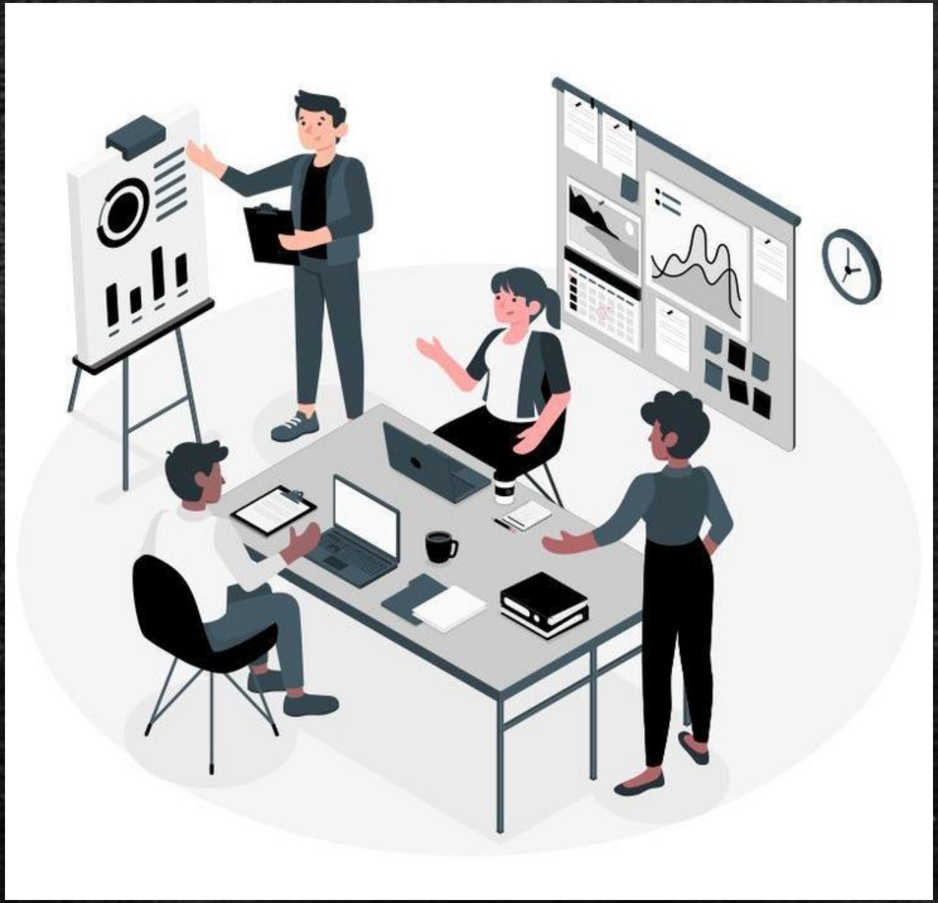
Aggregation in SQL

By : Marwa Ashraf

Learning Objectives

- Understand the concept of aggregation in SQL.
- Learn how to use aggregate functions such as SUM, AVG, COUNT, MIN, and MAX.
- Gain proficiency in writing queries involving aggregation.
- Practice using GROUP BY and HAVING clauses.
- Explore advanced aggregation techniques like nested queries and window functions.

Lesson Topics



- Lesson 1: Introduction to Aggregation
- Lesson 2: Basic Aggregate Functions
- Lesson 3: Grouping Data
- Lesson 4: Filtering Grouped Data
- Lesson 5: Advanced Aggregation Techniques

Lesson 1: Introduction to Aggregation

- Definition of aggregation in SQL
- Importance of aggregation in data analysis

Definition of aggregation in SQL

- An aggregate function is a function that performs a calculation on a set of values, and returns a single value
- aggregation refers to the process of applying a mathematical operation such as sum, average, count, minimum, or maximum to a set of values within a group. This grouping is typically done using the GROUP BY clause in SQL queries.
- `SELECT City_Name`
- `COUNT (Voter_ID) AS "Voter_Count"`
- `FROM Voter_List`
- `GROUP BY City_Name`
- `ORDER BY City_Name;`

Importance of aggregation in data analysis

- Aggregation plays a crucial role in data analysis for several reasons:
  - **Summarization of Data**: Aggregation allows you to condense large datasets into more manageable and understandable summaries
  - **Understanding Patterns and Trends**: Aggregating data helps in identifying patterns and trends that might not be apparent when looking at individual data points
  - **Comparing Data Segments**: Aggregation enables comparisons between different segments or groups within the dataset
  - **Supporting Decision Making**: Aggregated data provides valuable information for decision-making processes
  - **Data Visualization**: Aggregated data is often used for data visualization purposes
  - **Improving Performance**: Aggregation can also improve the performance of data analysis queries

