WEEK-1 NOTES

MYSQL

MySQL is an open-source relational database management system (RDBMS) that uses Structured Query Language (SQL) for database operations. It's one of the most popular database systems worldwide, known for its reliability, performance, and ease of use.

Key Features of MySQL

- **Open Source**: Free to use and modify under the GPL license
- **Relational Database**: Stores data in tables with relationships between them
- **Cross-Platform**: Runs on various operating systems (Windows, Linux, macOS)
- **Client-Server Architecture**: Supports multiple clients connecting to the database server
- **ACID Compliance**: Ensures reliability for transactions
- **Scalability**: Handles databases from small applications to enterprise-level systems
- **Replication**: Supports master-slave replication for data redundancy
- **Security**: Robust access control and encryption features

▼ MySQL Workbench

MySQL Workbench is the official integrated development environment (IDE) for MySQL. It provides a visual interface for database design, development, and administration.

MySQL Workbench Features

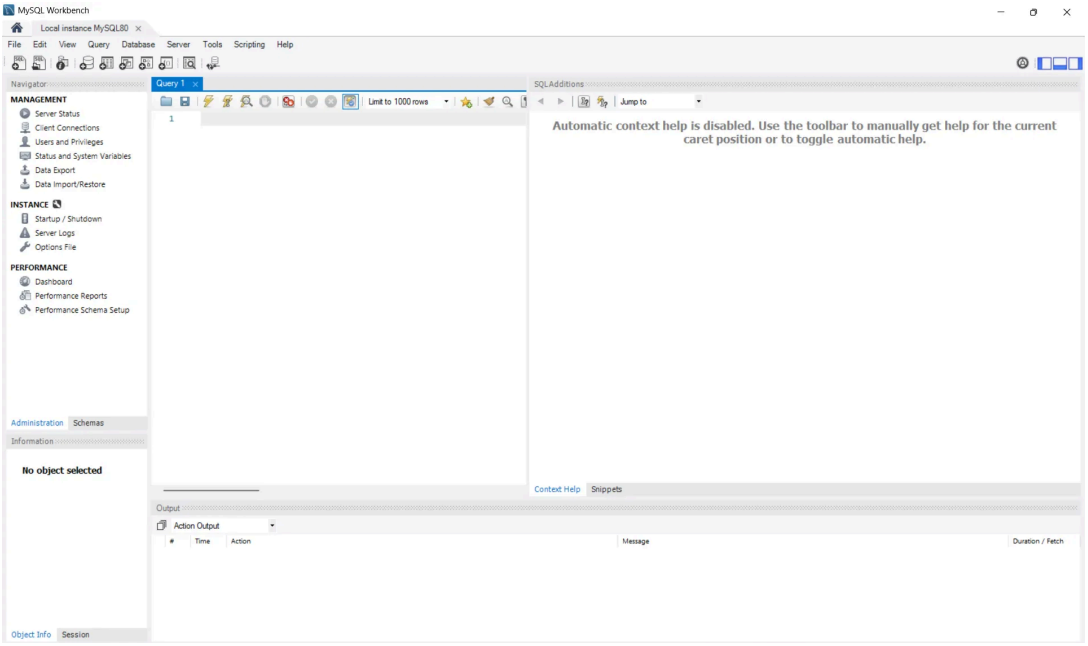
- **SQL Development**: Editor with syntax highlighting and query execution

- **Data Modeling**: Visual tools to create and modify database schemas
- **Database Administration**: User management, performance monitoring, and backup
- **Database Migration**: Tools to migrate from other database systems
- **Visual Explain Plan**: Graphical representation of query execution plans
- **Server Configuration**: Visual tools for configuring MySQL server settings
- **Data Import/Export**: Import and export functionality for various formats

Getting Started with MySQL Workbench

1. **Connection Setup**: Create connections to local or remote MySQL servers
2. **Schema Creation**: Design new databases with visual modeling tools
3. **Query Execution**: Write and run SQL statements with the SQL editor
4. **Result Management**: View, filter, and export query results
5. **Server Monitoring**: Monitor server health and performance metrics

MySQL Workbench simplifies database management tasks through its intuitive interface, making it accessible for both beginners and experienced database administrators.



