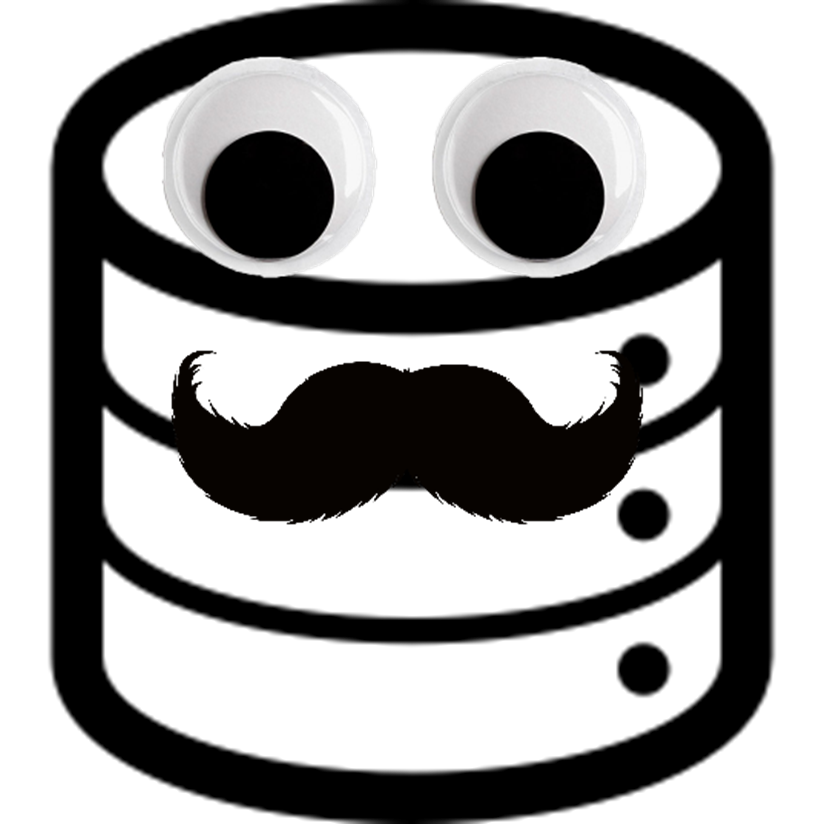
*WELCOME TO CLASS*



*Intermediate SQL Bootcamp*

Instructor: [Michael Valeri](https://www.linkedin.com/in/mvaleri/) / mvaleri12@gmail.com

Student: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **How to SELECT data out of Database – The Basic Syntax of SQL**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **KEYWORD** | **WRITTEN SEQUENCE** | **EXECUTION**  **SEQUENCE** | **WHAT IT DOES (Select A-G)** | **OTHER**  **NOTES** |
| SELECT | **1** | **3** |  |  |
| FROM | **2** | **1** |  |  |
| WHERE | **3** | **2** |  |  |
| GROUP BY | **4** | **4** |  |  |
| HAVING | **5** | **5** |  |  |
| ORDER BY | **6** | **6** |  |  |
| LIMIT | **7** | **7** |  |  |

WHAT IT DOES – Options (A-G)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *A* | *B* | *C* | *D* | *E* | *F* | *G* |
| Orders results by ASC (A-Z) by DESC (Z-A) order | Limits rows returned | Shows columns we want to see | Filters on aggregate functions. | Table(s) we want to pull from | Dimensions (columns) we want aggregates GROUPED BY | Filters on columns/dimensions of the table. |

SQL SYNTAX EXAMPLE

SELECT *Count*(DISTINCT artist) AS total\_artists

,genre

FROM ALBUMS

WHERE release\_date BETWEEN '2008-01-01' AND '2012-12-31'

GROUP BY genre

HAVING count(DISTINCT artist) > 1

ORDER BY total\_artists DESC

LIMIT 3

1. **REVIEW – Applying the Basic Syntax of SQL**

*“The VP of Marketing at Lime wants to dig into customers who signed up for the customer marketing list at a recent musical festival. She’s particularly interested in examining customers who don’t own either a bike or a car.*

*Write a query that pulls the total count of customers and their average age, grouped by ethnicity. Order the results from highest count of customers to lowest.*

