SQL – Structured Query Language (Well-Structured Notes)

1. What is SQL?

- **SQL (Structured Query Language)** is used to **store, retrieve, update, and manage data** in databases.
- SQL works on **tables** (rows & columns).

Popular SQL Databases (≈ 90% usage)

- **MySQL**
- **PostgreSQL**
- **Oracle**
- **SQL Server**

2. Software / Tools Used

- **Microsoft Excel** – Used for basic data storage (acts like a database)
- **MySQL Workbench** – GUI tool for MySQL
- **DB Viewer** – Database viewing tool
- **pgAdmin** – PostgreSQL administration tool

3. Database Connection (PATH)

To connect to a database, we need: - **IP Address** – Where the database server is located - **Port** – Communication channel (e.g., 3306 for MySQL) - **Username** – Login name - **Password** – Authentication

4. Excel vs Database

Excel	Database
File-based	Server-based
Sheet	Table
Column	Column
Row	Record

5. Basic SQL Concepts

Table Structure

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype
);
```

Example: Students Table

```
CREATE TABLE students (
    name VARCHAR(255),
    roll_num INT,
    subject VARCHAR(20),
    marks INT
);
```

Common Errors

- **Table already exists** → Table name duplication
- **Invalid input syntax** → Wrong datatype (e.g., inserting `80A` into INT column)

---

6. INSERT Data into Table

Insert selected columns

```
INSERT INTO students (name, roll_num)
VALUES ('Jay', 20);
```

Insert all columns

```
INSERT INTO students
VALUES ('Jay', 20, 'Maths', 80);
```

---

7. SELECT Query (Reading Data)

Select specific columns (Column Filter)

```
SELECT name, subject FROM students;
```

Select with condition (Row Filter)

```
SELECT * FROM students WHERE marks > 60;
```

---

8. Example: Courses Table

```
CREATE TABLE courses (
    course_id INT,
    course_name VARCHAR(80)
);

INSERT INTO courses
VALUES (1, 'SOC'), (2, 'BIGDATA');
```

Queries

```
SELECT * FROM courses;            -- All data
SELECT course_name FROM courses; -- Column filter
SELECT course_name FROM courses WHERE course_id = 2; -- Row filter
```

---

9. Filters in SQL

Column Filter

```
SELECT col1, col2 FROM table_name;
```

Row Filter

```
SELECT * FROM table_name WHERE condition;
```

---

10. Data Types & Conditions

NUMBER (INT, FLOAT)

Operators: - `>` Greater than - `<` Less than - `=` Equal - `>=`, `<=` - `!=` Not equal

STRING (VARCHAR)

- Exact match
- Pattern match using **LIKE**

11. Logical Conditions

- **AND** → All conditions must be true
- **OR** → Any one condition true

12. Employee Table Example

```
CREATE TABLE employee_info (
    emp_id INT,
    emp_name VARCHAR(50),
    department_name VARCHAR(50),
    emp_age INT,
    salary FLOAT
);
```

Insert Data

```
INSERT INTO employee_info
VALUES (5, 'Spcybersword', 'dep7', 90, 80000.0);
```

13. Numeric Conditions Examples

```
SELECT * FROM employee_info WHERE emp_age > 50;
SELECT * FROM employee_info WHERE emp_age >= 50;
SELECT * FROM employee_info WHERE emp_age < 50;
SELECT * FROM employee_info WHERE emp_age != 50;
SELECT * FROM employee_info WHERE emp_age > 50 AND salary > 60000;
SELECT * FROM employee_info WHERE emp_age > 50 OR salary > 60000;
```

IN Operator

```
SELECT * FROM employee_info WHERE emp_age IN (50, 60);
```

14. String Conditions (LIKE)

```
SELECT * FROM employee_info WHERE emp_name LIKE 'emp%';   -- Starts with emp
SELECT * FROM employee_info WHERE emp_name LIKE '%4';     -- Ends with 4
SELECT * FROM employee_info WHERE emp_name LIKE '%cyber%';-- Contains cyber
```

15. SQL Optimization (Basic Idea)

- **Early Push Down**: Apply filters as early as possible
- Reduces data processing
- Improves performance when tables are large (e.g., 30 × 100000 records)

NORMALIZATION (Very Simple Explanation)

What is Normalization?

Normalization is the process of **organizing data** to: - Remove duplicate data - Reduce redundancy - Improve data consistency

Original (Unnormalized) Table

StudentID	Name	Course	Duration	Subject	Marks
1	F1 M1 L1	PYTHON	3	A	20
1	F1 M1 L1	PYTHON	3	A	20
1	F1 M1 L1	PYTHON	3	B	19
2	F2 M2 L2	SOC	2	X	17
2	F2 M2 L2	SOC	2	Y	16

Problem: - Duplicate data - Data repeated many times - Hard to update

1NF (First Normal Form)

Rule: - Each cell should contain **only one value**

❌ `F1 M1 L1` → multiple values in one column

✔️Split into: - First Name - Middle Name - Last Name

2NF (Second Normal Form)

Rule: - Table must be in 1NF - Remove **partial dependency** - Each table must have a **Primary Key (PK)**

Example:

- Student table should not store Course details

Separate tables: - **Student(StudentID, Name)** - **Course(CourseID, CourseName, Duration)**

3NF (Third Normal Form)

Rule: - Table must be in 2NF - Remove **transitive dependency**

Meaning:

- Non-key column should not depend on another non-key column

Example:

❌Student → Course → Duration (Hidden dependency)

✔️ Correct Design: - Student(StudentID, Name) - Course(CourseID, CourseName, Duration) - Enrollment(StudentID, CourseID)

(Foreign Key relationship used)

Final Benefit of Normalization

✔️No duplicate data
✔️Easy updates
✔️Better performance
✔️Proper relationships between tables

16. Normalization – More Examples with Diagrams (Very Easy)

Example 1: Unnormalized Table (Problem Case)

Single Table:

StudentID	StudentName	Course	Subjects	Marks
1	Rahul	Python	A,B,C	20,19,18
2	Anita	SOC	X,Y	17,16

❌Problems: - Multiple values in one column - Difficult to query - Data duplication

Step 1: Convert to 1NF

Rule: One value per cell

Table after 1NF:

StudentID	StudentName	Course	Subject	Marks
1	Rahul	Python	A	20
1	Rahul	Python	B	19
1	Rahul	Python	C	18
2	Anita	SOC	X	17
2	Anita	SOC	Y	16

✔Atomic values achieved

Step 2: Convert to 2NF

Issue: Course depends only on StudentID, not on Subject

Split Tables:

Student Table | StudentID (PK) | StudentName | |--------------|------------| | 1 | Rahul | | 2 | Anita |

Course Table | CourseID (PK) | CourseName | |-------------|-----------| | 101 | Python | | 102 | SOC |

Marks Table | StudentID (FK) | CourseID (FK) | Subject | Marks | |--------------|--------------|---------|-------| | 1 | 101 | A | 20 | | 1 | 101 | B | 19 | | 2 | 102 | X | 17 |

✔Partial dependency removed

Step 3: Convert to 3NF

Issue: Course duration depends on CourseName

❌Hidden Dependency: CourseName → Duration

Final Tables:

Course Table | CourseID (PK) | CourseName | Duration | |-------------|-----------|---------| | 101 | Python | 3 | | 102 | SOC | 2 |

Enrollment Table | StudentID (FK) | CourseID (FK) | |--------------|--------------| | 1 | 101 | | 2 | 102 |

✔Transitive dependency removed

---

17. Diagram Representation (Text Diagram)

Before Normalization

```
STUDENT_TABLE
---------------------------------
StudentID | Name | Course | Subject | Marks
---------------------------------
1 | Rahul | Python | A,B,C | 20,19,18
```

After Normalization (3NF)

```
STUDENT              COURSE              MARKS
---------            ---------           ---------
StudentID (PK)       CourseID (PK)       StudentID (FK)
Name                 CourseName           CourseID (FK)
                     Duration             Subject
                                          Marks
```

18. Another Simple Real-Life Example

Without Normalization

OrderID	CustomerName	City	Product
1	Amit	Pune	Laptop
2	Amit	Pune	Mouse

❌City repeated

After Normalization

Customer Table | CustomerID | Name | City |

Order Table | OrderID | CustomerID | Product |

✔️Data stored once, reused everywhere

19. Quick Exam-Oriented Summary

- **1NF:** No multi-valued columns
- **2NF:** No partial dependency, PK required
- **3NF:** No indirect dependency

Easy Trick to Remember:

1NF → *One value*
2NF → *One table, one meaning*
3NF → *No hidden link*

End of Notes ✅