**Intermediate SQL**

This module helps you learn how to use string patterns and ranges to search data and how to sort and group data in result sets. You will also practice composing nested queries and execute select statements to access data from multiple tables.

**Learning Objectives**

* Use string patterns and ranges in SQL queries
* Group data into result sets
* Sort and order result sets
* Use in-built functions to refine query results
* Compose sub-queries and nested select statements

# **Refining your results**

## **Using String, Patterns and ranges**

* **Purpose of SELECT Statement**: The main goal is to retrieve data from a relational database table.
* **Basic SELECT Statement**:
  + Example: SELECT \* FROM Book; retrieves all rows and columns from the Book table.
* **Retrieving Specific Columns**:
  + You can select specific columns, e.g., SELECT Book\_ID, Title FROM Book;.
* **Using WHERE Clause**:
  + The WHERE clause restricts results based on conditions, e.g., SELECT Title FROM Book WHERE Book\_ID = 'B1';.
* **String Patterns**:
  + Use the LIKE predicate to search for patterns.
  + Wildcard character % substitutes for missing letters.
  + Example: SELECT FirstName FROM Author WHERE FirstName LIKE 'R%'; retrieves authors whose first name starts with 'R'.
* **Using Ranges**:
  + Instead of using multiple comparison operators, you can use BETWEEN for ranges.
  + Example: WHERE Pages BETWEEN 290 AND 300;.
* **Using IN Operator**:
  + The IN operator allows specifying a set of values.
  + Example: WHERE Country IN ('Australia', 'Brazil'); retrieves authors from specified countries.

These points summarize the techniques for simplifying SELECT statements using string patterns, ranges, and sets of values

## **Sorting result Sets**

* **Purpose of Database Management Systems**: Facilitate data retrieval, not just storage.
* **Basic SELECT Statement**:
  + Example: SELECT \* FROM book; retrieves all rows and columns from the "book" table.
* **Sorting Results**:
  + Use the **ORDER BY** clause to sort results.
  + Example: SELECT title FROM book ORDER BY title; sorts titles in **ascending** order by default.
  + To sort in **descending** order, use the keyword **DESC**: ORDER BY title DESC;.
* **Sorting by Column Sequence**:
  + You can also sort by the column's position in the SELECT statement.
  + Example: SELECT title, pages FROM book ORDER BY 2; sorts by the second column, which is "pages".
* **Result Set**: The output reflects the specified sorting order based on the chosen column.

## **Grouping Result Sets**

"Grouping Select Statement Result Sets":

* **DISTINCT Keyword**:
  + Used to eliminate duplicate values from a result set.
  + Example: SELECT DISTINCT country FROM author;
* **GROUP BY Clause**:
  + Groups result sets into subsets that have matching values for one or more columns.
  + Example: SELECT country, COUNT(\*) FROM author GROUP BY country;
* **COUNT Function**:
  + Counts the number of rows in each group.
  + Example: COUNT(\*) counts all authors from each country.
* **AS Keyword**:
  + Used to rename a derived column for clarity.
  + Example: SELECT country, COUNT(\*) AS author\_count FROM author GROUP BY country;
* **HAVING Clause**:
  + Used to filter results after grouping.
  + Works only with the **GROUP BY** clause.
  + Example: HAVING COUNT(\*) > 4 restricts results to groups with more than four authors.
* **Difference Between WHERE and HAVING**:
  + **WHERE**: Filters rows before grouping.
  + **HAVING**: Filters groups after aggregation.
* **Example Query**:
* SELECT country, COUNT(\*) AS author\_count
* FROM author
* GROUP BY country

HAVING COUNT(\*) > 4;

This query would return countries with more than four authors, along with the count of authors from those countries.

**Summary: Refining Your Results**

Congratulations! You have completed this lesson. At this point in the course, you know:

* You can use the WHERE clause to refine your query results.
* The search condition of the WHERE clause uses a predicate to refine the search.
* You can use the wildcard character (%) as a substitute for unknown characters in a pattern.
* You can use BETWEEN ... AND ... to specify a range of numbers.
* You can sort query results into ascending or descending order, using the ORDER BY clause to specify the column to sort on.
* You can group query results by using the GROUP BY clause.

# **Functions, Multiple Tables and Sub queries**

## **Built-in Database Functions**

SQL Functions Overview

* **Built-in Functions**: Most databases have built-in functions that can be used directly in SQL statements, allowing operations on data within the database.
* **Benefits**: Using these functions can reduce the amount of data retrieved, minimizing network traffic and bandwidth usage, especially with large datasets.

