




# Week 1 - MYSQL

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

## 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]

RESTRICT : Default Behaviour, If any other table is dependent it will throw an error and does not delete table

CASCADE : If any other table if dependent it will delete that table also

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;

# 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)**

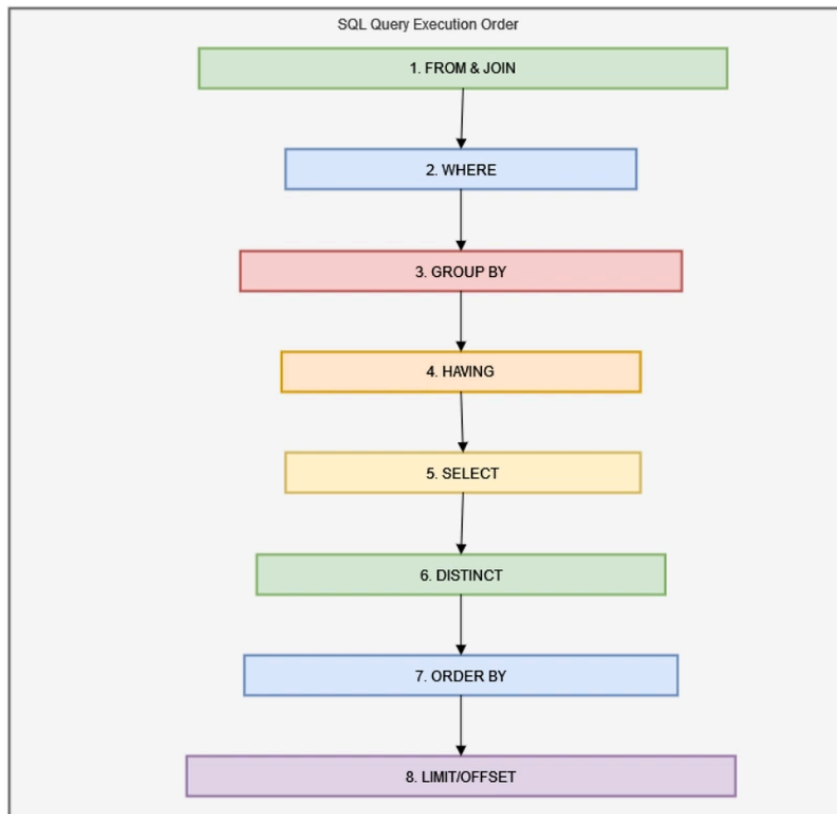
Syntax : DELETE FROM table\_name WHERE condition;

Ex : DELETE FROM T02 WHERE T02F01 = 108;

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

## 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