*Hint\* 🡪 Use CUSTOMER\_MARKETING\_LIST table*

***STEPS: Draw the Final Table, Build the Shell, and Architect the Final Query***

**Step 1: Draw the Final Table Output**

*Result Output Column Headers (SELECT)*

|  |  |  |
| --- | --- | --- |
| **(1)** | **(2)** | **(3)** |

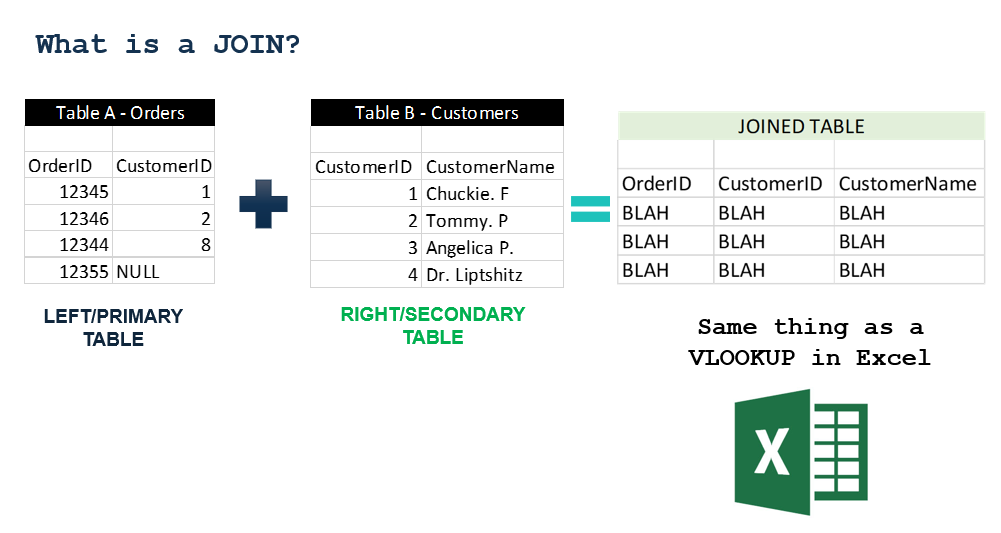


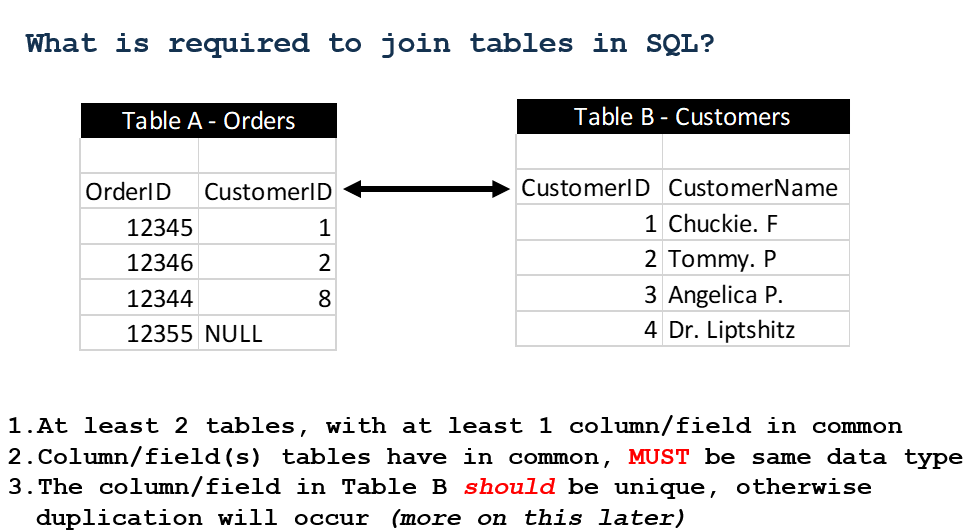
*Filters Running in the Background (WHERE)*

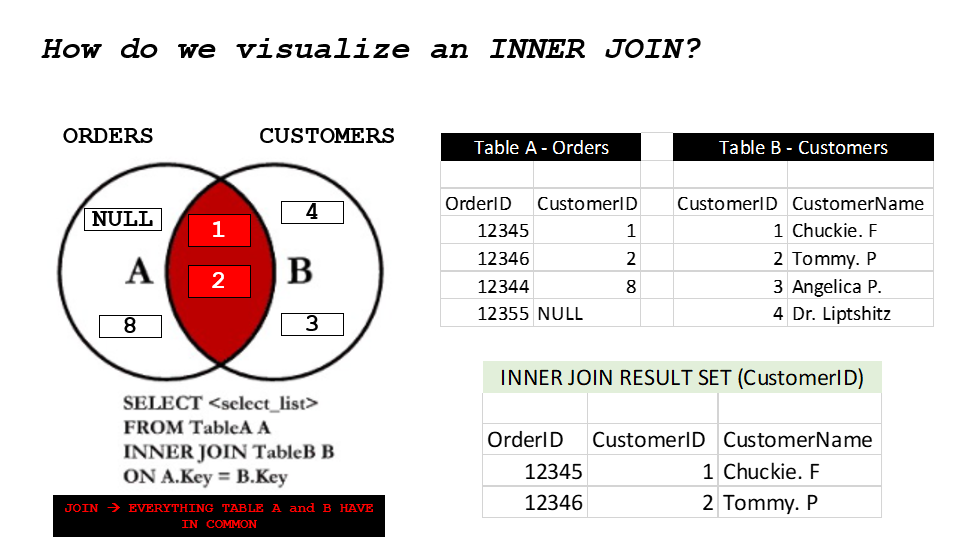
|  |  |
| --- | --- |
| **(1)** | **(2)** |

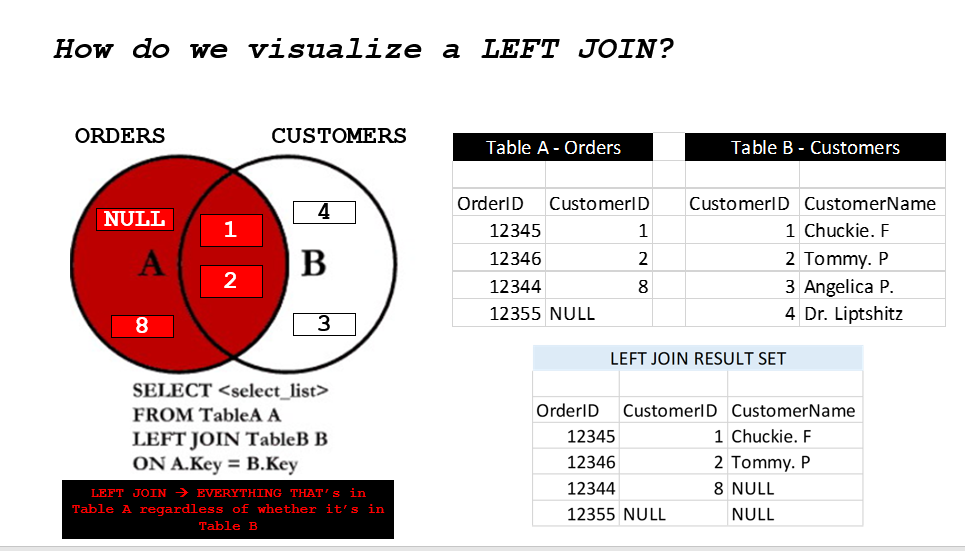
1. **Merging Two or More Tables in SQL – JOINS**

