Notes on SQL (Structured Query Language)

---

1. Introduction to SQL

- SQL = **Structured Query Language**.
- Standard language to **create, manipulate, and query databases**.
- Works with relational databases like **MySQL, Oracle, PostgreSQL, SQL Server**.

---

2. DDL (Data Definition Language)

Used to define the structure of the database.

Commands

- **CREATE** → Creates tables, views, indexes.
- **ALTER** → Modifies table structure.
- **DROP** → Deletes a table.
- **TRUNCATE** → Deletes all records but keeps structure.

```
-- Create table
CREATE TABLE Student (
  RollNo INT PRIMARY KEY,
  Name VARCHAR(50),
  Dept VARCHAR(20)
);


-- Add a new column
ALTER TABLE Student ADD Email VARCHAR(100);


-- Delete a column
ALTER TABLE Student DROP COLUMN Email;


-- Drop table
```

```
DROP TABLE Student;
```

---

3. DML (Data Manipulation Language)

Used to **modify data** in tables.

```
-- Insert record
INSERT INTO Student (RollNo, Name, Dept)
VALUES (1, 'Alice', 'CSE');

-- Update record
UPDATE Student SET Dept = 'IT' WHERE RollNo = 1;

-- Delete record
DELETE FROM Student WHERE RollNo = 1;
```

---

4. DQL (Data Query Language – SELECT)

Used to **retrieve data**.

```
-- Select all
SELECT * FROM Student;

-- Select specific columns
SELECT Name, Dept FROM Student;

-- Filtering
SELECT * FROM Student WHERE Dept = 'CSE';

-- Sorting
SELECT * FROM Student ORDER BY Name ASC;
```

-- Limiting

SELECT * FROM Student LIMIT 5;

---

5. DCL (Data Control Language)

Used for permissions.

-- Grant permission

GRANT SELECT, INSERT ON Student TO 'user1';

-- Revoke permission

REVOKE INSERT ON Student FROM 'user1';

---

6. TCL (Transaction Control Language)

Used for transactions.

BEGIN; -- start

UPDATE Student SET Dept = 'CSE' WHERE RollNo = 2;

ROLLBACK; -- undo changes

COMMIT; -- save changes

SAVEPOINT s1;

---

7. Joins

INNER JOIN

Returns common records.

SELECT Student.Name, Course.Title

FROM Student

INNER JOIN Course ON Student.Dept = Course.Dept;

LEFT JOIN

Returns all students, even if no course.

SELECT Student.Name, Course.Title

FROM Student

LEFT JOIN Course ON Student.Dept = Course.Dept;

RIGHT JOIN

Opposite of LEFT JOIN.

SELECT Student.Name, Course.Title

FROM Student

RIGHT JOIN Course ON Student.Dept = Course.Dept;

FULL OUTER JOIN

Returns all records from both tables.

SELECT Student.Name, Course.Title

FROM Student

FULL OUTER JOIN Course ON Student.Dept = Course.Dept;

---

8. Subqueries

-- Students in same department as RollNo = 2

SELECT Name FROM Student

WHERE Dept = (SELECT Dept FROM Student WHERE RollNo = 2);

-- IN operator

SELECT Name FROM Student

WHERE Dept IN (SELECT Dept FROM Course WHERE Title='DBMS');

---

9. Aggregate Functions & GROUP BY

-- Count students in each dept

SELECT Dept, COUNT(*) FROM Student GROUP BY Dept;

-- Average marks

SELECT AVG(Marks) FROM Result;

```
-- Max and Min
SELECT MAX(Marks), MIN(Marks) FROM Result;
```

---

10. Views, Indexes, and Sequences

```
-- Create view
CREATE VIEW CSE_Students AS
SELECT * FROM Student WHERE Dept = 'CSE';

-- Create index
CREATE INDEX idx_name ON Student(Name);

-- Sequence (Oracle/Postgres)
CREATE SEQUENCE RollNo_seq START WITH 1 INCREMENT BY 1;
```

---

11. Constraints

```
CREATE TABLE Employee (
  EmpID INT PRIMARY KEY,
  Name VARCHAR(50) NOT NULL,
  Salary DECIMAL(10,2) CHECK (Salary > 0),
  DeptID INT,
  FOREIGN KEY (DeptID) REFERENCES Department(DeptID),
  UNIQUE(Name)
);
```

---

12. Stored Procedures, Triggers, Functions

```
-- Stored Procedure
CREATE PROCEDURE GetStudents()
```

```
BEGIN
  SELECT * FROM Student;
END;

-- Trigger
CREATE TRIGGER before_insert_student
BEFORE INSERT ON Student
FOR EACH ROW
SET NEW.Name = UPPER(NEW.Name);

-- Function
CREATE FUNCTION getTotalStudents()
RETURNS INT
BEGIN
  DECLARE total INT;
  SELECT COUNT(*) INTO total FROM Student;
  RETURN total;
END;
```

---

13. Practice Problems (Solved)

Q1. Find names of students enrolled in CSE.

```
SELECT Name FROM Student WHERE Dept='CSE';
```

Q2. List students who scored above average marks.

```
SELECT Name FROM Result
WHERE Marks > (SELECT AVG(Marks) FROM Result);
```

Q3. Find department with maximum students.

```
SELECT Dept, COUNT(*) AS total
FROM Student
```

```
GROUP BY Dept

ORDER BY total DESC

LIMIT 1;
```

Q4. Display student name and course title using JOIN.

```
SELECT Student.Name, Course.Title

FROM Student

JOIN Course ON Student.Dept = Course.Dept;
```

Q5. Show employees earning more than all employees in ‘HR’.

```
SELECT Name FROM Employee

WHERE Salary > ALL (SELECT Salary FROM Employee WHERE Dept='HR');
```