Table Contents

Practice

1. Using SQL TOP

The TOP clause in SQL is used to limit the number of rows returned by a query. This is particularly useful when dealing with large datasets and you only want to retrieve a small subset of the results. It allows you to control how many rows are displayed based on the criteria provided in the query.

1A. Purpose of SQL TOP

The primary purposes of the TOP clause are:

- To **retrieve only the first few rows** from a result set.
- To **improve query performance** by limiting the number of rows processed.
- To **retrieve a sample** of the data, such as the top 10 highest-paid employees or the top 5 best-selling products.

1B. Syntax of SQL TOP

Here is the basic syntax of the TOP clause in SQL:

```
SELECT TOP (n) column1, column2, ...
FROM table_name
WHERE condition;
```

- n is the number of rows you want to retrieve.
- The column1, column2, ... part specifies which columns of data you want to select.
- The table_name specifies from which table you are retrieving the data.
- The WHERE clause (optional) specifies conditions that filter the data.

1C. Example Simple Use of SQL TOP

Explanation: This query selects the top 3 employees with the highest salaries. The ORDER BY Salary DESC clause ensures that the salaries are sorted in descending order, so the highest-paid employees appear first.

Imagine you have an Employees table that contains employee details. You want to retrieve the top 3 employees with the highest salaries.

```
SELECT TOP (3) EmployeeID, EmployeeName, Salary
FROM Employees
ORDER BY Salary DESC;
```

1D. Example SQL TOP with Percentage

Explanation: n represents the percentage of rows you want to retrieve

You can also use the TOP clause to retrieve a percentage of rows instead of a fixed number. Here's the syntax:

```
SELECT TOP (n) PERCENT column1, column2, ...
FROM table_name
WHERE condition;
```

1E. Example Using TOP with Percentage

Explanation: This query retrieves the top 10% of employees with the highest salaries. This is useful when the dataset size fluctuates, and you want to ensure that a proportional amount of data is retrieved.

Suppose you want to retrieve the top 10% of employees by salary.

```
SELECT TOP (10) PERCENT EmployeeID, EmployeeName, Salary
FROM Employees
ORDER BY Salary DESC;
```

1F. Alternative in Other Databases:

MySQL: MySQL uses the LIMIT clause instead of TOP:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column DESC
LIMIT n;
```

PostgreSQL and Oracle: PostgreSQL also uses LIMIT:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column DESC
LIMIT n;
```

Conclusion

- The TOP clause in SQL is useful for limiting the number of rows returned by a query.
- It is commonly used to improve performance, retrieve subsets of data, or get sample records.
- It works with both a specific number of rows or a percentage of rows and can be combined with other clauses like WHERE and ORDER BY.

2. Using SQL MIN() Function

The MIN() function in SQL is an aggregate function used to find the **smallest value** in a specified column. This is useful when you want to identify the minimum value within a set of values, such as the lowest price, the earliest date, or the smallest salary.

2A. Purpose of SQL MIN() Function

The main purposes of the MIN() function are:

- To **return the minimum value** from a specified column of numeric, date, or string data types.
- To **help in data analysis** by identifying the least or earliest value in a dataset.
- To **summarize data** by returning one specific value that represents the lower bound of a set.

2B. Syntax of SQL MIN() Function

- column_name: The name of the column from which you want to find the minimum value.
- table_name: The name of the table that contains the data.
- WHERE condition (optional): Any condition that filters the dataset before applying the MIN() function.

The basic syntax of the MIN() function is as follows:

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

2C. Example: Finding the Minimum Salary

Explanation: This query retrieves the smallest value in the Salary column from the Employees table. The result will be the lowest salary in the dataset.

Suppose you have an Employees table and you want to find the minimum salary among all employees.

```
SELECT MIN(Salary)
FROM Employees;
```

2D. Example: Minimum Salary in a Specific Department

Explanation: This query retrieves the smallest salary from the Employees table, but only for employees in the "Sales" department. The WHERE clause limits the rows considered to those in the specified department.

Let’s say you want to find the minimum salary in the "Sales" department.

```
SELECT MIN(Salary)
FROM Employees
WHERE Department = 'Sales';
```

2E. Example: Using MIN() with a Group By Clause

The MIN() function can be used in combination with the GROUP BY clause to find the minimum value for each group of data.

Explanation: This query groups the employees by their department and then finds the minimum salary within each group. The result will show the lowest salary for every department.

Suppose you want to find the minimum salary for each department.

```
SELECT Department, MIN(Salary)
FROM Employees
GROUP BY Department;
```

2F. Example: Using MIN() with a Date Column

Explanation: This query retrieves the earliest (minimum) date from the HireDate column, indicating the first employee who was hired.

The MIN() function can also be applied to date values.

For example, you may want to find the earliest hire date in the company.

```
SELECT MIN(HireDate)
FROM Employees;
```

2G. Example: Finding Minimum String Values

Explanation: This query retrieves the name that comes first in alphabetical order from the EmployeeName column.

In addition to numeric and date data types, the MIN() function can also be applied to **string** columns. When used on strings, the MIN() function will return the value that appears first in **lexicographical order** (alphabetical order).

```
SELECT MIN(EmployeeName)
FROM Employees;
```

2H. MIN() in Subqueries

Explanation: The subquery (SELECT MIN(Salary) FROM Employees) retrieves the minimum salary from the Employees table. The outer query then selects the details of the employee who has this minimum salary.

You can also use MIN() in subqueries to filter data in the outer query.

For example, if you want to find the details of the employee with the lowest salary:

```
SELECT EmployeeID, EmployeeName, Salary
FROM Employees
WHERE Salary = (SELECT MIN(Salary) FROM Employees);
```

2I. Combining MIN() with Other Aggregate Functions

Explanation: This query retrieves both the minimum and maximum salaries from the Employees table.

You can use the MIN() function alongside other aggregate functions like MAX(), AVG(), etc., to get multiple summary statistics.

For example, finding both the minimum and maximum salary:

```
SELECT MIN(Salary) AS MinSalary, MAX(Salary) AS MaxSalary
FROM Employees;
```

Conclusion

- The MIN() function is used to retrieve the smallest value from a specified column in SQL.
- It works with numeric, date, and string columns and can be combined with GROUP BY, WHERE, or other SQL clauses.
- You can use it to perform both simple queries, such as finding the smallest value, and more complex operations, like grouping data by category or finding minimum values in subqueries.

3. Using SQL MAX() Function

The MAX() function in SQL is an aggregate function that returns the **maximum value** from a specified column. It's used to find the largest value in a dataset, such as the highest salary, the latest date, or the most expensive product.

3A. Purpose of SQL MAX() Function

The main purposes of the MAX() function are:

- To **find the maximum value** in a column, either numeric, date, or string data types.
- To **summarize data** by retrieving the largest value in a set of records.
- To **identify key insights** from the dataset, such as finding the highest salary, the latest hire date, or the maximum price.

3B. Syntax of SQL MAX() Function

The basic syntax of the MAX() function is as follows:

- column_name: The column from which you want to retrieve the maximum value.
- table_name: The name of the table containing the data.
- WHERE condition: (optional) Used to filter rows before calculating the maximum value.

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

3C. Example: Finding the Maximum Salary

Explanation: This query retrieves the largest value in the Salary column from the Employees table. The result will be the highest salary among all employees.

Suppose you have an Employees table, and you want to find the highest salary among all employees.

```
SELECT MAX(Salary)
FROM Employees;
```

3D. Example: Maximum Salary in a Specific Department

Explanation: This query retrieves the maximum salary from the Salary column but only for employees in the "Sales" department. The WHERE Department = 'Sales' clause ensures that only the rows from the "Sales" department are considered.

If you want to find the maximum salary in a specific department, such as "Sales," you can add a WHERE clause to filter by department.

```
SELECT MAX(Salary)
FROM Employees
WHERE Department = 'Sales';
```

3E. Example: Using MAX() with GROUP BY

The MAX() function can also be used with the GROUP BY clause to find the maximum value for each group in a dataset.

Explanation: This query groups the data by department and finds the maximum salary in each group. The result will show the highest salary for each department.

For example, to find the maximum salary for each department:

```
SELECT Department, MAX(Salary)
FROM Employees
GROUP BY Department;
```

3F. Using MAX() with a Date Column

The MAX() function can be used to find the latest date in a dataset. Suppose you want to find the most recent hire date in the company:

Explanation: This query retrieves the latest date from the HireDate column, indicating the last employee to be hired.

```
SELECT MAX(HireDate)
FROM Employees;
```

3G. Finding Maximum String Values

The MAX() function can also be used with string data. When applied to strings, it returns the value that comes last in **lexicographical order** (alphabetical order).

Explanation: This query retrieves the employee name that comes last alphabetically in the EmployeeName column.

For example, to find the employee whose name comes last alphabetically:

```
SELECT MAX(EmployeeName)
FROM Employees;
```

3H. Using MAX() in Subqueries

The MAX() function can be used in a subquery to filter data in the outer query.

Explanation: The subquery (SELECT MAX(Salary) FROM Employees) retrieves the highest salary. The outer query then selects the details of the employee with that salary.

For example, to find the employee with the highest salary:

```
SELECT EmployeeID, EmployeeName, Salary
FROM Employees
WHERE Salary = (SELECT MAX(Salary) FROM Employees);
```

3I. Example: Combining MAX() with Other Aggregate Functions

You can use the MAX() function together with other aggregate functions such as MIN(), AVG(), and SUM() to get a comprehensive summary of the data.

Explanation: This query retrieves both the minimum and maximum salaries from the Employees table, providing insight into the salary range within the company.

For example, to find both the highest and lowest salaries:

```
SELECT MIN(Salary) AS MinSalary, MAX(Salary) AS MaxSalary
FROM Employees;
```

Conclusion

- The MAX() function is used to retrieve the largest value from a specified column.
- It can be applied to numeric, date, or string data and works well with GROUP BY, WHERE, and subqueries.
- The MAX() function helps in summarizing data and identifying key insights, such as the highest salary, the latest hire date, or the most expensive product.

4. Using SQL SUM() Function

The SUM() function in SQL is an aggregate function that calculates the **total sum** of a numeric column. It's commonly used to add up values such as the total sales, total salaries, or total quantity of products sold.

4A. Purpose of SQL SUM() Function

The primary purposes of the SUM() function are:

- To **calculate the total** value of a numeric column, like total revenue, total costs, or total quantity.
- To **summarize data** by adding up values across multiple rows.
- To **analyze numerical data** in databases by computing overall metrics, like the sum of orders, profits, or scores.

4B. Syntax of SQL SUM() Function

- column_name: The name of the column you want to sum.
- table_name: The name of the table containing the data.
- WHERE condition (optional): Used to filter rows before summing the values.

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

4C. Example: Calculating the Total Salary

Explanation: This query calculates the total sum of the Salary column, effectively summing up all employee salaries in the Employees table.

Suppose you have an Employees table, and you want to calculate the total salary paid to all employees.

```
SELECT SUM(Salary)
FROM Employees;
```

4D. Using SUM() with a Condition

You can also apply a WHERE clause to filter the rows before calculating the sum.

Explanation: This query sums up the salaries of only those employees who work in the "Sales" department. The WHERE Department = 'Sales' condition ensures that only these employees are considered.

For instance, to calculate the total salary paid to employees in the "Sales" department:

```sql
SELECT SUM(Salary)
FROM Employees
WHERE Department = 'Sales';
```

4E. Using SUM() with GROUP BY

The SUM() function can be combined with the GROUP BY clause to calculate the total for each group of data.

Explanation: This query groups the employees by department and calculates the total salary for each department. The result will show the total salaries for each department in the company.

For example, to calculate the total salary for each department:

```sql
SELECT Department, SUM(Salary)
FROM Employees
GROUP BY Department;
```

4F. Summing Product Quantities

Explanation: This query calculates the total quantity of products sold by summing the values in the Quantity column.

Suppose you have a Sales table with columns ProductID, Quantity, and Price. To calculate the total quantity of all products sold:

```sql
SELECT SUM(Quantity)
FROM Sales;
```

4G. Calculating Total Revenue

Explanation: This query multiplies the Quantity by the Price for each sale to get the revenue from each sale, and then sums up the total revenue. The alias AS TotalRevenue is used to give the result a descriptive column name.

To calculate the total revenue from all sales, you can multiply the Price by the Quantity and sum the result:

```
SELECT SUM(Quantity * Price) AS TotalRevenue
FROM Sales;
```

4H. Using SUM() in a Subquery

The SUM() function can be used in a subquery to filter data in the outer query.

Explanation: This query first groups employees by department and calculates the total salary for each department. The HAVING clause then filters the results to include only those departments where the total salary exceeds $100,000.

For example, to find the departments where the total salary exceeds $100,000:

```
SELECT Department
FROM Employees
GROUP BY Department
HAVING SUM(Salary) > 100000;
```

4I. Using SUM() with DISTINCT

The SUM() function can also be used with the DISTINCT keyword to sum only unique values.

Explanation: This query sums the unique salary values, meaning that if two employees have the same salary, it will only be counted once in the sum.

For example, to sum unique salary values:

```
SELECT SUM(DISTINCT Salary)
FROM Employees;
```

4J. Summing Data Over a Date Range

Explanation: This query sums the TotalAmount column for sales that occurred between January 1, 2024, and January 31, 2024.

Suppose you want to calculate the total sales for the month of January 2024. You can add a WHERE clause to filter the date range:

```
SELECT SUM(TotalAmount)
FROM Sales
WHERE SaleDate BETWEEN '2024-01-01' AND '2024-01-31';
```

4K. Combining SUM() with Other Aggregate Functions

You can use the SUM() function with other aggregate functions like COUNT(), AVG(), and MAX() to get comprehensive data insights.

Explanation: This query calculates the total salary, the average salary, and the number of employees in the Employees table.

For example, to calculate the total salary, the average salary, and the number of employees:

```
SELECT SUM(Salary) AS TotalSalary, AVG(Salary) AS AvgSalary, COUNT(EmployeeID) AS NumEmployees
FROM Employees;
```

Conclusion

- The SUM() function is used to calculate the total sum of a numeric column in SQL.
- It can be combined with conditions, grouping, and other aggregate functions to analyze and summarize data.
- It's useful for calculating totals such as revenue, salary, quantity, and more.

5. Using SQL AVG() Function

The AVG() function in SQL is an aggregate function that calculates the **average value** of a numeric column. It adds up all the values and then divides the sum by the number of records. This function is often used to determine things like the average salary, average product price, or the average score in a dataset.

5A. Purpose of SQL AVG() Function

The main purposes of the AVG() function are:

- To **calculate the mean value** of a numeric column.
- To **summarize data** by determining the average, like the average price, average score, or average salary.
- To **analyze trends** in datasets, such as identifying the average quantity sold over time.

5B. Syntax of SQL AVG() Function

- column_name: The name of the column you want to calculate the average for.
- table_name: The name of the table containing the data.
- WHERE condition: (optional) Used to filter rows before calculating the average.

The basic syntax for using the AVG() function is as follows:

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

5C. Example: Calculating the Average Salary

Explanation: This query calculates the average value of the Salary column in the Employees table. It adds up all the salaries and divides by the number of employees to get the average salary.

Suppose you have an Employees table, and you want to calculate the average salary of all employees.

```
SELECT AVG(Salary)
FROM Employees;
```

5D. Example: Average Salary in a Specific Department

Explanation: This query calculates the average salary of employees working in the "Sales" department. The WHERE Department = 'Sales' condition filters the dataset to only include employees in that department before computing the average.

You can use the AVG() function with a WHERE clause to calculate the average salary for a specific department, such as "Sales":

```
SELECT AVG(Salary)
FROM Employees
WHERE Department = 'Sales';
```

5E. Using AVG() with GROUP BY

The AVG() function can be combined with the GROUP BY clause to calculate the average value for each group of data.

Explanation: This query groups the employees by department and calculates the average salary for each group. The result will show the average salary for every department in the company.

For example, to find the average salary for each department:

```
SELECT Department, AVG(Salary)
FROM Employees
GROUP BY Department;
```

5F. Example:Calculating the Average Price of Products

Explanation: This query calculates the average value of the Price column in the Products table. The alias AS AvgPrice is used to give the result a more descriptive column name.

Suppose you have a Products table with columns ProductID, ProductName, and Price, and you want to calculate the average price of all products:

```
SELECT AVG(Price) AS AvgPrice
FROM Products;
```

5G. Example:Using AVG() in a Subquery

You can also use AVG() in subqueries to filter data in the outer query.

Explanation: The subquery (SELECT AVG(Salary) FROM Employees) calculates the average salary of all employees. The outer query then selects the details of employees who earn more than this average.

For instance, if you want to find all employees who earn more than the average salary:

```
SELECT EmployeeID, EmployeeName, Salary
FROM Employees
WHERE Salary > (SELECT AVG(Salary) FROM Employees);
```

5H. Example:Using AVG() with DISTINCT

You can use the DISTINCT keyword with the AVG() function to calculate the average of unique values.

Explanation: This query calculates the average salary by considering only unique salary values. If two employees have the same salary, it will only be counted once in the calculation.

For example, to calculate the average of distinct salaries:

```
SELECT AVG(DISTINCT Salary)
FROM Employees;
```

5I. Example:Calculating the Average Sales Amount in a Date Range

Explanation: This query calculates the average value of the TotalAmount column for sales made between January 1, 2024, and January 31, 2024. The result represents the average sales amount for that month.

Suppose you want to calculate the average sales amount for the month of January 2024. You can use a WHERE clause to filter the date range:

```
SELECT AVG(TotalAmount) AS AvgSales
FROM Sales
WHERE SaleDate BETWEEN '2024-01-01' AND '2024-01-31';
```

5J. Combining AVG() with Other Aggregate Functions

You can combine the AVG() function with other aggregate functions such as SUM(), COUNT(), and MAX() to analyze data more comprehensively.

Explanation: This query calculates the total salary, the average salary, and the total number of employees in the Employees table. It provides multiple insights from the same data.

For instance, to calculate the total salary, the average salary, and the number of employees:

```
SELECT SUM(Salary) AS TotalSalary, AVG(Salary) AS AvgSalary, COUNT(EmployeeID) AS NumEmployees
FROM Employees;
```

Conclusion

- The AVG() function is used to calculate the average of a numeric column in SQL.
- It can be combined with other clauses like GROUP BY, WHERE, and DISTINCT to calculate averages for specific subsets of data.
- It's useful for summarizing and analyzing numerical data, such as average prices, salaries, scores, and quantities.

4. Using SQL COUNT() Function

The COUNT() function in SQL is an aggregate function that returns the **number of rows** in a result set. It's used to count how many rows match a given condition or how many rows are present in a table.

4A. Purpose of SQL COUNT() Function

The primary purposes of the COUNT() function are:

- To **count the number of rows** in a table or in a subset of data.
- To **determine the number of non-null values** in a specific column.
- To **aggregate data** by counting occurrences, records, or matches in queries.

4B. Syntax of SQL COUNT() Function

- column_name: The column you want to count values from.
- table_name: The name of the table containing the data.
- WHERE condition: (optional) Used to filter the rows before counting them.

4C. Types of COUNT() Usage:

- **COUNT(*)**: Counts all rows in a table, including those with NULL values.
- **COUNT(column_name)**: Counts only non-NULL values in the specified column.
- **COUNT(DISTINCT column_name)**: Counts unique non-NULL values in the specified column.

4D. Example: Counting All Rows in a Table (COUNT(*))

Explanation: This query returns the total number of rows in the Employees table, including rows with NULL values in any column.

If you want to count the total number of rows in a table, use COUNT(*). This includes rows with NULL values:

```
SELECT COUNT(*)
FROM Employees;
```

4E. Example: Counting Non-NULL Values in a Column (COUNT(column_name))

Explanation: This query counts the number of rows where the Salary column is not NULL. If an employee does not have a salary assigned, that row will not be counted.

If you want to count how many non-NULL values exist in a particular column, you can use COUNT(column_name).

```sql
SELECT COUNT(Salary)
FROM Employees;
```

4F. Counting Rows Based on a Condition

Explanation: This query counts the number of employees who work in the "Sales" department. The WHERE clause filters the dataset, and only the rows where Department = 'Sales' are counted.

You can use the COUNT() function with a WHERE clause to count rows that satisfy a specific condition. For example, to count how many employees work in the "Sales" department:

```sql
SELECT COUNT(*)
FROM Employees
WHERE Department = 'Sales';
```

4G. Counting Unique Values (COUNT(DISTINCT column_name))

To count unique values in a column, use COUNT(DISTINCT column_name). This excludes duplicates from the count.

Explanation: This query counts the number of unique job titles in the Employees table, excluding any duplicate job titles.

For example, to count how many unique job titles there are in the company:

```sql
SELECT COUNT(DISTINCT JobTitle)
FROM Employees;
```

4H. Counting Rows with Multiple Conditions

Explanation: This query counts the number of employees who work in the "Sales" department and have a salary greater than $50,000. Both conditions must be true for the rows to be counted.

You can combine COUNT() with multiple conditions in the WHERE clause.

For instance, to count the number of employees in the "Sales" department who earn more than $50,000:

```
SELECT COUNT(*)
FROM Employees
WHERE Department = 'Sales' AND Salary > 50000;
```

4I. Using COUNT() with GROUP BY

Explanation: This query groups employees by department and counts the number of employees in each group. The result will display the department and the number of employees in each department.

You can use COUNT() with the GROUP BY clause to count rows in each group.

For instance, to count the number of employees in each department:

```
SELECT Department, COUNT(*)
FROM Employees
GROUP BY Department;
```

4J. Combining COUNT() with Other Aggregate Functions

You can combine COUNT() with other aggregate functions like SUM() and AVG() to analyze data comprehensively.

Explanation: This query counts the number of employees in each department and calculates the average salary for each department. The result will show the department, number of employees, and the average salary.

For example, to count the number of employees and the average salary in each department:

```
SELECT Department, COUNT(EmployeeID) AS NumEmployees, AVG(Salary) AS AvgSalary
FROM Employees
GROUP BY Department;
```

4K. Counting Null and Non-Null Values in a Column

Explanation: The first query counts how many employees have a salary, while the second query calculates the number of employees without a salary by subtracting the count of non-NULL values from the total number of rows.

You can count how many rows contain NULL or non-NULL values in a column.

For example, to count how many employees have and do not have salaries:

```
-- Count rows where Salary is not NULL
SELECT COUNT(Salary) AS NumWithSalary
FROM Employees;

-- Count rows where Salary is NULL
SELECT COUNT(*) - COUNT(Salary) AS NumWithoutSalary
FROM Employees;
```

Conclusion

- The COUNT() function is used to count the number of rows or values in a SQL table.
- It can be used to count all rows, non-NULL values, or unique values, and it works well with conditions (WHERE), grouping (GROUP BY), and subqueries.
- It's a versatile tool for aggregating data and gaining insights from datasets, such as counting employees, orders, unique values, or records that match specific criteria.

5. SQL Arithmetic Operators (Using with UPDATE)

In SQL, **arithmetic operators** are used to perform mathematical operations on numeric data. These operators can be used in various SQL queries like SELECT or UPDATE to modify data in a table. When used with the UPDATE statement, arithmetic operators allow you to modify the values of numeric columns in a database.

5A. Common SQL Arithmetic Operators

- **Addition (+)**: Adds two values.
- **Subtraction (-)**: Subtracts one value from another.
- **Multiplication (*)**: Multiplies two values.
- **Division (/)**: Divides one value by another.
- **Modulus (%)**: Returns the remainder when one value is divided by another.

5B. Purpose of SQL Arithmetic Operators in UPDATE

The main purpose of using arithmetic operators with the UPDATE statement is:

- To **update numeric columns** by performing mathematical operations.
- To **adjust existing values** by adding, subtracting, multiplying, dividing, or calculating the remainder.
- To **modify data dynamically** based on arithmetic calculations, without manually updating each record.

5C. Syntax for SQL Arithmetic Operators in UPDATE

- table_name: The name of the table.
- column_name: The column whose values you want to modify.
- <arithmetic_operator>: The arithmetic operation you want to perform (+, -, *, /, %).
- value: The value you want to use in the arithmetic operation.
- WHERE condition: (Optional) Used to filter which rows to update.

5D. Example: Increasing Employee Salaries by 10% (Multiplication)

Explanation: This query multiplies the existing value of the Salary column by 1.10 for all rows, increasing each employee's salary by 10%. The SET clause updates the Salary column with the new value.

Suppose you have an Employees table with a Salary column, and you want to increase all employees' salaries by 10%.

```
UPDATE Employees
SET Salary = Salary * 1.10;
```

5E. Adding a Bonus to All Employees (Addition)

Explanation: This query increases the value of the Salary column by 500 for all employees, effectively adding a $500 bonus to each employee's salary.

You want to give every employee in the company a $500 bonus.

```
UPDATE Employees
SET Salary = Salary + 500;
```

5F. Example: Reducing Inventory Stock After Sales (Subtraction)

Explanation: This query subtracts 20 from the Stock column for the product with ProductID = 101. This reduces the stock for that specific product by 20 units.

Assume you have a Products table with a Stock column, and you want to reduce the stock of a specific product (ProductID = 101) by 20 units after a sale.

```
UPDATE Products
SET Stock = Stock - 20
WHERE ProductID = 101;
```

5G. Example: Doubling the Price of a Specific Product (Multiplication)

Explanation: This query multiplies the current price by 2 for the product with ProductID = 105. The WHERE clause ensures that only the product with ProductID = 105 is updated.

Suppose you want to double the price of a product with ProductID = 105:

```
UPDATE Products
SET Price = Price * 2
WHERE ProductID = 105;
```

5H. Example: Dividing the Salary of a Group of Employees by 2 (Division)

Explanation: This query divides the salary by 2 for all employees whose role is "Intern", effectively reducing their salaries by half. The WHERE clause ensures that only employees in the "Intern" role are affected.

Suppose you want to halve the salaries of employees in the "Intern" role.

```
UPDATE Employees
SET Salary = Salary / 2
WHERE Role = 'Intern';
```

5I. Example: Using Modulus to Update Columns

Explanation: This query calculates the remainder of each employee's bonus when divided by 100 and stores it in the BonusRemainder column. For example, if an employee's bonus is 250, the BonusRemainder will be 50.

If you want to store the remainder when dividing an employee's bonus by 100 in a BonusRemainder column, you can use the modulus (%) operator:

```
UPDATE Employees
SET BonusRemainder = Bonus % 100;
```

5J. Example: Combining Multiple Arithmetic Operations in UPDATE

Explanation: This query first increases the salary by 10% (multiplying by 1.10), then subtracts $200 as a tax deduction, for the employee with EmployeeID = 3. The parentheses ensure that the salary is updated correctly before the subtraction.

You can combine multiple arithmetic operations in a single UPDATE query. For example, to increase an employee's salary by 10% and subtract a tax deduction of $200:

```
UPDATE Employees
SET Salary = (Salary * 1.10) - 200
WHERE EmployeeID = 3;
```

5K. Example: Conditional Update Using Arithmetic Operators

Explanation: This query multiplies the salary by 1.05 (increasing it by 5%) for employees whose current salary is less than $50,000. The WHERE clause ensures that only employees with a salary below $50,000 are updated.

You can use a conditional update to modify only certain rows based on the current value. For instance, to increase the salary by 5% only for employees who earn less than $50,000:

```
UPDATE Employees
SET Salary = Salary * 1.05
WHERE Salary < 50000;
```

Conclusion

- **Arithmetic operators** like +, -, *, /, and % allow you to perform calculations on numeric columns in SQL.
- When used with the UPDATE statement, they enable you to dynamically modify and adjust data in your tables.
- You can apply these operations on all rows, specific rows, or groups of rows based on conditions.
- Arithmetic operators are useful in scenarios like adjusting salaries, updating inventory levels, and calculating percentages or remainders.

Exercise

Question 1:

- **Table Name**: Publisher
- **Primary Key**: PubliserID

PubliserID	Name	Address
P01	Pearson	Bukit Jalil
P02	Deitel	Puchong
P03	Rainbow	Subang
P04	MacHill	Kuala Lumpur

Attributes	Data Type
PublisherID	nvarchar(50)
Name	nvarchar(50)
Address	nvarchar(50)

- **Table Name**: Book
- **Primary Key**: BookID
- **Foreign Key:** PublisherID

BookID	Name	Author	Price	PublishedDate	PublisherID
B01	Maths	J.Wenton	50.60	10 Jan 2016	P01
B02	Science	S.Hanson	100.00	12 Feb 2016	P01
B03	English	K.Vince	89.30	9 March 2016	P02
B03	Biology	K.Vince	150.80	24 April 2016	P03
B05	Computing	J.Denzin	NULL	NULL	NULL

Attributes	Data Type
BookID	nvarchar(50)
Name	nvarchar(50)
Author	nvarchar(50)
Price	decimal(10,2)
PublisherDate	date
PurblisherID	nvarchar(50)

a. Using MS SQL Server, create a new database Lab6

b. Write a query to create the tables given above

c. Write a query to add each row of data to the tables

Answer a:

CREATE DATABASE Lab6

Answer b:

CREATE TABLE Publisher(PublisherID nvarchar(50) PRIMARY KEY, Name nvarchar(50), Address nvarchar(50))	CREATE TABLE Book(BookID nvarchar(50) PRIMARY KEY, Name nvarchar(50), Author nvarchar(50), Price decimal(10,2), PublishedDate date, PublisherID nvarchar(50) FOREIGN KEY REFERENCES Publisher(PublisherID))

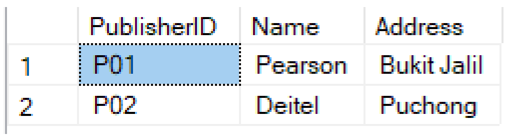
Answer c:

```
INSERT INTO Publisher
(PublisherID, Name, Address)
VALUES
('P01','Pearson','Bukit Jalil'),
('P02','Deitel','Puchong'),
('P03','Rainbow','Subang'),
('P04','Machill','Kuala Lumpur')
```

```
INSERT INTO Book
(BookID, Name,
Author,Price,PublishedDate,PublisherID)
Values
('B01', 'Math', 'J.Wenton', 50.60,'10 Jan
2016','P01'),
('B02', 'Science', 'S.Hanson', 100.00,'12 Feb
2016','P01'),
('B03', 'English', 'K.Vince', 89.30,'9 March
2016','P02'),
('B04', 'Biology', 'K.Vince', 150.80,'24 April
2016','P03'),
('B05', 'Computing', 'J.Denzin', NULL,NULL,NULL)
```

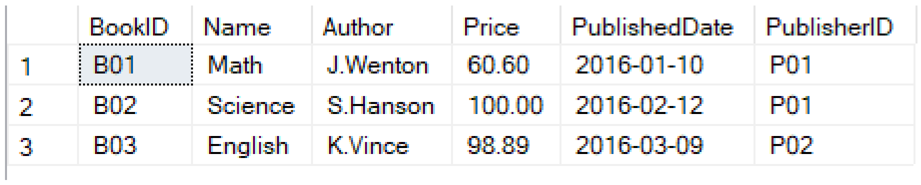
Question 2: Using SQL TOP

a. Select top 2 publishers

- SELECT TOP 2 *FROM Publisher;
- 

	PublisherID	Name	Address
1	P01	Pearson	Bukit Jalil
2	P02	Deitel	Puchong

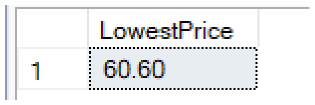
b. Select top 3 books where price is less than 130

- SELECT TOP 3 *FROM Book Where price < 130;
- 

	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B01	Math	J.Wenton	60.60	2016-01-10	P01
2	B02	Science	S.Hanson	100.00	2016-02-12	P01
3	B03	English	K.Vince	98.89	2016-03-09	P02

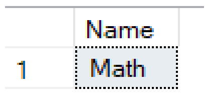
Question 3: Using SQL MIN() Function

a. Display the book which has the lowest price. (in the result table, give the column name as "LowestPrice")

- SELECT MIN(Price) AS LowestPrice From Book;
- 

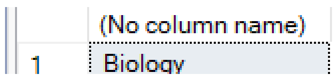
	LowestPrice
1	60.60

b. Display the book which has the lowest price, show the book name as well (in the result table, give the column name as "LowestPrice") *Hint: use a subquery

- SELECT Name From Book WHERE Price = (SELECT MIN(Price) AS LowestPrice From Book);
- 

	Name
1	Math

c. Try Select Min(Name) From Book (what do you get? Explain why)

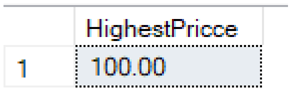
- SELECT MIN(Name) From Book;
- 

	(No column name)
1	Biology

- Because the MIN(Name) query returns the alphabetically smallest string from the Name column, and no column name is because the query did not assign a specific name for it.

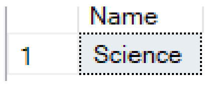
Question 4: Using SQL MAX() Function

a. Display the book which has the highest price. (in the result table, give the column name as "HighestPrice")

- **SELECT MAX(Price) AS HighestPricce From Book;**
- 

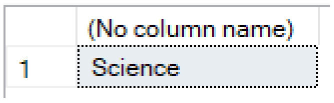
	HighestPricce
1	100.00

b. Display the book which has the highest price, show the book name as well (in the result table, give the column name as "HighestPrice") *Hint: use a subquery

- **SELECT Name From Book WHERE Price = (SELECT MAX(Price) AS HighestPricce From Book);**
- 

	Name
1	Science

c. Try Select Max(Name) From Book (what do you get? Explain why)

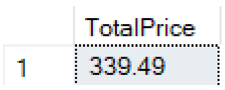
- **SELECT MAX(Name) From Book;**
- 

	(No column name)
1	Science

- **When you execute the query SELECT MAX(Name) FROM Book;, the result will be the alphabetically largest name from the Name column in the Book table.**

Question 5: Using SQL SUM() Function

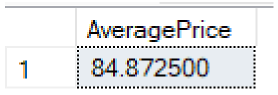
a. Get the total value of the "Price" column from the "Book" table. (in the result table, give the column name as "TotalPrice")

- **SELECT SUM(Price) AS TotalPrice From Book;**
- 

	TotalPrice
1	339.49

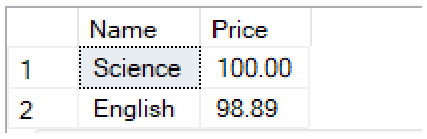
Question 6: Using SQL AVG() Function

a. Get the average value of the "Price" column from the "Book" table. (in the result table, give the column name as "AveragePrice")

- **SELECT AVG(Price) AS AveragePrice From Book;**
- 

	AveragePrice
1	84.872500

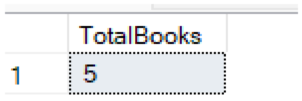
b. Display a list of books (showing name and price) which cost more than the average price. *Hint: use a subquery

- **SELECT Name,Price From Book WHERE Price > (SELECT AVG(Price) AS AveragePrice From Book);**
- 

	Name	Price
1	Science	100.00
2	English	98.89

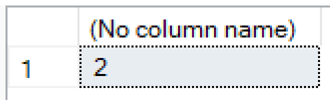
Question 7: Using SQL COUNT() Function

a. Find the total number of records in the Book table. (in the result table, give the column name as "TotalBooks")

- **SELECT COUNT(BookID) AS TotalBooks From Book;**
- **SELECT COUNT(Name) AS TotalBooks From Book;**
- 

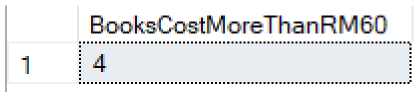
	TotalBooks
1	5

b. Find how many books written by 'K.Vince'

- **SELECT COUNT(BookID) FROM Book WHERE Author = 'K.Vince'**
- 

	(No column name)
1	2

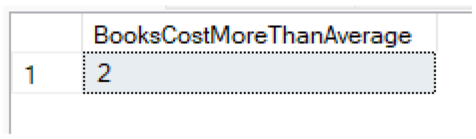
c. Find how many books cost more than 60 ringgit. (in the result table, give the column name as "BooksCostMoreThanRM60")

- **SELECT COUNT(Name) As BooksCostMoreThanRM60 FROM Book Where Price> 60;**
- 

	BooksCostMoreThanRM60
1	4

Question 8: Using SQL COUNT() Function & AVG() Function

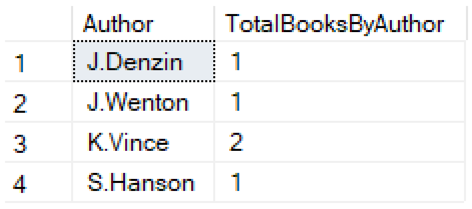
a. Find how many books cost more than the average price. (in the result table, give the column name as "BooksCostMoreThanAverage")

- SELECT COUNT(Name) AS BooksCostMoreThanAverage From Book Where Price > (SELECT AVG(Price)From Book);
- 

	BooksCostMoreThanAverage
1	2

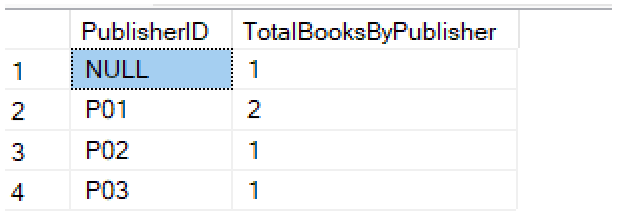
Question 9: Using SQL COUNT() Function & GROUP BY

a. Find the total number of records in the Book table group by each author. (in the result table, give the column name as "TotalBooksByAuthor")

- SELECT Author, COUNT(Name) AS TotalBooksByAuthor From Book GROUP BY Author;
- 

	Author	TotalBooksByAuthor
1	J.Denzin	1
2	J.Wenton	1
3	K.Vince	2
4	S.Hanson	1

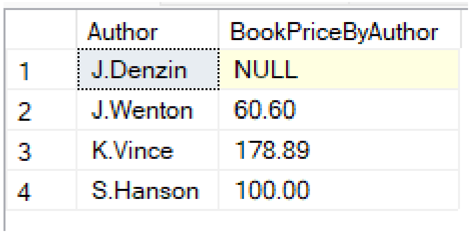
b. Find the total number of records in the Book table group by each PublisherID. (in the result table, give the column name as "TotalBooksByPublisher")

- SELECT PublisherID, COUNT(Name) AS TotalBooksByPublisher From Book GROUP BY PublisherID;
- 

	PublisherID	TotalBooksByPublisher
1	NULL	1
2	P01	2
3	P02	1
4	P03	1

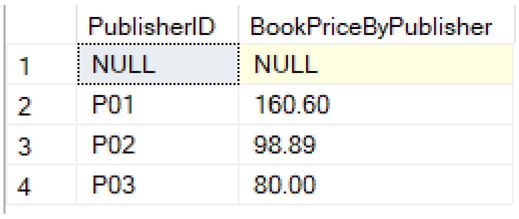
Question10: Using SQL SUM() Function & GROUP BY

a. Find the total Price of the books group by each author. (in the result table, give the column name as "BookPriceByAuthor")

- **SELECT Author, SUM(Price) AS BookPriceByAuthor FROM Book GROUP BY Author;**
- 

	Author	BookPriceByAuthor
1	J.Denzin	NULL
2	J.Wenton	60.60
3	K.Vince	178.89
4	S.Hanson	100.00

b. Find the total Price of the books group by each PublisherID. (in the result table, give the column name as "BookPriceByPublisher")

- **SELECT PublisherID, SUM(Price) AS BookPriceByPublisher FROM Book GROUP BY PublisherID;**
- 

	PublisherID	BookPriceByPublisher
1	NULL	NULL
2	P01	160.60
3	P02	98.89
4	P03	80.00

Question 11: SQL Arithmetic Operators (use UPDATE)

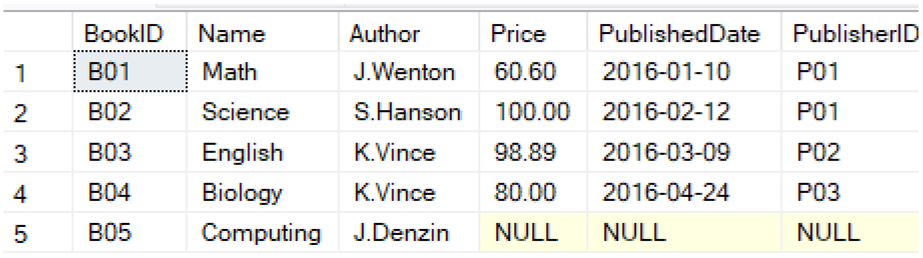
a. Change the price of Maths book, increase by 10

- **UPDATE Book**

  SET Price =50.60+ 10

  WHERE Name = 'Math';
- **UPDATE Book**

  SET Price += 10

  WHERE Name = 'Math';
- 

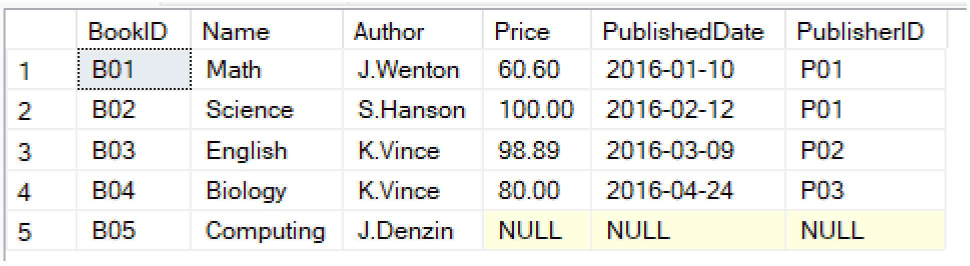
	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B01	Math	J.Wenton	60.60	2016-01-10	P01
2	B02	Science	S.Hanson	100.00	2016-02-12	P01
3	B03	English	K.Vince	98.89	2016-03-09	P02
4	B04	Biology	K.Vince	80.00	2016-04-24	P03
5	B05	Computing	J.Denzin	NULL	NULL	NULL

b. Change the price of Biology, decrease by 20

- UPDATE Book

  SET Price =100.00 -20

  WHERE Name = 'Biology';

- 

	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B01	Math	J.Wenton	60.60	2016-01-10	P01
2	B02	Science	S.Hanson	100.00	2016-02-12	P01
3	B03	English	K.Vince	98.89	2016-03-09	P02
4	B04	Biology	K.Vince	80.00	2016-04-24	P03
5	B05	Computing	J.Denzin	NULL	NULL	NULL

- UPDATE Book

  SET Price -= 20

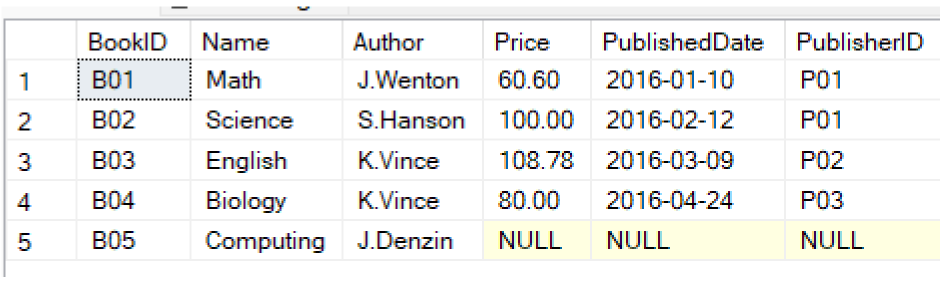
  WHERE Name = 'Biology';

c. Change the price of English book, increase by 10%

- UPDATE Book

  SET Price = Price + Price* 0.10

  WHERE Name = 'English';

- 

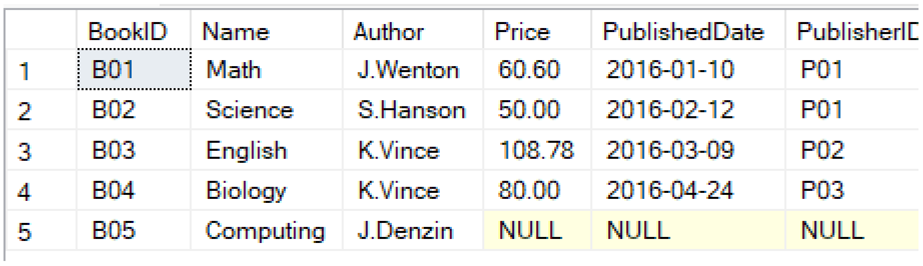
	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B01	Math	J.Wenton	60.60	2016-01-10	P01
2	B02	Science	S.Hanson	100.00	2016-02-12	P01
3	B03	English	K.Vince	108.78	2016-03-09	P02
4	B04	Biology	K.Vince	80.00	2016-04-24	P03
5	B05	Computing	J.Denzin	NULL	NULL	NULL

d. Change the price of Science book, divide by 2

- UPDATE Book

  SET Price = Price / 2

  WHERE Name = 'Science';

- 

	BookID	Name	Author	Price	PublishedDate	PublisherI
1	B01	Math	J.Wenton	60.60	2016-01-10	P01
2	B02	Science	S.Hanson	50.00	2016-02-12	P01
3	B03	English	K.Vince	108.78	2016-03-09	P02
4	B04	Biology	K.Vince	80.00	2016-04-24	P03
5	B05	Computing	J.Denzin	NULL	NULL	NULL