Lesson 2: Basic Aggregate Functions

- Understanding SUM, AVG, COUNT, MIN, and MAX functions
- Syntax and usage examples for each function
- Applying aggregate functions in simple queries

Understanding SUM, AVG, COUNT, MIN, and MAX functions

- **COUNT()**: This function counts the number of rows that meet a specified condition. It can also count all rows in a table if no condition is specified.
  - Syntax :
    - SELECT COUNT(*) FROM table_name;
  - Example:
    - SELECT COUNT(*)
      FROM Student
      WHERE Age<15
- **SUM()**: Computes the sum of values in a column.
  - Syntax :
    - SELECT SUM(Field) FROM Table_Name
  - Example:
    - SELECT SUM(sales_amount) FROM sales_data;

Understanding SUM, AVG, COUNT, MIN, and MAX functions

- **AVG()**: Calculates the average value of a numeric column.
  - Syntax :
    - SELECT AVG(Field) FROM Table_Name;
- Example:
  - SELECT AVG(salary) FROM employees;

- **MIN()**: Returns the minimum value of a column.
  - Syntax :
    - SELECT AVG(Field) FROM Table_Name;
  - Example:
    - SELECT MIN(age) FROM students;

Understanding SUM, AVG, COUNT, MIN, and MAX functions

- **MAX()**: Returns the maximum value of a column.
  - Syntax :
    - SELECT AVG(Field) FROM Table_Name;
  - Example:
    - SELECT MAX(price) FROM products;
- Aggregate functions ignore null values (except for COUNT ( )).

Lesson 3: Grouping Data

- Introduction to the GROUP BY clause
- Grouping data based on one or multiple columns
- Writing queries with GROUP BY and aggregate functions

Introduction to the GROUP BY clause

- The GROUP BY clause divides the rows returned from the SELECT statement into groups based on one or more columns. Rows with the same values in the specified column(s) are grouped together.
  - Example:

    -
      ```
      SELECT department, COUNT(*)
      FROM employees
      GROUP BY department;
      ```
    - This query groups employees by department and counts the number of employees in each department

Grouping data based on one or multiple columns

- You can group rows based on multiple columns by specifying those columns in the GROUP BY clause.
  - Example:
    - SELECT department, gender, AVG(salary)
      FROM employees
      GROUP BY department, gender;

    This query calculates the average salary for each combination of department and gender.

Writing queries with GROUP BY and aggregate functions

- the GROUP BY clause is used in conjunction with aggregate functions to perform calculations on each group of rows.
  - Example:
    - SELECT department, AVG(salary)
      FROM employees
      GROUP BY department;
    - This query calculates the average salary for each department

Lesson 4: Filtering Grouped Data

- Introduction to the HAVING clause
- Filtering grouped data using aggregate conditions
- Writing queries with GROUP BY, HAVING, and aggregate functions

Introduction to the HAVING clause

- The HAVING clause in SQL is used in combination with the GROUP BY clause to filter the rows that are returned by a GROUP BY operation based on specified conditions.
- While the WHERE clause filters rows before the aggregation, the HAVING clause filters groups after the aggregation

Filtering grouped data using aggregate conditions

- The HAVING clause allows you to apply conditions to the results of aggregate functions, such as SUM(), AVG(), COUNT(), MIN(), or MAX(), after the data has been grouped by the GROUP BY clause.
  - Example:

    -
      ```
      SELECT department, AVG(salary)
      FROM employees
      GROUP BY department
      HAVING AVG(salary) > 50000;
      ```
    - This query selects the departments with an average salary greater than $50,000

    -
      ```
      SELECT department, COUNT(*)
      FROM employees
      GROUP BY department
      HAVING COUNT(*) > 10;
      ```
    - This query selects departments with more than 10 employees.

Filtering grouped data using aggregate conditions

- **Logical Operators**: You can combine conditions in the HAVING clause using logical operators such as AND, OR, and NOT.
  - Example:

    -
      ```
      SELECT department, AVG(salary)
      FROM employees
      GROUP BY department
      HAVING AVG(salary) > 50000 AND COUNT(*) > 5;
      ```
    - This query selects departments with an average salary greater than $50,000 and more than 5 employees

- The HAVING clause allows you to filter groups based on aggregated values calculated from the filtered.
    - Example:

      -
        ```
        SELECT department, AVG(salary)
        FROM employees
        WHERE hire_year > 2010
        GROUP BY department
        HAVING AVG(salary) > 50000;
        ```
      - This query selects departments with an average salary greater than $50,000 among employees hired after 2010

Filtering grouped data using aggregate conditions

- Similar to the GROUP BY clause, you can also include an ORDER BY clause after the HAVING clause to order the results
  - Example:

    -
      ```
      SELECT department, AVG(salary)
      FROM employees
      GROUP BY department
      HAVING AVG(salary) > 50000
      ORDER BY AVG(salary) DESC;
      ```
    - This query selects departments with an average salary greater than $50,000 and orders the results by the average salary in descending order.

