations to fields using the order of operations.

INTRODUCTION: COMMENTING CODE

FILTERING AND AGGREGATING IN SQL

- Headers are a great way to keep a history of why the SQL query was built and who requested changes that have been made.
- Line-by-line usually works for calculations, and multi-line is a way to document more complicated processes and the reasoning behind them.

```
-- Basic inline commenting

/* Multiple Line comment is helpful for headers */

SELECT item_no, description /*In Line comments should
be used sparingly*/
```

EXAMPLE COMMENTING CODE

- Headers are a great practice to adopt.

```
/************************
** NAME: Name of report
** DESC: Description of report
** AUTH: Name of Author
** REQ: Name of requester
** DATE: Date report published
************************
**Change History
************************
** Version| Date     | Author |Description
**---------------------------------------
** 1.1        |10/15/16|Pat Doe| Description of change
*****************************/
```

WHERE CONDITIONS

WHERE CONDITIONS

- ✓ **`SELECT`** the columns
- ✓ **`FROM`** *points* to the table
- ✓ **`WHERE`** *filters* on rows
- ✓ `GROUP BY` *aggregates* across values of a variable
- ✓ `HAVING` *filters* groups
- ✓ `ORDER BY` *sorts or arranges* the results
- ✓ `LIMIT` *limits* result to the first n rows

WHERE CONDITIONS

- **WHERE** statement filters and focuses resulting information.
- We will be talking about ways to filter different types of data.
- When combining Logical Operators, be aware of the Order of Operation and *use parenthesis* to create groupings or priorities.

WHERE CONDITIONS

- Let's learn about a few new operators and apply them to our Iowa dataset:

`!=, <>`	Not equal to
`>, >=`	Greater than, greater than or equal to
`<, <=`	Less than, less than or equal to
`IN ( )`	Found in list of items
`NOT`	Negates a condition
`LIKE, ILIKE`	Contains item
`BETWEEN`	Within the range of
`%`	Wildcard
`_`	Wildcard, single character

WHERE CONDITIONS

- Notes on syntax for new operators:

`IN( )`

- specified as stored in table, separated by commas
-Ex: category_name `IN( ‘SPICED RUM’, ‘TEQUILA’)`

`ILIKE`

- disregards case (insensitive), may use wildcards
- Ex: category_name `ILIKE` ‘spiced%’

`BETWEEN`

-filters between two values, includes boundaries
- Ex: shelf_price `BETWEEN 5 AND 25`

GUIDED PRACTICE: WHERE CONDITIONS

WHERE CONDITIONS - GUIDED PRACTICE (Group 1)

- Follow along with these queries for each of the new predicate operators:

!=

- Which products are not from vendor 'Jim Beam Brands'
- `SELECT * FROM products WHERE vendor_name!='Jim Beam Brands';`

>, >=

- Which products are over 90 proof?
- `SELECT * FROM products WHERE CAST(proof as INT) > 90;`

<, <=

- Which products have a case cost of less than $60?
- `SELECT * FROM products WHERE case_cost < 60;`

WHERE CONDITIONS - GUIDED PRACTICE (Group 2)

- Follow along with these queries for each of the new predicate operators:

IN

- Which products are either Single Malt Scotches or Canadian Whiskies (based on category name)?

```
SELECT * FROM products WHERE category_name IN ('SINGLE MALT
SCOTCH', 'CANADIAN WHISKIES');
```

LIKE(ILIKE), NOT LIKE

- Which products have 'Whiskies' in the category name?

```
SELECT * FROM products WHERE category_name LIKE '%WHISKIES';
```

- Which products don't have 'Whiskies' in the category name?

```
SELECT * FROM products WHERE category_name NOT LIKE '%WHISKIES';
```

WHERE CONDITIONS - GUIDED PRACTICE (Group 3)

- Follow along with these queries for each of the new predicate operators:

BETWEEN (includes boundary values)

- Which products have a shelf_price between $4 and $10?

```
SELECT * FROM products WHERE shelf_price BETWEEN 4 AND 10;
(2701 rows)
```

- Which products have a bottle_price between $4 and $10?

```
SELECT * FROM products WHERE CAST(bottle_price as DEC) BETWEEN 4 AND 10;
(4011 rows)
```

Note: Bottle_price is a money data type, it cannot be compared to an integer.
CAST money types to decimal for comparison to numbers.

INDEPENDENT PRACTICE: WHERE CONDITIONS

WHERE CONDITIONS - INDEPENDENT PRACTICE

EXERCISE

DIRECTIONS

In your new project at Deloitte, your boss has asked you a few more initial questions about the new Iowa Liquor dataset.

Please write queries that answer the following questions:

1. Which products have a case cost of more than $100?
2. Which tequilas have a case cost of more than $100?
3. Which tequilas or scotch whiskies have a case cost of more than $100?
4. Which tequilas or scotch whiskies have a case cost between $100 and $120?
5. Which whiskies of any kind cost more than $100?
6. Which whiskies of any kind cost between $100 and $150?
7. Which products except tequilas cost between $100 and $120?

COMBINING LOGICAL OPERATORS

COMBINING LOGICAL OPERATORS

- In some cases, **OR** is a great way to include additional data. However, in some cases it may include more data than intended.
- We want to see any Washington state sales, or any sales in the U.S. greater than 500.

```
WHERE Country = 'US'
AND State = 'Washington'
OR cost > 500
```

- This query will not properly answer the question. Why?

COMBINING LOGICAL OPERATORS

- The presumption is you will only get U.S. and everything in Washington OR any states with cost greater than 500.
- However, because the OR is not grouped, it will instead override the Country = US and bring back ANY country with sales greater than 500.

