Index

Experiment No: 01

Experiment Name: Study and Implementation of DML Commands of SQL with Suitable (Insert, Delete, Update)

Objectives:

i. To insert elements in a database
ii. To delete elements in a database
iii. To update element in a database

Theory:

Structured collection of data that is organized in a way that allows for efficient storage, retrieval, and manipulation of information. SQL (Structured Query Language) is a programming language used for managing and manipulating relational databases. In this experiment, focusing on Data Manipulation Language (DML) commands of SQL, which are used to interact with the data stored in the database. The three main DML commands, i.e., insertion, deletion, and updating of data in a database

The SQL statement INSERT INTO is used to insert new rows of data into a table in the database. Almost all the RDBMS provide this SQL query to add the records in database tables.

The syntax of INSERT INTO statement is

INSERT INTO TABLE_NAME (column1, column2...columnN) VALUES (value1, value2...valueN);

The SQL statement insert new column the SQL query is

ALTER TABLE *table_name* ADD *column_name datatype*;

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

ALTER TABLE *table_name* DROP COLUMN *column_name*;

Delete a record from table

DELETE FROM table_name WHERE condition;

To rename a column in a table, use the following syntax:

ALTER TABLE *table_name* RENAME COLUMN *old_name* to *new_name*;

To update records in a table using SQL, you can use the UPDATE statement. Here's the basic syntax:

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

Example

```
UPDATE customers SET email = 'newemail@example.com'
WHERE customer_id = 5;
```

Code:

```
create database university
use university
create table department(
                        dept_name varchar(20),
                        building varchar (15),
                        budget numeric(8,2),
                        primary key(dept_name)
);
insert into department values('ICE','Engineering',87000)
insert into department values('CSE','Engineering',90000)
insert into department values('EEE','Science',95000)
insert into department values('EECE','Science',80000)
insert into department values('BANGLA','BANGLA',68000)
insert into department values('ENGLISH','ENGLISH',55000)

select * from department
--deleting
delete from department where dept_name ='CSE'
select * from department
 --update
 update department set budget = budget + budget*1.05 where budget <90000
 select * from department
```

Output:
After inserting the values the table is

	dept_name	building	budget
1	BANGLA	BANGLA	139400.00
2	CSE	Engineering	90000.00
3	EECE	Science	164000.00
4	EEE	Science	95000.00
5	ENGLISH	ENGLISH	112750.00
6	ICE	Engineering	178350.00

After deleting value the table is

	dept_name	building	budget
1	BANGLA	BANGLA	139400.00
2	EECE	Science	164000.00
3	EEE	Science	95000.00
4	ENGLISH	ENGLISH	112750.00
5	ICE	Engineeri…	178350.00

After updating the table is

	dept_name	building	budget
1	BANGLA	BANGLA	139400.00
2	EECE	Science	164000.00
3	EEE	Science	95000.00
4	ENGLISH	ENGLISH	112750.00
5	ICE	Engineeri…	178350.00

Experiment No: 02

Experiment Name: Study and Implementation of DDL Commands of SQL with Suitable. Example (Create, Alter, Drop)

Objectives:

(i) To study and implement how to create a table in a database
(ii) To study and implement how to alter a table in database
(iii) To study and implement how to drop a record or attribute

Theory: In a database the for implementing the DDL commands of SQL with suitable are given below.

(i) CREATE:

The CREATE command in SQL is used to create objects in a database. The primary object that is created using CREATE is a table, but it can also be used to create other objects like indexes, views, and databases (depending on the DBMS).

Examples

For creating a table

```
CREATE TABLE table_name (
   column1 datatype1 constraints,
   column2 datatype2 constraints,
   ...
);
```

For creating an index

```
CREATE INDEX index_name ON table_name (column_name);
```

(i) ALTER:

The ALTER command is used to modify the structure of an existing database object. It can be used to add, modify, or drop columns, constraints, indexes, etc.

