SUMMARY: AGGREGATE FUNCTIONS

SESSION OVERVIEW:

By the end of this session, the students will be able to:

- Understand different types of aggregate functions.
- Understand the GROUP BY function.
- Understand the use of the HAVING clause.
- Understand a few concepts related to scalar functions.

KEY TOPICS AND EXAMPLES:

Understanding different types of aggregate functions:

1. **What are aggregate functions?**

   Aggregate functions in SQL are unique functions that work on a group of rows in a table and produce a single value as a result. These operations are used to calculate a set of numbers and produce summary statistics like sum, count, average, maximum, and minimum. SQL queries frequently employ aggregate procedures to condense data from one or more tables.

   After grouping and aggregating, aggregate functions can also be used with the HAVING or WHERE clause to further filter the data. A condition involving an aggregate function can be used to filter the results of a query using the HAVING or WHERE clause.

   Below are a few frequently used aggregate functions in SQL. Let us understand each of these functions with the help of various examples.

   - Count()
   - Sum()
   - Avg()
   - Min()
   - Max()
   - Variance()
   - STDDEV()

2. **Uses of aggregate functions:**

   - **Summarizing Data:** Aggregate functions can be used to summarize data by calculating totals, averages, counts, minimums, or maximums of a set of values.
     For example: you can use SUM to calculate the total sales for a product or AVG to calculate the average salary of employees.

   - **Grouping Data:** Aggregate functions are often used with the GROUP BY clause to group rows that have the same values in specified columns. This allows you to perform aggregate calculations on each group separately.
     For example, you can use SUM with GROUP BY to calculate the total sales for each

product category.

- **Filtering Data:** Aggregate functions can also be used in the HAVING clause to filter groups based on aggregate values. This allows you to filter the result set based on the results of aggregate calculations.
  For example: you can use HAVING SUM(sales) > 10000 to only show product categories with total sales greater than 10,000.

- **Calculating Percentages:** Aggregate functions can be used to calculate percentages by dividing an aggregate value by another aggregate value or a constant.
  For example: you can use SUM(sales) / (SELECT SUM(sales) FROM sales) to calculate the percentage of total sales for each product.

- **Handling NULL Values**: Aggregate functions automatically ignore NULL values in their calculations. This can be useful when dealing with incomplete or missing data.

3. **Concepts on COUNT functions:**

   The SQL COUNT() function returns the number of rows in a table satisfying the criteria specified in the WHERE clause. It sets the number of rows or non-NULL column values. COUNT() returns 0 if there are no matching rows.

Basic syntax:

```
SELECT COUNT(expression)
FROM table_name
WHERE condition;
```

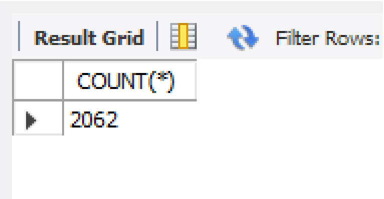
NOTE:

- ***expression is the column or expression to be counted. If you use COUNT(*), it will count all rows in the table.***
- ***table_name is the name of the table from which you want to count rows.***
- ***condition is an optional condition that must be met for a row to be included in the count. If no condition is specified, all rows are counted.***

Example 1:

```
# Query to find the total customers in the company.
SELECT COUNT(*) AS Total_count
FROM customers;
```

Output:



	COUNT(*)
▶	2062

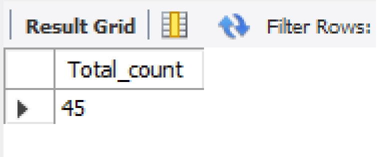
The above output indicates that the customers table has 2062 records.

Example 2:

```
# Query to find the total products whose cost is less than 1000 and
price is more than 1000.

SELECT COUNT(*) AS Total_count
FROM Products
WHERE ProductCost< 1000 AND ProductPrice> 1000;
```

Output:



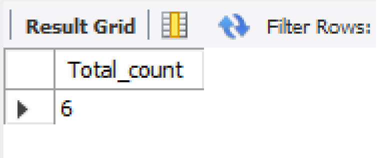
Total_count
45

Example 3:

```
# Query to find the unique subcategories whose product cost is less than
1000 and price is more than 1000.

SELECT COUNT(DISTINCT productsubcategorykey) AS Total_count
FROM Products
WHERE ProductCost< 1000 AND ProductPrice> 1000;
```

Output:



Total_count
6

4. **Concepts on SUM function:**

   The SUM function in SQL is used to calculate the sum of values in a column. It is an aggregate function, meaning it operates on a set of values and returns a single value.

