SQL FOR EVERYONE
THE DEFINITIVE GUIDE

Table of Contents

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.

8. Modifying Database Information

SQL allows you to insert, update, and delete data with `INSERT`, `UPDATE`, and `DELETE` commands respectively. Be careful when using these commands as you can change your data permanently.

8.1 INSERT

```
INSERT INTO Employees (EmployeeID, FirstName, LastName, Position)
VALUES (4, 'Mark', 'Anderson', 'Analyst');
```

This query adds a new row to the Employees table.

8.2 UPDATE

```
UPDATE Employees
SET Position = 'Senior Analyst'
WHERE EmployeeID = 4;
```

This query changes Mark Anderson's position to Senior Analyst.

8.3 DELETE

```
DELETE FROM Employees WHERE EmployeeID = 4;
```

This query deletes Mark Anderson's record from the Employees table.

9. Advanced SQL Techniques

Let's delve into more complex techniques with the help of examples.

9.1 Handling NULL values

NULL value in SQL means no or zero value. Here's how you can use `IS NULL` and `IS NOT NULL`:

```
SELECT *
FROM Employees
WHERE DepartmentID IS NULL;
```

This query selects all employees who don't belong to any department.

```
SELECT *
FROM Employees
WHERE DepartmentID IS NOT NULL;
```

This query selects all employees who belong to a department.

9.2 String Functions

SQL offers several functions to manipulate strings. Some examples include:

- CONCAT(): Concatenates two or more strings.
- TRIM(): Removes leading and trailing spaces of a string.
- LENGTH(): Returns the length of a string.

```
SELECT CONCAT(FirstName, ' ', LastName) as FullName, TRIM(Position),
LENGTH(FirstName) as NameLength
FROM Employees;
```

This query retrieves a full name by combining first and last names, the position after removing leading and trailing spaces, and the length of the first name.

9.3 Date and Time Functions

SQL provides many functions to work with date and time. Some examples include:

- NOW(): Returns the current date and time.
- CURDATE(): Returns the current date.
- CURTIME(): Returns the current time.

```
SELECT OrderID, OrderTotal, NOW() as QueryTime
FROM Orders
WHERE OrderDate = CURDATE();
```

This query retrieves today's orders along with the query execution time.

9.4 Case Statements

Case statements help in implementing conditional logic in SQL:

```
SELECT FirstName, Position,
CASE
    WHEN Position = 'Analyst' THEN 'Junior Level'
    WHEN Position = 'Engineer' THEN 'Mid Level'
    ELSE 'Senior Level'
END as JobLevel
FROM Employees;
```

This query categorizes employees into job levels based on their positions.

9.5 Window Functions

Window functions perform calculations across a set of table rows that are related to the current row:

```
SELECT FirstName, Position, Salary,
RANK() OVER (PARTITION BY Position ORDER BY Salary DESC) as Rank
FROM Employees;
```

This query ranks employees within their respective positions based on their salaries.

10. Optimization and Performance Tuning

Here are some examples demonstrating SQL optimization techniques:

10.1 EXPLAIN

Most SQL databases support the EXPLAIN command, which shows the execution plan of an SQL statement. This can help you understand how your SQL query will be executed and where you can optimize it.

```
EXPLAIN SELECT * FROM Employees;
```

10.2 Avoid SELECT *

Rather than using SELECT *, specify the columns you need. This reduces the amount of data that needs to be read from the disk.

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

10.3 Use LIMIT

If you only need a specific number of rows, use LIMIT to prevent reading unnecessary data.

```
SELECT * FROM Employees ORDER BY Salary DESC LIMIT 10;
```

This query gets the top 10 employees with the highest salaries.

10.4 Index your data

Indexing your data can significantly speed up data retrieval times. Here's how you can add an index:

```
CREATE INDEX idx_employees_position ON Employees(Position);
```