Examples:

(a)Adding a Column:

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

(b) Modifying a Column:

```
ALTER TABLE table_name MODIFY column_name new_datatype;
```

This allows you to change the datatype of an existing column.

(c) Dropping a Column:

ALTER TABLE table_name DROP COLUMN column_name;

(iii) DROP:

The DROP command is used to remove objects from the database. Be cautious when using this command, as it permanently deletes data.

Examples:

Dropping a Table:

DROP TABLE table_name;

This deletes an entire table from the database.

Dropping an Index:

DROP INDEX index_name;

Code:

```
create database university
use university
create table instructor(
ID varchar(20),
name varchar(15) not null,
dept_name varchar(15),
salary numeric(8,2),
primary key(ID));
insert into instructor(ID,name,dept_name,salary) values('200610','alamin','ICE','86000')
insert into instructor(ID,name,dept_name,salary) values('200611','Nirob','CSE','80000')
insert into instructor(ID,name,dept_name,salary) values('200601','Naima
Islam','EEE','70000')
insert into instructor(ID,name,dept_name,salary) values('200622','Sajeeb
kumur','EECE','90000')
insert into instructor(ID,name,dept_name,salary) values('200605','Uamme
kulsum','CE','95000')
insert into instructor(ID,name,dept_name,salary) values('200600','Gopal bhar','Arct','68000')
select * from instructor
alter table instructor add course_id varchar(20)

select * from instructor
drop table instructor

select * from instructor
```

Output:

Create a table

	ID	name	dept_name	salary
1	200600	Gopal bhar	Arct	68000.00
2	200601	Naima Islam	EEE	70000.00
3	200605	Uamme kulsu…	CE	95000.00
4	200610	alamin	ICE	86000.00
5	200611	Nirob	CSE	80000.00
6	200622	Sajeeb kumur	EECE	90000.00

Alter a table

	ID	name	dept_name	salary	course_id
1	200600	Gopal bhar	Arct	68000.00	NULL
2	200601	Naima Islam	EEE	70000.00	NULL
3	200605	Uamme kulsu…	CE	95000.00	NULL
4	200610	alamin	ICE	86000.00	NULL
5	200611	Nirob	CSE	80000.00	NULL
6	200622	Sajeeb kumur	EECE	90000.00	NULL

Drop a table

Experiment No: 03

Experiment Name: Study and Implementation of DML Commands of (Select Clause , From Clause, Where Clause)

Objectives:

(i) To study and implement how select clause table in a database
(ii) To study and implement how from clause table in a database
(iii) To study and implement how where clause table in a database

Theory:

DML (Data Manipulation Language) commands: SELECT, FROM, and WHERE clauses in SQL.

(i) SELECT Clause:

The SELECT statement is used to retrieve data from a database. It is one of the most fundamental and frequently used commands in SQL.

SELECT column1, column2, ...FROM table_name;

Example:

SELECT first_name, last_name FROM customers;

This query retrieves the first_name and last_name columns from the customers table.

(ii) FROM Clause:

The FROM clause specifies the source table or tables from which to retrieve data.

Syntax:

SELECT column1, column2, ...FROM table1, table2, ...;

(iii) WHERE Clause:

The WHERE clause is used to filter records based on a specified condition.

Syntax:

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

Example:

SELECT *FROM products WHERE category = 'Electronics' AND price > 500;

This query retrieves all columns from the products table where the category is 'Electronics' and the price is greater than 500.

