




Week 1 - MYSQL NOTES

Created by  Dakshil Gorasiya

Overview of Workbench

- MySQL Workbench is a graphical tool for working with MySQL servers and databases.

MySQL Workbench offers five key functional areas:

1. **SQL Development:** Allows you to create and manage database server connections. You can configure connection parameters and execute SQL queries using the built-in SQL Editor.
2. **Data Modeling (Design):** Enables graphical creation of database schema models, reverse and forward engineering between schemas and live databases, and comprehensive database editing. The Table Editor offers intuitive tools for modifying Tables, Columns, Indexes, Triggers, Partitioning, Options, Inserts, Privileges, Routines, and Views.
3. **Data Migration:** Facilitates migration of tables, objects, and data from various database systems to MySQL, including Microsoft SQL Server, Microsoft Access, Sybase ASE, SQLite, SQL Anywhere, PostgreSQL, and other RDBMS. It also supports migration between different MySQL versions.
4. **Server Administration:** Helps you manage MySQL server instances through user administration, backup and recovery, audit data inspection, database health monitoring, and server performance tracking.
5. **MySQL Enterprise Support:** Provides support for Enterprise products including MySQL Enterprise Backup, MySQL Firewall, and MySQL Audit.

| Default port for MYSQL is 3306

Type of SQL - DDL, DML, DQL, DCL, TCL

DDL - Data Definition Language

CREATE DATABASE

Used to create a new database or schema

Syntax : `CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name`

ex : `CREATE DATABASE EMPLOYEE;`

DDL (CREATE, ALTER, DROP tables)

CREATE - Used to create a new table

Syntax : CREATE [TEMPORARY] TABLE [IF NOT EXISTS] *tbl_name* (*create_definition*,...)

We can use the **TEMPORARY** keyword when creating a table. A **TEMPORARY** table is visible only within the current session, and is dropped automatically when the session is closed.

```
Ex : CREATE TABLE T01
(
T01F01 INT PRIMARY KEY,
T01F02 VARCHAR(50),
T01F03 VARCHAR(50)
);
```

Data type:

Numeric

- DECIMAL(m,d) : Best for financial data where precision is needed
- FLOAT(m, d), DOUBLE(m,d): Best for scientific data
- TINYINT : -128 to 127
- BOOL, BOOLEAN : true false
- SMALLINT, MEDIUMINT, INT, INTEGER, BIGINT,

Dates

- DATE : Format yyyy-mm-dd
- DATETIME : yyyy-mm-dd hh:mm:ss [.fraction]
- TIMESTAMP : Store number of seconds since 1970-01-01 00:00:00 0.000000
- TIME : hh:mm:ss [.fraction]
- YEAR : yyyy

To assign automatic time stamp

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  dt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

String

- CHAR(length) : Fixed length
- VARCHAR(length) : variable length

- BLOB : Binary Large Object (TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB) used to store raw data like image, video, objects
- TEXT : Used to store long strings like articles
- ENUM:

```
CREATE TABLE shirts (
  name VARCHAR(40),
  size ENUM('x-small', 'small', 'medium', 'large', 'x-large')
);
```

JSON:

```
CREATE TABLE t1 (jdoc JSON);

INSERT INTO t1 VALUES('{"key1": "value1", "key2": "value2"}');
```

DROP - Used to drop (delete) table

Syntax: DROP [TEMPORARY] TABLE [IF EXISTS] *tbl_name* [,*tbl_name*] [RESTRICT | CASCADE]

There is no effect of CASCADE and RESTRICT it just added to make it compatible with other SQL

Ex: DROP TABLE IF EXISTS T01;

Constraints

- Primary key : unique, not null, only one per table, used to uniquely identify row
- Foreign key : Create a link between two tables, prevents action that destroy that relationship, Foreign key must be refer to unique column of other table, can be restrict or cascade or set null or no action
- Unique : Make sure no duplicate entry in table
- Not Null : Make sure field always some value
- Check : To add some condition which must be true to insert data

ALTER - Used to change table specification

Rename table

```
ALTER TABLE DEPARTMENTS RENAME T01;
```

Add new column

ALTER TABLE T01 ADD T01F04 TIMESTAMP;