▼ DDL

DDL stands for **Data Definition Language**.

It is used to **define or modify the structure of database objects** such as:

- Databases
- Tables
- Indexes
- Constraints
- Views

1. CREATE DATABASE

```
CREATE DATABASE database_name;
```

Edge Cases:

- If the database already exists →

```
CREATE DATABASE CompanyDB;
```

Results in error:

ERROR 1007 (HY000): Can't create database 'CompanyDB'; database exists

Fix:

```
CREATE DATABASE IF NOT EXISTS CompanyDB;
```

2. DROP DATABASE

```
DROP DATABASE database_name;
```

Edge Cases:

- If the database doesn’t exist →

```
DROP DATABASE NonExistentDB;
```

Results in error:

ERROR 1008 (HY000): Can't drop database 'NonExistentDB'; database doesn't exist

Fix:

```
DROP DATABASE IF EXISTS CompanyDB;
```

3. CREATE TABLE

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

Example:

```
CREATE TABLE tb01(
   f01 INT AUTO_INCREMENT PRIMARY KEY,
   f02 VARCHAR(100) NOT NULL,
   f03 VARCHAR(100) UNIQUE,
   f04 DATE,
   f05 DECIMAL(10,2)
);
```

4. DROP TABLE

```
DROP TABLE table_name;
```

5. ALTER TABLE

Used to modify an existing table.

Add Column:

```
ALTER TABLE tb01 ADD COLUMN f06 VARCHAR(50);
```

Modify Column:

```
ALTER TABLE tb01 MODIFY COLUMN f03 DECIMAL(12,2);
```

Rename Column :

```
ALTER TABLE tb01 RENAME COLUMN f05 TO f07;
```

Drop Column:

```
ALTER TABLE tb01 DROP COLUMN f07;
```

Add Primary Key:

```
ALTER TABLE tb01 ADD PRIMARY KEY (f01);
```

Add Foreign Key:

```
ALTER TABLE tb01 ADD CONSTRAINT fk_dept FOREIGN KEY (f02) REFE
RENCES tb02(f08);
```

6. TRUNCATE TABLE

```
TRUNCATE TABLE table_name;
```

Edge Cases:

- Unlike `DELETE FROM Employees`, **TRUNCATE is faster but cannot be rolled back** if not in transaction.
- Using TRUNCATE when there are FOREIGN KEY constraints → ERROR
- Must drop foreign key constraint first or use `DELETE`.

7. RENAME TABLE

```
RENAME TABLE old_name TO new_name;
```

▼ DML

DML stands for **Data Manipulation Language**.

It is used to **manipulate data inside tables**:

- Insert new records
- Update existing data
- Delete data

1. INSERT

```
INSERT INTO table_name (column1, column2, ...) VALUES (value1, value
2, ...);
```

Example:

```
INSERT INTO tb01(f01, f02, f03, f04)
VALUES ('Nisharg Soni', 'ncsoni04@gmail.com', '2025-09-09', 55000.0
0);
```

2. UPDATE

```
UPDATE table_name SET column1 = value1, column2 = value2 WHERE c
ondition;
```

Example:

```
UPDATE tb01 SET f04 = 60000 WHERE name = 'Nisharg Soni';
```

Update all rows (Dangerous!):

```
UPDATE tb01 SET f05 = f05 * 1.1;
```

Always double-check the WHERE clause.

- Using LIMIT (MySQL-specific feature):

  ```
  UPDATE tb01 SET f04 = 50000 WHERE f04 < 40000 LIMIT 5;
  ```

  → Updates only first 5 matched rows.