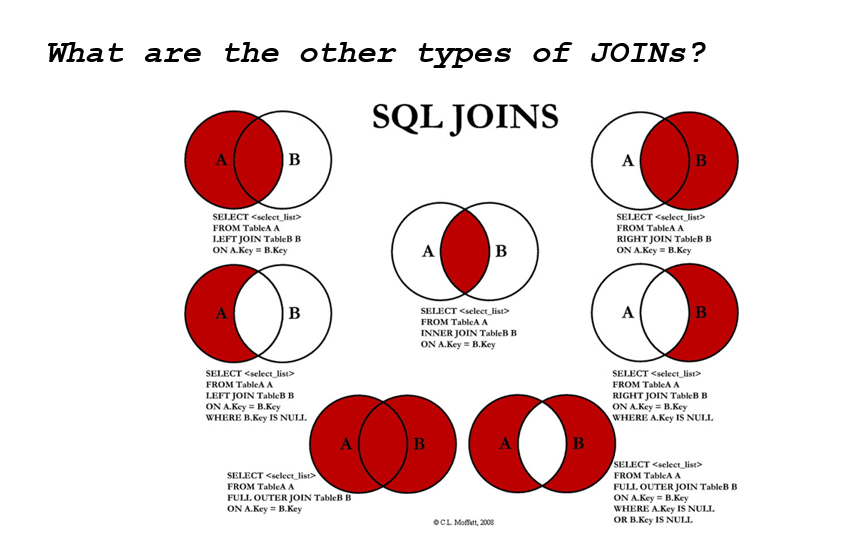


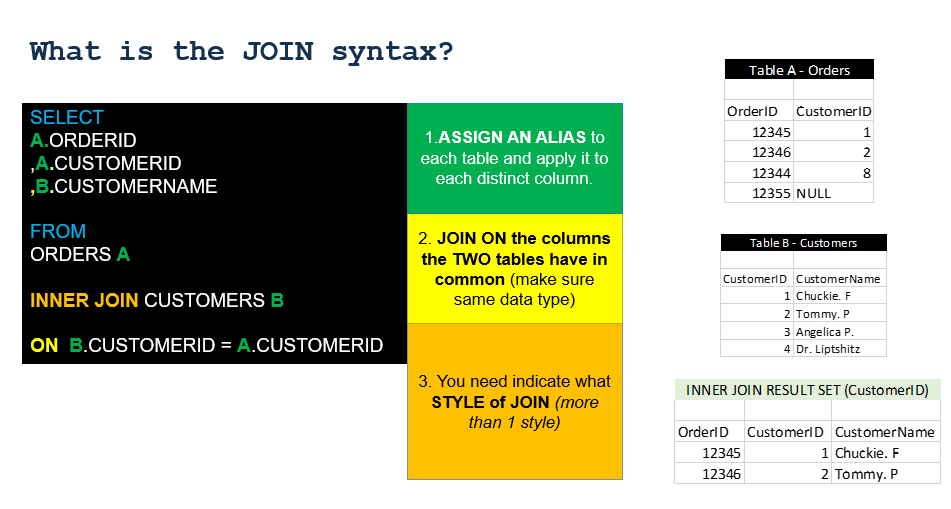










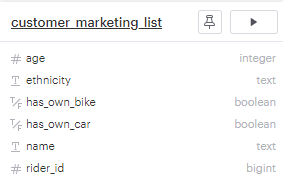
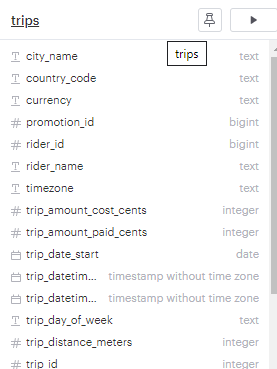


1. **JOIN WORKSHOP – Writing JOINs**

Instructor Guided Question

“I want to see total trips grouped by bike ownership”

Step 1: What table can you find trips? What table can you find bike ownership?



Step 2: What column(s) do these two tables have in common? Draw a line connecting them…

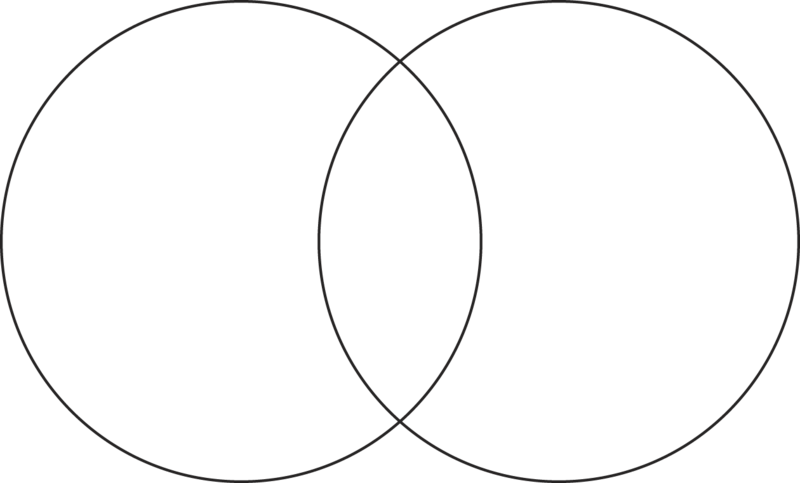
Step 3: Assign your primary table (A) and secondary table (B).

|  |  |
| --- | --- |
| **TABLE A:** | **TABLE B:** |
| Primary Table (most important) | Secondary Table (nice to have) |

Step 4: Label Table A and Table B in both Venn diagrams. Shade the area for an INNER JOIN vs. a LEFT JOIN.

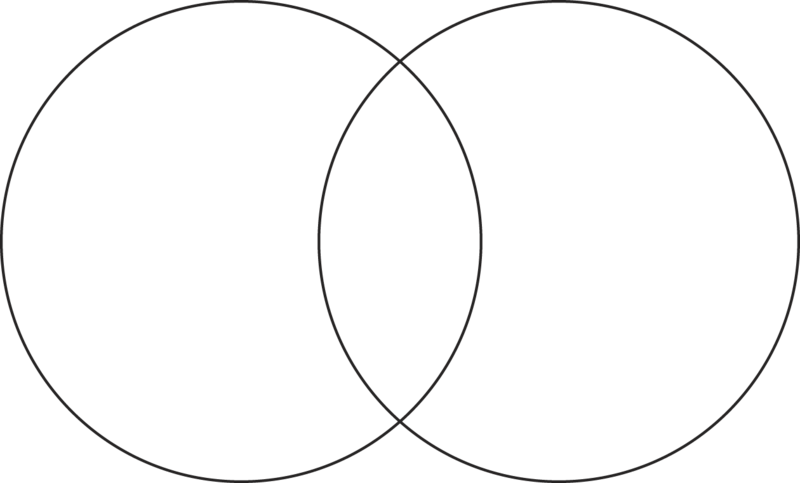
INNER JOIN

|  |  |
| --- | --- |
| **TABLE A:** | **TABLE B:** |



LEFT JOIN

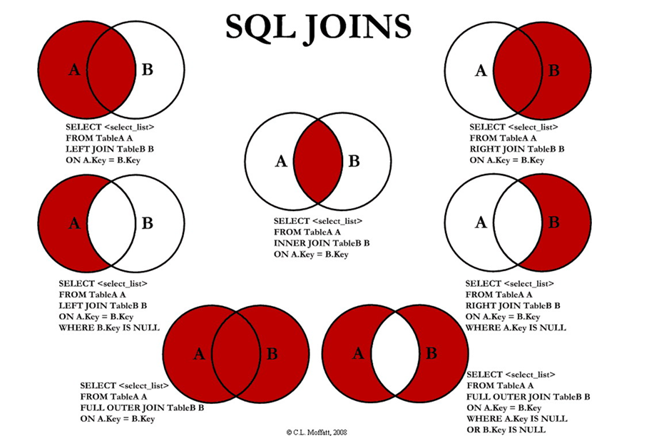
|  |  |
| --- | --- |
| **TABLE A:** | **TABLE B:** |



