PABNA UNIVERSITY OF SCIENCE AND TECHNOLOGY

Faculty of Engineering and Technology

Department of Information and Communication Engineering

Lab Report

Course Title: Database Management Systems Sessional

Course Code: ICE-3106

Submitted by

Name: Md.Ashikur Rahman
Roll No: **200607**
Reg No: **1065372**
Session:2019-2020
3rd year 1st semester
Department of Information and
Communication Engineering,
Pabna University of Science and Technology

Submitted to

Sohag Sarker
Associate Professor
Department of Information and
Communication Engineering,
Pabna University of Science and Technology

Submission Date: 05-11-2023

…………………………

Signature

Index

SL No	Experiment Name	Page No
01	Study and Implement the following DML Commands of SQL with suitable examples • Insert • Delete • Update	02
02	Study and Implement the following DDL Commands of SQL with suitable examples • Create • Alter • Drop	05
03	Study and Implement the following DML Commands • Select Clause • From Clause • Where Clause	08
04	Study and Implement the following DML Commands • Group By and Having Clause • Order By Clause • Create View, Indexing and Procedure Clause	11
05	Study and Implement the following SQL Commands of Join Operations with examples • Cartesian Product • Nutural Join • Left Outer Join • Right Outer Join • Full Outer Join	16
06	Study and Implement the following Aggregate Function with example • Count Function • Max Function • Min Function • Avg Function	22
07	Study and Implement the Triggering System on Database Table using SQL commands with sxamples.	26
08	Study and Implement the SQL Commands to connect MySQL Database with Java or PHP.	29

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:

```
use master;
--To create database
create database university_2000607
use university_200607
-- Create Table
create table department(
dept_name varchar(20),
building varchar(15),
budget numeric(12,2),
primary key(dept_name));
--Insert Value in Table
insert into department values('ICE','Watson',90000);
insert into department values('CSE','Taulor',100000);
insert into department values('EECE','Packard',50000);
--To Display all value of Department Table
select * from department;
--To Delete Department Table with one tuple
delete from department where dept_name='EECE';
--Delete Table value
delete from department;
-- To update Department table
update department set budget=budget*1.5 where budget<85000;select * from department
```

Output:

After inserting the values the table is

	dept_name	Name	budget
1	ICE	Watson	90000
2	CSE	Taulor	100000
3	EECE	packard	50000

After deleting value the table is

	dept_name	Name	budget
1	ICE	Watson	90000
2	CSE	Taulor	100000

After updating the table is

	dept_name	Name	budget
1	ICE	Watson	139400.00
2	CSE	Taulor	164000.00

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:

```
use master;
--Create Database
create database university_200607
use university_200607
-- Create Table
create table instructor(
ID varchar(5),
name varchar(20) not NULL,
dept_name varchar(20),
salary numeric(8,2),
primary key(ID));
--Insert Value in Table
insert into instructor values('101','ASHIK','ICE',90000);
insert into instructor values('102','NOYON','CSE',20000);
insert into instructor values('103','SOBUZ','EEE',100000);

--To Display all value of Department Table
```

```
select * from instructor;

--To alter (add, rename, drop)

alter table instructor add course_no char(20) default 'ICE';

insert into instructor(ID,name,dept_name,salary)  values('104','SUHAN','EEE',100000);

alter table instructor drop column course_no;

--Drop Table

drop table instructor;
```

Output:

Create a table

	ID	name	dept_name	salary
1	101	ASHIK	ICE	90000
2	102	NAYON	CSE	20000
3	103	SOBUZ	EEE	100000

