SQL Functions and Advanced Querying

Welcome to the SQL Manual for Aggregating Data, Group by, and Conditional Statements. In this session, we'll delve deeper into the techniques used to manipulate data within SQL databases effectively.

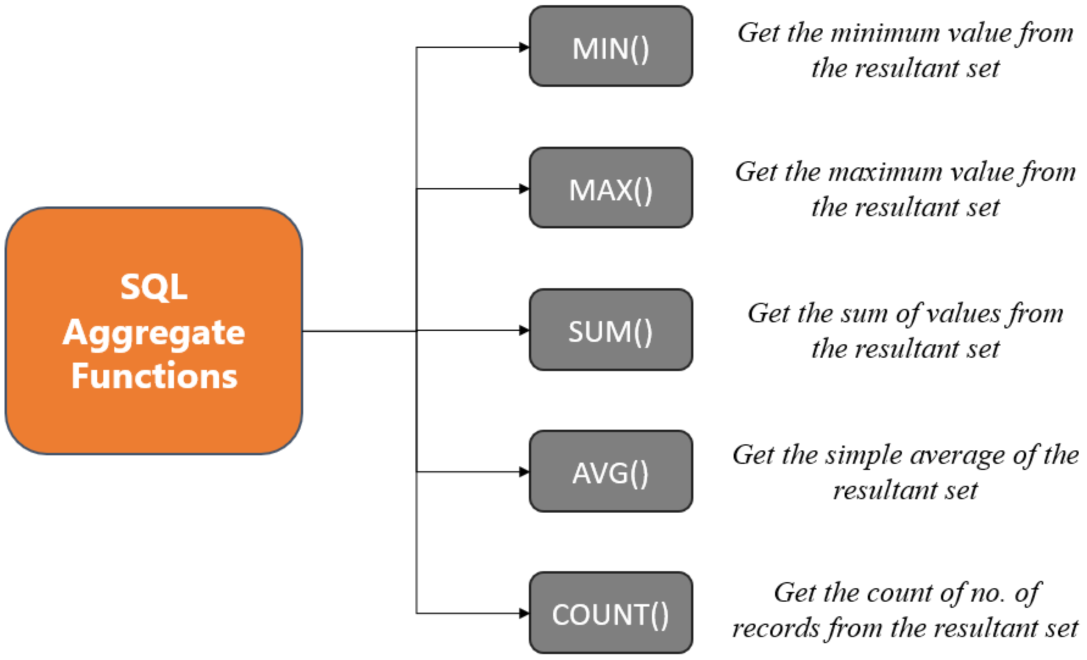
Objective:

• SQL Functions:

- Explore common aggregate functions:
  - Aggregate Functions: COUNT(), SUM(), AVG(), MIN(), MAX()
  - String Functions: CONCAT(), SUBSTRING(), LENGTH()
  - Date Functions: NOW(), DATE_ADD(), DATEDIFF()
  - Mathematical Functions: ROUND(), CEIL(), FLOOR()

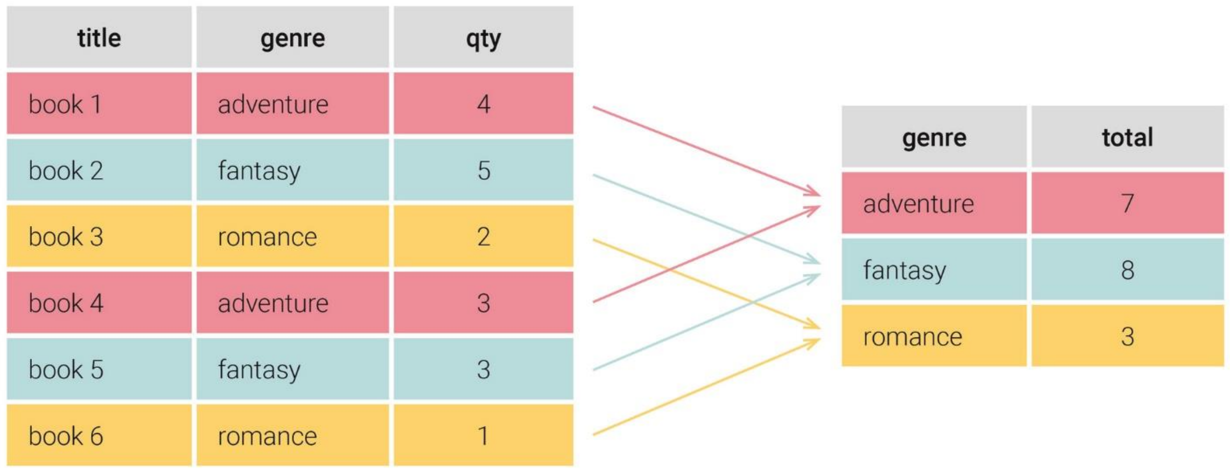
• Aggregate Functions:

- Explore common aggregate functions such as COUNT, SUM, AVG, MIN, and MAX.
- Understand how to perform calculations on a set of values to return a single summary value.
- Use aggregate functions with the GROUP BY clause for grouped data analysis


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

• Implement the GROUP BY Clause:

- Learn to group rows with the same values into summary rows.
- Understand the necessity of including non-aggregate function columns in the GROUP BY clause.
- Practice using the GROUP BY clause with aggregate functions to calculate summary statistics.



title	genre	qty
book 1	adventure	4
book 2	fantasy	5
book 3	romance	2
book 4	adventure	3
book 5	fantasy	3
book 6	romance	1

genre	total
adventure	7
fantasy	8
romance	3

• Utilize the HAVING Clause:

- Learn to filter grouped data based on specified conditions.
- Understand the difference between the WHERE and HAVING clauses.
- Apply the HAVING clause to refine results of grouped data and aggregate functions.

• Case Statement:

- Understand the purpose and syntax of CASE statements in SQL.
- Learn to apply conditional logic using CASE statements.

• Ensure SQL Statement Best Practices:

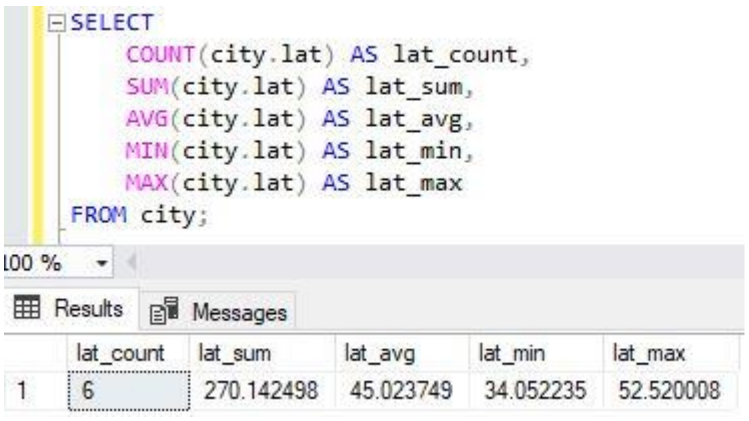
- Remember to end SQL statements with a semicolon (;).
- Use appropriate whitespace and indentation for readability.
- Combine learned techniques to efficiently filter, sort, and aggregate data in SQL databases.

1. SQL Functions:

Function Type	Function	Description	Example
Aggregate	COUNT()	Counts number of rows	SELECT COUNT(*) FROM students;
	SUM()	Sums up values in a column	SELECT SUM(marks) FROM results;
	AVG()	Calculates average	SELECT AVG(salary) FROM employees;
	MIN()	Finds minimum value	SELECT MIN(age) FROM users;
	MAX()	Finds maximum value	SELECT MAX(score) FROM tests;
String	CONCAT()	Combines strings	SELECT CONCAT(first_name, ' ', last_name);
	SUBSTRING()	Extracts part of a string	SELECT SUBSTRING(name, 1, 3);
	LENGTH()	Returns string length	SELECT LENGTH(email) FROM users;
Date	NOW()	Returns current date and time	SELECT NOW();
	DATE_ADD()	Adds interval to date	SELECT DATE_ADD(hire_date, INTERVAL 1 MONTH);
	DATEDIFF()	Returns days between two dates	SELECT DATEDIFF(end_date, start_date);
Mathematical	ROUND()	Rounds number to nearest integer or decimal	SELECT ROUND(price, 2);
	CEIL()	Rounds up to next whole number	SELECT CEIL(4.2);
	FLOOR()	Rounds down to previous whole number	SELECT FLOOR(4.8);

2. Aggregate Functions:

Aggregate functions perform calculations on a set of values and return a single value. Common aggregate functions include COUNT, SUM, AVG, MIN, and MAX.



```
SELECT
    COUNT(city.lat) AS lat_count,
    SUM(city.lat) AS lat_sum,
    AVG(city.lat) AS lat_avg,
    MIN(city.lat) AS lat_min,
    MAX(city.lat) AS lat_max
FROM city;
```

	lat_count	lat_sum	lat_avg	lat_min	lat_max
1	6	270.142498	45.023749	34.052235	52.520008

Syntax:

SELECT AGGREGATE_FUNCTION(column_name)

FROM table_name;

Example:

SELECT AVG(salary) FROM employees;

Key Points:

- Aggregate functions are often used in conjunction with the GROUP BY clause to calculate summary statistics for groups of data.
- Be cautious when using aggregate functions, as they can hide details and nuances in the underlying data.

3. GROUP BY Clause:

The GROUP BY clause is used to group rows that have the same values into summary rows, which is essential for performing aggregate functions on grouped data.

Syntax:

SELECT column1, AGGREGATE_FUNCTION(column2)
FROM table_name
GROUP BY column1;

Example:

```
SELECT department, AVG(salary)
FROM employees
GROUP BY department;
```

Key Points:

- Columns specified in the SELECT clause that are not part of an aggregate function must be included in the GROUP BY clause.
- GROUP BY operations are typically followed by aggregate functions to calculate summary statistics for each group.

4. HAVING Clause:

The HAVING clause filters grouped data based on specified conditions, allowing you to further refine the results of aggregate functions.

Syntax:

```
SELECT column1, AGGREGATE_FUNCTION(column2)
FROM table_name
GROUP BY column1
HAVING condition;
```

Example:

```
SELECT department, AVG(salary)
FROM employees
GROUP BY department
HAVING AVG(salary) > 50000;
```

Key Points:

- Unlike the WHERE clause, which filters individual rows, the HAVING clause filters groups of rows resulting from the GROUP BY operation.
- Conditions in the HAVING clause are applied after the data has been grouped and aggregated.

5. CASE Statement:

Key Points:

- **Purpose**: To create conditional logic in SQL queries.
- **Syntax**:

```
SELECT
  column_name,
  CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ...
    ELSE resultN
  END as new_column_name
FROM
  table_name;
```

Examples:

- Simple `CASE` statement:

```
SELECT
  employee_name,
  CASE
    WHEN salary > 50000 THEN 'High Salary'
    WHEN salary BETWEEN 30000 AND 50000 THEN 'Medium Salary'
    ELSE 'Low Salary'
  END as salary_category
FROM
  employees;
```

- Nested `CASE` statements:

```
SELECT
  employee_name,
  CASE
    WHEN salary > 50000 THEN 'High Salary'
    WHEN salary BETWEEN 30000 AND 50000 THEN
      CASE
        WHEN department = 'Sales' THEN 'Medium Salary - Sales'
        ELSE 'Medium Salary - Other'
      END
    ELSE 'Low Salary'
  END as salary_category
FROM
  employees;
```

Usage in Different Clauses:

- **SELECT**:

```
SELECT
  employee_name,
  CASE
    WHEN salary > 50000 THEN 'High Salary'
    ELSE 'Low Salary'
  END as salary_category
FROM
  employees;
```

- **WHERE**:

```
SELECT
  employee_name
FROM
  employees
WHERE
  CASE
    WHEN salary > 50000 THEN 'High Salary'
    ELSE 'Low Salary'
  END = 'High Salary';
```

- **ORDER BY**:

```
SELECT
  employee_name, salary
FROM
  employees
ORDER BY
  CASE
    WHEN salary > 50000 THEN 'High Salary'
    ELSE 'Low Salary'
  END;
```

Remember to always end your SQL statements with a semicolon (;) and to use appropriate whitespace and indentation for readability. Utilize these techniques to efficiently data analysis in your SQL databases. Happy querying!