

SQL FOR EVERYONE

THE DEFINITIVE GUIDE

Table of Contents

1. Introduction to SQL
2. Basic SQL Syntax
3. Querying Data
4. Filtering and Sorting Data
5. Joining Tables
6. Aggregation Functions
7. Subqueries and Nested Queries
8. Modifying Database Information
9. Advanced SQL Techniques
10. Optimization and Performance Tuning

1. Introduction to SQL

SQL is a standard language designed for managing data in relational databases. It's commonly used to query, insert, update, and modify data. Most RDBMS (Relational Database Management System) like MySQL, SQLite, Oracle, and PostgreSQL use SQL.

As a data analyst, you'll often work with large volumes of data stored in these databases. SQL becomes an essential tool to retrieve, manipulate, and analyze this data.

1.1 RDBMS and Tables

In SQL, data is stored in tables, just like an Excel spreadsheet. A table is made up of rows (records) and columns (fields). Here's an example of a table, `Employees`:

EmployeeID	FirstName	LastName	Position
1	John	Doe	Analyst
2	Jane	Doe	Engineer
3	Mary	Johnson	Manager

2. Basic SQL Syntax

Let's look at the fundamental SQL commands: SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY.

2.1 SELECT and FROM

The `SELECT` statement is used to select data from a database, and the `FROM` statement specifies which table to get the data from.

```
SELECT FirstName, LastName  
FROM Employees;
```

This query retrieves all first and last names from the `Employees` table.

If you want to select all columns, use the `*` symbol:

```
SELECT * FROM Employees;
```

2.2 WHERE

The `WHERE` clause is used to filter records:

```
SELECT *  
FROM Employees  
WHERE Position = 'Analyst';
```

This query retrieves all data for employees who are analysts.

2.3 GROUP BY and HAVING

GROUP BY groups rows that have the same values in specified columns into aggregated data. **HAVING** is used instead of **WHERE** with aggregated data.

```
SELECT Position, COUNT(*)
FROM Employees
GROUP BY Position
HAVING COUNT(*) > 1;
```

This query shows positions held by more than one employee.

2.4 ORDER BY

ORDER BY is used to sort the data in ascending or descending order:

```
SELECT *
FROM Employees
ORDER BY LastName ASC;
```

This query sorts employees by their last name in ascending order.

3. Querying Data

The **SELECT** statement is not just for selecting simple rows. We can use it to perform calculations, concatenations, and more.

```
SELECT FirstName || ' ' || LastName as FullName, Position
FROM Employees;
```

This query concatenates the first and last names, separated by a space, and displays it as **FullName**.

4. Filtering and Sorting Data

Apart from **WHERE** and **ORDER BY**, SQL offers **BETWEEN**, **LIKE**, and **IN** to filter data.

4.1 BETWEEN

BETWEEN is used to filter by a range:

```
SELECT *  
FROM Orders  
WHERE OrderDate BETWEEN '2023-01-01' AND '2023-12-31';
```

This query selects all orders placed in the year 2023.

4.2 LIKE and ILIKE

LIKE is used in a **WHERE** clause to search for a specified pattern in a column. The "%" sign is used to define wildcards (missing letters) both before and after the pattern. Also, note that **LIKE** is case sensitive. **ILIKE** can be used for case-insensitive search.

```
SELECT *  
FROM Employees  
WHERE FirstName LIKE 'J%';
```

This query selects all employees with a first name starting with 'J'.

4.3 IN

IN allows you to specify multiple values in a **WHERE** clause:

```
SELECT *  
FROM Employees  
WHERE Position IN ('Analyst', 'Engineer');
```

This query selects all analysts and engineers.

5. Joining Tables

JOIN statements are used to combine rows from two or more tables based on a related column. The different types of joins include **INNER JOIN**, **LEFT (OUTER) JOIN**, **RIGHT (OUTER) JOIN**, and **FULL (OUTER) JOIN**.

Consider this additional table, `Departments`:

DepartmentID	DepartmentName
1	IT
2	Sales
3	HR

And suppose we add a `DepartmentID` field to the `Employees` table. Here's how we can use different types of joins:

5.1 INNER JOIN

```
SELECT Employees.LastName, Employees.FirstName, Departments.DepartmentName
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

This query retrieves the list of employees along with their respective department names.

5.2 LEFT (OUTER) JOIN

```
SELECT Employees.LastName, Employees.FirstName, Departments.DepartmentName
FROM Employees
LEFT JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

This query retrieves all employees and their departments, including employees with no department (the `DepartmentName` for them will be `NULL`).

5.3 RIGHT (OUTER) JOIN

```
SELECT Employees.LastName, Employees.FirstName, Departments.DepartmentName
FROM Employees
RIGHT JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

This query retrieves all departments and their employees, including departments with no employees.

5.4 FULL (OUTER) JOIN

```
SELECT Employees.LastName, Employees.FirstName, Departments.DepartmentName
FROM Employees
FULL JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

This query retrieves all combinations of employees and departments, including employees with no department and departments with no employees.

6. Aggregation Functions

SQL provides several functions to perform calculations on data, such as `COUNT()`, `SUM()`, `AVG()`, `MIN()`, `MAX()`, and `GROUP_CONCAT()`.

```
SELECT COUNT(*)
FROM Orders
WHERE OrderDate BETWEEN '2023-01-01' AND '2023-12-31';
```

This query returns the total number of orders placed in the year 2023.

7. Subqueries and Nested Queries

A subquery is a SQL query nested inside a larger query. A subquery may occur in:

- A `SELECT` clause
- A `FROM` clause
- A `WHERE` clause

The subquery can be nested inside a `SELECT`, `INSERT`, `UPDATE`, or `DELETE` statement or inside another subquery.

```
SELECT EmployeeID, FirstName, Position
FROM Employees
WHERE EmployeeID IN (SELECT EmployeeID FROM Orders WHERE OrderTotal > 1000);
```

This query selects all employees who have made orders totaling more than 1000.