
☑ Introduction to SQL

SQL Overview

1. What is SQL and how is it used?
2. List different types of SQL commands.
3. What are the advantages of using SQL?

Database Concepts

1. What is a relational database?
2. Define table, row, and column in database context.
3. Explain Primary Key and Foreign Key.

Setting Up a Database

1. Create a database named Company.
2. Create a table Employees with columns: ID, Name, Age, Department.
3. Drop the Company database.

☑ Basic Queries

SELECT Statement

1. Retrieve all data from a table named Students.
2. Retrieve only the Name and Marks from Students.
3. Retrieve all columns from Products where Price > 1000.

WHERE Clause

1. Get employees with age greater than 30.
2. Find customers from city 'Delhi'.
3. Get orders with status 'Pending'.

ORDER BY Clause

1. Display employees sorted by salary.
2. Show students sorted by name alphabetically.
3. Sort products by price in descending order.

☑ Data Manipulation

INSERT Statement

1. **Insert a record into Employees.**
2. **Add a new student to the Students table.**
3. **Insert multiple rows into Orders.**

UPDATE Statement

1. **Update salary of an employee with ID 101.**
2. **Change status to 'Completed' for a given order.**
3. **Update the department of an employee named 'John'.**

DELETE Statement

1. **Delete a student with ID 5.**
2. **Remove all orders with status 'Cancelled'.**
3. **Delete employees aged below 20.**

☑ **Joins**

INNER JOIN

1. **Get all customers and their orders.**
2. **List employees with their departments using INNER JOIN.**
3. **Join Orders and Products on ProductID.**

LEFT JOIN

1. **Get all employees and their assigned projects, including unassigned.**
2. **List all customers and any orders they may have placed.**
3. **Show all departments and their managers.**

RIGHT JOIN

1. **Show all projects and their assigned employees.**
2. **Get all orders and any linked products.**
3. **List all suppliers even if they haven’t supplied any products.**

FULL JOIN

1. **List all employees and projects including those without matches.**
2. **Show all customers and orders, including unmatched records.**
3. **Combine two tables A and B with full outer join logic.**

☑ **Aggregation & Grouping**

Aggregate Functions

1. **Find the maximum salary in Employees.**
2. **Count the number of orders placed.**
3. **Get the average marks from Students.**

GROUP BY Clause

1. **Count number of employees in each department.**
2. **Get total sales for each product.**
3. **Show average salary by department.**

HAVING Clause

1. **Show departments with more than 5 employees.**
2. **List products with average sales greater than 1000.**
3. **Display customers who placed more than 2 orders.**

☑ Subqueries & Advanced Topics

Subquery Basics

1. **Find employees who earn more than the average salary.**
2. **List products priced above the average.**
3. **Get names of students scoring above class average.**

Nested Queries

1. **Find departments with the highest-paid employee.**
2. **Get employees who are not managers.**
3. **List orders placed by customers from 'Mumbai'.**

Views

1. **Create a view HighSalary for employees with salary > 50000.**
2. **Create a view for customer names and their orders.**
3. **Drop a view named TopProducts.**

☑ Database Design

Normalization

1. **Normalize a table containing repeated customer address data.**
2. **Identify anomalies in an unnormalized table.**

3. **Convert a table to 3NF.**

Indexing

1. **Create an index on EmployeeID.**
2. **Create a composite index on FirstName and LastName.**
3. **Drop an index from Orders.**

Stored Procedures & Triggers

1. **Create a stored procedure to update employee salary.**
2. **Create a trigger to log deletions from Students.**
3. **Write a procedure to insert a record into Orders.**

Solutions:

-- ☑ Introduction to SQL

-- SQL Overview

-- Q1: What is SQL used for?

-- Q2: List SQL command types

-- Q3: Advantages of SQL

-- Database Concepts

-- Q1: What is a relational database?

-- Q2: Define table, row, and column

-- Q3: Primary vs Foreign Key

-- Setting Up a Database

CREATE DATABASE Company;

CREATE TABLE Employees (ID INT, Name VARCHAR(100), Age INT, Department VARCHAR(50));

DROP DATABASE Company;

-- ☑ Basic Queries

-- SELECT Statement

```
SELECT * FROM Students;
SELECT Name, Marks FROM Students;
SELECT * FROM Products WHERE Price > 1000;

-- WHERE Clause
SELECT * FROM Employees WHERE Age > 30;
SELECT * FROM Customers WHERE City = 'Delhi';
SELECT * FROM Orders WHERE Status = 'Pending';

-- ORDER BY Clause
SELECT * FROM Employees ORDER BY Salary;
SELECT * FROM Students ORDER BY Name;
SELECT * FROM Products ORDER BY Price DESC;

-- ☑ Data Manipulation
-- INSERT Statement
INSERT INTO Employees VALUES (101, 'John', 28, 'IT');
INSERT INTO Students (ID, Name, Marks) VALUES (1, 'Asha', 85);
INSERT INTO Orders VALUES (201, 'Laptop', 2), (202, 'Phone', 1);

-- UPDATE Statement
UPDATE Employees SET Salary = 60000 WHERE ID = 101;
UPDATE Orders SET Status = 'Completed' WHERE OrderID = 105;
UPDATE Employees SET Department = 'HR' WHERE Name = 'John';

-- DELETE Statement
DELETE FROM Students WHERE ID = 5;
DELETE FROM Orders WHERE Status = 'Cancelled';
DELETE FROM Employees WHERE Age < 20;

-- ☑ Joins
```