Alter a table

	ID	name	dept_name	salary	course_no
1	101	ASHIK	ICE	90000	NULL
2	102	NAYON	CSE	20000	NULL
3	103	SOBUZ	EEE	100000	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
create database university_200607
use university_190609
-- Create Table
create table instructor(
ID varchar(5),
name varchar(20) not NULL,
dept_name varchar(20),
salary numeric(8,2),
primary key(ID));
--Insert Value in Table
insert into instructor values('101','ASHIK','ICE',90000);
insert into instructor values('102','NOYON','CSE',20000);
insert into instructor values('103','SOBUZ','EEE',100000);
--To Display all value of Department Table
select * from instructor;
select dept_name from instructor;
select dept_name from instructor where dept_name = 'ICE';
```

Output:

Select clause

	ID	Name	dept_name	salary
1	101	ASHIK	ICE	90000

2	102	NOYON	CSE	20000
3	103	SOBUZ	EEE	100000

From clause

	dept_name
1	ICE
2	CSE
3	EEE

Where clause

	dept_name
1	ICE

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 Database
create database university_200607
use university_200607
-- Create Table
create table instructor(
ID varchar(5),
name varchar(20) not NULL,
dept_name varchar(20),
salary numeric(8,2),
primary key(ID));
--Insert Value in Table
insert into instructor values('101','AHIK','ICE',90000);
insert into instructor values('102','NOYON','CSE',20000);
insert into instructor values('103',SOBUZ','EEE',100000);
--To Display all value of Department Table
select * from instructor;
--group by clause
select dept_name,AVG(salary) as avg_salary from instructor group by
dept_name;
--having clause
select dept_name,AVG(salary) as avg_salary from instructor group by
dept_name having AVG(salary)>42000;
--order by clause
select * from instructor order by salary desc,name asc;
--create view
create view faculty as select ID,name,dept_name from instructor;
--to display
select * from faculty;
--Indexing
select salary from instructor;
--create indexing
create index salary_index on instructor(salary asc);
select salary from instructor;
--delete indexing
drop index instructor.salary_index;
--create procedure
CREATE PROCEDURE instructor_proc
AS
BEGIN
select name as authors_name from instructor where ID='101';
END
EXEC instructor_proc --Call procedure
--to display
select * from instructor;

drop procedure instructor_proc;
```

Output:

Group By Clause

	dept_name	avg_salary
1	EEE	100000
2	ICE	90000.
3	CSE	20000

Order By (Ascending order)

	ID	name	dept_name	salary
1	101	ASHIK	ICE	90000
2	102	NOYON	CSE	20000
3	102	SOBUZ	EEE	100000

Create view as faculty

	ID	name	dept_name
1	101	ASHIK	ICE
2	102	NOYON	CSE
3	101	SOBUZ	EEE

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 1 Alice Math 2 Bob Science	ID Grade 1 A 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 2 Bob 3 Charlie	1 A 2 B

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 1 Alice 2 Bob	ID Grade 1 A 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:

```
--Create Database
create database university_200607
use university_200607;
 -- Create Table
create table department(
dept_name varchar(20),
building varchar(15),
budget numeric(12,2),
primary key(dept_name));
--Insert Value in Table
insert into department values('ICE','Watson',90000);
insert into department values('CSE','Taulor',100000);
insert into department values('EECE','Packard',50000);
--To Display all value of Department Table
select * from department;
-- Create instructor Table
create table instructor(
ID varchar(5),
name varchar(20) not NULL,
dept_name varchar(20),
salary numeric(8,2),
primary key(ID));
--Insert Value in Table
insert into instructor values('101','ASHIK','ICE',90000);
insert into instructor values('102','N0YON','CSE',20000);
insert into instructor values('103','SOBUZ','EEE',100000);
--To Display all value of instructor Table
select * from instructor;
--To display all
print('Instructor Table:');
select * from instructor;
select * from department;
```

--Cartesian Product

```
select building, department.dept_name,salary from department,instructor where
department.dept_name=instructor.dept_name;
```

--Join Operation

```
select salary,building from department join instructor on
department.dept_name=instructor.dept_name;
```

```
select * from department join instructor on department.dept_name=instructor.dept_name;
```

--left outer join

```
select * from department left outer join instructor on department.dept_name=instructor.dept_name;
```

--Right outer join

```
select * from department right outer join instructor on department.dept_name=instructor.dept_name;
```

--Full outer join

```
select * from department full outer join instructor on department.dept_name=instructor.dept_name;
```

Output:

---cartesian product

	bulding	salary
1	Watson	1000.00
2	Science	1001.00

----join product

	ID	name	budget
1	101	ASHIK	90000.00
2	3245	NOYO	85000.00

---left outer join

	ID	name	dept_name	salary	dept_name	bulding	budget
1	101	ASHIK	ICE	90000	ICE	Watson	90000.00
2	102	NOYON	CSE	20000	CSE	Science	100000
3	103	SOBUZ	EEE	100000	EECE	Packard	50000