3. DELETE

Used to remove records from a table.

```
DELETE FROM table_name WHERE condition;
```

▼ DQL

DQL stands for **Data Query Language**.

It’s primary purpose is to **retrieve data from database tables** using the ***SELECT*** statement and apply filtering, sorting, grouping, and conditions.

1. SELECT

```
SELECT column1, column2 FROM table_name;
```

Example:

```
SELECT f01, f02, f03 FROM Employees;
```

Select All Columns:

```
SELECT * FROM Employees;
```

2. WHERE Clause

Used to filter records.

```
SELECT column1 FROM table_name WHERE condition;
```

Examples:

```
SELECT * FROM tb01 WHERE f04 > 50000;
SELECT * FROM tb01 WHERE f03 BETWEEN '2025-09-09' AND '2026-09-09';
SELECT * FROM tb02 WHERE f02 LIKE 'N%';
SELECT * FROM tb02 WHERE f04 IS NULL;
SELECT * FROM tb02 WHERE f04 > 50000 AND f03 < '2023-01-01';
```

3. ORDER BY

Sorts the result set by one or more columns.

```
SELECT column1 FROM table_name ORDER BY column1 ASC|DESC;
```

Examples:

```
SELECT f01, f02 FROM tb01 ORDER BY f02 DESC;
SELECT f01, f03 FROM tb02 ORDER BY f03 ASC;
SELECT f01, f02, f03 FROM tb01 ORDER BY f02 DESC, f03 ASC;
```

4. GROUP BY

Groups rows sharing a common attribute and allows aggregate functions.

```
SELECT column1, AGG_FUNC(column2) FROM table_name GROUP BY c
olumn1;
```

Examples:

```
SELECT f05, COUNT(*) AS emp_count FROM tb01 GROUP BY f05;
SELECT f05, AVG(f02) AS avg_salary FROM tb01 GROUP BY f05;
```

5. HAVING

Works like WHERE but applies conditions on aggregated data.

```
SELECT column1, AGG_FUNC(column2) FROM table_name GROUP BY c
olumn1 HAVING condition;
```

Examples:

```
SELECT f05, AVG(f02) AS avg_salary FROM tb01 GROUP BY f05 HAVIN
G AVG(f02) > 50000;
SELECT f05, COUNT(*) AS emp_count FROM tb01 GROUP BY f05 HAVI
NG emp_count >= 3;
```

▼ JOINs

In relational databases, **JOINs** combine rows from two or more tables based on related columns (usually using foreign keys).

Used when you want to query data from multiple tables at once.

Table 1 – tb01 (Employees)

f01	f02	f03
1	Nisharg Soni	101
2	Dakshil Gorasiya	102
3	Hemang Patel	103
4	Manish Jadav	NULL

Table 2 – tb02 (Departments)

f03	f04
101	HR
102	Engineering
104	Sales

1. INNER JOIN

Returns rows where there is a **match in both tables**.

```
SELECT e.f02, d.f04
FROM tb01 e
INNER JOIN tb02 d ON e.f03 = d.f03;
```

Result:

f02	f04
Nisharg Soni	HR
Dakshil Gorasiya	Engineering

2. LEFT JOIN (or LEFT OUTER JOIN)

Returns **all rows from the left table (Employees)**, and matching rows from the right table (Departments).

If no match → NULL in columns from right table.

```
SELECT e.f02, d.f04
FROM tb01 e
LEFT JOIN tb02 d ON e.f03 = d.f03;
```

Result:

f02	f04
Nisharg Soni	HR
Dakshil Gorasiya	Engineering
Hemang Patel	NULL
Manish Jadav	NULL

3. RIGHT JOIN (or RIGHT OUTER JOIN)

Returns **all rows from the right table (Departments)**, and matching rows from the left table (Employees).

If no match → NULL in columns from left table.

```
SELECT e.f02, d.f04
FROM tb01 e
RIGHT JOIN tb02 d ON e.f03 = d.f03 ;
```

Result:

f02	f04
Nisharg Soni	HR
Dakshil Gorasiya	Engineering
NULL	Sales

4. FULL OUTER JOIN

MySQL doesn’t support FULL OUTER JOIN natively.