Code:

```
create database university
use university
create table insertvalue(
                        dept_name varchar(15),
                        bulding varchar(15),
                        budget numeric(8,2)
                        primary key(dept_name)

);
insert into insertvalue values('ICE','Engineering',87000)
insert into insertvalue values('CSE','Engineering',90000)
insert into insertvalue values('EEE','JHON',95000)
insert into insertvalue values('EECE','Watson',80000)
insert into insertvalue   values('BANGLA','BANGLA',68000)
insert into insertvalue values('ENGLISH','ENGLISH',55000)

select * from insertvalue
select dept_name from insertvalue
select dept_name from insertvalue where dept_name = 'EECE'
```

Output:

Select clause

	dept_name	bulding	budget
1	BANGLA	BANGLA	68000.00
2	CSE	Engineeri…	90000.00
3	EECE	Watson	80000.00
4	EEE	JHON	95000.00
5	ENGLISH	ENGLISH	55000.00
6	ICE	Engineeri…	87000.00

From clause

	dept_name
1	BANGLA
2	CSE
3	EECE

4	EEE
5	ENGLISH
6	ICE

Where clause

dept_name

1	EEC

Experiment No:04

Experiment Name: Study and Implementation of DML Commands of

- Group By & Having Clause
- Order By Clause
- Create View, Indexing & Procedure Clause

Objectives:

(i) To understand and implement the data definition language for using the Group by & Having Clause

(ii) To understand and implement the data definition language for using the Order By clause

(iii) To understand and implement the data definition language for using the Create View, Indexing & Procedure Clause

Theory:

DML (Data Manipulation Language) commands in SQL.

Group By & Having Clause:

1. Group By Clause:

The GROUP BY clause is used to group rows with identical data into summary rows. It is often used with aggregate functions like COUNT, SUM, AVG, etc.

Syntax:

SELECT column1, aggregate_function(column2)

FROM table_name

GROUP BY column1;

column1: The column by which you want to group the data.

aggregate_function(column2): An aggregate function applied to column2.

2. Having Clause:

The HAVING clause works like a WHERE clause but is used specifically with aggregate functions. It filters the results after they have been grouped.

Syntax:

SELECT column1, aggregate_function(column2)

FROM table_name

GROUP BY column1

HAVING condition;

condition: The condition that must be met for a group to be included in the result set.

3.Order By Clause:

The ORDER BY clause is used to sort the result set based on one or more columns.

SELECT column1, column2, ...

FROM table_name

ORDER BY column1 [ASC | DESC], column2 [ASC | DESC], ...;

column1, column2, ...: The columns by which you want to sort.

ASC: Ascending order (default).

DESC: Descending order.

Create View, Indexing & Procedure:

(i). Create View:

A view is a virtual table that is based on the result of a SELECT query. It does not store the data itself, but it provides a way to represent complex queries in a simplified form.

Syntax:

CREATE VIEW view_name AS

SELECT column1, column2, ...

FROM table_name

WHERE condition;

(ii). Indexing:

Indexes are data structures that improve the speed of data retrieval operations on a table at the cost of additional storage and decreased performance on data modification operations (like INSERT, UPDATE, DELETE).

Syntax to Create an Index:

CREATE INDEX index_name

ON table_name (column1, column2, ...);

Syntax to Drop an Index:

DROP INDEX index_name;

(iii). Procedure:

A stored procedure is a set of SQL statements that can be stored in a database and executed by calling the procedure. It helps in modularizing and reusing code.

Syntax to Create a Procedure:

CREATE PROCEDURE procedure_name

AS

BEGIN

  -- SQL Statements

END;

Syntax to Execute a Procedure:

EXEC procedure_name;

These DML commands provide advanced capabilities for querying and managing data in a database. Remember to replace column_name, table_name, and other placeholders with actual names from your database.