Rename column

ALTER TABLE T01 RENAME COLUMN T01F04 TO TIMECREATED;

Drop column

ALTER TABLE T01 DROP COLUMN TIMECREATED;

Change data type

ALTER TABLE T01 MODIFY COLUMN T01F03 VARCHAR(100);

Add primary key

ALTER TABLE T01 ADD CONSTRAINT PK_T01 PRIMARY KEY(T01F01);

Drop foreign key

ALTER TABLE T02 DROP CONSTRAINT T02_IBFK_1;

Add foreign key

ALTER TABLE T02 ADD CONSTRAINT FK_T02F08 FOREIGN KEY (T02F08) REFERENCES T01(T01F01);

Drop unique constraint

ALTER TABLE T02 DROP CONSTRAINT T02F04;

Add unique constraint

ALTER TABLE T02 ADD CONSTRAINT T02F04_UNIQUE UNIQUE(T02F04);

Add check constraint

ALTER TABLE T02 ADD CONSTRAINT T02F07_CHECK CHECK(T02F07 > 0);

Add not null constraint

ALTER TABLE T02 MODIFY COLUMN T02F02 VARCHAR(50) NOT NULL;

Set default value

ALTER TABLE T02 ALTER COLUMN T02F05 SET DEFAULT (CURDATE());

Drop default values

```
ALTER TABLE T02 ALTER COLUMN T02F05 DROP DEFAULT;
```

TRUNCATE

Truncate is a DDL command

It reset the table delete all data from table

It cannot be rollbacked

It does not invoke trigger

It is very fast compared to DELETE

Syntax:

```
TRUNCATE TABLE table_name;
```

Ex;

```
START TRANSACTION;
TRUNCATE TABLE T04;
ROLLBACK;
SELECT * FROM T04;
```

It will delete all data of table and does not rollback

DML - Data Manipulation Language

INSERT, UPDATE, DELETE ROWS

INSERT : Used to insert data into table

Syntax : INSERT INTO tbl_name [(col_names)] VALUES (values)

```
Ex : INSERT INTO T01 (T01F01, T01F02, T01F03) VALUES
(1, 'Engineering', 'Bengaluru'),
(2, 'Sales', 'Mumbai'),
(3, 'Human Resources', 'Rajkot'),
(4, 'Marketing', 'Delhi'),
(5, 'Finance', 'Mumbai');
```

To add data from other table

```
INSERT INTO target_table (column1, column2, column3)
SELECT source_column_a, source_column_b, source_column_c
```

FROM source_table
WHERE condition;

UPDATE - Used to update data of table usually for particular row

Syntax : UPDATE table_name SET field_name = value1 [field_name = value2...] [WHERE condition]

Ex : UPDATE T02 SET T02F08 = 5 WHERE T02F01 = 108;

DELETE - Used to delete rows from table (Use it with "WHERE" clause otherwise entire table data will be deleted)

Syntax : DELETE FROM table_name WHERE condition;

