07 Aggregate Functions - Nikhil X KirkYagami

Introduction to Aggregate Functions

Aggregate functions in MySQL perform calculations on a set of values and return a single value. These functions are essential for data analysis, reporting, and summarizing information in databases.
They allow you to answer questions like "What is the average salary of our programmers?" or "How many software products did we sell in total?"

Common Aggregate Functions

1. COUNT()

The `COUNT()` function returns the number of rows that match a specified criterion.

Basic Syntax:

```
COUNT(expression)
```

Real-world Examples:

Example 1: Count total programmers

```
SELECT COUNT(*) AS TotalProgrammers
FROM programmers;
```

Example 2: Count programmers by gender

```
SELECT GENDER, COUNT(*) AS Count
FROM programmers
GROUP BY GENDER;
```

Example 3: Count programmers with Python as primary language

```
SELECT COUNT(*) AS PythonProgrammers
FROM programmers
WHERE Primary_Language = 'Python';
```

Example 4: Find the number of different programming languages used as primary languages

```
SELECT COUNT(DISTINCT Primary_Language) AS DistinctLanguages
FROM programmers;
```

2. SUM()

The `SUM()` function calculates the total sum of a numeric column.

Basic Syntax:

```
SUM(expression)
```

Real-world Examples:

Example 1: Calculate total salary budget

```
SELECT SUM(Salary) AS TotalSalaryBudget
FROM programmers;
```

Example 2: Calculate total sales revenue

```
SELECT SUM(Software_Cost * Sold) AS TotalRevenue
FROM software;
```

Example 3: Calculate total development costs by primary language

```
SELECT p.Primary_Language,
SUM(s.Development_Cost) AS TotalDevelopmentCost
FROM programmers p
JOIN software s ON p.Programmer_Name = s.Programmer_Name
GROUP BY p.Primary_Language;
```

Example 4: Calculate profit from each software (revenue - development cost)

```
SELECT Programmer_Name, Software_Name,
        SUM(Software_Cost * Sold) as revenue,
        Development_Cost,
       SUM(Software_Cost * Sold) - Development_Cost AS Profit
FROM software
GROUP BY Programmer_Name, Software_Name, Development_Cost;
```

3. AVG()

The `AVG()` function calculates the average value of a numeric column.

Basic Syntax:

```
AVG(expression)
```

Real-world Examples:

Example 1: Calculate average programmer salary

```
SELECT AVG(Salary) AS AverageSalary
FROM programmers;
```

Example 2: Calculate average salary by primary programming language

```
SELECT Primary_Language, AVG(Salary) AS AverageSalary
FROM programmers
GROUP BY Primary_Language;
```

Example 3: Calculate average software development cost

```
SELECT AVG(Development_Cost) AS AverageDevelopmentCost
FROM software;
```

Example 4: Calculate average software units sold by language

```
SELECT Developed_In, AVG(Sold) AS AverageUnitsSold
FROM software
GROUP BY Developed_In;
```

4. MIN()

The `MIN()` function returns the smallest value in a selected column.

Basic Syntax:

```
MIN(expression)
```

Real-world Examples:

Example 1: Find lowest programmer salary

```
SELECT MIN(Salary) AS LowestSalary
FROM programmers;
```

Example 2: Find minimum course fee

```
SELECT MIN(Course_Fee) AS MinimumCourseFee
FROM studies;
```

Example 3: Find minimum software units sold by development language

```
SELECT Developed_In, MIN(Sold) AS MinimumUnitsSold
FROM software
GROUP BY Developed_In;
```

Example 4: Find youngest programmer (most recent date of birth)

```
SELECT MAX(DOB) AS YoungestProgrammerDOB
FROM programmers;
```

5. MAX()

The `MAX()` function returns the largest value in a selected column.

Basic Syntax:

```
MAX(expression)
```

Real-world Examples:

Example 1: Find highest programmer salary

```sql
SELECT MAX(Salary) AS HighestSalary
FROM programmers;
```

Example 2: Find most expensive software

```sql
SELECT MAX(Software_Cost) AS MostExpensiveSoftware
FROM software;
```

Example 3: Find most sold software product

```sql
SELECT Programmer_Name, Software_Name, MAX(Sold) AS MostUnitsSold
FROM software
group by Programmer_Name, Software_Name
order by MAX(Sold)  desc
limit 1;
```

Example 4: Find oldest programmer (earliest date of birth)

```sql
SELECT MIN(DOB) AS OldestProgrammerDOB
FROM programmers;
```

Advanced Usage with GROUP BY

The `GROUP BY` clause groups rows that have the same values in specified columns. This is often used with aggregate functions to perform calculations on each group.