Code:

```
----create alter and drop
create database uinversity
use university
create table instructor(
        ID varchar(20),
```

```
        name varchar(20) not null,
        dept_name varchar(20),
        salary numeric(8,2),
        primary key(ID)
);
insert into instructor  values ('10101','Srinivasan','Comp.Sci',65000);
insert into instructor  values ('12121','Wu','Finance',90000);
insert into instructor values ('15151','Mozart','Music',40000);
insert into instructor  values ('22222','Einstein','Physics',95000);
insert into instructor values ('32343','EI Said','History',60000);
insert into instructor values ('33456','Gold','Physics',87000);
select * from instructor
select dept_name from instructor
---group by
select name from instructor group by name;
select dept_name,avg(salary) as avg_salary from instructor group by dept_name
select dept_name,count(*) from instructor group by dept_name
select * from instructor
---having clause
select dept_name,avg(salary) as avg_salary from instructor group by dept_name having
avg(salary)>55000;
----order by clause
select * from instructor order by salary asc,name desc;
---view
create view faculty as
select ID,name,dept_name from instructor
select * from instructor
----index
create index dept_inx on instructor(dept_name)
---procedure
create procedure instruct_proc
AS
BEGIN
select name as authors_name from instructor where ID = '15151'
END
exec instruct_proc
```

Output:

Group By Clause

	dept_name	avg_salary
1	Comp.Sci	65000.000000
2	Finance	90000.000000
3	History	60000.000000
4	Music	40000.000000
5	Physics	91000.000000

Order By (Ascending order)

	ID	name	dept_name	salary
1	15151	Mozart	Music	40000.00
2	32343	EI Said	History	60000.00
3	10101	Srinivas…	Comp.Sci	65000.00
4	33456	Gold	Physics	87000.00
5	12121	Wu	Finance	90000.00
6	22222	Einstein	Physics	95000.00

Create view as faculty

	ID	name	dept_name
1	10101	Srinivas…	Comp.Sci
2	12121	Wu	Finance
3	15151	Mozart	Music
4	22222	Einstein	Physics
5	32343	EI Said	History
6	33456	Gold	Physics

Experiment No: 05

Experiment Name: Study and Implementation of SQL Commands of Join Operations with Example
□Cartesian Product, Natural Join ,□Left Outer Join ,□Right Outer Join ,□Full Outer Join

Objectives:

(i)To understand and implement the Cartesian product, Natural join in a database

(ii) To understand and implement the Left Outer Join ,□Right Outer Join in a database

(iii) To understand and implement the Full Outer Join in a database

Theory:

1. Cartesian Product:

The Cartesian Product combines every row from the first table with every row from the second table. It results in a table with m x n rows (where m is the number of rows in the first table and n is the number of rows in the second table).

Syntax:

SELECT * FROM table1 CROSS JOIN table2;

Example

Consider two tables A and B:

Table A:		Table B:	
ID	Name	Course	Grade
1	Alice	Math	A
2	Bob	Science	B

The Cartesian Product (A x B) will be:

ID	Name	Course	Grade
1	Alice	Math	A
1	Alice	Science	B
2	Bob	Math	A
2	Bob	Science	B

2. Natural Join:

A Natural Join combines two tables based on columns with the same name and data type. It eliminates duplicate columns, keeping only one instance of each column.

Syntax:

SELECT * FROM table1 NATURAL JOIN table2;

Example:

Consider two tables A and B:

Table A:			Table B	
ID	Name	Course	ID	Grade
1	Alice	Math	1	A
2	Bob	Science	2	B

The Natural Join (A NATURAL JOIN B) will be:

ID	Name	Course	Grade
1	Alice	Math	A
2	Bob	Science	B

3. Left Outer Join:

A Left Outer Join returns all the records from the left table (first table) and the matched records from the right table (second table). The result will contain NULL values for the columns from the right table where there is no match.

Syntax:

SELECT * FROM table1

LEFT JOIN table2 ON table1.column = table2.column;

4. Right Outer Join:

A Right Outer Join is similar to a Left Outer Join, but it returns all the records from the right table and the matched records from the left table. The result will contain NULL values for the columns from the left table where there is no match

Syntax:

SELECT * FROM table1 RIGHT JOIN table2 ON table1.column = table2.column;

Example:

Consider two tables A and B:

Table A:		Table B:	
ID	Name	ID	Grade
1	Alice	1	A
2	Bob	2	B
3	Charlie		

The Left Outer Join (A LEFT JOIN B ON A.ID = B.ID) will be:

ID	Name	Grade
1	Alice	A
2	Bob	B
3	Charlie	NULL

The Right Outer Join (A RIGHT JOIN B ON A.ID = B.ID) will be:

ID	Name	Grade
1	Alice	A
2	Bob	B
NULL	NULL	C

5. Full Outer Join:

A Full Outer Join returns all records when there is a match in either left or right table. It returns NULL values for unmatched columns on either side.