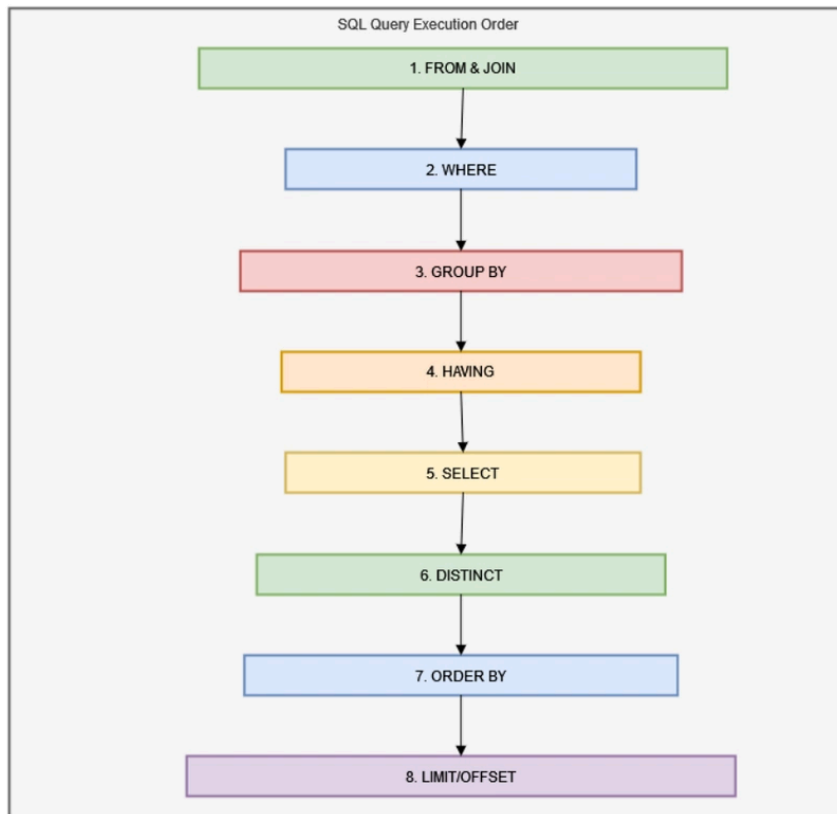
Ex : DELETE FROM T02 WHERE T02F01 = 108;

To remove all null entry : DELETE FROM T02 WHERE T02F08 IS NULL;

DQL - Data Query Language

SELECT, WHERE, ORDER BY, GROUP BY, Having

Order of execution



SELECT - Used to retrieve data from one or more tables.

Syntax: SELECT Fields FROM Table Name (Alias) JOIN WHERE Condition Group BY Fields Order By Fields

Ex:

```
SELECT * FROM T01;
```

| * represent all columns

WHERE - Used to add condition in query

Only return data which satisfy given condition

Ex:

```
SELECT
  T01F01,
  T01F02,
  T01F03
FROM
  T01
WHERE
  T01F01 = 1;
```

ORDER BY - Used to sort column in ascending or descending order.

If there is multiple order by clause in single query it will first sort by first column and for duplicate in that it will sort by second column.

Ex:

```
SELECT
    T02F01,
    T02F02,
    T02F03
FROM
    T02
ORDER BY
    T02F02,
    T02F03;
```

GROUP BY - Used to collect and group rows that have the same values in specified columns into a summary row.

Its primary purpose is to be used with **aggregate functions** (like COUNT, SUM, AVG, MAX, MIN) to perform a calculation on each of these groups.

If multiple columns is given it find unique tuple of all columns and form groups Ex: GROUP BY(T01F01, T01F02) will form group for each unique value of (T01F01, T01F02)

Ex:

```
SELECT
    T01F02,
    COUNT(T02F01) AS NUMBER_OF_EMPLOYEE
FROM
    T02
    INNER JOIN T01 ON (T01F01 = T02F08)
GROUP BY
    T02F08;
```

HAVING - Used to add condition in group by

Ex:

```
SELECT
    T01F02,
    SUM(T02F07)
FROM
    T02
    INNER JOIN T01 ON (T01F01 = T02F08)
GROUP BY
    T02F08
HAVING
    SUM(T02F07) > 100000;
```


LIMIT - LIMIT is used at last of sql which limit number of records is result reducing network bandwidth

OFFSET - OFFSET is used to skip number of records before using LIMIT

Ex :

```
SELECT
    T02F02,
    T02F01,
    T02F07
FROM
    T02
ORDER BY
    T02F07
LIMIT
    1
OFFSET
    2;
```

SUM, COUNT, AVG, MIN, MAX

These are aggregate function usually used with GROUP BY clause

COUNT(column_name) : count number of column where given column name is not null

count will assume all NULL as 1 record

```
# Count number of employee department wise
SELECT
    T01F02,
    COUNT(T02F01) AS NUMBER_OF_EMPLOYEE
FROM
    T02
    INNER JOIN T01 ON (T01F01 = T02F08)
GROUP BY
    T02F08;
```

