**PROMPT : Give me detailed document on select statement in SQL with various examples**

The SELECT statement is the most fundamental and widely used command in SQL (Structured Query Language). Its primary purpose is to retrieve or query data from one or more tables in a database. You can select all data, or use various clauses to filter, sort, group, and format the results.

## Basic Syntax

The simplest form of a SELECT statement retrieves specific columns from a single table.

**Syntax:**

SELECT column1, column2, ...  
FROM table\_name;

* **SELECT**: Specifies the columns you want to retrieve.
* **FROM**: Specifies the table from which to retrieve the data.

To demonstrate, let's use a sample table called Employees.

Sample Table: Employees

| EmployeeID | FirstName | LastName | Department | Salary | HireDate |

| :--- | :--- | :--- | :--- | :--- | :--- |

| 101 | Anjali | Sharma | HR | 60000 | 2021-06-15 |

| 102 | Vikram | Singh | Engineering | 95000 | 2020-02-10 |

| 103 | Priya | Patel | Engineering | 85000 | 2022-08-01 |

| 104 | Rohan | Kumar | Sales | 72000 | 2021-11-20 |

| 105 | Sneha | Reddy | Sales | 78000 | 2020-07-30 |

| 106 | Amit | Mishra | HR | 55000 | 2023-01-05 |

| 107 | Pooja | Verma | Engineering | 110000 | 2019-04-12 |

Example 1: Selecting Specific Columns

To get just the first name, last name, and salary of all employees:

SELECT FirstName, LastName, Salary  
FROM Employees;

Result:

| FirstName | LastName | Salary |

| :--- | :--- | :--- |

| Anjali | Sharma | 60000 |

| Vikram | Singh | 95000 |

| ... | ... | ... |

Example 2: Selecting All Columns

The asterisk (\*) is a wildcard to select all columns from a table.

SQL

SELECT \*  
FROM Employees;

This query returns all rows and all columns from the Employees table.

## Filtering Data with WHERE

The WHERE clause is used to extract only those records that fulfill a specified condition. 🧐

**Syntax:**

SQL

SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

**Common Operators used with WHERE:**

| Operator | Description | Example |
| --- | --- | --- |
| = | Equal to | WHERE Department = 'Sales' |
| <> or != | Not equal to | WHERE Department <> 'HR' |
| > | Greater than | WHERE Salary > 80000 |
| < | Less than | WHERE Salary < 60000 |
| >= | Greater than or equal to | WHERE Salary >= 95000 |
| <= | Less than or equal to | WHERE Salary <= 72000 |
| BETWEEN | Between a certain range (inclusive) | WHERE Salary BETWEEN 70000 AND 80000 |
| LIKE | Search for a pattern (e.g., % for multiple characters, \_ for a single character) | WHERE FirstName LIKE 'A%' |
| IN | To specify multiple possible values for a column | WHERE Department IN ('HR', 'Sales') |

You can also combine conditions using logical operators **AND**, **OR**, and **NOT**.

Example 3: Filtering with a Single Condition

To find all employees in the 'Engineering' department:

SQL

SELECT FirstName, LastName, Salary  
FROM Employees  
WHERE Department = 'Engineering';

Result:

| FirstName | LastName | Salary |

| :--- | :--- | :--- |

| Vikram | Singh | 95000 |

| Priya | Patel | 85000 |

| Pooja | Verma | 110000 |

Example 4: Filtering with AND

To find all employees in the 'Engineering' department who earn more than $90,000:

SQL

SELECT FirstName, LastName, Salary  
FROM Employees  
WHERE Department = 'Engineering' AND Salary > 90000;

Result:

| FirstName | LastName | Salary |

| :--- | :--- | :--- |

| Vikram | Singh | 95000 |

| Pooja | Verma | 110000 |

## Sorting Results with ORDER BY

The ORDER BY clause sorts the result set in ascending or descending order.