Syntax:

SELECT * FROM table1 FULL JOIN table2 ON table1.column = table2.column;

Example

Consider two tables A and B:

Table A:		Table B:	
ID	Name	ID	Grade
1	Alice	1	A
2	Bob	3	B

The Full Outer Join (A FULL JOIN B ON A.ID = B.ID) will be:

ID	Name	Grade
1	Alice	A
2	Bob	NULL
NULL	NULL	B

Code:

```
use university

create table depart(

        dept_name varchar(20),

                bulding varchar(20),
```

```
                budget numeric(8,2),
                primary key(dept_name)
);
insert into depart values('ICE','Watson','90000')
insert into depart values('CSE','Science','85000')
insert into depart values('EEE','Engineering','80000')
insert into depart values('CE','Engineering','68000')
insert into depart values('EECE','Science','55000')
insert into depart values('Arct','Painter','95000')
create table instruct(
        ID varchar(20),
                name varchar(15) not null,
                dept_name varchar(15),
                salary numeric(8,2),
                primary key(ID));
insert into instruct(ID,name,dept_name,salary) values('1012','sumu','ICE','1000')
insert into instruct(ID,name,dept_name,salary) values('3245','summuu','CSE','1001')
insert into instruct(ID,name,dept_name,salary) values('3865','raiyan','BANGLA','1002')
insert into instruct(ID,name,dept_name,salary) values('4755','RIYA','ENGLISH','1003')
insert into instruct(ID,name,dept_name,salary) values('6789','MAHI','PHYSICS','10004')
select * from depart
select * from instruct
---cartesian product
select bulding,salary from instruct,depart where depart.dept_name = instruct.dept_name;
----join product
select ID,name,budget from instruct join depart on depart.dept_name = instruct.dept_name;
---left outer join
select * from instruct left outer join depart on depart.dept_name=instruct.dept_name;
---right outer join
select * from instruct right outer join depart on depart.dept_name=instruct.dept_name;
```

---full outer join

select * from instruct full outer join depart on depart.dept_name=instruct.dept_name;

Output:

---cartesian product

	bulding	salary
1	Watson	1000.00
2	Science	1001.00

----join product

	ID	name	budget
1	1012	sumu	90000.00
2	3245	summ	85000.00

---left outer join

	ID	name	dept_name	salary	dept_name	bulding	budget
1	1012	sumu	ICE	1000.00	ICE	Watson	90000.00
2	3245	sumu	CSE	1001.00	CSE	Science	85000.00
3	3865	raiyan	BANGLA	1002.00	NULL	NULL	NULL
4	4755	RIYA	ENGLISH	1003.00	NULL	NULL	NULL
5	6789	MAHI	PHYSICS	10004.00	NULL	NULL	NULL

---right outer join

	ID	name	dept_name	salary	dept_name	bulding	budget
1	NULL	NULL	NULL	NULL	Arct	Painter	95000.00
2	NULL	NULL	NULL	NULL	CE	Engineeri…	68000.00
3	3245	sumu	CSE	1001.00	CSE	Science	85000.00
4	NULL	NULL	NULL	NULL	EECE	Science	55000.00
5	NULL	NULL	NULL	NULL	EEE	Engineeri…	80000.00
6	1012	sumu	ICE	1000.00	ICE	Watson	90000.00