Example Table: PETRESCUE

* **Columns**: ID, animal, quantity, cost, rescue date.
* **Purpose**: Used to demonstrate SQL functions.

Aggregate Functions

* **Definition**: Aggregate functions take a collection of values (like a column) and return a single value or null.
* **Common Aggregate Functions**:
  + **SUM**: Adds up all values in a column.
    - Example: SELECT SUM(COST) FROM PETRESCUE;
    - To rename the output: SELECT SUM(COST) AS SUM\_OF\_COST FROM PETRESCUE;
  + **MIN**: Returns the lowest value.
    - Example: SELECT MIN(QUANTITY) FROM PETRESCUE;
  + **MAX**: Returns the highest value.
    - Example: SELECT MAX(QUANTITY) FROM PETRESCUE;
  + **AVG**: Returns the average value.
    - Example: SELECT AVG(COST) FROM PETRESCUE;
    - For average cost per dog: SELECT AVG(COST / QUANTITY) FROM PETRESCUE WHERE ANIMAL = 'Dog';

Scalar and String Functions

* **Scalar Functions**: Operate on individual values.
  + Example: Rounding values: SELECT ROUND(COST) FROM PETRESCUE;
* **String Functions**: Operate on string values (char and varchar).
  + **LENGTH**: Retrieves the length of each value.
    - Example: SELECT LENGTH(ANIMAL) FROM PETRESCUE;
  + **UPPERCASE**: Converts values to uppercase.
    - Example: SELECT UPPER(ANIMAL) FROM PETRESCUE;
  + **LOWERCASE**: Converts values to lowercase.
    - Example: SELECT \* FROM PETRESCUE WHERE LOWER(ANIMAL) = 'cat';
  + **DISTINCT**: Retrieves unique values.
    - Example: SELECT DISTINCT(UPPER(ANIMAL)) FROM PETRESCUE;

Summary

* The video covered built-in SQL aggregate functions (SUM, MIN, MAX, AVG) and scalar/string functions (ROUND, LENGTH, UPPER, LOWER).
* These functions are essential for performing operations directly within the database, enhancing efficiency and reducing data transfer.

## **Data and Time Built-in Functions**

* **Date and Time Functions**: SQL has special data types for dates and times, including:
  + **Date**: 8 digits (YYYYMMDD)
  + **Time**: 6 digits (HHMMSS)
  + **Timestamp**: 20 digits (YYYYMMDDHHMMSSZZZZZZ)
* **Functions Available**: You can extract various components from date and time, such as:
  + Day
  + Month
  + Day of the week
  + Hour
  + Minute
  + Second
* **Using Functions in Queries**:
  + Example to extract the day from a date:

SELECT DAY(rescue\_date) FROM pet\_rescue WHERE animal = 'cat';

* + Example to count sales in May:

SELECT COUNT(\*) FROM pet\_rescue WHERE MONTH(rescue\_date) = 5;

* **Date Arithmetic**: You can perform calculations with dates:
  + Example to find a date three days after a rescue date:

SELECT DATE\_ADD(rescue\_date, INTERVAL 3 DAY) FROM pet\_rescue;

* **Current Date and Time**: You can use special registers to get the current date and time:
  + Example to find how many days have passed since a rescue date:

SELECT CURRENT\_DATE - rescue\_date FROM pet\_rescue;

**Date Functions**

1. Write a query that displays the rescue date.

For this query, we will use the function DAY(COLUMN\_NAME). The output of this query will be only the DAY part of the date in the column. The query for this question can be written as:

1. 1
2. SELECT DAY(RESCUEDATE) FROM PETRESCUE;

Copied!

In case the query was asking for MONTH of rescue, the query would change to:

1. 1
2. SELECT MONTH(RESCUEDATE) FROM PETRESCUE;

Copied!

In case the query was asking for YEAR of rescue, the query would change to:

1. 1
2. SELECT YEAR(RESCUEDATE) FROM PETRESCUE;

Copied!

1. Animals rescued should see the vet within three days of arrival. Write a query that displays the third day of each rescue.

For this query, we will use the function DATE\_ADD(COLUMN\_NAME, INTERVAL Number Date\_element). Here, the quantity in Number and in Date\_element will combine to form the interval to be added to the date in the column. For the given question, the query would be:

1. 1
2. SELECT DATE\_ADD(RESCUEDATE, INTERVAL 3 DAY) FROM PETRESCUE

Copied!

If the question was to add 2 months to the date, the query would change to:

1. 1
2. SELECT DATE\_ADD(RESCUEDATE, INTERVAL 2 MONTH) FROM PETRESCUE

Copied!

Similarly, we can retrieve a date before the one given in the column by a given number using the function DATE\_SUB. By modifying the same example, the following query would provide the date 3 days before the rescue.

1. 1
2. SELECT DATE\_SUB(RESCUEDATE, INTERVAL 3 DAY) FROM PETRESCUE

Copied!

1. Write a query that displays the length of time the animals have been rescued, for example, the difference between the current date and the rescue date.

For this query, we will use the function DATEDIFF(Date\_1, Date\_2). This function calculates the difference between the two given dates and gives the output in number of days. For the given question, the query would be:

1. 1
2. SELECT DATEDIFF(CURRENT\_DATE, RESCUEDATE) FROM PETRESCUE

Copied!

CURRENT\_DATE is also an inbuilt function that returns the present date as known to the server.

To present the output in a YYYY-MM-DD format, another function FROM\_DAYS(number\_of\_days)can be used. This function takes a number of days and returns the required formatted output. The query above would thus be modified to

1. 1
2. SELECT FROM\_DAYS(DATEDIFF(CURRENT\_DATE, RESCUEDATE)) FROM PETRESCUE

## **Sub-Queries and Nested-Selects**

1. **Definition of Subqueries**:
   * Subqueries are queries nested within parentheses inside another SQL query.
   * They allow for more complex queries by enabling the use of results from one query in another.
2. **Purpose of Subqueries**:
   * To perform operations that cannot be done with a single query.
   * To evaluate aggregate functions like AVG, SUM, etc., in the context of another query.
3. **Example Scenario**:
   * **Goal**: Retrieve a list of employees who earn more than the average salary.
   * **Incorrect Approach**:

SELECT \* FROM employees WHERE salary > AVG(salary);

* + - This results in an error because aggregate functions cannot be used directly in the WHERE clause.

1. **Correct Use of Subquery**:
   * Use a subquery to calculate the average salary:
   * SELECT employee\_id, first\_name, last\_name, salary
   * FROM employees

WHERE salary > (SELECT AVG(salary) FROM employees);

* + Here, the subquery (SELECT AVG(salary) FROM employees) calculates the average salary, which is then used in the WHERE clause.

1. **Column Expressions**:
   * Subqueries can also be used in the SELECT statement to create new columns.
   * Example:
   * SELECT employee\_id, salary, (SELECT AVG(salary) FROM employees) AS average\_salary

FROM employees;

* + This allows you to compare each employee's salary with the average salary.

1. **Derived Tables**:
   * Subqueries can be part of the FROM clause, creating derived tables.
   * Example:
   * SELECT \*

FROM (SELECT employee\_id, first\_name, last\_name, department\_id FROM employees) AS employee\_info;

* + This derived table can be used as a data source for the outer query.

1. **Benefits of Using Subqueries**:
   * They simplify complex queries by breaking them down into manageable parts.
   * They enhance readability and maintainability of SQL code.

Conclusion:

* Subqueries and nested selects are powerful tools in SQL that allow for more complex data retrieval and manipulation.
* They can be used in various parts of a SQL statement, including the WHERE clause, SELECT list, and FROM clause.

## **Working with Multiple Tables**

* **Sub-queries**:
  + Example: Retrieve employee records from the Employees table where the Department ID exists in the Departments table.
  + Another example: Get employees from a specific location using a sub-query on the Departments table.
* **Implicit Joins**:
  + You can specify multiple tables in the FROM clause, resulting in a full join.
  + Example: SELECT \* FROM employees, departments WHERE employees.DEP\_ID = departments.DEPT\_ID\_DEP;
  + You can use aliases for table names to simplify queries.

These methods allow you to work with data across different tables effectively.

**Summary: Functions, Multiple Tables, and Sub-queries**

Congratulations! You have completed this lesson. At this point in the course, you know:

* Tools for database management that offer built-in functions for performing operations on data within the database itself.
* That when working with large datasets, you may save time by using built-in functions rather than first retrieving the data into your application and then executing functions on the retrieved data.
* You can use sub-queries to form more powerful queries than otherwise.
* You can use a sub-select expression to evaluate some built-in aggregate functions like the average function.
* Derived tables or table expressions are sub-queries where the outer query uses the results of the sub-query as a data source.