SOLUTION:

```
WHERE Country = 'US'
AND (State = 'Washington'
OR cost > 500)
```

- PostgreSQL order of operation for logical operators, generally: NOT–AND–OR.

COMBINING OPERATORS: MATH'S ORDER OF OPERATION

How Do I Remember It All ? ... PEMDAS !

P	**P**arentheses first
E	**E**xponents (Powers and Square Roots)
MD	**M**ultiplication and **D**ivision (left-to-right)
AS	**A**ddition and **S**ubtraction (left-to-right)

INDEPENDENT PRACTICE

COMBINING LOGICAL OPERATORS

DIRECTIONS

- Your boss at Deloitte has another question for you to research:

"From the Iowa Liquor Database I only want information about vendor 305. Can you get me the bottle price and proof? Price should be less than 5 OR the proof is greater than 100, either is fine."

```
SELECT vendor, bottle_price, proof
FROM products
WHERE vendor = 305
  AND (cast(bottle_price AS decimal) <5
  OR cast(proof AS integer)>100);
```

AGGREGATIONS INTRODUCTION

AGGREGATIONS

- Let's look at common SQL aggregate function commands:
    - **`MIN`**, **`MAX`**, **`SUM`**, **`COUNT`**, **`AVG`**.
- Companion tools with aggregations are **`GROUP BY`** and **`HAVING`**:
    - **`GROUP BY`** indicates the dimensions by which you want to group your data (e.g., a category to sort into subgroups).
    - **`HAVING`** is how you can filter measures you have aggregated (e.g., where a **`SUM`** is further evaluated against a criteria).

AGGREGATIONS

Considerations when working with *Aggregated Values*:

- What data type is involved? You may need to **`CAST`** before an aggregation can be done.

  Ex: **`AVG(CAST(bottle_price AS DEC))`**

- Does the aggregated value create a binning or higher level of classification? If so, **`GROUP BY`** is required.
  - Aggregating functions are typically performed on **numeric measures** like sales, prices, miles, heights, etc.
  - **`GROUP BY`** is on the **dimensions or categories** that are being measured, like stores, vendor, or territory.

AGGREGATIONS

- Let's look at an example using a table called 'all_athletes' with detailed information about this college's athletes, including height and playing field. The request is to find the average tallest team, with a minimum average of 6 feet tall, who compete outside of a gymnasium.
- The code would look something like this:

```
SELECT sport_team, AVG(height)
FROM all_athletes
WHERE playing_field !='GYM'
GROUP BY sport_team
HAVING AVG(height) >6
ORDER BY sport_team DESC;
```

← Consider the impact of this filter

GUIDED PRACTICE: AGGREGATIONS

AGGREGATIONS: GUIDED PRACTICE

- Let's practice building these together:

```
SELECT MAX(total) FROM sales;

SELECT AVG(state_btl_cost) FROM sales;
```

- Is `CAST` needed for successful execution? Why or why not?
- Use `ROUND` to control the number of digits beyond the decimal.
- Use `ROUND` to limit the average state bottle cost to two decimals.

AGGREGATIONS: GUIDED PRACTICE

- Let's build one together that groups the results:

```
SELECT vendor,vendor_name,AVG(bottle_price)
FROM products
GROUP BY vendor, vendor_name
ORDER BY 3 DESC;
```

- Is `CAST` needed for successful execution? Why or why not.
- Use `ROUND` to control the number of digits beyond the decimal.
- Use `ROUND` to limit the average bottle cost to two decimals.

INDEPENDENT PRACTICE: AGGREGATIONS

AGGREGATIONS: INDEPENDENT PRACTICE

EXERCISE

DIRECTIONS

Using our aggregation tools find the largest and smallest bottle price offered by each vendor. Further refine the results by placing a minimum bottle price of $10 and **limiting the output** to the top twenty of the most expensive bottle prices.

Starter Code:

```
SELECT vendor_name, (aggregate of large bottle_price),
        (aggregate of smallest bottle_price)
FROM products
GROUP BY vendor.
```

Further refine results to those which *have* a minimum bottle_price of $10.

Sort the results by maximum bottle_price column.

Limit the report data to the top twenty.

Keep in mind that the data type of bottle_price is *money*.

AGGREGATIONS: INDEPENDENT PRACTICE SOLUTION

SOLUTION

```
SELECT vendor_name,
    ROUND(MAX(CAST(bottle_price AS DEC)),2) AS max_cost,
    ROUND(MIN(CAST(bottle_price AS DEC)),2) AS min_cost
FROM products
GROUP BY vendor_name
HAVING ROUND(MIN(CAST(bottle_price AS DEC)),2) > 10
ORDER BY 2 DESC
LIMIT 20;
```

CONCLUSION

RECAP

- Applying commenting to code using `--` and `/* */`
- Used SQL conditional operators `=`,`!=`,`>`,`<`, `IN`, `NOT`,`IN`, and `BETWEEN`.
- Used SQL Boolean operators `OR` to include only data desired to be included.
- Introduced SQL commands, `GROUP BY` and `HAVING` to filter data.
- Used aggregate functions `MIN`, `MAX`, `SUM`, `AVG`, `COUNT`.
- Applied calculations to fields using the order of operations.

FILTERING AND AGGREGATING IN SQL

Q&A

RESOURCES

RESOURCES

- SQL Server Logical Operators: https://goo.gl/2Q9gmH
- "Query Results Using Boolean Logic", essentialSQL: https://www.essentialsql.com/get-ready-to-learn-sql-server-query-results-using-boolean-logic/
- The SQL `HAVING` clause, by Mode Analytics, https://goo.gl/Je3M85
- "Difference between `WHERE`, `GROUP BY` and `HAVING` clauses" article by Manoj Pandey, https://goo.gl/cNCtBa