---full outer join

	ID	name	dept_name	salary	dept_name	bulding	budget
1	1012	sumu	ICE	1000.00	ICE	Watson	90000.00
2	3245	sumu	CSE	1001.00	CSE	Science	85000.00
3	3865	raiyan	BANGLA	1002.00	NULL	NULL	NULL
4	4755	RIYA	ENGLISH	1003.00	NULL	NULL	NULL
5	6789	MAHI	PHYSICS	10004.00	NULL	NULL	NULL
6	NULL	NULL	NULL	NULL	Arct	Painter	95000.00
7	NULL	NULL	NULL	NULL	CE	Engineeri…	68000.00
8	NULL	NULL	NULL	NULL	EECE	Science	55000.00
9	NULL	NULL	NULL	NULL	EEE	Engineeri…	80000.00

Experiment No: 06

Experiment Name: Study and Implementation of Aggregate Function with Example (Count Function, Max Function, Min Function Avg Function)

Objectives:

(i) To understand the different issues in the design and implementation of a database system
(ii) To apply and implement the Aggregate Function

Theory: SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value. It is also used to summarize the data. The five type of aggregation function is Count Function, Max Function ,Min Function Avg Function ,Sum

Function.

1. COUNT Function:

The COUNT function is used to count the number of rows that meet a specified condition.

Syntax:

SELECT COUNT(column_name) FROM table_name WHERE condition;

2. MAX Function:

The MAX function returns the highest value in a column.

Syntax:

SELECT MAX(column_name) FROM table_name WHERE condition;

Example:

Consider a table products:

ID	Product_Name	Price
1	Laptop	1200
2	Smartphone	800
3	Tablet	500

SELECT MAX(Price) FROM products;

This will return:

MAX(Price)
1200

3. MIN Function:

The MIN function returns the lowest value in a column.

Syntax:

SELECT MIN(column_name) FROM table_name WHERE condition;

Example:

Using the same products table:

SELECT MIN(Price) FROM products;

This will return:

markdown

Copy code

MIN(Price)

500

4. AVG Function:

The AVG function calculates the average value of a column.

Syntax:

SELECT AVG(column_name) FROM table_name WHERE condition;

Example:

Using the same products table:

SELECT AVG(Price) FROM products;

This will return:

markdown

AVG(Price)

833.33

These aggregate functions are invaluable when you need to perform calculations on sets of data, such as finding totals, averages, maximum and minimum values, etc. They allow you to summarize and analyze your data effectively

Code:

```
create database university
use university
create table  instructorSalary(
        ID varchar(20),
                        dept_name varchar(20),
                        salary numeric(8,2),
                        primary key(ID)
);

insert into instructorSalary values('1212','ICE','60000')
insert into instructorSalary values('1215','CE','77000')
insert into instructorSalary values('1219','CSE','85000')
insert into instructorSalary values('1214','EEE','65000')

select * from instructorSalary

select count(ID) as count_ID from instructorSalary
select max(salary) as max_salary from instructorSalary
select min(salary) as min_salary from instructorSalary
select avg(salary) as avg_salary from instructorSalary
select SUM(salary) as total_salary from instructorSalary
```

Output:

--Table

	ID	dept_name	salary
1	1212	ICE	60000.00
2	1214	EEE	65000.00
3	1215	CE	77000.00
4	1219	CSE	85000.00

--count

	count_ID
1	4

--max

	max_salary
1	85000.00

--min

	min_salary
1	60000.00

--Avg

	avg_salary
1	71750.000000

--sum

	total_salary
1	287000.00

Experiment No: 07

Experiment Name: Study and Implementation of Triggering System on Database Table Using SQL Commands with Example

Objectives:

- **(i)** To understand and implement the triggering system on database table using sql
- **(ii)** To applying triggering on database.
- (iii) To understand how to access the data within the trigger

Theory: An SQL trigger is a database object that is associated with a table and automatically executes a set of SQL statements when a specific event occurs on that table. Triggers are used to enforce business rules, maintain data integrity, and automate certain actions within a database. They can be triggered by various events, such as inserting, updating, or deleting data in a table, and they allow you to perform additional operations based on those events

A triggering system in a database allows you to define actions that should be automatically executed when certain events occur on a table, such as inserting, updating, or deleting records. These actions are defined using triggers, which are blocks of SQL code associated with a specific event on a table.