Lesson 5: Advanced Aggregation Techniques

- Nested queries for aggregation
- Using subqueries in aggregate functions
- Introduction to window functions for advanced aggregation

Nested queries for aggregation

- These nested queries allow for more complex analyses and can be particularly useful when the desired aggregation cannot be achieved with a single query.
- Nested queries for aggregation provide SQL users with a flexible and powerful way to perform complex analyses and calculations on their data.

Using subqueries in aggregate functions

- Subqueries can be used within aggregate functions to perform more complex calculations based on the results of those subqueries. This technique allows for dynamic and conditional aggregation, enabling SQL users to derive insights from their data in a more sophisticated manner.

- Examples:

```
▪ SELECT department,
   AVG(salary) AS avg_salary_top_10
  FROM employees
  WHERE salary > (SELECT PERCENTILE_CONT(0.9) WITHIN GROUP (ORDER BY salary)
FROM employees)
  GROUP BY department;
```

Here, a subquery is used within the WHERE clause to filter employees whose salary is greater than the 90th percentile of salaries in the entire dataset. Then, the AVG function calculates the average salary for each department based on this filtered set of employees.

Using subqueries in aggregate functions

- **Using Subquery in Aggregate Function Parameters:**

  -
    ```
    SELECT department,
        AVG(
          CASE
            WHEN salary > (SELECT AVG(salary) FROM employees WHERE department = e.department)
            THEN salary
            ELSE NULL
        END
        ) AS avg_above_department_avg
      FROM employees e
      GROUP BY department;
    ```
  - In this example, a subquery is used within a CASE statement inside the AVG function. It calculates the average salary for each department but only considers salaries that are greater than the average salary of the respective department.

Using subqueries in aggregate functions

- **Using Subquery Result as an Argument in Aggregate Function**:

```
SELECT department,
 AVG(salary * 1.1) AS avg_salary_with_bonus
FROM employees
GROUP BY department;
```

  - In this example, a subquery is not explicitly used, but the calculation within the AVG function is effectively an aggregate operation based on the result of the subquery `salary * 1.1`. This calculates the average salary with a 10% bonus for each department.

Introduction to window functions for advanced aggregation

- Window functions in SQL are advanced aggregation techniques that allow for calculations across a set of rows related to the current row within the result set.
- **Syntax**:
  - Window functions are typically used in conjunction with the OVER clause, which defines the window or subset of rows over which the function operates

```
– SELECT column1, column2,
   window_function() OVER (PARTITION BY partition_column ORDER BY order_column)
  FROM table_name;
```

Introduction to window functions for advanced aggregation

- **PARTITION BY**: The PARTITION BY clause divides the result set into partitions to which the window function is applied independently. It allows you to perform calculations on distinct subsets of data.
- **ORDER BY**: The ORDER BY clause specifies the ordering of rows within each partition. It determines the logical order of the rows used by the window function.

Introduction to window functions for advanced aggregation

- **Common Window Functions**:
  - **ROW_NUMBER()**: Assigns a unique integer to each row within a partition.
  - **RANK()**: Assigns a rank to each row within a partition, with gaps in the ranking when there are ties.
  - **DENSE_RANK()**: Similar to RANK() but assigns consecutive ranks without gaps.
  - **NTILE()**: Divides the rows within a partition into a specified number of equally sized groups.
  - **LEAD() and LAG()**: Accesses data from subsequent or preceding rows within the partition, respectively.
  - **SUM(), AVG(), MIN(), MAX()**: These aggregate functions can also be used as window functions to compute aggregates over a window of rows.

Introduction to window functions for advanced aggregation

- **Example**:

  -
    ```
    SELECT department,
      employee_name,
      salary,
      RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS salary_rank
    FROM employees;
    ```
  - This query calculates the rank of each employee's salary within their department, ordering employees by salary in descending order within each department.