Step 5: Write the query that JOINs these two tables…Record the rider counts…

|  |  |
| --- | --- |
| **INNER JOIN TRIP COUNT:** | **LEFT JOIN TRIP COUNT:** |

**Match the JOINs to their business use cases**



|  |  |
| --- | --- |
| **TABLE A: TRIPS** | **TABLE B: CUSTOMER\_MARKETING\_LIST** |
| *Customer has taken a ride* | *Customer has signed up for marketing list* |

*Match the Statement to the SQL Logic*

Only customers who signed up the marketing list and have taken a ride. \_\_\_\_\_\_\_\_\_

All Customers \_\_\_\_\_\_\_\_\_\_

Only customers who haven’t completed both actions \_\_\_\_\_\_\_\_\_\_

Only customers who have signed up for the marketing list and not taken a ride \_\_\_\_\_\_\_\_\_\_\_\_

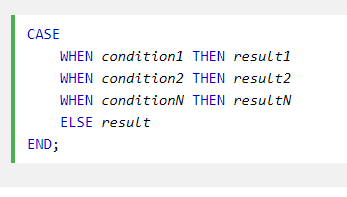
Only customers who have taken a ride and not signed up for the marketing list \_\_\_\_\_\_\_\_\_\_\_\_

All Customers who have taken a ride \_\_\_\_\_\_\_\_\_\_\_\_\_

All Customers who signed up for the marketing list \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

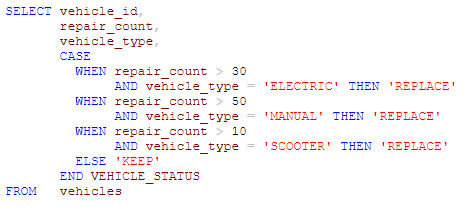
**BONUS: What join would be useful for figuring out which customers to promote rides too?**

1. **Writing Conditional Statements – CASE Statements**

The CASE statement goes through conditions and returns a value when the first condition is met (like an IF-THEN-ELSE statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

**CASE STATEMENT EXAMPLE – Identify Vehicles to Replace**



|  |  |  |  |
| --- | --- | --- | --- |
| VEHICLE\_ID | REPAIR\_COUNT | VEHICLE\_TYPE | VEHICLE\_STATUS |
| 12345 | 20 | ELECTRIC | KEEP |
| 12346 | 35 | ELECTRIC | REPLACE |
| 12347 | 101 | MANUAL | REPLACE |
| 12348 | 9 | SCOOTER | KEEP |

What syntax rules do CASE Statements follow? 🡪 Similar to the WH

1. **CASE Statement Workshop – OK, MILLENNIAL…**

*The VP of Marketing at Lime wants to identify the total number of customers in the CUSTOMER\_MARKETING\_LIST by the following buckets.*

GENERATION

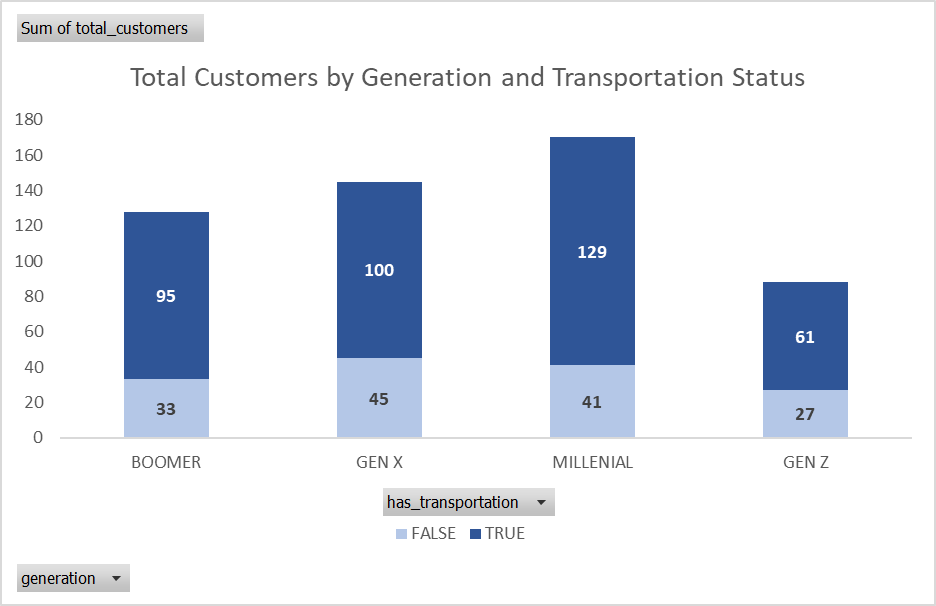
* *Over 75: Silent*
* *55-75 years old: Boomer*
* *40-54 years old: Gen X*
* *25-39 years old: Millennial*
* *Under 25: Gen Z*

HAS\_TRANSPORTATION

* TRUE (OWNS BIKE OR CAR)
* FALSE (DOESN’T OWN BIKE and Don’t OWN A CAR)