SUM(column_name) : It sums up all values of given column and ignore null

Only work on int, decimal, float

```
# Get department having total salary greater than 100000
SELECT
    T01F02,
    SUM(T02F07)
FROM
    T02
    INNER JOIN T01 ON (T01F01 = T02F08)
GROUP BY
    T02F08
HAVING
    SUM(T02F07) > 100000;
```

AVG(column_name) : It find average of given column and ignore null

Only work on int, decimal, float

```
# To calculate average salary of department
SELECT
    T01F02,
    AVG(T02F07)
FROM
    T02
    INNER JOIN T01 ON (T01F01 = T02F08)
GROUP BY
    T02F08;
```

MIN(column_name) : It find minimum of given column

When used on varchar it work on alphabetical order for example MIN(first_name) will return name starting with A

When used on date it give earliest date

MAX(column_name) : It find maximum of given column

When used on varchar it work on alphabetical order for example MAX(first_name) will return name starting with Z

When used on date it give latest date

```
# To find minimum and maximum salary of department
SELECT
    T01F02,
    MIN(T02F07),
    MAX(T02F07)
FROM
    T02
```

```
INNER JOIN T01 ON (T01F01 = T02F08)
GROUP BY
    T02F08;
```

INNER, LEFT, RIGHT JOIN with examples

Join is used to combine two or more tables based on a related column between them

INNER JOIN

It is like intersection, it only give rows that satisfy condition

Syntax: `tbl_1 INNER JOIN tbl_2 [ON condition];`

Ex:

```
# To list employees with their department name
SELECT
    T02F02,
    T02F03,
    T01F02
FROM
    T01
    INNER JOIN T02 ON T01F01 = T02F08;
```

LEFT JOIN

A LEFT JOIN (or LEFT OUTER JOIN) returns all rows from the left table, and the matched rows from the right table. If there is no match in the right table, the columns from the right table will have NULL values.

Syntax: `tbl_1 LEFT JOIN tbl_2 ON condition;`

```
# To list all employee (also who is not associated with any department)
SELECT
    T02F02,
    T02F03,
    T01F02
FROM
    T02
    LEFT JOIN T01 ON T01F01 = T02F08;
```

RIGHT JOIN

A RIGHT JOIN (or RIGHT OUTER JOIN) returns all rows from the right table, and the matched rows from the left table. If there is no match in the left table, the columns from the left table will have NULL values.

Syntax: tbl_1 RIGHT JOIN tbl_2 ON condition;

```
# To list department with no employee
SELECT
    T01F02
FROM
    T02
    RIGHT JOIN T01 ON T01F01 = T02F08
WHERE
    T02F01 IS NULL;
```

NATURAL JOIN

A natural join is same as inner join but automatically joins table based on all columns that share the same name

Not recommended to use as it give less control on query

Syntax: tbl_1 NATURAL JOIN tbl_2;

CROSS JOIN

A cross join is cartesian product of tables it return all possible combination of rows

Syntax: tbl_1 CROSS JOIN tbl_2 [WHERE condition];

It is often used with where clause

Example:

```
# To list employee with their department name
SELECT
    T02F02,
    T02F03,
    T01F02
FROM
    T01
    CROSS JOIN T02
WHERE
    T01F01 = T02F08;
```

To get data from two table having foreign key inner join is more fast and effective as it use index to quickly get data while cross join first get cartesian product and then apply where condition

TCL - Transaction Control Language

Transaction is operation on database which follow ACID property

Atomicity - Either transaction is fully completed or rollbacked it can be in intermediate state for longer duration

Consistency - Once transaction is complete database should be in valid state means no database constraint should be violated

Isolation - One transaction should not interfere with other (Ex. Ticket booking)

Durability - Once transaction is committed data should be **permanently** stored in database and can't be corrupted if system crash or shut down

Commands used in TCL are START TRANSACTION, COMMIT, ROLLBACK

START TRANSACTION - It will start new transaction and any update to data will not be done in DB if other user try to fetch data it will give old data

COMMIT - It write data back to DB and now transaction is finished if other user will try to fetch data he will get updated data

ROLLBACK - It will revert transaction (undo its effect) It will restore DB back to state of before transaction started