Syntax:

create trigger [trigger_name]

[before | after]

{insert | update | delete}

on [table_name]

[for each row]

[trigger_body]

Insert Trigger: When data is inserted into the original table, a trigger can automatically add the same data to a backup table. This ensures that a copy of the data is kept for future reference or recovery.

Delete Trigger: When data is deleted from the original table, a trigger can add the deleted data to a backup table. This is valuable for maintaining an audit trail or keeping a history of changes.

Code:

```
use University
create table instructor
( ID int, name nvarchar(50), dept_name nvarchar(50),salary int )
select * from instructor
insert into instructor values(22222,'Einstein','Physics',95000)
insert into instructor values(12121,'We','Finance',90000)
insert into instructor values(32343,'El Said','History',60000)
insert into instructor values(45565,'Katz','CSE',75000)
insert into instructor values(98345,'Kim','EEE',80000)
insert into instructor values(98346,'AL AMIN','ICE',80000)
select * from instructor
--create another table for update value keeping
create table backup_ins
( ID int, name nvarchar(50), dept_name nvarchar(50),salary int )
select * from backup_ins
--create another table for deleted value keeping
create table backup_del
( ID int, name nvarchar(50), dept_name nvarchar(50),salary int )
select * from backup_del
--creating trrigger
create trigger ins_trigger
on instructor
after insert
as begin
 print'The tigger inserted successfully'
end
--update trigger
alter TRIGGER ins_trigger
ON instructor
AFTER INSERT
AS
BEGIN
   INSERT INTO backup_ins(ID, name, dept_name, salary)
   SELECT ID, name, dept_name, salary
   FROM inserted;
END;
--deleted tigger
create TRIGGER del_trigger
ON instructor
AFTER DELETE
AS
BEGIN
   INSERT INTO backup_del (ID, name, dept_name, salary)
   SELECT ID, name, dept_name, salary
   FROM deleted;
END;
DELETE FROM instructor WHERE ID = 32343;
select * from backup_del
```

Output:

The orginal table instructor is

	ID	name	dept_name	salary
1	22222	Einstein	Physics	95000
2	12121	We	Finance	90000
3	32343	El Said	History	60000
4	45565	Katz	CSE	75000
5	98345	Kim	EEE	80000

After inserting one element the inserted table

	ID	name	dept_name	salary
1	98346	AL AMIN	ICE	80000

After deleted tuple from instructor table the backup table is

	ID	name	dept_name	salary
1	32343	El Said	History	60000

Experiment No: 08

Experiment Name: Study and Implementation of SQL Commands to Connect MySQL Database with Java or PHP.

Objectives:

1. To study and implement the php and html form for inserting information to the database in local server xampp

2. To create a database in xampp Mysql and connect with php

Theory: PHP is a server-side scripting language commonly used for web development. It is particularly well-suited for database interactions. MySQL is a popular open-source relational database management system. Connecting PHP with MySQL allows web applications to dynamically interact with and manipulate data stored in a MySQL database

The objective is to learn and apply SQL commands for connecting a MySQL database with PHP. This involves creating an HTML form to input data, establishing a connection to a local XAMPP server with MySQL, and executing PHP scripts to insert information into the database. Additionally, the goal is to create a database within XAMPP's MySQL environment and establish a connection using PHP, facilitating the seamless interaction between web-based forms and the underlying database. This exercise aims to provide hands-on experience in setting up a functional database-driven web application locally.