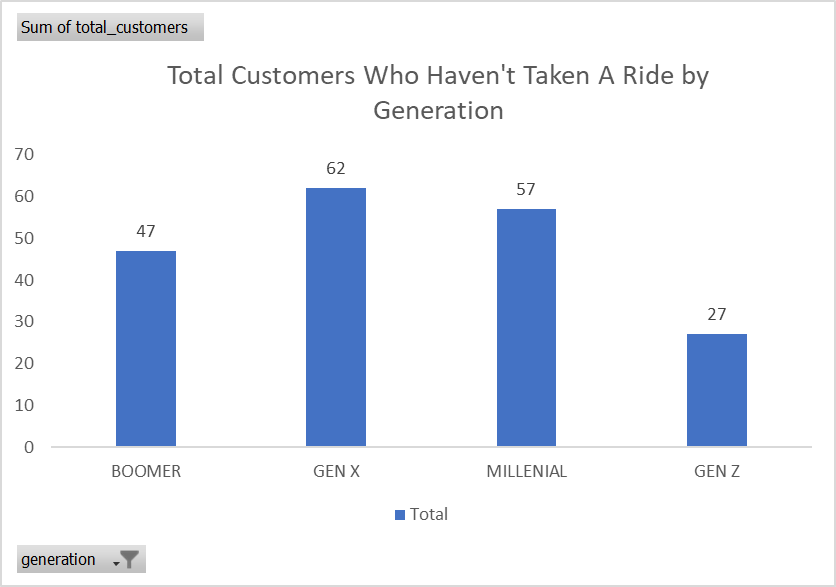
***EXERCISE 1***

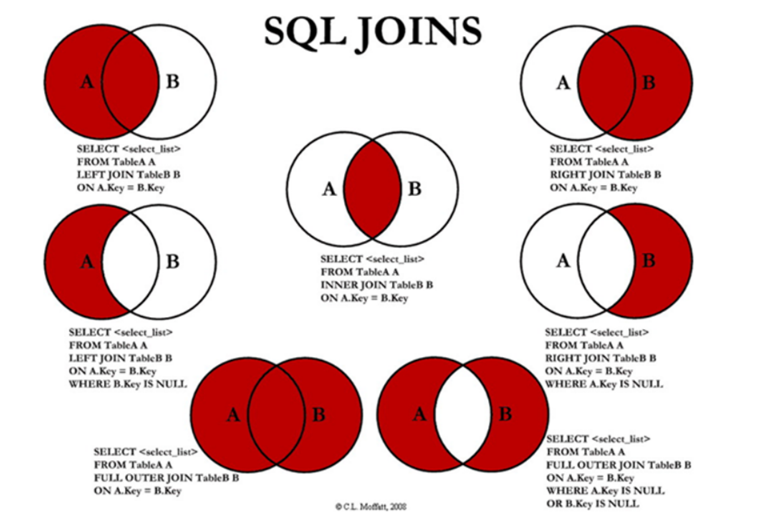
*Write a SQL query that replicates the numbers in the below chart. Hint, use BETWEEN, AND, and OR to construct your CASE statements…*



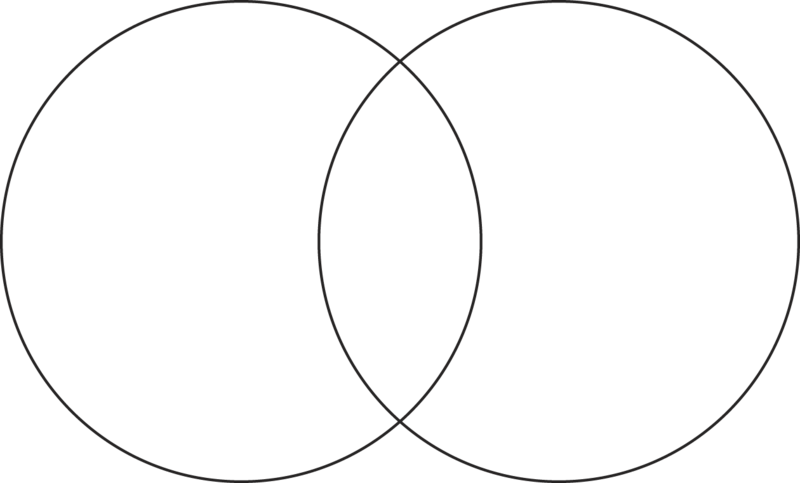
***EXERCISE 2***

*Write a SQL query that replicates the numbers in the below chart. Hint, you’ll need to JOIN to the TRIPS table. Draw a VENN DIAGRAM TO HELP…*

**



|  |  |
| --- | --- |
| **TABLE A** | **TABLE B** |



1. **Ranking Data in SQL – Introduction to Window Functions**

*Table: PEOPLE\_IN\_LIMO*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Networth\_MM | Gender | Location | Networth Rank | Networth Rank by Location |
| Jay-Z | 1,000 | M | Backseat |  |  |
| Beyonce | 400 | F | Backseat |  |  |
| Oprah | 2,800 | F | Front |  |  |
| Driver | 1 | M | Front |  |  |

**Opening Questions**

1. What is a partition? 🡪
2. How many ways can you rank the data in the above table based on Networth\_MM (in millions $)?

|  |  |  |  |
| --- | --- | --- | --- |
|  | Ranking Definition | Granularity Change? | Partitions Added? |
| 1 | Rank People by Net Worth | FALSE | FALSE |
| 2 | Rank People by Net Worth within a Location |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 | Rank Locations by Net Worth | TRUE | FALSE |
| 7 |  |  |  |
| 8 |  |  |  |

1. Organize the above table into the following buckets…

|  |  |  |
| --- | --- | --- |
|  | Keep Granularity | Change Granularity |
| No Partitions | 1  RANK() OVER (ORDER BY Networth\_MM DESC) | 6  RANK () OVER |
| Has Partitions |  |  |

**VII. Ranking Data Workshop…The “Best” Mechanic**

*The GM of Operations at Lime wants the following 4 tables of data created*

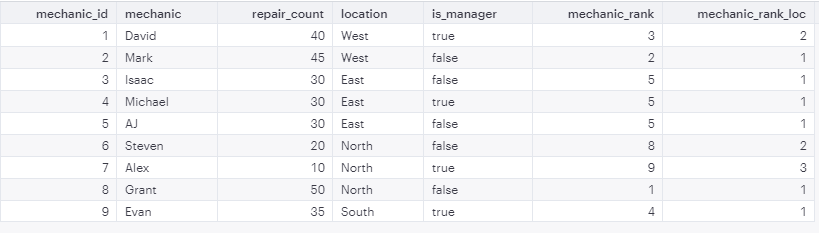
1. *Mechanics Ranked by Repair Count*
2. *Locations Ranked by Repair Count*
3. *Mechanics Ranked Within Location by Repair Count*
4. *Managers/Non Managers Ranked Within Location by Repair Count*