Ex:

```
START TRANSACTION;
DELETE FROM T04;
SELECT * FROM T04;
ROLLBACK;
SELECT * FROM T04;
```

at end T04 is not updated and data is safe

Ex:

```
START TRANSACTION;
DELETE FROM T04 WHERE T04F01 = 1;
SELECT * FROM T04;
SAVEPOINT 1_DELETED;
DELETE FROM T04;
```

```
SELECT * FROM T04;  
ROLLBACK TO SAVEPOINT 1_DELETED;  
COMMIT;  
SELECT * FROM T04;
```

at end only one row is deleted from T04;

AUTOCOMMIT : It is property of DATABASE which can be changed by SET AUTOCOMMIT = 0 or 1;

When we run any command on database it will create a new transaction and committed if AUTOCOMMIT is ON

But if its off we have to manually COMMIT or ROLLBACK it

When we commit or rollback a new transaction is always created so we can run COMMIT and ROLLBACK any time without getting error

DCL - Data Control Language

DCL is used to manage access rights and permission within database

It include two main commands GRANT and REVOKE

GRANT

Grant give specific permission to a user

Syntax :

```
GRANT privilege(s)  
ON object_type.object_name  
TO 'user'@'host' [WITH GRANT OPTION];
```

privileges → SELECT, INSERT, UPDATE, DELETE, ALL PRIVILEGES

object_type object_name → database_name.* or database_name.table_name or *.*

user - username

host - ip address, localhost or 192.168.1.100 or % (can be connect from anywhere)

WITH GRANT OPTION - allow user to give the permission to other user

REVOKE

Revoke takeaway permission from user

```
REVOKE privilege(s)  
ON object_type object_name  
FROM 'user'@'host';
```

When we revoke permission from user it will also remove WITH GRANT OPTION implicitly as user can only permission to other which he have

To only remove WITH GRANT OPTION

```
REVOKE GRANT OPTION
ON object_level
FROM 'user'@'host';
```

use command FLUSH PRIVILEGES to apply it on DB

To view current permission

```
SHOW GRANTS FOR 's_dev'@'localhost';
```

In output of Show Grants

```
GRANT USAGE ON *.* TO 'TEST'@'localhost'
```

this line indicates that user has permission to connect to database it is minimum privilege given by MYSQL to all users

```
CREATE USER 'TEST'@'LOCALHOST' IDENTIFIED BY 'PASSWORD';
CREATE USER 'TEST1'@'LOCALHOST' IDENTIFIED BY 'PASSWORD';
```

```
DROP USER 'TEST'@'LOCALHOST';
DROP USER 'TEST1'@'LOCALHOST';
```

```
SHOW GRANTS FOR 'TEST'@'LOCALHOST';
SHOW GRANTS FOR 'TEST1'@'LOCALHOST';
```

```
GRANT SELECT ON LibraryDB.* TO 'TEST'@'LOCALHOST';
REVOKE SELECT ON LIBRARYDB.* FROM 'TEST'@'LOCALHOST';
```

```
GRANT ALL PRIVILEGES ON *.* TO 'TEST1'@'LOCALHOST' WITH GRANT OPTION; -- to give all
privileges (similar to admin)
```

```
GRANT SELECT ON LIBRARYDB.* TO 'TEST1'@'LOCALHOST' WITH GRANT OPTION; -- to give
select privileges
```

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'TEST1'@'LOCALHOST'; -- to revoke all privil
eges
```

```
FLUSH PRIVILEGES;
```

Logs

There are many types of logs

Error log

The Error Log is the most important log for basic troubleshooting. It's where MySQL writes information about critical errors, warnings, and informational messages that occur during the server's lifecycle.

- **What it logs:**
 - Server startup and shutdown events.
 - Critical errors that cause the server to shut down unexpectedly (e.g., running out of memory, table corruption).
 - Warnings about deprecated features or potentially problematic configurations.
 - Information about the replication threads on a replica server.
 - Events from the InnoDB storage engine, such as crash recovery processes.
- **Why it's used:**
 - It's the very first place you should look when the MySQL server fails to start or shuts down unexpectedly.
 - Diagnosing and resolving server-side issues.
 - Monitoring the health of the server.
- **How to manage it:**
 - The error log is **enabled by default** and cannot be disabled.
 - The location is controlled by the `log_error` system variable.

To get location