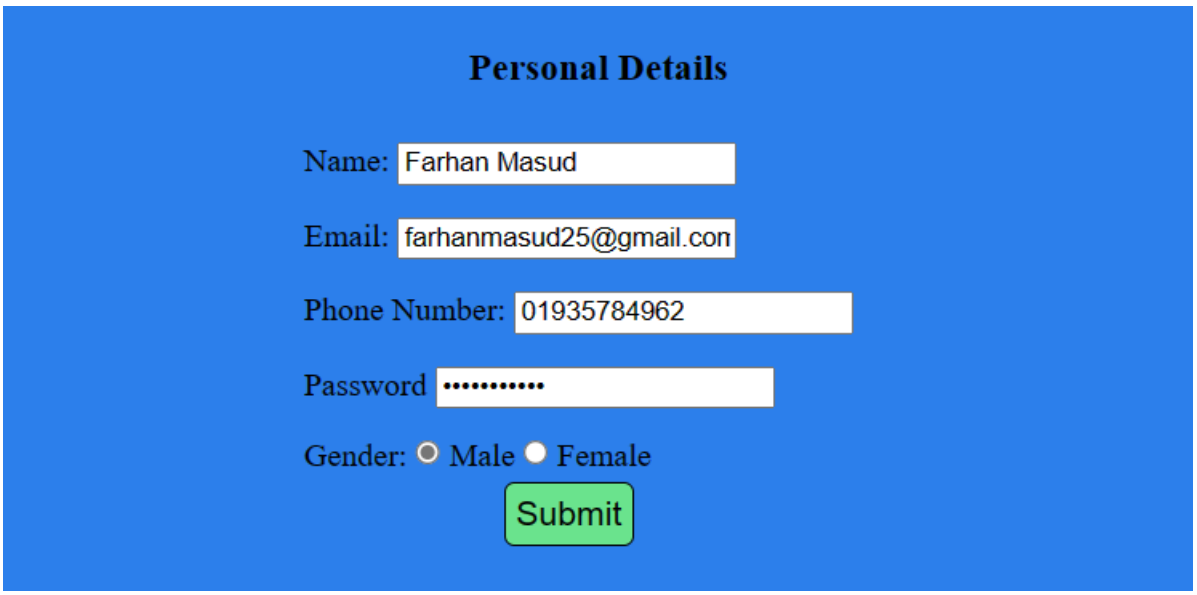
Code:

```
<?php
$base = mysqli_connect('localhost', 'root', '', 'insert');
if(isset($_POST['submit'])){
   $name = $_POST['name'];
   $email = $_POST['email'];
   $password = $_POST['password'];
   $sql = "INSERT INTO insertform(name, email, password) VALUES ('$name', '$email',
'$password')";
   if(mysqli_query($base, $sql)){
      echo "Inserted successfully";
   }
   else{
      echo "Insertion failed: " . mysqli_error($base); // Added error message for debugging
   }
}

mysqli_close($base); // Close the connection after use
?>
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
   <title>indert form</title>
   <style>
      body{
         background-color: antiquewhite;
         font-family: Arial, Helvetica, sans-serif;
      }
      h1{
         text-align: center;
      }

      label {
         font-weight: bold;
         margin-bottom: 5px;

      }

      input {
         width: 100%;
         padding: 8px;
         margin-bottom: 10px;
         border-radius: 8px;
         border-color: green;
      }
      input[type="submit"] {
         background-color: blueviolet;
         color: white;
         cursor: pointer;
         padding: 5px 5px;
         margin: 0 auto;
         display: block;

      }
   </style>
</head>
<body>
   <h1>Personal Details</h1>

   <form action="insert.php" method="POST">

      <label for="name">First Name : </label>
      <input type="text" id="name" name="name" placeholder="Enter your name"><br>
      <label for="email">Email : </label>
      <input type="email" id="email" name="email" placeholder="Enter valid email "><br>
      <label for="passward">Passward : </label>
      <input type="password" id="password" name="password" placeholder="Enter 6 digit
password"><br>
      <input type="submit" name="submit" value="submit">
   </form>
</body>
</ht
```

Output:

php form:



Mysql database: