**Execute OVER and PARTITION BY Clause in SQL Queries**

**SQL PARTITION BY Clause:**

The SQL PARTITION BY clause is a subclause of the OVER clause, commonly used in conjunction with window functions such as AVG(), MAX(), and RANK(). It is employed in scenarios where window functions operate on window frames—sets of rows that can vary for each record in the query result.

**Syntax:**

**SELECT**

**<column>,**

**<window function> OVER(PARTITION BY <column> [ORDER BY <column>])**

**FROM table;**

The PARTITION BY clause divides the result set into partitions based on specified columns, and the window function is then applied within each partition. This allows for more granular analysis within distinct groups of data.

**1. Basic SELECT Query:**

The basic SELECT query retrieves specific columns from the City table where a certain condition is met (e.g., CountryCode is 'AUS' or 'NZL'). This is a standard query for data retrieval.

**2. GROUP BY with WITH ROLLUP:**

The query demonstrates the use of the GROUP BY clause with ROLLUP, providing subtotals and a grand total for city populations. This is helpful for summarizing data and generating reports with hierarchical totals.

**3. Query Without Labels (using NULLs):**

Similar to the previous query, but without explicit labels. NULL values represent subtotal and grand total rows. This showcases an alternative presentation style.

**Key Concepts:**

**GROUP BY Clause:**

* Used to group rows based on specific criteria, allowing the application of aggregate functions to yield one result per group.

**WITH ROLLUP:**

* Extends the GROUP BY functionality by including extra rows that represent subtotals and a grand total.

**Aggregation Functions (SUM, AVG):**

* Functions like SUM() and AVG() are employed to perform calculations on groups of rows, providing aggregated results.

**PARTITION BY Clause:**

* Used within the OVER clause to define partitions for window functions, enabling analysis within distinct subsets of data.

**Window Functions:**

* Functions like AVG(), MAX(), etc., operate on window frames, and the PARTITION BY clause helps delineate these frames for specific analyses.

**ORDER BY Clause:**

* Optionally used within the PARTITION BY clause to establish the order of records within the window frame. Some window functions, like LEAD() and LAG(), require an ORDER BY clause.

**Window Frame Bounds:**

* Specifies the limits of the window frame using expressions like UNBOUNDED PRECEDING, n PRECEDING, CURRENT ROW, n FOLLOWING, and UNBOUNDED FOLLOWING.

USE coding\_challenge1;

-- Create another table for car\_sales

CREATE TABLE IF NOT EXISTS car\_sales (

sale\_id INT PRIMARY KEY,

car\_make VARCHAR(50),

car\_model VARCHAR(50),

sale\_price DECIMAL(10, 2),

sale\_date DATE

);

-- Insert data into the car\_sales table

INSERT INTO car\_sales VALUES

(1, 'Ford', 'Mondeo', 20000, '2023-01-15'),

(2, 'Renault', 'Fuego', 18000, '2023-01-20'),

(3, 'Citroen', 'Cactus', 22000, '2023-02-05'),

(4, 'Ford', 'Falcon', 9500, '2023-02-10'),

(5, 'Ford', 'Galaxy', 13000, '2023-03-01'),

(6, 'Renault', 'Megane', 15000, '2023-03-05'),

(7, 'Citroen', 'Picasso', 25000, '2023-03-10');