```
SHOW VARIABLES LIKE 'log_error';
```

General Query Log

This log record everything from connection disconnection to all sql statements

It is very slow as required too much disk write and should be used for debugging only

to enable

```
SET GLOBAL general_log = 'ON';
```

to disable

```
SET GLOBAL general_log = 'OFF';
```

To get its location


```
SHOW VARIABLES LIKE 'general_log_file';
```

Slow query log

This log is the primary tool for performance optimization. It helps you find and fix inefficient queries that are slowing down application.

It logs query that take more time than specified or we can also enable to log query that do not use index

This log is used to optimize query

To enable

```
SET GLOBAL slow_query_log = 'ON';
```

To set time

```
SET GLOBAL long_query_time = 2;
```

To set it to index mode

```
SET GLOBAL log_queries_not_using_indexes = 'ON';
```

To get slow log file

```
SHOW VARIABLES LIKE 'slow_query_log_file';
```

Binary log

It store all statement that change database state like CREATE, ALTER, DROP, UPDATE, INSERT, DELETE but it does not log SELECT query which does not modify database

It store log in binary format

Uses :

- **Replication:** This is its primary purpose. A source (master) server writes all changes to its binary log. Replica (slave) servers read this log and replay the events to keep their own data synchronized.
- **Point-in-Time Recovery (PITR):** The binary log is essential for disaster recovery. You can restore a full backup and then "replay" the binary logs up to a specific moment in time (e.g., right before a developer accidentally dropped a table).

format:

STATEMENT : logs SQL statements can be non-deterministic in case of function like (UUID())

ROW : Logs the change in individual rows, default mode

MIXED : STATEMENT based but if its unsafe it switch to ROW

Relay Log

This log is specific to replica servers in a replication setup.

- It decouples the process of reading from the source and executing the events on the replica. This allows the SQL thread on the replica to execute the events from a local file without having to wait for network I/O from the source, improving replication performance and resilience.
- It's a copy of the events from the source server's binary log. The replica's I/O thread reads events from the source's binlog and writes them to its own relay log.
- The relay log is managed automatically by the replica server when replication is configured. Its file name is controlled by the relay_log variable.

LOCKS

Lock is used to feature ISOCATION property of transaction

There is two types of lock

1. Shared (SELECT)
2. Exclusive (UPDATE, DELETE, INSERT)

There is two levels of lock

1. Table level : Use less memory, not convient for concurrent opeation, simple
2. row level : Use more memory, better for concurrent operation, complex, Used by INNODB

Locks may cause deadlock but INNODB handles it, it identify deadlock find victim and rollback it

When LOCK TABLE is executed it implicitly release all lock

When START TRANSACTION is executed it implicitly release all locks

If you lock a table explicitly with `LOCK TABLES`, any tables related by a foreign key constraint are opened and locked implicitly. For foreign key checks, a shared read-only lock (`LOCK TABLES READ`) is taken on related tables. For cascading updates, a shared-nothing write lock (`LOCK TABLES WRITE`) is taken on related tables that are involved in the operation.

If you lock a table explicitly with `LOCK TABLES`, any tables used in triggers are also locked implicitly

To put lock on table

```
LOCK TABLE table_name WRITE | READ;
```

Ex:

```
LOCK TABLE T01 WRITE;
```

To release lock (It will release all locks hold by current session)

```
UNLOCK TABLE;
```

To put lock on specific rows

```
START TRANSACTION;  
SELECT * FROM T01 WHERE T01F01 = 1 FOR UPDATE | FOR SHARE;  
COMMIT;
```

Row level update lock does not block read operation but it stops other transaction to acquire lock however if we try to access data it return last committed data.

If you want to update value after reading by SELECT get lock and then update else you get last committed data which may change because you get data by INNODB's MVCC (Multi version concurrency control)

INNODB

It is storage engine used by MYSQL

It provide feature like ACID combability, row level locking, foreign key support, crash recovery support, buffer pool

Earlier version of MYSQL use MyISAM storage engine which lacks in above features