But you can simulate it using `UNION`.

```
SELECT e.f02, d.f04
FROM tb01 e
LEFT JOIN tb02 d ON e.f03 = d.f03

UNION

SELECT e.f02, d.f04
FROM tb01 e
RIGHT JOIN tb02 d ON e.f03 = d.f03;
```

Result:

f02	f04
Nisharg Soni	HR
Dakshil Gorasiya	Engineering
Hemang Patel	NULL
Manish Jadav	NULL
NULL	Sales

5. CROSS JOIN

Returns **Cartesian product of both tables** (all combinations).

Syntax:

```
SELECT e.f02, d.f04
FROM tb01 e
CROSS JOIN tb02 d;
```

Result:

f01	f04
Nisharg Soni	HR

f01	f04
Nisharg Soni	Engineering
Nisharg Soni	Sales
Dakshil Gorasiya	HR
Dakshil Gorasiya	Engineering
Dakshil Gorasiya	Sales
Hemang Patel	HR
Hemang Patel	Engineering
Hemang Patel	Sales
Manish Jadav	HR
Manish Jadav	Engineering
Manish Jadav	Sales

6. SELF JOIN

Joining a table to itself (useful for hierarchical data).

Example:

Employees table has a f06 (manager_id) column referring to f05 (emp_id) (self-referencing).

```
SELECT e1.f01 AS Employee, e2.f01 AS Manager
FROM tb01 e1
LEFT JOIN tb01 e2 ON e1.f06 = e2.f05;
```

▼ AGGREGATE FUNCTONS

Aggregate functions are used to **compute a single value from multiple rows of data**.

They are commonly used in combination with ***GROUP BY*** but can also be used alone.

1. SUM() – Total Sum of Values

```
SELECT SUM(column_name) FROM table_name WHERE condition;
```

Example:

Get total salary of all employees:

```
SELECT SUM(f04) AS total_salary FROM tb01;
```

If matching rows Returns `NULL`.

Use `COALESCE()` to avoid NULL:

```
SELECT COALESCE(SUM(f04), 0) AS total_salary FROM tb01 WHERE f0
4 > 1e6;
```

2. COUNT() – Count Number of Rows

```
SELECT COUNT(column_name) FROM table_name WHERE condition;
```

Examples:

- Count total employees:

  ```
  SELECT COUNT(*) AS total_employees FROM tb01;
  ```

- Count employees in a specific department:

  ```
  SELECT COUNT(f01) AS engineering_count FROM tb01 WHERE f03
  = 101;
  ```

Edge Cases:

- `COUNT(column_name)` ignores NULL values in the column.

  Example:

  ```
  SELECT COUNT(f03) FROM tb01;
  ```

  → Does NOT count rows where `f03 IS NULL`.

- Use `COUNT(*)` to count all rows, regardless of NULLs.

3. AVG() – Average Value

```
SELECT AVG(column_name) FROM table_name WHERE condition;
```

Example:

Average salary of all employees:

```
SELECT AVG(f02) AS avg_salary FROM tb01;
```

Edge Cases:

- AVG ignores NULL values by default.
- If no rows → Returns NULL.

Use `COALESCE()` for default fallback:

```
SELECT COALESCE(AVG(f02), 0) AS avg_salary FROM tb01 WHERE f03 = 999;
```

4. MIN() – Minimum Value

```
SELECT MIN(column_name) FROM table_name WHERE condition;
```

Example:

Find the smallest salary:

```
SELECT MIN(f02) AS min_salary FROM tb01;
```

5. MAX() – Maximum Value

```
SELECT MAX(column_name) FROM table_name WHERE condition;
```

Example:

Find the highest salary:

```
SELECT MAX(f02) AS max_salary FROM tb01;
```