Basic Syntax:

```
SELECT SUM(column_name) AS total_sum
FROM table_name;
```

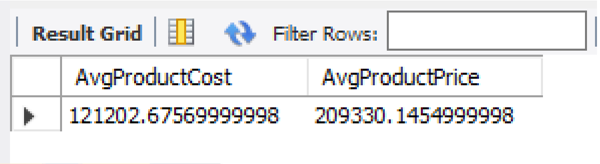
NOTE:

- ***column_name is the name of the column for which you want to calculate the sum.***
- ***table_name is the name of the table containing the column.***
- ***total_sum is the alias for the result, which will be the sum of all values in the specified column.***

Example 1:

```
SELECT SUM(ProductCost) AS AvgProductCost,
       SUM(Productprice) AS AvgProductPrice
FROM Products;
```

Output:



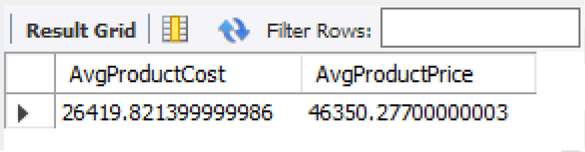
	AvgProductCost	AvgProductPrice
▶	121202.67569999998	209330.1454999998

Example 2:

```
# Query to find the total cost which is less than 800 and the total
price of the products which is more than 1000.

SELECT SUM(ProductCost) AS AvgProductCost,
      SUM(Productprice) AS AvgProductPrice
FROM Products
WHERE ProductCost<800 AND ProductPrice> 1000;
```

Output:



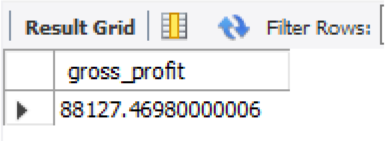
	AvgProductCost	AvgProductPrice
▶	26419.821399999986	46350.27700000003

Example 3:

```
# Query to find the profit of all the products.

SELECT SUM(ProductPrice - ProductCost) AS gross_profit
FROM Products;
```

Output:



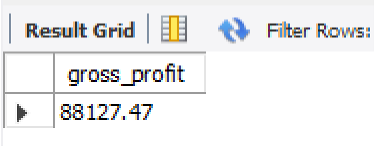
	gross_profit
▶	88127.46980000006

Example 4:

```
# Query to find the decimal value of the profit made by the company on
the products.

SELECT CAST(SUM(ProductPrice - ProductCost) AS DECIMAL(10,2)) AS
gross_profit
FROM Products;
```

Output:



	gross_profit
▶	88127.47

5. **Concepts on AVG function:**

   The AVG() method in MySQL is used to find the average value of an expression or column. The AVG() method takes a column or an expression as input and returns the average of the values in that column combined. The MySQL AVG() function can perform calculations on large data sets and the users can get valuable insights from the data.

Basic syntax:

```
SELECT AVG(column_name) AS total_avg
FROM table_name
WHERE condition;
```

Note:

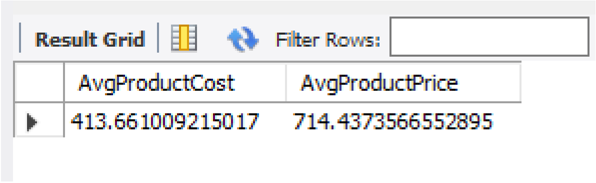
- ***column_name is the name of the column for which you want to calculate the sum.***
- ***table_name is the name of the table containing the column.***
- ***total_avg is the alias for the result, which will be the average of all values in the specified column.***

Example 1:

```
# Query to find the average cost and prices of the products.

SELECT AVG(ProductCost) AS AvgProductCost,
      AVG(Productprice) AS AvgProductPrice
FROM Products;
```

Output:



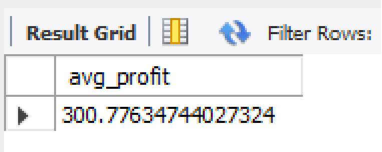
	AvgProductCost	AvgProductPrice
▶	413.661009215017	714.4373566552895

Example 2:

```
# Query to find the average profit made by the company.

SELECT AVG(ProductPrice - ProductCost) AS avg_profit
FROM products;
```

Output:



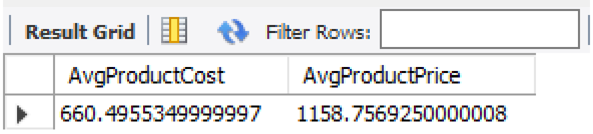
avg_profit
300.77634744027324

Example 3:

```
# query to find the average cost and price of the product whose cost is
less than 800 and price is more than 1000.

SELECT AVG(ProductCost) AS AvgProductCost,
      AVG(Productprice) AS AvgProductPrice
FROM Products
WHERE ProductCost<800 AND ProductPrice> 1000;
```

Output:



AvgProductCost	AvgProductPrice
660.4955349999997	1158.7569250000008

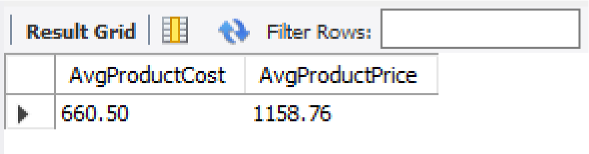
Example 4:

In the above example we have noticed that there is the value that is reflected is having recurring decimals which is not very important for the analysis purpose. Thus to make the query look clean we will use this particular query to reduce the decimals.

```
# Query to find the average cost and price of the product whose cost is
less than 800 and price is more than 1000.

SELECT CAST(AVG(ProductCost) AS DECIMAL(10,2))  AS AvgProductCost,
      CAST(AVG(Productprice) AS DECIMAL(10,2))AS AvgProductPrice
FROM Products
WHERE ProductCost<800 AND ProductPrice> 1000;
```

Output:



AvgProductCost	AvgProductPrice
660.50	1158.76

6. Concepts on MAX function:

The aggregate function SQL MAX() is used to find the maximum value or highest value of a certain column or expression. This function is useful to determine the largest of all selected values of a column.

Basic syntax:

```
SELECT MAX(column_name) AS max_value
FROM table_name
WHERE condition;
```

NOTE:

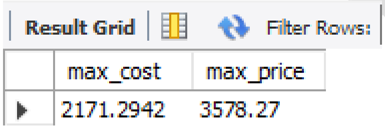
- *column_name is the name of the column for which you want to find the maximum value.*
- *table_name is the name of the table containing the column.*
- *max_value is the alias for the result, which will be the maximum value in the specified column.*

Example 1:

```
# Query to find the maximum product cost and product price.

SELECT MAX(productCost) AS max_cost,
       MAX(ProductPrice) AS max_price
FROM products;
```

Output:



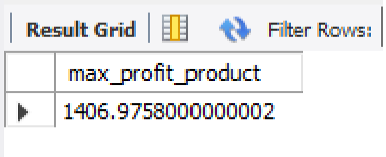
max_cost	max_price
2171.2942	3578.27

Example 2:

```
# Query to find the maximum profit made by the company.

SELECT MAX(ProductPrice) - MAX(ProductCost) AS max_profit_product
FROM products;
```

Output:



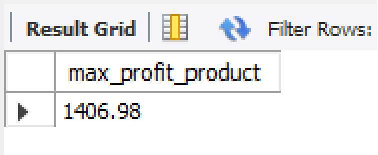
max_profit_product
1406.9758000000002

Example 3:

```
# Query to find the reduced decimals oriented maximum profit made by the
company.

SELECT CAST(MAX(ProductPrice) - MAX(ProductCost) AS DECIMAL(10,2)) AS
max_profit_product
FROM products;
```

Output:



max_profit_product
1406.98

7. Concepts on MIN function:

The aggregate function SQL MIN() is used to find the minimum value or lowest value of a column or expression. This function is useful to determine the smallest of all selected values of a column.

Basic syntax:

```
SELECT MIN(column_name) AS min_value
FROM table_name
WHERE condition;
```

Note:

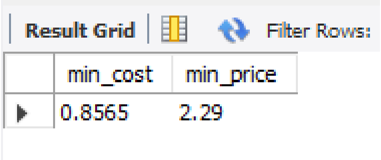
- ***column_name is the name of the column for which you want to find the minimum value.***
- ***table_name is the name of the table containing the column.***
- ***min_value is the alias for the result, which will be the minimum value in the specified column.***

Example 1:

```
# Query to find the minimum product cost and product price.

SELECT MIN(productCost) AS min_cost,
       MIN(ProductPrice) AS min_price
FROM products;
```

Output:



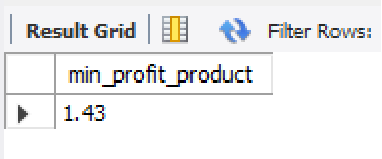
min_cost	min_price
0.8565	2.29

Example 2:

```
# Query to find the reduced decimal-oriented minimum profit made by the
company.

SELECT CAST(MIN(ProductPrice) - MIN(ProductCost) AS DECIMAL(10,2)) AS
min_profit_product
FROM products;
```

Output:



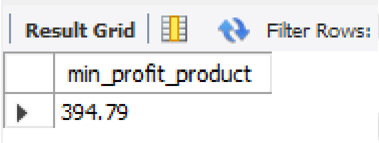
min_profit_product
1.43

Example 3:

```
# Query to find the reduced decimal-oriented minimum profit made by the
company where the price of the product is more than 1000 and the cost of
the product is less than 800.

SELECT CAST(MIN(ProductPrice - ProductCost) AS DECIMAL(10,2)) AS
min_profit_product
FROM products
WHERE ProductPrice>1000 AND ProductCost<800;
```

Output:



min_profit_product
394.79

8. **Concepts on VARIANCE():**

   MySQL VARIANCE() function returns the population standard variance of an expression. It considers the entire dataset rather than just a sample. Variance is a statistical measure that indicates the spread or dispersion of a dataset.

   This function is useful in -

   - It's essential for understanding the distribution of data.
   - High variance indicates greater dispersion, while low variance suggests data points are closer to the mean.
   - Variance can identify outliers or extreme values in the dataset. Unusually high variances might signal data quality issues or anomalies.
   - In finance and investment, variance is used to assess the risk associated with an investment.
   - Variance can be used to analyze the consistency of performance metrics. It might indicate how consistent the product quality is.

Basic Syntax:

```
VARIANCE(expr);
```

Syntax with select statement:

```
SELECT VARIANCE([DISTINCT] expression)
[FROM table_name];
```

NOTE:

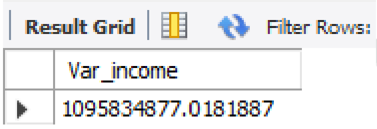
- ***SELECT: This is the standard keyword used to retrieve data from the database.***

- *VARIANCE(): This is the function that calculates the variance of the specified expression.*
- *[DISTINCT]: This is an optional keyword. If included, it calculates the variance of the distinct (unique) values in the expression.*
- *expression: This is the column or expression for which you want to calculate the variance. It can be a column name, a mathematical expression, or a function.*
- *[FROM table_name]: This is an optional clause that specifies the table from which to retrieve the data. If you include this clause, you can use column names directly in the expression.*

Example 1:

```
SELECT VARIANCE(annualIncome) AS Var_income
FROM Customers;
```

Output:



```
Var_income
1095834877.0181887
```

9. Concepts on STDDEV():

The STDDEV() function is used to calculate statistical information for a specified numeric field in a query. It returns NULL if no matching rows are found.
This function is useful in -

- It's used when you have a sample of data and want to estimate the variability in the entire population.
- STDDEV() quantifies the spread or dispersion of values around the mean (average) in a sample.
- When analyzing sample data, STDDEV() helps you assess the distribution of values and identify potential outliers or anomalies within the sample.

Syntax:

```
SELECT STDDEV([DISTINCT] expression) [FROM table_name];
```

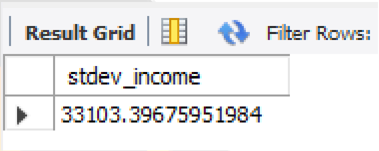
NOTE:

- *SELECT: This is the standard keyword used to retrieve data from the database.*
- *STDDEV(): This is the function that calculates the standard deviation of the specified expression.*
- *[DISTINCT]: This is an optional keyword. If included, it calculates the standard deviation of the distinct (unique) values in the expression.*
- *expression: This is the column or expression for which you want to calculate the standard deviation. It can be a column name, a mathematical expression, or a function.*
- *[FROM table_name]: This is an optional clause that specifies the table from which to retrieve the data. If you include this clause, you can use column names directly in the expression.*

Example 1:

```
SELECT stddev(annualIncome) AS stdev_income
FROM Customers;
```

Output:



```
stdev_income
33103.39675951984
```

10. Concepts on GROUP_CONCAT:

GROUP_CONCAT is a MySQL aggregate function that concatenates values from multiple rows into a single string result. This function is particularly useful for generating comma-separated lists, aggregating text data, or transforming row values into a single string.

Basic Syntax:

```
GROUP_CONCAT([DISTINCT] expression [ORDER BY expression [ASC | DESC]]
[SEPARATOR str_val])
```

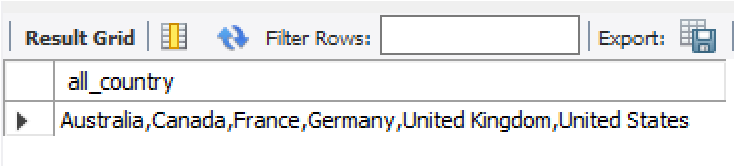
Note:

- ***DISTINCT:*** *Optional keyword to remove duplicate values from the concatenated result.*
- ***expression:*** *The column or expression whose values you want to concatenate.*
- ***ORDER BY:*** *Optional clause to sort the values before concatenating them.*
- ***SEPARATOR:*** *Optional clause to specify a custom separator string between the concatenated values (default is a comma,).*

Example:

```
SELECT GROUP_CONCAT(distinct Country) AS all_country
FROM territories;
```

Output:



```
all_country
Australia,Canada,France,Germany,United Kingdom,United States
```

Understanding the concept of GROUP BY:

GROUP BY is a clause in SQL that is used to group rows that have the same values in one or more columns. It is often used with aggregate functions like COUNT, SUM, AVG, MIN, and MAX to perform operations on each group of rows. Here are the key concepts of GROUP BY:

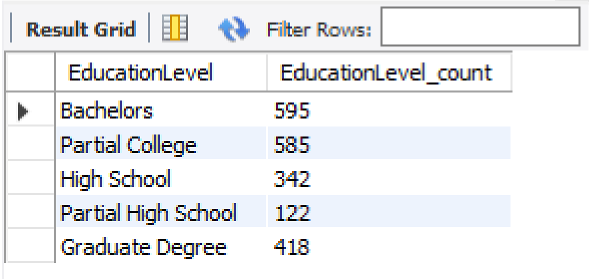
- **Grouping Rows:** The GROUP BY clause groups rows based on the values in specified columns. Rows with the same values in these columns are treated as a single group.
- **Aggregate Functions:** GROUP BY is commonly used with aggregate functions to perform calculations on each group of rows. For example, you can use SUM to calculate the total sales amount for each product category.
- **Columns in SELECT:** When using GROUP BY, any column in the SELECT statement that is not an argument of an aggregate function must be included in the GROUP BY clause. This is because SQL requires that all non-aggregated columns in the SELECT list be part of the GROUP BY clause to ensure the query is unambiguous.
- **Filtering Groups:** You can use the HAVING clause to filter groups based on aggregate values. This is similar to the WHERE clause but is applied after the GROUP BY clause and is used to filter groups, not individual rows.

Example 1:

```
# Query to find the total count of customers in different educational
levels and categorize using the education level criteria.

SELECT EducationLevel,
  COUNT(*) AS EducationLevel_count
FROM customers
GROUP BY EducationLevel;
```

Output:



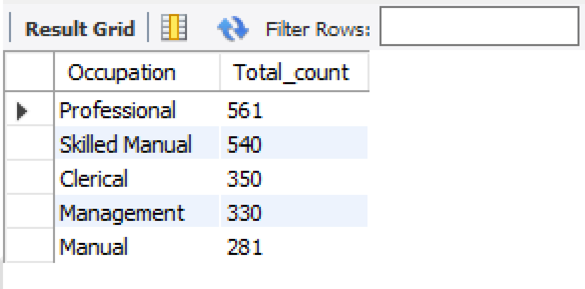
EducationLevel	EducationLevel_count
Bachelors	595
Partial College	585
High School	342
Partial High School	122
Graduate Degree	418

Example 2:

```
# Query to find the total count of the customers in occupations and
categorize using the occupation criteria. Also, sort it in descending
order.

SELECT Occupation, COUNT(*) AS Total_count
FROM Customers
GROUP BY Occupation
ORDER BY Total_count DESC;
```

Output:



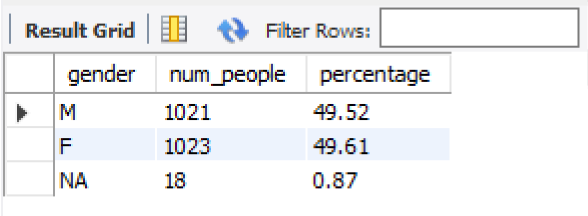
Occupation	Total_count
Professional	561
Skilled Manual	540
Clerical	350
Management	330
Manual	281

Example 3:

```
# Query to find the count as well as the percentage of people and
categorize with the help of gender criteria.

SELECT gender, COUNT(*) AS num_people,
       CAST(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM customers) AS
DECIMAL(5,2)) AS percentage
FROM customers
GROUP BY gender;
```

Output:



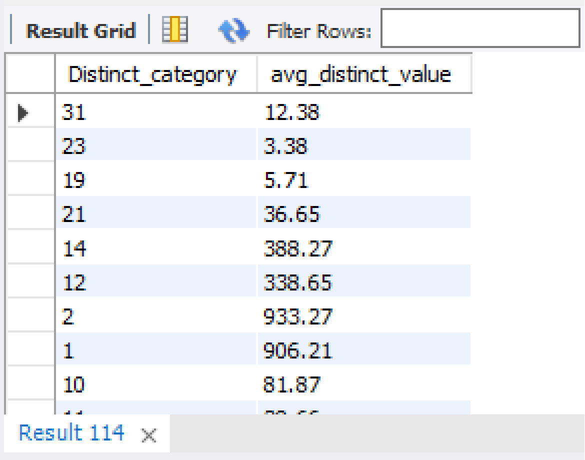
gender	num_people	percentage
M	1021	49.52
F	1023	49.61
NA	18	0.87

Example 4:

```
# query to find the reduced decimal-oriented average of the product cost
on categorization of the unique subcategories.

SELECT DISTINCT (ProductSubcategoryKey) AS Distinct_category,
         CAST(AVG(ProductCost) AS DECIMAL(10,2)) AS avg_distinct_value
FROM products
GROUP BY Distinct_category;
```

Output:



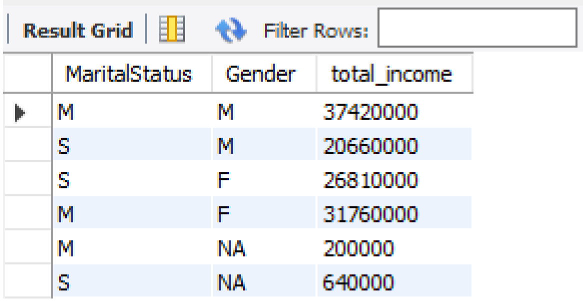
Distinct_category	avg_distinct_value
31	12.38
23	3.38
19	5.71
21	36.65
14	388.27
12	338.65
2	933.27
1	906.21
10	81.87

Example 5:

```
# Query to find the total income of the customers using a grouping of
more than one criteria.

SELECT MaritalStatus, Gender, SUM(annualincome) AS total_income
FROM customers
GROUP BY MaritalStatus, Gender;
```

Output:



MaritalStatus	Gender	total_income
M	M	37420000
S	M	20660000
S	F	26810000
M	F	31760000
M	NA	200000
S	NA	640000

Understanding the concept of the HAVING clause:

The MySQL HAVING Clause, utilized alongside the GROUP BY clause, filters grouped rows based on a specified condition, exclusively returning rows where the condition is TRUE. Acting on groups formed by GROUP BY, the HAVING clause allows the application of conditions, evaluating each group, and including those meeting the criteria. Unlike the WHERE clause, which operates on individual rows, HAVING functions on groups. When GROUP BY is omitted, HAVING behaves akin to WHERE. MySQL processes the HAVING clause after FROM, WHERE, SELECT, and GROUP BY, preceding DISTINCT, SELECT, ORDER BY, and LIMIT clauses, as per SQL standard specifications.

- **Filtering Grouped Data:** The HAVING clause filters the grouped rows returned by a GROUP BY clause based on specified conditions. It allows you to filter groups based on aggregated values such as COUNT, SUM, AVG, MIN, and MAX.

- **Aggregate Functions:** The HAVING clause is commonly used with aggregate functions like SUM, AVG, COUNT, MIN, and MAX. These functions are applied to groups of rows, and the HAVING clause filters the groups based on the result of these functions.

Basic syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

NOTE:

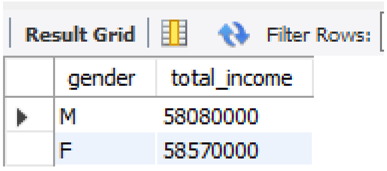
- ***The above syntax consists of all the relevant functions in it. It is not compulsory that every time whenever you use the HAVING clause you have to include the ORDER BY clause. You can use it accordingly whenever required.***

Example 1:

```
# Query to find the total income of the customers on gender criteria
whose income is more than 1000000.

SELECT gender, SUM(annualincome) AS total_income
FROM customers
GROUP BY gender
HAVING total_income > 1000000;
```

Output:



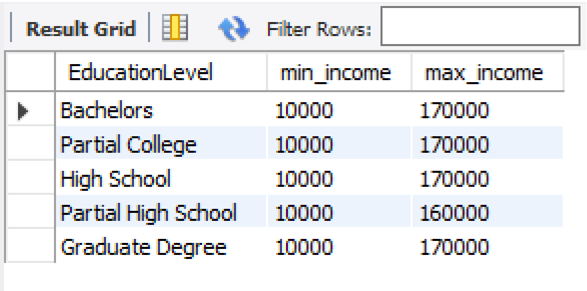
gender	total_income
M	58080000
F	58570000

Example 2:

```
# Query to find the minimum and maximum income of the customers whose
income is more than the different of the minimum and maximum income on
the educational level categorization.

SELECT EducationLevel, MIN(annualincome) AS min_income,
MAX(annualincome) AS max_income
FROM customers
GROUP BY educationlevel
HAVING MAX(annualincome) - MIN(annualincome) > 50000;
```

Output:



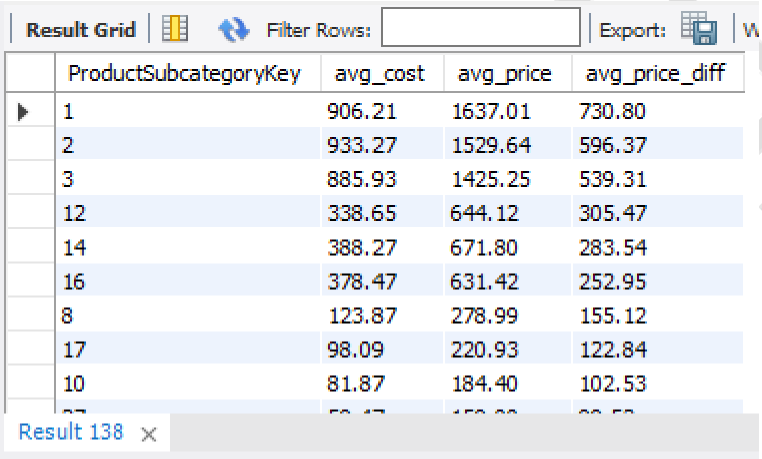
	EducationLevel	min_income	max_income
▶	Bachelors	10000	170000
	Partial College	10000	170000
	High School	10000	170000
	Partial High School	10000	160000
	Graduate Degree	10000	170000

Example 3:

```
# Query to find the reduced decimal oriented average of the product cost
and price. Also, find the average price difference which is more than 20
and sort by descending order.

SELECT ProductSubcategoryKey,
       CAST(AVG(productcost)AS DECIMAL(10,2)) AS avg_cost,
       CAST(AVG(productprice)AS DECIMAL(10,2)) AS avg_price,
       CAST(AVG(productprice - productcost)AS DECIMAL(10,2)) AS
avg_price_diff
FROM products
GROUP BY ProductSubcategoryKey
HAVING AVG(productprice - productcost) > 20
ORDER BY avg_price_diff DESC;
```

Output:



	ProductSubcategoryKey	avg_cost	avg_price	avg_price_diff
▶	1	906.21	1637.01	730.80
	2	933.27	1529.64	596.37
	3	885.93	1425.25	539.31
	12	338.65	644.12	305.47
	14	388.27	671.80	283.54
	16	378.47	631.42	252.95
	8	123.87	278.99	155.12
	17	98.09	220.93	122.84
	10	81.87	184.40	102.53

IMPORTANT:
Order of Execution: In SQL, the order of execution of a query is FROM -> WHERE -> GROUP BY -> HAVING -> SELECT -> ORDER BY. This means that the GROUP BY clause is applied after the WHERE clause and before the SELECT clause.

Difference between HAVING and WHERE clauses in SQL:

S NO.	WHERE	HAVING

1	Filters rows before groups are aggregated.	Filters groups after the aggregation process.
2	WHERE Clause can be used without GROUP BY Clause	HAVING Clause can be used with GROUP BY Clause
3	WHERE Clause implements in row operations	HAVING Clause implements in column operation
4	WHERE Clause cannot contain aggregate function	HAVING Clause can contain aggregate function
5	WHERE Clause can be used with SELECT, UPDATE, and DELETE statements.	HAVING Clause can only be used with a SELECT statement.
6	WHERE Clause is used before GROUP BY Clause	HAVING Clause is used after the GROUP BY Clause
7	WHERE Clause is used with single-row functions like UPPER, LOWER, etc.	HAVING Clause is used with multiple row functions like SUM, COUNT etc.

Understanding scalar functions like ROUND and ABS:

1. **Concepts on ROUND function:**

   In MySQL, the ROUND function is used to round a numeric value to a specified number of decimal places. The ROUND function is commonly used to round numeric values for display purposes or to simplify calculations. It is often used in conjunction with aggregate functions to round aggregated values.

   a. **Rounding Rules**: The ROUND function uses standard rounding rules:
      - If the decimal portion is 0.5 or greater, the number is rounded up.
      - If the decimal portion is less than 0.5, the number is rounded down.

   b. **Positive and Negative Decimals**: If decimals is positive, the ROUND function rounds number to the specified number of decimal places. If decimals is negative, number is rounded to the left of the decimal point (e.g., rounding to the nearest ten, hundred, etc.).

   c. Examples:
      - ROUND(3.14159, 2) returns 3.14 (rounding to 2 decimal places).
      - ROUND(123.456, -1) returns 120 (rounding to the nearest ten).
      - ROUND(123.456, -2) returns 100 (rounding to the nearest hundred).

Basic syntax:

```
ROUND(number, decimals)
```

NOTE:

- ***Number is the numeric value to be rounded***
- ***Decimals is the number of decimal places to round to. If decimals is omitted, the ROUND function rounds to the nearest integer.***

Syntax with SELECT statement:

```
SELECT ROUND(column_name, decimals) AS rounded_value
```

```
FROM table_name;
```

NOTE:

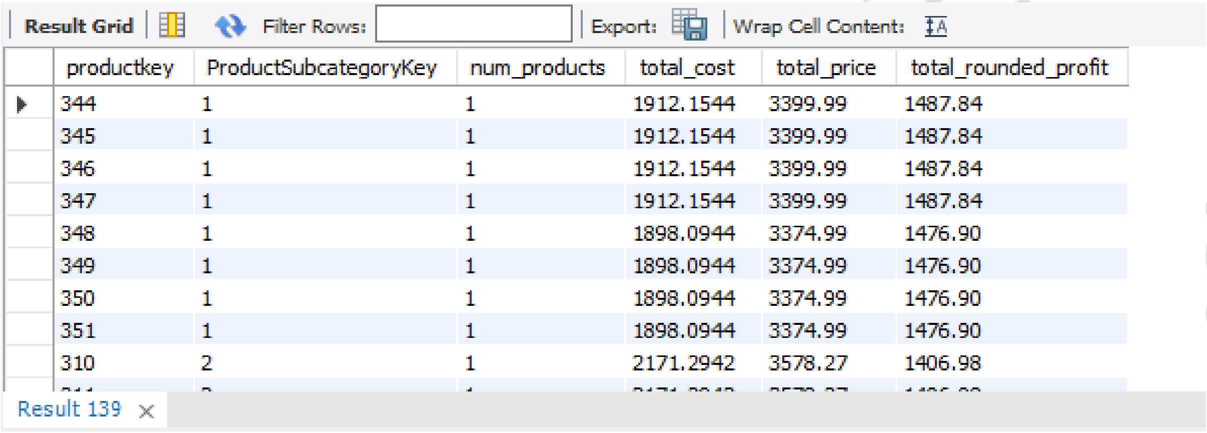
- *column_name is the name of the column containing the numeric value you want to round.*
- *decimals is the number of decimal places to round to.*
- *table_name is the name of the table containing the column.*

Example 1:

```
# Query to find the count of the total products, the total cost of the
products, the total price of the products, and the reduced
decimal-oriented total sum of the difference between product price and
product cost which need to be categorized on product key and
subcategorykey. Also, sort the difference between product price and
product cost in descending order.

SELECT
    productkey,
    ProductSubcategoryKey,
    COUNT(*) AS num_products,
    SUM(productcost) AS total_cost,
    SUM(productprice) AS total_price,
    CAST(ROUND(SUM(productprice) - SUM(productcost), 2) AS
DECIMAL(10,2)) total_rounded_profit
FROM
    products
GROUP BY
    productkey,
    ProductSubcategoryKey
ORDER BY
    total_rounded_profit DESC;
```

Output:



productkey	ProductSubcategoryKey	num_products	total_cost	total_price	total_rounded_profit
344	1	1	1912.1544	3399.99	1487.84
345	1	1	1912.1544	3399.99	1487.84
346	1	1	1912.1544	3399.99	1487.84
347	1	1	1912.1544	3399.99	1487.84
348	1	1	1898.0944	3374.99	1476.90
349	1	1	1898.0944	3374.99	1476.90
350	1	1	1898.0944	3374.99	1476.90
351	1	1	1898.0944	3374.99	1476.90
310	2	1	2171.2942	3578.27	1406.98

NOTE: The output format does not contain all the rows from the returned table.

2. Concepts on ABS function:

In MySQL, the ABS function is used to return the absolute value of a numeric expression. The ABS function is commonly used when you need to ignore the sign of a numeric value and work with its magnitude. For example, when calculating differences between values or distances from a reference point.

Return Value: The ABS function returns the absolute value of the specified number. If the number is positive, the result is the same as the input. If the number is negative, the result is the positive equivalent.

Examples:

- ABS(5) returns 5.
- ABS(-5) returns 5.
- ABS(0) returns 0.
- ABS(-10.75) returns 10.75.

Basic syntax:

```
ABS(number)
```

Syntax with SELECT statement:

```
SELECT ABS(column_name) AS abs_value
FROM table_name;
```

NOTE:

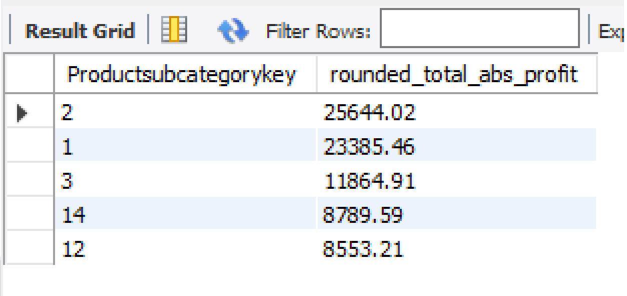
- **column_name is the name of the column or numeric expression for which you want to calculate the absolute value.**
- **table_name is the name of the table containing the column (if applicable).**

Example 1:

```
# Query to find the absolute sum of the difference between the product
price and the product cost which is categorized on
productsubcategorykey. Also, find the total on those whose difference
betwee product price and cost is more than 5000 in absolute terms and
sort by descending order.

SELECT Productsubcategorykey,
       CAST(ROUND(SUM(ABS(productprice - productcost)), 2) AS
DECIMAL(10, 2)) AS rounded_total_abs_profit
FROM products
GROUP BY Productsubcategorykey
HAVING SUM(ABS(productprice - productcost)) > 5000
ORDER BY Rounded_total_abs_profit DESC;
```

Output:



Productsubcategorykey	rounded_total_abs_profit
2	25644.02
1	23385.46
3	11864.91
14	8789.59
12	8553.21

Note: The student can play around with the tables available in the dataset to practice the above mentioned functions and clauses which will strengthen the fundamental understanding of the concepts.