Learning Objectives

- GROUP BY on a single table
- GROUP BY on two or more tables
- GROUP BY with JOINS

LO1: Understanding GROUP BY clause on a Single Table

The **GROUP BY** statement in SQL is used to arrange identical data into groups based on specified columns. If a particular column has the same values in multiple rows, the **GROUP BY** clause will group these rows together.

Key Points About GROUP BY:

- GROUP BY clause is used with the **SELECT** statement.
- In the query, the GROUP BY clause is placed after the **WHERE** clause.
- In the query, the GROUP BY clause is placed before the **ORDER BY** clause if used.
- In the query, the Group BY clause is placed before the Having clause.
- Conditions should be placed in the **HAVING clause.**

Syntax:

```sql
SELECT column1, function_name(column2)
FROM table_name
GROUP BY column1, column2;
```

Example: Let us consider the following tables:

Example 1: Group By Single Column:

employees

EmployeeID	LastName	FirstName	BirthDate
1	Davolio	Nancy	1968-12-08
2	Fuller	Andrew	1952-02-19
3	Leverling	Janet	1963-08-30
4	Peacock	Margaret	1958-09-19
5	Buchanan	Steven	1955-03-04
6	Suyama	Michael	1963-07-02
7	King	Robert	1960-05-29
8	Callahan	Laura	1958-01-09

Requirement: Get the number of employees born in each year
Query:

```
SELECT  YEAR(BirthDate)  AS  BirthYear,  COUNT(EmployeeID)  AS
Total_employees
FROM employees
GROUP BY BirthYear;
```

Result:

BirthYear	Total_employees
1968	1
1952	1
1963	2
1958	2
1955	1

Example 2: Group By Multiple Columns

customers

CustomerID	CustomerName	Country	City
1	Alfreds Futterkiste	Germany	Berlin

2	Furia Bacalhau e Frutos do Mar	Portugal	Lisboa
3	Bólido Comidas preparadas	Spain	Madrid
4	Blauer See Delikatessen	Germany	Mannheim
5	Princesa Isabel Vinhoss	Portugal	Lisboa
6	Galería del gastrónomo	Spain	Madrid
7	Godos Cocina Típica	Spain	Madrid

Requirement: count customers in each country and city.
Query:

```
SELECT Country, City, COUNT(CustomerID) AS Total_customers
FROM customers
GROUP BY Country, City;
```

Result:

Country	City	Total_customers
Germany	Berlin	1
Portugal	Lisboa	2
Spain	Madrid	3
Germany	Mannheim	1

HAVING Clause in GROUP BY Clause

The **WHERE clause** is used to place conditions on columns but what if we want to place conditions on groups? This is where the HAVING clause comes into use. We can use the **HAVING clause** to place conditions to decide which group will be part of the final result set.

The WHERE clause filters rows before grouping, whereas the HAVING clause filters groups after they have been created. If we use WHERE instead of HAVING with an aggregate function, the query will result in an error because WHERE does not work with grouped data.

Syntax:

```
SELECT column1, function_name(column2)
FROM table_name
WHERE condition
GROUP BY column1, column2
HAVING condition
ORDER BY column1, column2;
```

Example: Consider the customers table.

Requirement: Find cities in each country where more than 1 customer exist (customers table).

Query:

```
SELECT Country, City, COUNT(CustomerID) AS Total_customers
FROM customers
GROUP BY Country, City
HAVING Total_customers > 1;
```

Result:

Country	**City**	**Total_customers**
Portugal	Lisboa	2
Spain	Madrid	3

LO2: Understanding GROUP BY clause on multiple tables

The **GROUP BY** clause in SQL helps organize data by grouping similar values. When working with multiple tables **without JOIN**, you can apply GROUP BY separately to each table. If needed, results from different tables can be combined using **UNION** or **UNION ALL**. This method is useful for summarizing data independently while keeping it structured and easy to analyze.

The UNION ALL operator keeps all duplicate records, whereas the UNION operator removes duplicates. In this case, UNION ALL is used to maintain all records from both tables without filtering out any duplicate customer IDs.

Syntax:

```
SELECT column1, column2
FROM Table1
GROUP BY column1

UNION ALL/ UNION

SELECT column1, column2
FROM Table2
GROUP BY column2;
```

Example: Let us consider the following tables.

customers

CustomerID	CustomerName	Country	City
1	Alfreds Futterkiste	Germany	Berlin
2	Furia Bacalhau e Frutos do Mar	Portugal	Lisboa
3	Bólido Comidas preparadas	Spain	Madrid
4	Blauer See Delikatessen	Germany	Mannheim
5	Princesa Isabel Vinhoss	Portugal	Lisboa

suppliers

SupplierID	SupplierName	Country	City
1	Exotic Liquid	UK	Londona

2	New Orleans Cajun Delights	USA	New Orleans
3	Grandma Kelly's Homestead	USA	Ann Arbor
4	Tokyo Traders	Japan	Tokyo
5	Cooperativa de Quesos 'Las Cabras'	Spain	Oviedo

Requirement: count how many entities are in each country (combining both tables).

Query:

Query 1: Grouping Customers Table

```
SELECT Country, COUNT(CustomerID) AS total_customers,
'Customer' AS entity_type
FROM customers
GROUP BY Country;
```

Result:

country	total_customers	entity_type
Germany	2	customers
Portugal	2	customers
Spain	1	customers

Query 2: Grouping Suppliers Table

```
SELECT Country, COUNT(SupplierID) AS total_suppliers,
'Supplier' AS entity_type
FROM suppliers
GROUP BY Country;
```

Result:

country	total_customers	entity_type
USA	2	suppliers
Japan	1	suppliers

Spain	1	suppliers
UK	1	suppliers

Query 3: Combining Results with UNION

```
SELECT Country, COUNT(CustomerID) AS total_customers,
'Customer' AS entity_type
FROM customers
GROUP BY Country

UNION ALL

SELECT Country, COUNT(SupplierID) AS total_suppliers,
'Supplier' AS entity_type
FROM suppliers
GROUP BY Country;
```

Result:

country	**total_customers**	**entity_type**
Germany	2	customers
Portugal	2	customers
Spain	1	customers
USA	2	suppliers
Japan	1	suppliers
Spain	1	suppliers
UK	1	suppliers

LO3: Understanding GROUP BY clause with JOINS

In SQL, **JOIN** combines data from multiple tables, and **GROUP BY** organizes and summarizes it. When used together, JOIN merges related records first, and GROUP BY groups them to perform aggregate calculations like SUM, COUNT, or AVG. This method is essential for analyzing relationships, generating reports, and extracting meaningful insights efficiently.

Syntax:

```
SELECT Table1.column1, Table2.column2,
aggregate_function(Table1.column3)
FROM Table1
INNER JOIN/ CROSS JOIN/ NATURAL JOIN/ LEFT JOIN/ RIGHT JOIN/
JOIN Table2
ON Table1.common_column = Table2.common_column
GROUP BY Table1.column1, Table2.column2;
```

Example: Let us consider the following tables.

employees

EmployeeID	LastName	FirstName	BirthDate
1	Davolio	Nancy	1968-12-08
2	Fuller	Andrew	1952-02-19
3	Leverling	Janet	1963-08-30
4	Peacock	Margaret	1958-09-19
5	Buchanan	Steven	1955-03-04
6	Suyama	Michael	1963-07-02
7	King	Robert	1960-05-29
8	Callahan	Laura	1958-01-09

orders

OrderID	CustomerID	EmployeeID	OrderDate
10248	90	5	1996-07-04
10249	81	6	1996-07-05
10250	34	4	1996-07-08
10258	20	1	1996-07-17
10259	13	4	1996-07-18
10260	55	4	1996-07-19
10265	7	2	1996-07-25
10269	89	5	1996-07-31

Requirement: count how many orders each employee handled

Query:

```
SELECT e.EmployeeID, e.LastName, COUNT(o.OrderID) AS
total_orders
FROM employees e
```

```
JOIN orders o ON e.EmployeeID = o.EmployeeID
GROUP BY e.EmployeeID, e.LastName;
```

Result:

EmployeeID	LastName	total_orders
1	Davolio	1
2	Fuller	1
3	Leverling	0
4	Peacock	3
5	Buchanan	2
6	Suyama	1

Conclusion:

- GROUP BY is used to organize and summarize data in SQL.
- HAVING filters groups, while WHERE filters rows before grouping.
- JOIN and GROUP BY together help in analyzing related data.

Tasks:

1. Find the number of customers in each country.
   (Country, Total_Customers)
2. Retrieve the total number of products in each category.
   (CategoryID, CategoryName, Total_Products)
3. List the total number of orders handled by each employee.
   (EmployeeID, LastName, Total_Orders)
4. Find the total quantity ordered for each product.
   (ProductID, ProductName, Total_Quantity)
5. Retrieve the total number of orders shipped by each shipper.
   (ShipperID, ShipperName, Total_Orders)
6. List customers who have placed more than 5 orders.
   (CustomerID, CustomerName, Total_Orders)
7. Get the average quantity per order for each product.
   (ProductID, ProductName, Avg_Quantity)
8. List of products ordered along with their total quantity arrange them in descending order.

(ProductID, ProductName, Total_Quantity)

9. List the number of orders for each month.
   (OrderMonth, Total_Orders)
10. List product categories along with their total sales quantity.
    (CategoryID, CategoryName, Total_SalesQuantity)
11. Find employees who have handled more than 20 orders.
    (EmployeeID, LastName, Total_Orders)
12. Identify the products that have been ordered more than 50 times.
    (ProductID, ProductName, Total_Quantity)
13. List the total number of orders for each year.
    (OrderYear, Total_Orders)
14. List employees along with the number of orders they handled.
    (EmployeeID, LastName, Total_Orders)
15. Find the products that have never been ordered.
    (ProductID, ProductName)
16. List the total number of orders placed in each country.
    (Country, Total_Orders)
17. List employees along with the number of unique products they have worked on.
    (EmployeeID, LastName, Unique_Products_Count)
18. List shipping methods along with the number of orders shipped.
    (ShipperID, ShipperName, Total_Orders)

Hackerrank Task:

1. Average-Population
2. Revising-Aggregations-The-Averagefunction
3. Revising-Aggregations-Sum
4. Revising-Aggregations-The-Countfunction
5. Weather-Observation-Station-6
6. Weather-Observation-Station-7
7. Weather-Observation-Station-8
8. Weather-Observation-Station-9
9. Weather-Observation-Station-10

10. Weather-Observation-Station-11
11. Weather-Observation-Station-12
12. More-Than-75-Marks
13. Population-Density-Difference
14. The-Blunder
15. Earnings-Of-Employees
16. What-Type-Of-Triangle

What to Submit:

Submit the following file in Zip on Eduko:

- 2024-CS-X.txt