*Use the following window functions (ranking) to create each of the 4 tables. Use the MECHANICS table…*

* *RANK()*
* *DENSE\_RANK()*
* *ROW\_NUMBER()*

*Dissection…*

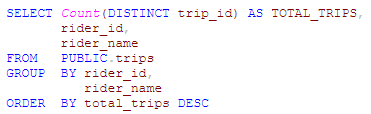
*A window function performs a calculation across a set of table rows that are somehow related to the current row. It is used for ranking and grand totals. Example function: RANK()*





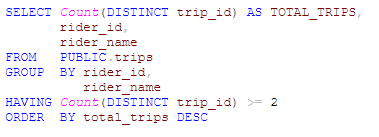
**IX. Querying Your Own Query – Having, Subqueries, and CTEs in SQL**

*“Show me a list of riders that have taken at least 2 trips”*



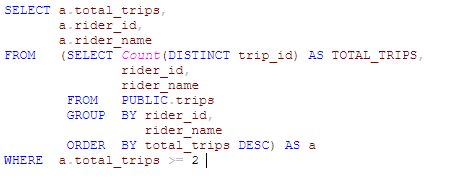
How do we remove the riders that have less than 2 trips from this query?

*Approach 1: Using HAVING*



HAVING allows us to filter on: (**Circle one) aggregates/dimensions**

WHERE allows us to filter on: (**Circle one) aggregates/dimensions**

*Approach 2: Using a Subquery*

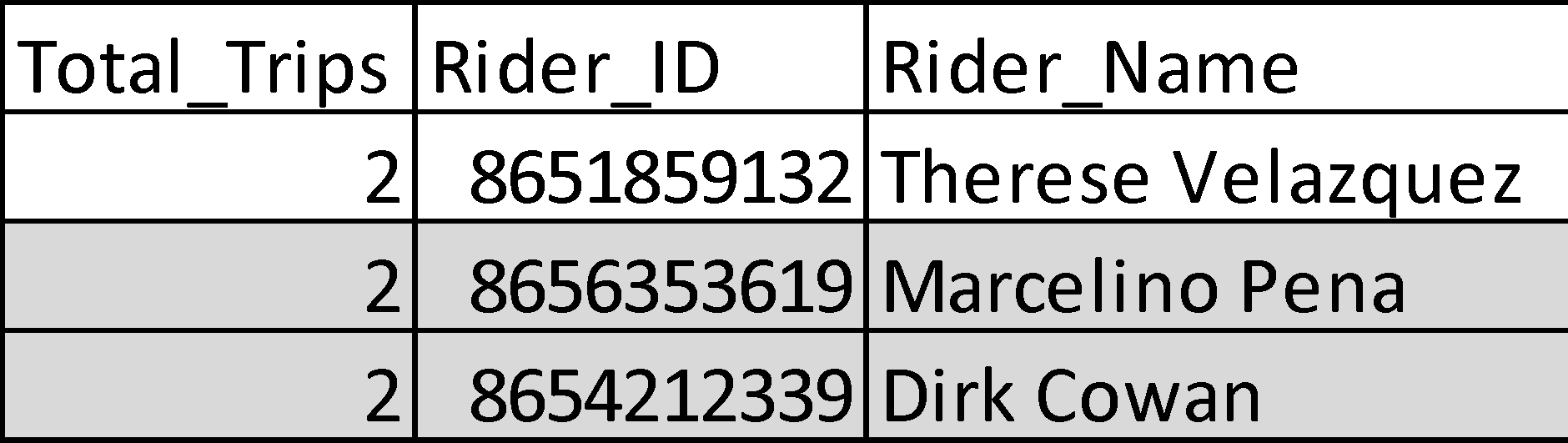




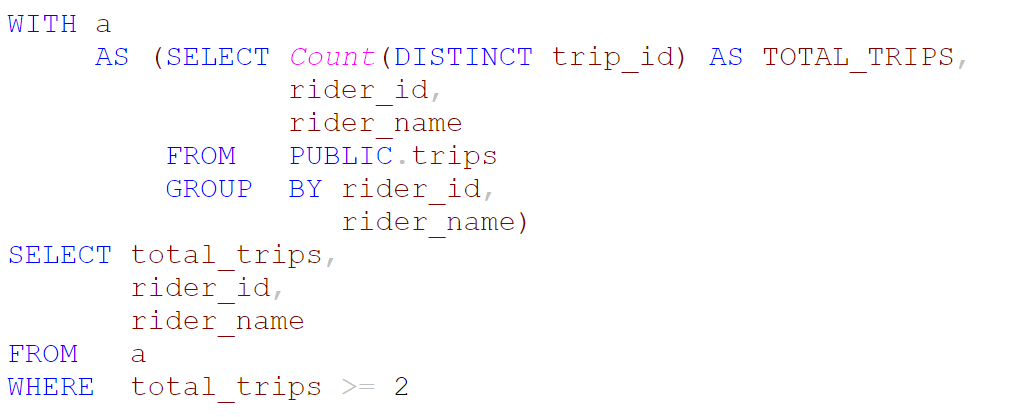








*Approach 3: Using a CTE (Common Table Expression)*

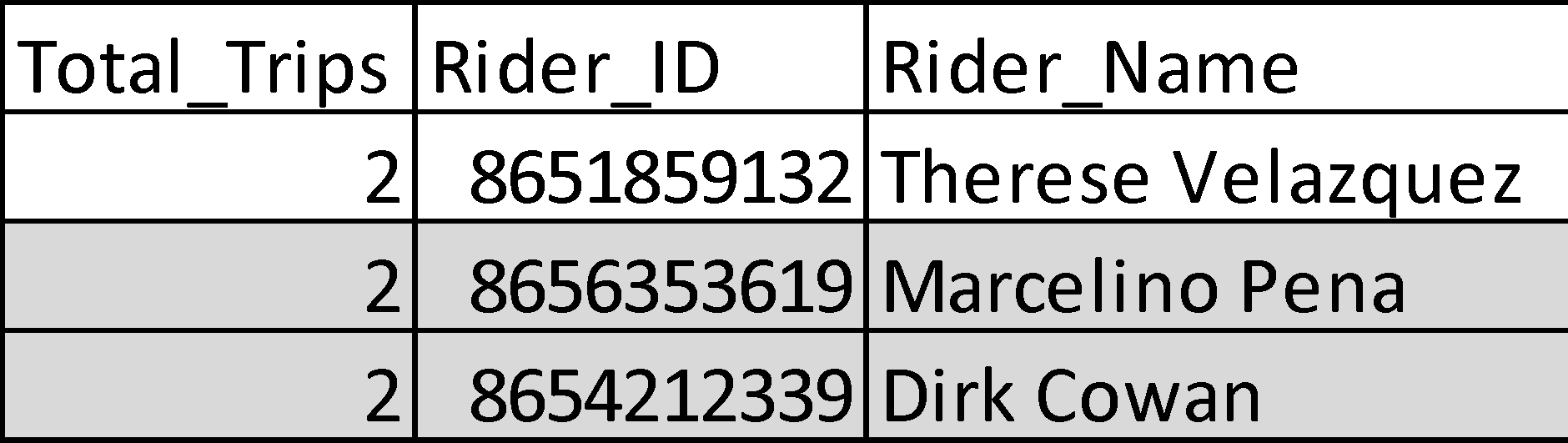




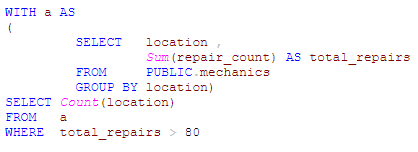


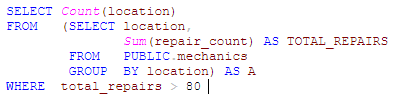






*In the following examples identify whether the query is a CTE or a Subquery, and label the components of each query. Describe what each step in each query is doing.*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



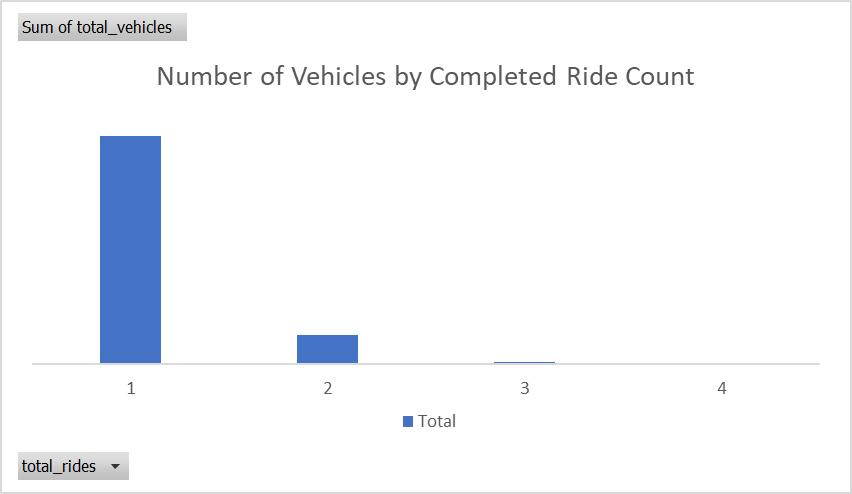
**X. Querying Your Own Query – Writing Subqueries and CTEs**

*The VP of Operations at Lime wants to understand how many vehicles have only:*

* *1 completed ride 🡪 # of vehicles*
* *2 completed rides 🡪 # of vehicles*
* *3 completed rides 🡪 # of vehicles*
* *4 completed rides 🡪 # of vehicles*

*Hints*

* *Your query only requires these columns from the trips table;*
  + *vehicle\_id*
  + *trip\_id*
  + *trip\_status*
* *You’ll need to use a subquery or a CTE…*
* *Your final result output will have only two columns*
  + *Ride Count*
  + *Number of Vehicles*



**XI. Putting it all together – Architect the Final Query**

*Write a query that counts the total number of completed scooter rides grouped by trip month and trip description\*.*

*AGG: Total Number of Scoot Rides*

*DIM\_1: Trip Description (Based on Trip Rating 5= Excellent*

*Dim\_2: Trip Month*

