SQL Querying Commands

Overview

SQL querying allows users to retrieve, filter, and manipulate data from a database using various commands. This document covers different types of SQL queries along with their execution results.

Basic Queries

Selecting All Columns

The `SELECT *` command retrieves all columns from a table.

```
SELECT * FROM Employees;
```

Table Output:

EmpID	Name	Age	DepartmentID	Salary
1	Alice	30	1	60000.00
2	Bob	28	2	70000.00
3	Charlie	35	3	75000.00
4	David	40	1	80000.00
5	Emma	26	2	50000.00

Selecting Specific Columns

The `SELECT` command can also be used to retrieve specific columns.

```
SELECT Name, Salary FROM Employees;
```

Table Output:

Name	Salary
Alice	60000.00
Bob	70000.00
Charlie	75000.00
David	80000.00
Emma	50000.00

Filtering Data

Using WHERE Clause

The `WHERE` clause filters records based on a specified condition.

```
SELECT * FROM Employees WHERE DepartmentID = 2;
```

Table Output:

EmpID	Name	Age	DepartmentID	Salary
2	Bob	28	2	70000.00
5	Emma	26	2	50000.00

Using LIKE Operator

The `LIKE` operator is used to filter records based on pattern matching.

```
SELECT * FROM Employees WHERE Name LIKE 'A%';
```

Table Output:

EmpID	Name	Age	DepartmentID	Salary
1	Alice	30	1	60000.00

Sorting and Limiting Results

Sorting by Salary (Descending Order)

The `ORDER BY` clause sorts the results in ascending or descending order.

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

Table Output:

EmpID	Name	Age	DepartmentID	Salary
4	David	40	1	80000.00
3	Charlie	35	3	75000.00
2	Bob	28	2	70000.00
1	Alice	30	1	60000.00
5	Emma	26	2	50000.00

Limiting Results

The `LIMIT` clause restricts the number of rows returned.

```
SELECT * FROM Employees ORDER BY Age ASC LIMIT 3;
```

Table Output:

EmpID	Name	Age	DepartmentID	Salary
5	Emma	26	2	50000.00
2	Bob	28	2	70000.00
1	Alice	30	1	60000.00

Aggregate Queries

Counting Employees

The `COUNT()` function returns the number of rows that match a specified condition.

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

Table Output:

EmployeeCount
5

Average Salary in HR Department

The `AVG()` function calculates the average value of a numeric column.

```
SELECT AVG(Salary) AS AvgSalary FROM Employees WHERE DepartmentID = 1;
```

Table Output:

AvgSalary
70000.00

Grouping and Aggregation

Grouping Employees by Department

The `GROUP BY` clause groups rows that have the same values into summary rows.

```
SELECT DepartmentID, COUNT(*) AS EmployeeCount FROM Employees GROUP BY DepartmentID;
```

Table Output:

DepartmentID	EmployeeCount
1	2
2	2
3	1

Filtering Groups Using HAVING

The `HAVING` clause is used to filter groups based on a condition.

```
SELECT DepartmentID, COUNT(*) AS EmployeeCount FROM Employees GROUP BY DepartmentID HAV
```

Table Output:

DepartmentID	EmployeeCount
1	2
2	2

Nested Queries (Subqueries)

Finding Employees with Above-Average Salary

A subquery is a query nested inside another query.

```
SELECT * FROM Employees WHERE Salary > (SELECT AVG(Salary) FROM Employees);
```

Table Output:

EmpID	Name	Age	DepartmentID	Salary
3	Charlie	35	3	75000.00
4	David	40	1	80000.00

Using Subquery to Find Maximum Salary

Subqueries can also be used in the `SELECT` clause.

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

Table Output:

Name	MaxSalary
Alice	80000.00
Bob	80000.00
Charlie	80000.00
David	80000.00
Emma	80000.00

Joins (Combining Tables)

Inner Join

The `INNER JOIN` keyword selects records that have matching values in both tables.

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

Table Output:

Name	DepartmentName
Alice	HR
Bob	IT
Charlie	Finance
David	HR
Emma	IT

Left Join

The `LEFT JOIN` keyword returns all records from the left table and the matched records from the right table.

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

Table Output:

Name	DepartmentName
Alice	HR

Name	DepartmentName
Bob	IT
Charlie	Finance
David	HR
Emma	IT

Complex Queries

Checking Existence Using EXISTS

The `EXISTS` operator checks for the existence of any record in a subquery.

```
SELECT Name FROM Employees WHERE EXISTS (SELECT 1 FROM Departments WHERE Employees.Depa
```

Table Output:

Name
Alice
Bob
Charlie
David
Emma

Using CASE for Categorization

The `CASE` statement is used to create conditional logic in SQL.

```
SELECT Name,
       CASE
           WHEN Salary > 80000 THEN 'High'
           WHEN Salary BETWEEN 50000 AND 80000 THEN 'Medium'
           ELSE 'Low'
       END AS SalaryCategory
FROM Employees;
```

Table Output:

Name	SalaryCategory
Alice	Medium
Bob	Medium

Name	SalaryCategory
Charlie	Medium
David	Medium
Emma	Medium

Summary

- SQL provides a variety of querying techniques, including basic selection, filtering, sorting, aggregation, and joins.
- Nested queries and complex conditions enhance query flexibility.
- Joins help in combining data across multiple tables.

Run these queries in MySQL or PostgreSQL to observe the results.