**-- Query 1: Average price across all car sales and average price by car type**

**SELECT**

**car\_make,**

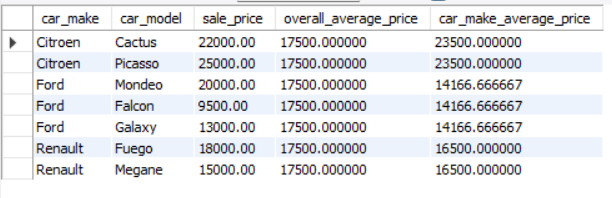
**car\_model,**

**sale\_price,**

**AVG(sale\_price) OVER() AS "overall\_average\_price",**

**AVG(sale\_price) OVER (PARTITION BY car\_make) AS "car\_make\_average\_price"**

**FROM car\_sales;**

****

**-- Query 2: Average price and top price per car make**

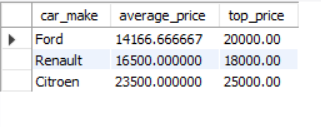
**SELECT**

**car\_make,**

**AVG(sale\_price) AS average\_price,**

**MAX(sale\_price) AS top\_price**

**FROM car\_sales**

**GROUP BY car\_make;  
  
**

**-- Query 3: Average make price using window functions**

**SELECT**

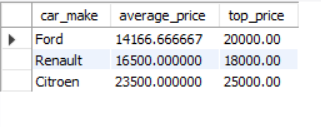
**car\_make,**

**car\_model,**

**sale\_price,**

**AVG(sale\_price) OVER (PARTITION BY car\_make) AS average\_make**

**FROM car\_sales;**

****

**-- Query 4: Monthly variation in car sales prices**

**WITH year\_month\_data AS (**

**SELECT DISTINCT**

**DATE\_FORMAT(sale\_date, '%Y') AS year,**

**DATE\_FORMAT(sale\_date, '%m') AS month,**

**AVG(sale\_price) AS average\_price**

**FROM car\_sales**

**GROUP BY 1, 2**

**)**

**SELECT year,**

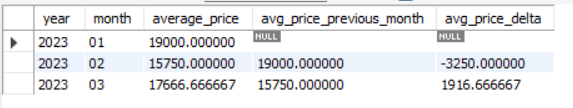
**month,**

**average\_price,**

**LAG(average\_price) OVER (ORDER BY year, month) AS avg\_price\_previous\_month,**

**average\_price - LAG(average\_price) OVER (ORDER BY year, month) AS avg\_price\_delta**

**FROM year\_month\_data;**

****

**Totals and Subtotals**

**Subtotals and Total Aggregations:**

Subtotals are generated by the ROLLUP operator, which creates aggregated results for subsets of the grouping columns.

The grand total represents the overall sum of the specified column (Population in this case).

The GROUP BY clause, in conjunction with the ROLLUP operator, allows for efficient calculation of subtotals and grand totals in a single query.

**ROLLUP Operator:**

The ROLLUP operator is used in the GROUP BY clause to generate subtotals and a grand total for a result set.

It is used to create multiple grouping sets in a single query, allowing you to aggregate data at different levels of granularity.

**Concepts in the Query:**

IF(GROUPING(CountryCode), 'All Countries', CountryCode):

The GROUPING function is used to check if a column is part of the grouping set.

If GROUPING(CountryCode) evaluates to 1, it means that the row represents a subtotal or grand total, and 'All Countries' is displayed; otherwise, the actual CountryCode is displayed.

**SUM(Population):**

Aggregates the population values for each grouping set.

**-- Create a new database**

**CREATE DATABASE IF NOT EXISTS city\_population\_Subtotals;**

**USE city\_population\_Subtotals;**

-- Create a table for City

CREATE TABLE IF NOT EXISTS City (

CountryCode CHAR(3),

District VARCHAR(255),

Name VARCHAR(255),

Population INT

);

-- Insert values into the City table

INSERT INTO City (CountryCode, District, Name, Population) VALUES

('AUS', 'New South Wales', 'Sydney', 3276207),

('AUS', 'Victoria', 'Melbourne', 2865329),

('AUS', 'Queensland', 'Brisbane', 1291117),

('AUS', 'West Australia', 'Perth', 1096829),

('AUS', 'South Australia', 'Adelaide', 978100),

('AUS', 'Capital Region', 'Canberra', 322723),

('AUS', 'Queensland', 'Gold Coast', 311932),

('AUS', 'New South Wales', 'Newcastle', 270324),

('AUS', 'New South Wales', 'Central Coast', 227657),

('AUS', 'New South Wales', 'Wollongong', 219761),

('AUS', 'Tasmania', 'Hobart', 126118),

('AUS', 'Victoria', 'Geelong', 125382),

('AUS', 'Queensland', 'Townsville', 109914),

('AUS', 'Queensland', 'Cairns', 92273),

('NZL', 'Auckland', 'Auckland', 381800),

('NZL', 'Canterbury', 'Christchurch', 324200),

('NZL', 'Auckland', 'Manukau', 281800),

('NZL', 'Auckland', 'North Shore', 187700),

('NZL', 'Auckland', 'Waitakere', 170600),

('NZL', 'Wellington', 'Wellington', 166700),

('NZL', 'Dunedin', 'Dunedin', 119600),

('NZL', 'Hamilton', 'Hamilton', 117100),

('NZL', 'Wellington', 'Lower Hutt', 98100);