*Window function: Rank Trip Desc based on Total Ride within a Month*

*Remove people who went to the music festival (Exception Join)*

*Assign a rank to each trip description within a trip month based on total rides. Exclude riders who are in the customer\_marketing\_list table.*

*Hint: You’ll need to use a JOIN, CASE statement, rank function and a CTE/subquery to complete. Draw a picture of a VENN Diagram to help…*

*Use the following columns to construct your query*

***trips***

* *trip\_id*
* *trip\_date\_start*
* *trip\_rating*
* *rider\_id*
* *trip\_status*
* *vehicle\_type*

***customer\_marketing\_list***

* *rider\_id*

*Use the following functions*

* *rank() 🡪 Ranks data within a window*
* *to\_char() 🡪 Converts one date type to another*
* *count(distinct) 🡪 Counts distinct number of records*
* *CASE WHEN X THEN Y ELSE 🡪 Conditional statement for creating buckets*

# SQL Cheat Sheet Part 1 - Common Functions And Operators

*FUNCTIONS are used to* ***manipulate columns of data****. They can be placed in the SELECT and WHERE clauses. Remember, FUNCTIONS are columns of data...*

|  |  |  |
| --- | --- | --- |
| **FUNCTION** | **WHAT IT DOES** | **DOCUMENTATION** |
| DISTINCT COLUMN\_A | Will remove duplicate entries from one or more columns. Useful when working with columns that are not the primary key. | <https://www.postgresqltutorial.com/postgresql-select-distinct/> |
| CAST(COLUMN\_A AS TEXT)  CAST(COLUMN\_A AS INT)  CAST(COLUMN\_A AS NUMERIC) | Converts one data type to another. A common application when adding two columns of data together with different data types. | <https://www.postgresqltutorial.com/postgresql-cast/> |
| TO\_CHAR(COLUMN\_DATE,'YYYY') AS YEAR  TO\_CHAR(COLUMN\_DATE,'MM') AS MONTH  TO\_CHAR(COLUMN\_DATE,'MON') AS MONTH  TO\_CHAR(COLUMN\_DATE,'WW') AS WEEK | Useful for truncating date columns to specifically formatted strings, like year, month, day, week, or year-month. Exhaustive list of all date formatting options can be found in the documentation. | <https://www.postgresql.org/docs/9.6/functions-formatting.html> |
| String Functions | Useful for examining and manipulating string values. Examples include LEFT, RIGHT, TRIM, SUBSTRING, UPPER, LOWER | <https://www.postgresql.org/docs/9.1/functions-string.html> |