Basic Syntax:

```sql
SELECT column1, column2, aggregate_function(column3)
FROM table
GROUP BY column1, column2;
```

Real-world Examples:

Example 1: Count programmers by institute

```sql
SELECT Institute, COUNT(*) AS NumberOfProgrammers
FROM studies
GROUP BY Institute;
```

Example 2: Calculate total course fees by institute

```sql
SELECT Institute, SUM(Course_Fee) AS TotalCourseFees
FROM studies
GROUP BY Institute;
```

Example 3: Calculate average salary by gender and primary language

```sql
SELECT GENDER, Primary_Language, AVG(Salary) AS AverageSalary
FROM programmers
GROUP BY GENDER, Primary_Language;
```

Example 4: Calculate total revenue by development language

```
SELECT Developed_In, SUM(Software_Cost * Sold) AS TotalRevenue
FROM software
GROUP BY Developed_In
ORDER BY TotalRevenue DESC;
```

Using HAVING with Aggregate Functions

The `HAVING` clause is used to filter groups based on aggregate function results, similar to how `WHERE` filters individual rows.

Basic Syntax:

```
SELECT column1, aggregate_function(column2)
FROM table
GROUP BY column1
HAVING condition;
```

Real-world Examples:

Example 1: Find primary languages used by more than 2 programmers

```
SELECT Primary_Language, COUNT(*) AS NumberOfProgrammers
FROM programmers
GROUP BY Primary_Language
HAVING NumberOfProgrammers > 2;
```

Example 2: Find institutes with average course fee above $5000

```
SELECT Institute, AVG(Course_Fee) AS AverageCourseFee
FROM studies
GROUP BY Institute
HAVING AverageCourseFee > 5000;
```

Example 3: Find development languages with total revenue greater than $1 million

```
SELECT Developed_In, SUM(Software_Cost * Sold) AS TotalRevenue
FROM software
GROUP BY Developed_In
HAVING TotalRevenue > 1000000;
```

Example 4: Find programmers who developed software with a combined profit over $500,000

```
SELECT p.Programmer_Name, SUM(s.Software_Cost * s.Sold - s.Development_Cost) AS TotalProfit
FROM programmers p
JOIN software s ON p.Programmer_Name = s.Programmer_Name
GROUP BY p.Programmer_Name
HAVING TotalProfit > 500000;
```

Combining Multiple Aggregate Functions

You can use multiple aggregate functions in a single query to get different statistics in one result set.

Real-world Examples:

Example 1: Calculate salary statistics for all programmers

```
SELECT
    COUNT(*) AS TotalProgrammers,
    MIN(Salary) AS MinimumSalary,
    MAX(Salary) AS MaximumSalary,
    AVG(Salary) AS AverageSalary,
    SUM(Salary) AS TotalSalaryBudget
FROM programmers;
```

Example 2: Calculate software statistics by development language

```
SELECT
    Developed_In,
    COUNT(*) AS NumberOfSoftware,
    AVG(Software_Cost) AS AverageCost,
    SUM(Sold) AS TotalSold,
    SUM(Software_Cost * Sold) AS TotalRevenue
FROM software
GROUP BY Developed_In;
```

Example 3: Calculate course statistics by institute

```
SELECT
    Institute,
    COUNT(*) AS NumberOfCourses,
    MIN(Course_Fee) AS MinCourseFee,
    MAX(Course_Fee) AS MaxCourseFee,
    AVG(Course_Fee) AS AvgCourseFee,
    SUM(Course_Fee) AS TotalCourseFees
FROM studies
GROUP BY Institute;
```

Example 4: Calculate comprehensive programmer statistics by gender

```
SELECT
    GENDER,
    COUNT(*) AS Count,
    MIN(Salary) AS MinSalary,
    MAX(Salary) AS MaxSalary,
    AVG(Salary) AS AvgSalary,
    SUM(Salary) AS TotalSalary,
    AVG(YEAR(CURRENT_DATE) - YEAR(DOB)) AS AvgAge
FROM programmers
GROUP BY GENDER;
```

Using Aggregate Functions with Subqueries

Aggregate functions can be used in subqueries to create more complex queries.

Real-world Examples:

Example 1: Find programmers with salary above average

```sql
SELECT Programmer_Name, Salary
FROM programmers
WHERE Salary > (SELECT AVG(Salary) FROM programmers);
```

Example 2: Find software with above-average sales

```sql
SELECT Software_Name, Sold
FROM software
WHERE Sold > (SELECT AVG(Sold) FROM software);
```

Example 3: Find programmers who developed software with the highest revenue

```sql
SELECT Programmer_Name
FROM software
WHERE Software_Cost * Sold = (
    SELECT MAX(Software_Cost * Sold)
    FROM software
);
```

Example 4: Find programmers whose course fee is above the average for their institute

```sql
SELECT s1.Programmer_Name, s1.Institute, s1.Course_Fee
FROM studies s1
WHERE s1.Course_Fee > (
    SELECT AVG(s2.Course_Fee)
    FROM studies s2
    WHERE s2.Institute = s1.Institute
);
```

Common Table Expressions (CTEs) with Aggregate Functions

Common Table Expressions (CTEs) provide a way to create temporary result sets that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement.

Basic Syntax:

```sql
WITH cte_name AS (
    SELECT column1, aggregate_function(column2)
    FROM table
    GROUP BY column1
)
SELECT * FROM cte_name;
```

Real-world Examples:

Example 1: Categorize programmers by salary tiers

```sql
WITH SalaryCategories AS (
    SELECT
        Programmer_Name,
        Salary,
        CASE
            WHEN Salary < 14000 THEN 'Low'
            WHEN Salary BETWEEN 14000 AND 16000 THEN 'Medium'
```

```
            ELSE 'High'
        END AS SalaryTier
    FROM programmers
)
SELECT
    SalaryTier,
    COUNT(*) AS ProgrammerCount,
    AVG(Salary) AS AverageSalary
FROM SalaryCategories
GROUP BY SalaryTier;
```

Example 2: Calculate software profitability

```
WITH ProfitAnalysis AS (
    SELECT
        Programmer_Name,
        Software_Name,
        Software_Cost * Sold AS Revenue,
        Development_Cost AS Cost,
        (Software_Cost * Sold) - Development_Cost AS Profit
    FROM software
)
SELECT
    Programmer_Name,
    SUM(Revenue) AS TotalRevenue,
    SUM(Cost) AS TotalCost,
    SUM(Profit) AS TotalProfit,
    ROUND(SUM(Profit) / SUM(Cost) * 100, 2) AS ROI_Percentage
FROM ProfitAnalysis
GROUP BY Programmer_Name
ORDER BY ROI_Percentage DESC;
```

Example 3: Compare programmer performance against institute averages

```
WITH InstituteBenchmarks AS (
    SELECT
        Institute,
        AVG(s.Course_Fee) AS AvgCourseFee,
        COUNT(*) AS StudentCount
    FROM studies s
    GROUP BY Institute
)
SELECT
    s.Programmer_Name,
    s.Institute,
    s.Course_Fee,
    ib.AvgCourseFee,
    ROUND((s.Course_Fee - ib.AvgCourseFee) / ib.AvgCourseFee * 100, 2) AS
PercentageDifference
FROM studies s
JOIN InstituteBenchmarks ib ON s.Institute = ib.Institute
ORDER BY s.Institute, PercentageDifference DESC;
```

Example 4: Calculate age demographics and experience levels

```sql
WITH Demographics AS (
    SELECT
        Programmer_Name,
        YEAR(CURRENT_DATE) - YEAR(DOB) AS Age,
        YEAR(CURRENT_DATE) - YEAR(DOJ) AS YearsOfExperience,
        CASE
            WHEN YEAR(CURRENT_DATE) - YEAR(DOJ) < 10 THEN 'Junior'
            WHEN YEAR(CURRENT_DATE) - YEAR(DOJ) BETWEEN 10 AND 20 THEN 'Mid-level'
            ELSE 'Senior'
        END AS ExperienceLevel
    FROM programmers
)
SELECT
    ExperienceLevel,
    COUNT(*) AS ProgrammerCount,
    MIN(Age) AS MinAge,
    MAX(Age) AS MaxAge,
    AVG(Age) AS AvgAge,
    AVG(YearsOfExperience) AS AvgExperience
FROM Demographics
GROUP BY ExperienceLevel;
```

Window Functions vs. Aggregate Functions

Window functions perform calculations across a set of table rows that are related to the current row, while aggregate functions operate on a group of rows and return a single value.

Example: Difference between aggregate and window function

Using aggregate function:

```sql
SELECT Primary_Language, AVG(Salary) AS AvgSalary
FROM programmers
GROUP BY Primary_Language;
```

Using window function:

```sql
SELECT
    Programmer_Name,
    Primary_Language,
    Salary,
    AVG(Salary) OVER (PARTITION BY Primary_Language) AS AvgLanguageSalary,
    Salary - AVG(Salary) OVER (PARTITION BY Primary_Language) AS SalaryDifference
FROM programmers;
```

The window function version shows each programmer's salary alongside the average for their language group, plus the difference between their salary and the group average.

Practical Business Questions Using Aggregate Functions

Example 1: Cost-benefit analysis of education

```sql
SELECT
    s.Institute,
    COUNT(*) AS ProgrammerCount,
```

```
    AVG(p.Salary) AS AvgSalary,
    AVG(s.Course_Fee) AS AvgCourseFee,
    AVG(p.Salary) / AVG(s.Course_Fee) AS SalaryToFeeRatio
FROM studies s
JOIN programmers p ON s.Programmer_Name = p.Programmer_Name
GROUP BY s.Institute
ORDER BY SalaryToFeeRatio DESC;
```

Example 2: Most profitable programming languages

```
SELECT
    p.Primary_Language,
    COUNT(DISTINCT p.Programmer_Name) AS ProgrammerCount,
    COUNT(DISTINCT s.Software_Name) AS SoftwareCount,
    SUM(s.Software_Cost * s.Sold) AS TotalRevenue,
    SUM(s.Development_Cost) AS TotalDevelopmentCost,
    SUM(s.Software_Cost * s.Sold - s.Development_Cost) AS TotalProfit,
    AVG(p.Salary) AS AvgProgrammerSalary
FROM programmers p
LEFT JOIN software s ON p.Programmer_Name = s.Programmer_Name
GROUP BY p.Primary_Language
ORDER BY TotalProfit DESC;
```

Example 3: Find the most efficient programmers (revenue per salary dollar)

```
SELECT
    p.Programmer_Name,
    p.Salary,
    SUM(s.Software_Cost * s.Sold) AS TotalRevenue,
    ROUND(SUM(s.Software_Cost * s.Sold) / p.Salary, 2) AS RevenuePerSalaryDollar
FROM programmers p
JOIN software s ON p.Programmer_Name = s.Programmer_Name
GROUP BY p.Programmer_Name, p.Salary
ORDER BY RevenuePerSalaryDollar DESC;
```

Example 4: Return on investment for each institute's graduates

```
SELECT
    st.Institute,
    COUNT(DISTINCT st.Programmer_Name) AS GraduateCount,
    SUM(st.Course_Fee) AS TotalInvestment,
    SUM(s.Software_Cost * s.Sold) AS TotalRevenue,
    ROUND(SUM(s.Software_Cost * s.Sold) / SUM(st.Course_Fee), 2) AS ROI
FROM studies st
JOIN programmers p ON st.Programmer_Name = p.Programmer_Name
JOIN software s ON p.Programmer_Name = s.Programmer_Name
GROUP BY st.Institute
ORDER BY ROI DESC;
```

Common Mistakes and Gotchas

1. **Mixing aggregate and non-aggregate columns without GROUP BY**

```sql
-- This will cause an error:
SELECT Primary_Language, AVG(Salary)
FROM programmers;

-- Correct way:
SELECT Primary_Language, AVG(Salary)
FROM programmers
GROUP BY Primary_Language;
```

2. **Using WHERE with aggregate functions**

```sql
-- This will cause an error:
SELECT Primary_Language, AVG(Salary)
FROM programmers
WHERE AVG(Salary) > 15000
GROUP BY Primary_Language;

-- Correct way (using HAVING):
SELECT Primary_Language, AVG(Salary) AS AvgSalary
FROM programmers
GROUP BY Primary_Language
HAVING AVG(Salary) > 15000;
```

3. **Misunderstanding NULL handling**

```sql
-- COUNT(*) counts all rows
SELECT COUNT(*) FROM programmers;

-- COUNT(column) counts non-NULL values in the column
SELECT COUNT(Secondary_Language) FROM programmers;

-- Add a NULL value to demonstrate
UPDATE programmers SET Secondary_Language = NULL WHERE Programmer_Name = 'Tony Stark';

-- Now COUNT(*) and COUNT(Secondary_Language) will return different values
SELECT COUNT(*) AS TotalRows, COUNT(Secondary_Language) AS NonNullLanguages
FROM programmers;
```

4. **Forgetting that aggregate functions exclude NULL values**

```sql
-- If some salary values are NULL:
SELECT AVG(Salary) FROM programmers; -- Ignores NULL salaries

-- To include NULL values (treating them as 0):
SELECT AVG(COALESCE(Salary, 0)) FROM programmers;
```

Summary

- **COUNT()**: Counts rows or non-NULL values
- **SUM()**: Calculates the total of numeric values
- **AVG()**: Calculates the average of numeric values
- **MIN()**: Finds the minimum value
- **MAX()**: Finds the maximum value

- **GROUP BY**: Groups rows with the same values for aggregate calculations
- **HAVING**: Filters groups based on aggregate function results
- **Window Functions**: Provide more detailed calculations while preserving row detail

These aggregate functions are essential tools for data analysis in MySQL, allowing you to extract meaningful insights from your database and answer complex business questions efficiently.