LESSON 3

# AGGREGATE FUNCTIONS AND GROUPING COUNT, SUM, AVG, MIN, MAX, AND GROUP BY

Ihor Liutak

# Content

**Short description**

Use aggregate functions (count, sum, avg, min, max) to perform data calculations. Introduce grouping data using the group by clause for summary reports. provide hands-on exercises to reinforce the concepts

**Kurzbeschreibung**

Verwendung von aggregate-funktionen (count, sum, avg, min, max) zum durchführen von daten berechnungen. Einführung in die gruppierung von daten mit der group by-klausel für zusammenfassende berichte. bieten sie praxisübungen zur verstärkung der konzepte

# Introduction to Aggregate Functions

Introduction to Aggregate Functions:

**What Are Aggregate Functions?**

Functions that perform calculations on a set of rows and return a single value

**Why Use Them?:**

Useful for summarizing data (e.g., totals, averages, counts).

**Overview of Functions:**

COUNT, SUM, AVG, MIN, MAX

## Using Aggregate Functions

COUNT:

```
SELECT COUNT(column_name) FROM table_name;
SELECT COUNT(student_id) AS total_students FROM students;
```

SUM:

```
SELECT SUM(column_name) FROM table_name;
SELECT SUM(salary) AS total_salary FROM employees;
```

AVG:

```
SELECT AVG(column_name) FROM table_name;
SELECT AVG(score) AS average_score FROM tests;
```

MIN and MAX:

```
SELECT MIN(column_name), MAX(column_name) FROM table_name;
SELECT MIN(price) AS lowest_price, MAX(price) AS highest_price
    FROM products;
```

## Grouping Data with GROUP BY

Purpose of GROUP BY:

To group rows that have the same values in specified columns

Syntax:

```
SELECT column_name, aggregate_function(column_name)
    FROM table_name
    GROUP BY column_name;
```

Grouping sales by product:

```
SELECT product_id, SUM(sales) AS total_sales
    FROM sales
    GROUP BY product_id;
```

Counting students by enrollment year::

```
SELECT enrollment_year, COUNT(student_id) AS total_students
    FROM students
    GROUP BY enrollment_year;
```

# Introduction to JOINs

What Are JOINs:

Combining rows from two or more tables based on a related column

Why Use JOINs:

To query data spread across multiple tables efficiently

Types of JOINs:

**`INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN`**

# Understanding INNER JOIN

Definition:

Retrieves records that have matching values in both tables

Syntax:

```
SELECT columns
    FROM table1
    INNER JOIN table2
    ON table1.column = table2.column;
```

Types of JOINs:

```
INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN
```

# Understanding INNER JOIN - Example

Query:

```
SELECT students.name, enrollments.course
    FROM students
    INNER JOIN enrollments
    ON students.student_id = enrollments.student_id;
```

Result:

```
name  | course
--------------
John  | Math
Alice | Science
```

students

```
student_id | name
-----------------
1          | John
2          | Alice
3          | Bob
```

enrollments

```
student_id | course
-------------------
1          | Math
2          | Science
```

## Practical Use Cases

Joining Two Tables, Retrieve orders along with customer details:

```
SELECT orders.order_id, customers.name
    FROM orders
    INNER JOIN customers
    ON orders.customer_id = customers.customer_id;
```

Using Aliases for Readability:

```
SELECT o.order_id, c.name
FROM orders AS o
INNER JOIN customers AS c
ON o.customer_id = c.customer_id;
```

Filtering with WHERE in JOINs:

```
SELECT s.name, e.course
FROM students s
INNER JOIN enrollments e
ON s.student_id = e.student_id
WHERE e.course = 'Math';
```

## Common Pitfalls and Best Practices

Avoid using columns in SELECT that are not in the GROUP BY or aggregate functions.

Missing ON Clause:
  Results in a cross join (Cartesian product) and often unintended large results.

Mismatched Column Data Types:
  Columns in ON clause must be of compatible types.

Duplicate Columns:
  Resolve conflicts by specifying table names or aliases (e.g., table1.column).

# Creating Tables

Basic Syntax:

```
CREATE TABLE [table_name] (
    [column_name] [data_type] [constraints],
    [column_name] [data_type] [constraints],
    PRIMARY KEY ([column_name])
);
```

*Example*

```
CREATE TABLE students (
    student_id INT AUTO_INCREMENT
        PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    enrollment_date DATE
);
```

Common Constraints: `NOT NULL, UNIQUE, DEFAULT, PRIMARY KEY, FOREIGN KEY`

Dropping a Table: `DROP TABLE [table_name];`

Viewing Table Structure: `DESCRIBE [table_name];`