-- INNER JOIN

SELECT Customers.Name, Orders.OrderID FROM Customers INNER JOIN Orders ON Customers.ID = Orders.CustomerID;

SELECT E.Name, D.DeptName FROM Employees E INNER JOIN Departments D ON E.DeptID = D.ID;

SELECT Orders.OrderID, Products.ProductName FROM Orders INNER JOIN Products ON Orders.ProductID = Products.ID;

-- LEFT JOIN

SELECT Employees.Name, Projects.ProjectName FROM Employees LEFT JOIN Projects ON Employees.ID = Projects.EmployeeID;

SELECT Customers.Name, Orders.OrderID FROM Customers LEFT JOIN Orders ON Customers.ID = Orders.CustomerID;

SELECT Departments.Name, Managers.Name FROM Departments LEFT JOIN Managers ON Departments.ManagerID = Managers.ID;

-- RIGHT JOIN

SELECT Projects.ProjectName, Employees.Name FROM Employees RIGHT JOIN Projects ON Employees.ID = Projects.EmployeeID;

SELECT Orders.OrderID, Products.ProductName FROM Orders RIGHT JOIN Products ON Orders.ProductID = Products.ID;

SELECT Suppliers.Name, Products.Name FROM Products RIGHT JOIN Suppliers ON Products.SupplierID = Suppliers.ID;

-- FULL JOIN

SELECT * FROM Employees FULL OUTER JOIN Projects ON Employees.ID = Projects.EmployeeID;

SELECT * FROM Customers FULL OUTER JOIN Orders ON Customers.ID = Orders.CustomerID;

SELECT * FROM A FULL OUTER JOIN B ON A.ID = B.ID;

-- ☑ Aggregation & Grouping

-- Aggregate Functions

SELECT MAX(Salary) FROM Employees;

SELECT COUNT(*) FROM Orders;

SELECT AVG(Marks) FROM Students;

```
-- GROUP BY Clause
SELECT Department, COUNT(*) FROM Employees GROUP BY Department;
SELECT ProductID, SUM(Amount) FROM Sales GROUP BY ProductID;
SELECT Department, AVG(Salary) FROM Employees GROUP BY Department;

-- HAVING Clause
SELECT Department, COUNT(*) FROM Employees GROUP BY Department HAVING COUNT(*) > 5;
SELECT ProductID, AVG(Amount) FROM Sales GROUP BY ProductID HAVING AVG(Amount) > 1000;
SELECT CustomerID, COUNT(*) FROM Orders GROUP BY CustomerID HAVING COUNT(*) > 2;

-- ☑ Subqueries & Advanced Topics
-- Subquery Basics
SELECT * FROM Employees WHERE Salary > (SELECT AVG(Salary) FROM Employees);
SELECT * FROM Products WHERE Price > (SELECT AVG(Price) FROM Products);
SELECT Name FROM Students WHERE Marks > (SELECT AVG(Marks) FROM Students);

-- Nested Queries
SELECT DeptID FROM Employees WHERE Salary = (SELECT MAX(Salary) FROM Employees);
SELECT * FROM Employees WHERE ID NOT IN (SELECT ManagerID FROM Departments);
SELECT * FROM Orders WHERE CustomerID IN (SELECT ID FROM Customers WHERE City = 'Mumbai');

-- Views
CREATE VIEW HighSalary AS SELECT * FROM Employees WHERE Salary > 50000;
CREATE VIEW CustomerOrders AS SELECT Customers.Name, Orders.OrderID FROM Customers JOIN
Orders ON Customers.ID = Orders.CustomerID;
DROP VIEW TopProducts;

-- ☑ Database Design
-- Normalization
-- (No SQL command, theoretical practice)
```

```
-- Indexing
CREATE INDEX idx_emp_id ON Employees(ID);
CREATE INDEX idx_name ON Employees(FirstName, LastName);
DROP INDEX idx_order_status ON Orders;

-- Stored Procedures & Triggers
CREATE PROCEDURE UpdateSalary(IN emp_id INT, IN new_salary INT)
BEGIN
  UPDATE Employees SET Salary = new_salary WHERE ID = emp_id;
END;

CREATE TRIGGER LogStudentDelete
AFTER DELETE ON Students
FOR EACH ROW
INSERT INTO DeletedStudentsLog(StudentID, DeletedAt) VALUES (OLD.ID, NOW());

CREATE PROCEDURE InsertOrder(IN pid INT, IN qty INT)
BEGIN
  INSERT INTO Orders (ProductID, Quantity) VALUES (pid, qty);
END;
```