Lesson 4-Grouping Rows & Aggregate Functions

Lesson 4 Concepts

1. We can group rows together to see aggregated results in our result set.
2. There are built-in functions that can perform tasks on column data.
3. The HAVING clause acts like a WHERE clause for GROUP BY statements.

We Can Group Rows Together To See Aggregated Results in Our Result Set

With the queries that we have created so far, we have retrieved a number of rows from the database. Each row in our result set has represented one row from the database. By grouping the rows of our result set, we can have each row of our result set represent multiple rows from our database. For context, this is most often used with category columns, or columns where there are multiple repeating values in the column. For example, by grouping our class list by the *status* column, we could get two rows in our result set, one for Domestic students (and how many there are), and one for International students (and how many there are). Questions such as, “Which category of vehicle has the most rentals?”, and, “Which regional office has the highest sales average per quarter?” are answerable by grouping the data by these categories. The clause for grouping rows is, **GROUP BY**. We will look at this clause more closely in the next section.

There Are Built-in Functions That Can Perform Tasks on Columns

If you had to count how many fiction or non-fiction books were borrowed from a library, you would have to order by category and then manually count the rows. Wonderfully, there are built in SQL functions that can be performed on multiple rows to do this work for us. A function, just like in JavaScript, is a chunk of code that has already been written that we can call by name to do a job for us. The grouping (or “aggregate”) functions that we are looking at today are: **AVG()**, **MIN()**, **MAX()**, **SUM()**, and **COUNT()**. Note the parentheses after each keyword. Just like in JavaScript, the parentheses are where we put the thing that we want the function to do its job on.

Below, we have added **AVG(price)** to the **SELECT** clause of our statement. This will return the average of the values in the *price* column.

```
SELECT AVG(price) FROM stock_items;
```

The **MIN()** and **MAX()** functions will return the smallest value or the largest value in the provided column. This will work for string values (alphabetically, with A being the smallest value), and with dates (with more recent dates having a larger value than dates in the past). **SUM()** will *total* the values in a column, and **COUNT()** will return the *number of database rows* that are included in the grouping.

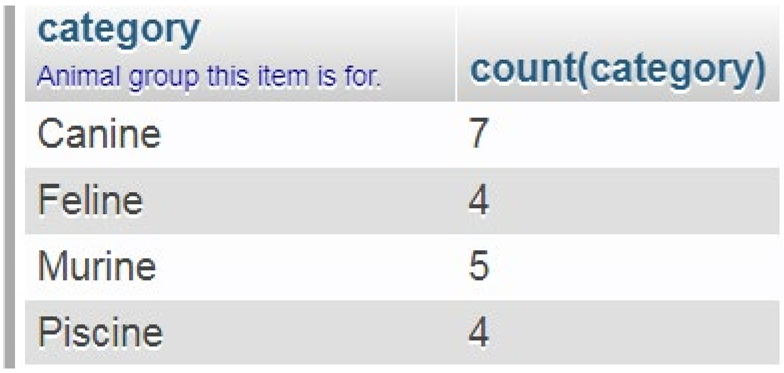
```
SELECT MIN(inventory) FROM stock_items;

SELECT MAX(price) FROM stock_items;

SELECT SUM(inventory) FROM stock_items;

SELECT COUNT(*) FROM stock_items;
```

The above queries will return a single digit without any further information. To provide a richer context (and turn data into knowledge ☺), these functions are typically used along with the **GROUP BY** clause.

The **GROUP BY** clause is the clause that will return multiple rows from the database grouped into individual rows in your result set, based on your specifications.

Compared to the last query above, the query below uses **COUNT()** and **GROUP BY** to return the number of rows for each unique value it finds in the category column.

```
SELECT category, count(category) FROM stock_items GROUP
BY category;
```



category Animal group this item is for.	count(category)
Canine	7
Feline	4
Murine	5
Piscine	4

The HAVING clause acts like a WHERE clause for GROUP BY statements

The **WHERE** clause lets us add additional conditions to our search query. Likewise, we can add conditions to our **GROUP BY** clause using the **HAVING** clause. This serves to limit or refine the groups of rows that are included in the result set.

The following query limits the number of groups by specifying that there must be more than one in the grouping of rows to be included in the result set. Note, we are using **COUNT()** with **SELECT** *and* with **HAVING**!!

```
SELECT COUNT(role), role FROM employees GROUP BY role
HAVING COUNT(role) > 1;
```

We can combine our keywords, functions, and clauses to give more succinct answers: "How many customers do we have in Germany and Britain?"

```
SELECT COUNT(country), country FROM Customers GROUP BY
Country HAVING country IN("Germany", UK");
```

References

Thomson, L., & Welling, L. (2003). *MySQL tutorial*. Sams.
The above reference is available through the Humber Library (library.humber.ca).

W3Schools SQL GROUP BY Statement
https://www.w3schools.com/sql/sql_groupby.asp

W3Schools SQL HAVING Clause
https://www.w3schools.com/sql/sql_having.asp