LAB Sheet:

Setting Up and Connecting to the MySQL Server in XAMPP

Using DB with xampp shell

1. (Create, Select, show, drop) Database, (Create, alter, rename, drop) on table

Create: CREATE DATABASE databasename;

Show: Show databases;

Select Database: use databasename;

Drop: DROP DATABASE databasename;

Create Table:

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

Example:

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

Alter Table- Add Column:

```
ALTER TABLE table_name
ADD column_name datatype;
```

Example:

```
ALTER TABLE Customers
ADD Email varchar(255);
```

ALTER TABLE - DROP COLUMN

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Example:

```
ALTER TABLE Customers
DROP COLUMN Email;
```

Rename Table:

```
ALTER TABLE old_table_name RENAME new_table_name;
```

Drop Table: `DROP TABLE table_name;`

2. (Insert, Update, Delete, Select) record, add primary key

Insert:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

Example:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City,
PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen
21', 'Stavanger', '4006', 'Norway');
```

Update and where:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Example:

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

Delete:

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

Example:

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

Select:

```
SELECT column1, column2, ...
FROM table_name;
//
SELECT * FROM table_name;
```

Example:

```
SELECT CustomerName, City FROM Customers;
```

Add primary Key:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

//

```
ALTER TABLE table_name
ADD PRIMARY KEY (column_name);
```

Example:

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

Drop Primary Key Constraint:

```
ALTER TABLE table_name
DROP PRIMARY KEY;
```

3. MySQL(where clause, Distinct clause, from clause, Group By, Having clause)

Where Clause and Form Clause:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example:

```
SELECT * FROM Customers
WHERE Country='Mexico';
//
SELECT * FROM Customers
WHERE CustomerID=1;
```

Distinct clause:

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

Example:

```
SELECT DISTINCT Country FROM Customers;
```

Count:

```
//Number of the distinct (Different) Country
SELECT COUNT(DISTINCT Country) FROM Customers;
```

Group By and Order By:

```
SELECT column_name(s)
FROM table_name
WHERE condition
```

```
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Example:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
//
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

Having clause

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

Example:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

4. (AND, OR, LIKE, NOT) Condition

AND:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

Example:

```
SELECT * FROM Customers
WHERE Country='Germany' AND City='Berlin';
```

OR:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

Example:

```
SELECT * FROM Customers
WHERE City='Berlin' OR City='München';
```

NOT:

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Example:

```
SELECT * FROM Customers
WHERE NOT Country='Germany';
```

Like:

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Example:

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
//
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

5. Join operation (Cross, self) two tables (Find the maximum, minimum value), Union

Join Cross:

```
SELECT column_name(s)
FROM table1
CROSS JOIN table2;
```

Example:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
CROSS JOIN Orders;
```

Join (Self):

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

Example:

```
SELECT A.CustomerName AS CustomerName1,
B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City;
```

Max:

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

Example:

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

Min:

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

Example:

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

Union:

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

Example:

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers;
```

6. Aggregate Function (sum, avg, min, max)

Sum:

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

Example:

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

Avg:

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

Example:

```
SELECT AVG(Price)
FROM Products;
```

Max:

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

Example:

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

Min:

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

Example:

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

7. Triggers Operations (Create, show, drop, Before, After)

Types of SQL triggers:

1. Row level trigger – An event is triggered at row level i.e. for each row updated, inserted or deleted.
2. Statement level trigger – An event is triggered at table level i.e. for each sql statement executed.

Syntax for creating a trigger:

```
CREATE [OR REPLACE] TRIGGER trigger_name
```

```
 {BEFORE | AFTER | INSTEAD OF }
 {INSERT [OR] | UPDATE [OR] | DELETE}
 [OF col_name]
 ON table_name
 [REFERENCING OLD AS o NEW AS n]
 [FOR EACH ROW]
 WHEN (condition)
 BEGIN
   --- sql statements
 END;
/
```

Where:

CREATE [OR REPLACE] TRIGGER trigger_name – It creates a trigger with the given name or overwrites an existing trigger with the same name.
{BEFORE | AFTER | INSTEAD OF } – It specifies the trigger get fired. i.e before or after updating a table. INSTEAD OF is used to create a trigger on a view.
{INSERT [OR] | UPDATE [OR] | DELETE} – It specifies the triggering event. The trigger gets fired at all the specified triggering event.
[OF col_name] – It is used with update triggers. It is used when we want to trigger an event only when a specific column is updated.
[ON table_name] – It specifies the name of the table or view to which the trigger is associated.
[REFERENCING OLD AS o NEW AS n] – It is used to reference the old and new values of the data being changed. By default, you reference the values as :old.column_name or :new.column_name. The old values cannot be referenced when inserting a record and new values cannot be referenced when deleting a record, because they do not exist.
[FOR EACH ROW] – It is used to specify whether a trigger must fire when each row being affected (Row Level Trigger) or just once when the sql statement is executed (Table level Trigger).
WHEN (condition) – It is valid only for row level triggers. The trigger is fired only for rows that satisfy the condition specified.

Example:

Existing data:

Select * from employees;

```
EMP_ID      NAME  AGE   ADDRESS     SALARY
1     Shveta      23    Delhi 50000
2     Bharti      22    Karnal      52000
3     Deepika     24    UP    54000
4     Richi 25    US    56000
5     Bharat      21    Paris 58000
6     Sahdev      26    Delhi 60000
```

Trigger:

```
CREATE OR REPLACE TRIGGER show_salary_difference
BEFORE DELETE OR INSERT OR UPDATE ON employees
FOR EACH ROW
WHEN (NEW.EMP_ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

Note: The above trigger will execute for every INSERT, UPDATE or DELETE operations performed on the EMPLOYEES table.

Drop a trigger:

```
DROP TRIGGER trigger_name;
```

8. Connect Database with website (PHP)

Connect.php file:

```
<?php
$serverName = "localhost";
$username = "root";
$password = "";
```

```
$dbName = "DB_Name";

//creating connection
$conn = new mysqli($serverName, $username, $password);
//check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
} else {
    mysqli_select_db($conn, $dbName);
   // echo "Connection Successful";
}
?>
```

For Each Php connection just call

```
require_once('DBconnect.php');
```