*OPERATORS are used* ***when filtering data*** *in the WHERE and HAVING clause or creating CASE statements (intermediate SQL concept). Remember, the column data types dictate the syntax used…*

|  |  |
| --- | --- |
| **COLUMN DATA TYPE** | **SYNTAX RULE** |
| INTEGER / NUMERIC / DECIMAL / BIGINT / SMALLINT | *NO SINGLE QUOTES / WHERE RATING > 4* |
| BOOLEAN | *NO SINGLE QUOTES / WHERE BURNED = TRUE* |
| TEXT | *SINGLE QUOTES / WHERE GENRE = 'RAP'* |
| DATE / TIMESTAMP | *SINGLE QUOTES / WHERE RELEASE\_DATE = '2009-03-01'* |

|  |  |  |
| --- | --- | --- |
| **OPERATOR** | **WHAT IT DOES** | **DOCUMENTATION** |
| Basic Comparison Operators   * = (Equals) * <> (Doesn’t equal) * > (Greater than) * < (Less than) * >= (Greater than or equal to) * <= (Less than or equal to)   -- GENRE = 'Rock'  -- GENRE <> 'Rock' | A comparison (or relational) operator is a mathematical symbol which is used to compare two values. Comparison operators are used in conditions that compare one expression with another. | <https://www.postgresqltutorial.com/postgresql-select-distinct/> |
| IN  -- COLUMN\_A IN ('ITEM\_A','ITEM\_B'...) | The SQL IN condition allows you to easily test if an expression matches any value in a list of values within a single column. It is used to help reduce the need for multiple [OR](https://www.techonthenet.com/sql/or.php) | <https://www.techonthenet.com/sql/in.php> |
| BETWEEN  -- COLUMN\_DATE BETWEEN '2008-01-01' AND '2012-12-31'  -- TO\_CHAR(COLUMN\_DATE,'YYYY') BETWEEN '2008' AND '2012' | The SQL BETWEEN condition allows you to easily test if an expression is within a range of values (inclusive). | <https://www.techonthenet.com/sql/between.php> |
| IS  -- COLUMN\_A IS NULL | The IS NULL condition is used in SQL to test for a NULL value. It returns TRUE if a NULL value is found, otherwise it returns FALSE. Commonly confused with IN. | <https://www.techonthenet.com/sql/is_null.php> |
| NOT  -- COLUMN\_A IS NOT NULL  -- COLUMN\_A NOT IN ('A','B','C') | The SQL NOT condition (sometimes called the NOT Operator) is used to negate a condition | <https://www.techonthenet.com/sql/not.php> |
| ILIKE  -- COLUMN\_A ILIKE 'the %' -- Starts with 'the'  -- COLUMN\_A ILIKE '% the %' -- Middle has 'the'  -- COLUMN\_A ILIKE '% the' -- Ends with 'the' | The SQL ILIKE condition allows you to use wildcards to perform pattern matching in a query. Case sensitivity is ignored with ILIKE. % is used to indicate any character (wildcard). Remember a SPACE ( ) is a character... | <https://www.postgresql.org/docs/8.3/functions-matching.html> |
| AND  -- GENRE = 'Rap' AND RATING > 4 | The SQL AND condition is used to test for two or more conditions in two or more columns. All conditions must be met for the record to be pulled. *Output: Will give a list of rap albums that have over a 4 rating.* | <https://www.techonthenet.com/sql/and.php> |
| OR (UNCONTAINED)  -- GENRE = 'Rap' AND RATING > 4 OR BURNED = TRUE | The SQL OR condition is used to test multiple conditions. Anyone of the conditions must be met for a record to be selected. *Output: Will give a list of rap albums that have over a 4 rating and a list of all burned albums.* | <https://www.techonthenet.com/sql/or.php> |
| OR (CONTAINED)  -- GENRE = 'Rap' AND (RATING > 4 OR BURNED = TRUE) | SQL OR operator can be wrapped in () to modify existing logic *Output: Will give a list of rap albums that have over a 4 rating and rap albums that are burned.* | <https://www.techonthenet.com/sql/or.php> |

# SQL Cheat Sheet Part 2 - Aggregate Functions

*AGGREGATES are* ***used to summarize columns of data****, typically numeric/integer data types. They can be placed in the SELECT and HAVING clauses. They will never go in the GROUP BY or WHERE clause…*

*Example Table*

|  |  |
| --- | --- |
| **ORDER\_ID** | **AMOUNT** |
| 1 | 10 |
| 2 | 0 |
| 3 | 5 |
| 4 | 10 |
| 5 | NULL |

|  |  |  |
| --- | --- | --- |
| **AGGREGATE** | **WHAT IT DOES** | **DOCUMENTATION** |
| SUM  -- SUM(AMOUNT) -- 25 -- meaningful  -- SUM(ORDER\_ID) -- 15 | Returns the sum of all the values in a column | <https://www.techonthenet.com/sql/sum.php> |
| MIN  -- MIN(AMOUNT) -- 0  -- MIN(ORDER\_ID) -- 1 | Returns the minimum value in a column, ignores nulls | <https://www.techonthenet.com/sql/min.php> |
| MAX  -- MAX(AMOUNT) -- 10  -- MAX(ORDER\_ID) -- 5 | Returns the maximum value in a column, ignores nulls | <https://www.techonthenet.com/sql/max.php> |
| COUNT  -- COUNT(AMOUNT) -- 4  -- COUNT(ORDER\_ID) -- 5 | Returns the count of records in a column, ignores nulls | <https://www.postgresqltutorial.com/postgresql-count-function/> |
| COUNT(DISTINCT)  -- COUNT(DISTINCT AMOUNT) -- 3  -- COUNT(DISTINCT ORDER\_ID) -- 5 | Returns the unique count of records in a column, ignores nulls. | <https://www.postgresqltutorial.com/postgresql-count-function/> |

# 