* **ASC**: Ascending order (default).
* **DESC**: Descending order.

**Syntax:**

SQL

SELECT column1, column2, ...  
FROM table\_name  
ORDER BY column1 ASC|DESC, column2 ASC|DESC, ...;

Example 5: Sorting by a Single Column

To list all employees sorted by their salary in descending order:

SQL

SELECT FirstName, Salary  
FROM Employees  
ORDER BY Salary DESC;

Result:

| FirstName | Salary |

| :--- | :--- |

| Pooja | 110000 |

| Vikram | 95000 |

| Priya | 85000 |

| ... | ... |

## Aggregating Data with GROUP BY

The GROUP BY statement groups rows that have the same values in specified columns into summary rows. It is often used with aggregate functions to perform calculations on each group.

**Common Aggregate Functions:**

* **COUNT()**: Counts the number of rows.
* **SUM()**: Calculates the sum of values.
* **AVG()**: Calculates the average of values.
* **MIN()**: Returns the minimum value.
* **MAX()**: Returns the maximum value.

**Syntax:**

SELECT column\_name(s), aggregate\_function(column\_name)  
FROM table\_name  
GROUP BY column\_name(s);

Example 6: Using GROUP BY and COUNT

To count the number of employees in each department:

SELECT Department, COUNT(EmployeeID) AS NumberOfEmployees  
FROM Employees  
GROUP BY Department;

Result:

| Department | NumberOfEmployees |

| :--- | :--- |

| HR | 2 |

| Engineering | 3 |

| Sales | 2 |

Example 7: Using GROUP BY with AVG and SUM

To find the average salary and total salary payout for each department:

SQL

SELECT  
 Department,  
 AVG(Salary) AS AverageSalary,  
 SUM(Salary) AS TotalSalary  
FROM Employees  
GROUP BY Department;

Result:

| Department | AverageSalary | TotalSalary |

| :--- | :--- | :--- |

| HR | 57500 | 115000 |

| Engineering | 96666.67 | 290000 |

| Sales | 75000 | 150000 |

### Filtering Groups with HAVING

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions. HAVING allows you to filter the results of a GROUP BY query.

Example 8: Using HAVING

To find departments where the average salary is greater than $80,000:

SQL

SELECT  
 Department,  
 AVG(Salary) AS AverageSalary  
FROM Employees  
GROUP BY Department  
HAVING AVG(Salary) > 80000;

Result:

| Department | AverageSalary |

| :--- | :--- |

| Engineering | 96666.67 |

## Additional Useful Clauses

### DISTINCT

The DISTINCT keyword is used to return only unique (different) values.

Example 9: Selecting Distinct Departments

SELECT DISTINCT Department  
FROM Employees;

Result:

| Department |

| :--- |

| HR |

| Engineering |

| Sales |

### AS (Aliases)

Aliases are used to give a table or a column in a table a temporary, more readable name.

Example 10: Using Column Aliases

We already used this in the GROUP BY examples (AS NumberOfEmployees).

SQL

SELECT FirstName AS First, LastName AS Last  
FROM Employees;

Result:

| First | Last |

| :--- | :--- |

| Anjali | Sharma |

| ... | ... |

### JOIN

The JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Imagine we have another table Departments:

Table: Departments

| DeptName | DeptManager |

| :--- | :--- |

| HR | Rahul Gupta |

| Engineering| Meera Iyer |

| Sales | John Doe |

Example 11: Using INNER JOIN

To get the employee's name and their department manager:

SQL

SELECT  
 E.FirstName,  
 E.LastName,  
 D.DeptManager  
FROM Employees AS E  
INNER JOIN Departments AS D ON E.Department = D.DeptName;

Result:

| FirstName | LastName | DeptManager |

| :--- | :--- | :--- |

| Anjali | Sharma | Rahul Gupta |

| Vikram | Singh | Meera Iyer |

| ... | ... | ... |