**-- Query to select all records from the City table**

**SELECT**

**CountryCode,**

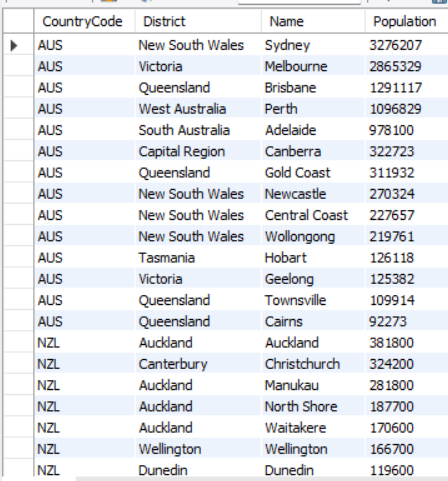
**District,**

**Name,**

**Population**

**FROM City**

**WHERE CountryCode IN ('AUS', 'NZL');**

****

**-- Query to retrieve city populations with subtotals and grand total**

**SELECT**

**IF(GROUPING(CountryCode), 'All Countries', CountryCode) AS CountryCode,**

**IF(GROUPING(District), 'All Districts', District) AS District,**

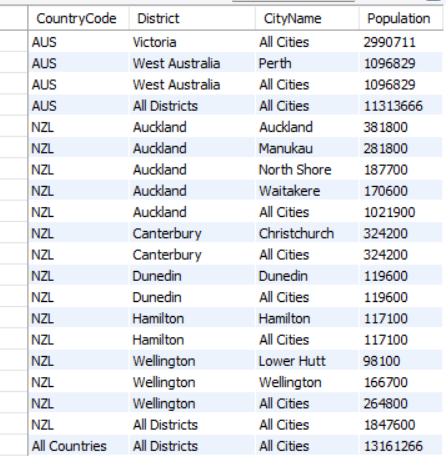
**IF(GROUPING(Name), 'All Cities', Name) As CityName,**

**SUM(Population) AS Population**

**FROM City**

**WHERE CountryCode IN ('AUS', 'NZL')**

**GROUP BY CountryCode, District, Name WITH ROLLUP;**

****

**-- Query without labels (using NULLs) to get the same output**

**SELECT**

**CountryCode,**

**District,**

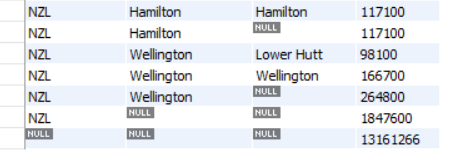
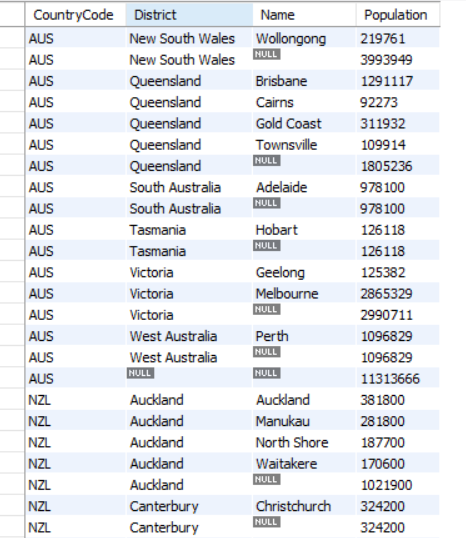
**Name,**

**SUM(Population) AS Population**

**FROM City**

**WHERE CountryCode IN ('AUS', 'NZL')**

**GROUP BY CountryCode, District, Name WITH ROLLUP;**

****