**Mastering SQL: A Complete Roadmap**

**1. Introduction to Databases and SQL**

**What is SQL?**

SQL (Structured Query Language) is a standard language used to interact with relational databases. It allows users to store, retrieve, manipulate, and manage data efficiently. SQL is widely used in applications ranging from simple data storage to complex data analytics.

**Creating a Database**

A database is a structured collection of data stored in an organized manner.

CREATE DATABASE CompanyDB;

This command creates a new database named CompanyDB.

**Selecting a Database**

Before working with a database, you must select it:

USE CompanyDB;

This tells SQL that all subsequent commands should be executed within CompanyDB.

**2. Creating Tables and Defining Schema**

Tables store data in rows and columns. Each column has a specific data type.

**Creating a Table**

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY AUTO\_INCREMENT,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Age INT CHECK (Age >= 18),

Salary DECIMAL(10,2) DEFAULT 50000.00,

DepartmentID INT,

HireDate DATE

);

* EmployeeID INT PRIMARY KEY AUTO\_INCREMENT: Unique ID that auto-increments.
* FirstName VARCHAR(50) NOT NULL: First name, cannot be NULL.
* Age INT CHECK (Age >= 18): Employees must be at least 18 years old.
* Salary DECIMAL(10,2) DEFAULT 50000.00: Default salary if none is provided.
* DepartmentID INT: Links to another table (to be explained in foreign keys).

**Altering a Table**

Modify an existing table structure:

ALTER TABLE Employees ADD COLUMN Email VARCHAR(100);

This adds an Email column to the Employees table.

**Deleting a Table**

DROP TABLE Employees;

This removes the table permanently.

**3. Data Types in SQL**

Common SQL data types include:

* **INTEGER (INT)**: Whole numbers.
* **VARCHAR(n)**: Variable-length strings (e.g., VARCHAR(255)).
* **TEXT**: Large text data.
* **DECIMAL(m, d)**: Fixed-point numbers (e.g., DECIMAL(10,2)).
* **DATE, DATETIME, TIMESTAMP**: Store date and time values.
* **BOOLEAN**: Stores TRUE or FALSE values.

**4. Inserting and Managing Data**

**Inserting Data into a Table**

INSERT INTO Employees (FirstName, LastName, Age, Salary, DepartmentID, HireDate)

VALUES ('John', 'Doe', 30, 60000.00, 2, '2023-01-15');

This inserts a new employee record.

**Updating Existing Records**

UPDATE Employees SET Salary = 70000 WHERE EmployeeID = 1;

This updates the salary of the employee with EmployeeID = 1.

**Deleting Records**

DELETE FROM Employees WHERE EmployeeID = 1;

This deletes the employee with EmployeeID = 1.

**5. Constraints in SQL**

Constraints ensure data integrity in the database.

**Primary Key Constraint**

Ensures each record has a unique identifier:

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY AUTO\_INCREMENT,

DepartmentName VARCHAR(100) NOT NULL

);

**Foreign Key Constraint**

Links two tables together to maintain relationships:

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY AUTO\_INCREMENT,

FirstName VARCHAR(50),

DepartmentID INT,

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

);

This ensures that the DepartmentID in Employees exists in Departments.

**Unique Constraint**

Ensures that a column has unique values:

ALTER TABLE Employees ADD CONSTRAINT unique\_email UNIQUE (Email);

**Check Constraint**

Ensures that values meet specific conditions:

ALTER TABLE Employees ADD CONSTRAINT check\_salary CHECK (Salary > 30000);

**6. Retrieving Data (SELECT Queries)**

**Basic SELECT Query**

SELECT FirstName, LastName FROM Employees;

Retrieves first and last names of all employees.

**Filtering Data using WHERE**

SELECT \* FROM Employees WHERE Age > 30;

Retrieves employees older than 30.

**Sorting Data using ORDER BY**

SELECT \* FROM Employees ORDER BY Salary DESC;

Retrieves employees sorted by salary in descending order.

**Limiting Results using LIMIT**

SELECT \* FROM Employees LIMIT 5;

Retrieves the first 5 employees.

**7. Joins (Combining Tables)**

**Inner Join**

SELECT Employees.FirstName, Employees.LastName, Departments.DepartmentName

FROM Employees

INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

Retrieves employees along with their department names.

**Left Join**

SELECT Employees.FirstName, Employees.LastName, Departments.DepartmentName

FROM Employees

LEFT JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

Retrieves all employees, including those without a department.

**Right Join**

SELECT Employees.FirstName, Employees.LastName, Departments.DepartmentName

FROM Employees

RIGHT JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

Retrieves all departments, including those without employees.

**8. Subqueries and Nested Queries**

A subquery is a query inside another query.

**Example: Retrieve employees with a salary above the company average**

SELECT FirstName, LastName, Salary

FROM Employees

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

**9. Indexing for Performance Optimization**

Indexes improve query performance.

CREATE INDEX idx\_salary ON Employees (Salary);

Creates an index on the Salary column to speed up searches.

**10. Transactions (Ensuring Data Consistency)**

A transaction is a sequence of queries that must be executed together.

START TRANSACTION;

UPDATE Employees SET Salary = 90000 WHERE EmployeeID = 2;

DELETE FROM Employees WHERE EmployeeID = 3;

COMMIT;

If an error occurs, rollback the changes:

ROLLBACK;

**11. Advanced SQL Techniques**

**Stored Procedures**

Reusable SQL blocks:

CREATE PROCEDURE GetEmployees()

BEGIN

SELECT \* FROM Employees;

END;

Call it using:

CALL GetEmployees();

**🚀 Final Notes**

* Practice daily with **real-world datasets**.
* Learn **query optimization** to handle large data efficiently.
* Master **database design** for better data structuring.
* Use **SQL in applications** (Python, Java, etc.) for hands-on experience.

Would you like practice exercises for each section? Let me know! 🚀😊