Activities

Let's Practise Together

Activities

- Restore and using AdventureWorks2019 Dataset
  1. **Calculate Total Sales Revenue by Year**
  2. **Find the Most Popular Products by Quantity Sold**
  3. **Identify the Top Selling Product Categories**
  4. **Find Customers with High Total Sales**
  5. **Identify Customers who Have Not Made Purchase**
  6. **List Products with Quantity Sold Greater Than Average**
  7. **Calculate Running Total Sales Amount**
  8. **Calculate Moving Average Sales Amount**
  9. **Calculate Percentile Rank of Sales Amount**
  10. **Find Employees with No Sales Orders**

Assignments

Good Luck

Assignments

1. Write SQL queries to calculate total sales, average order value, and highest-selling product from a sales database.
2. Analyze a customer database and identify the top 5% of customers based on their total purchase amount.
3. Write a query to find the month with the highest revenue for a given year from an orders table
4. Identify the top 10 products with the highest total sales revenue, including the product name, product category, and total revenue

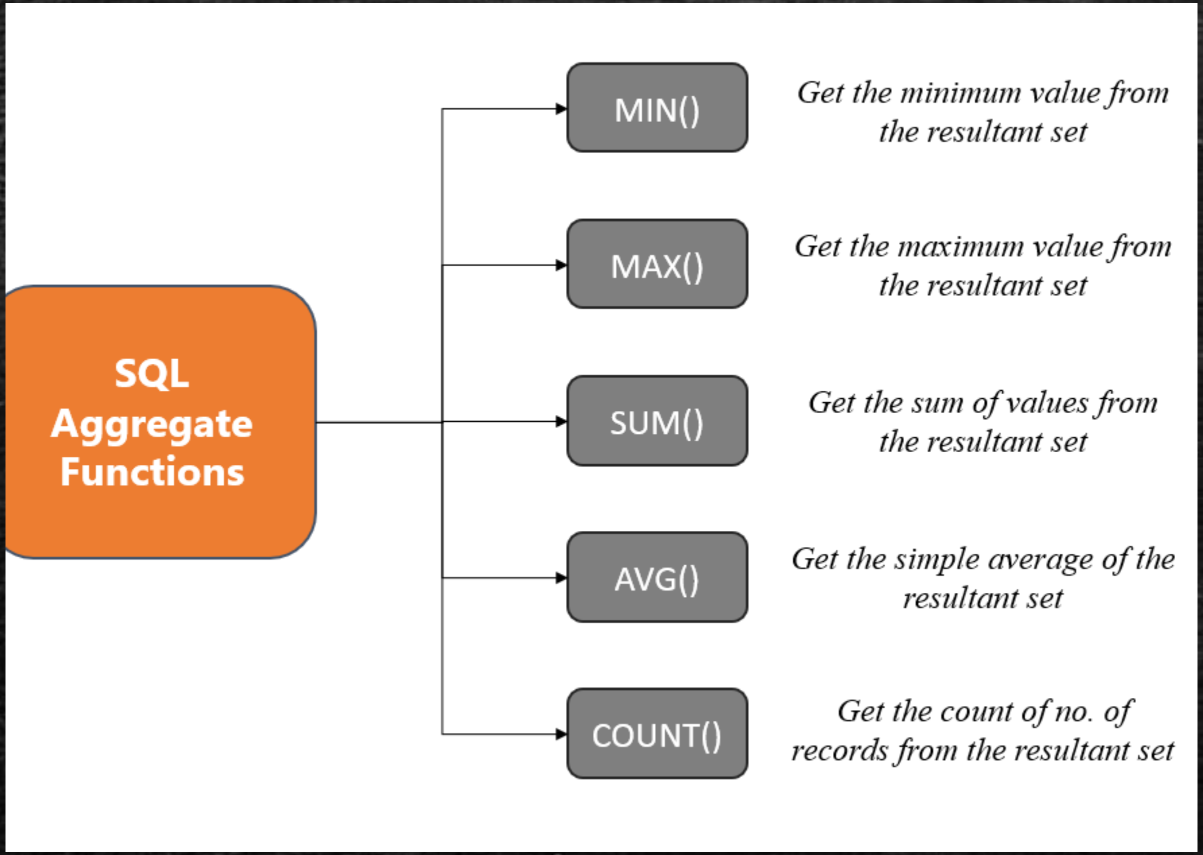
Assignments

5- Determine the average number of days it takes for orders to be shipped after they are placed, categorized by shipping method

6-How does sales performance vary across the product categories?

7-Which region generated the highest revenue?

Thankyou


SQL Aggregate Functions
MIN()
Get the minimum value from the resultant set
MAX()
Get the maximum value from the resultant set
SUM()
Get the sum of values from the resultant set
AVG()
Get the simple average of the resultant set
COUNT()
Get the count of no. of records from the resultant set

By Marwa Ashraf