---right outer join

	ID	name	dept_name	salary	dept_name	bulding	budget
1	101	ASHIK	ICE	90000	ICE	Watson	90000.00
2	102	NOYON	CSE	20000	CSE	Science	100000
3	103	SOBUZ	EEE	100000	EECE	Packard	50000

---full outer join

	ID	name	dept_name	salary	dept_name	bulding	budget
1	101	ASHIK	ICE	90000	ICE	Watson	90000.00
2	102	NOYON	CSE	20000	CSE	Science	100000
3	103	SOBUZ	EEE	100000	EECE	Packard	50000

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
create database university_200607
use university_200607
-- Create instructor Table
create table instructor(
ID varchar(5),
name varchar(20) not NULL,
dept_name varchar(20),
salary numeric(8,2),
primary key(ID));

--Insert Value in Table
insert into instructor values('101','ASHIK','ICE',90000);
insert into instructor values('102','NOYON','CSE',20000);
insert into instructor values('103','SOBUZ','EEE',100000);

--To Display all value of instructor Table
select * from instructor;

--Aggregate function
select COUNT(ID) as count_ID from instructor;
select MAX(salary) as max_salary from instructor;
select MIN(salary) as min_salary from instructor;
select AVG(salary) as avg_salary from instructor;
```

Output:

```
--Table
```

	ID	dept_name	salary
1	101	ICE	90000
2	102	CSE	20000
3	103	EEE	100000

```
--count
```

	count_ID
1	3

```
--max
```

	max_salary
1	100000

```
--min
```

	min_salary
1	20000

```
--Avg
```

	avg_salary
1	70000

```
--sum
```

	total_salary
1	210000

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:**

```
--Create Database

create database university_200607

use university_200607
```

```
GO
--customers Table Creating Start
CREATE TABLE customers
(
cusl_id CHAR (6) PRIMARY KEY CHECK (cusl_id LIKE '[A-Z][0-9][0-9][0-9][0-9][0-9]'),
cusl_fname CHAR(15) NOT NULL,
cusl_lname VARCHAR (15),
cusl_address TEXT,
cusl_telno CHAR (20) CHECK (cusl_telno LIKE '+88[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
cusl_city CHAR (22) DEFAULT 'Lalmonirhat',
sales_amnt MONEY CHECK (sales_amnt>=0),
proc_amnt MONEY CHECK (proc_amnt>=0)
);

--customers Table Insert Start
insert into customers
(cusl_id,cusl_fname,cusl_lname,cusl_address,cusl_telno,cusl_city,sales_amnt,proc_amnt) VALUES
('P00002','RAHATUL','RABBI','221/B Dhanmondi','+88017-00000000','Dhaka',0,0);

insert into customers VALUES
('C00005','IMRAN','Hossain','221/B Dhanmondi','+88017-00000000','Dhaka',0,0);

--customers Table Display Start
select * from customers;

--create trigger
create trigger test on Transactions for insert
```

```
as

begin

DECLARE @item_id char(6), @tranamount int, @tran_type char(1),@cusld_id
char(6),@price money

SELECT @item_id=item_id, @tranamount=tran_quantity,
@tran_type=tran_type,@cusld_id =cusl_id FROM INSERTED

Select @price=item_price from items where item_id=@item_id

 IF (@tran_type ='S')

     begin

       UPDATE Items SET item_qoh=item_qoh- @tranamount WHERE item_id=@item_id

       update  CustomerAndSuppliers set sales_amnt=sales_amnt+@price*@tranamount
where cusl_id=@cusld_id

     end

 ELSE

   begin

    UPDATE Items SET item_qoh=item_qoh+ @tranamount WHERE item_id=@item_id

    update  CustomerAndSuppliers set proc_amnt=proc_amnt+@tranamount*@price where
cusl_id=@cusld_id

    end

end


--delete triggering

drop  TRIGGER test
```

Output:

The orginal table instructor is

ID	name	dept_name	salary

1	101	Ashik	ICE	95000
2	102	noyon	EEE	90000
3	103	sobuz	CSE	60000
4	104	Katz	CSE	75000
5	105	Kim	EEE	80000

After inserting one element the inserted table

	ID	name	dept_name	salary
1	106	MD.ASHIK	ICE	80000

After deleted tuple from instructor table the backup table is

	ID	name	dept_name	salary
1	103	SOBUZ	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.

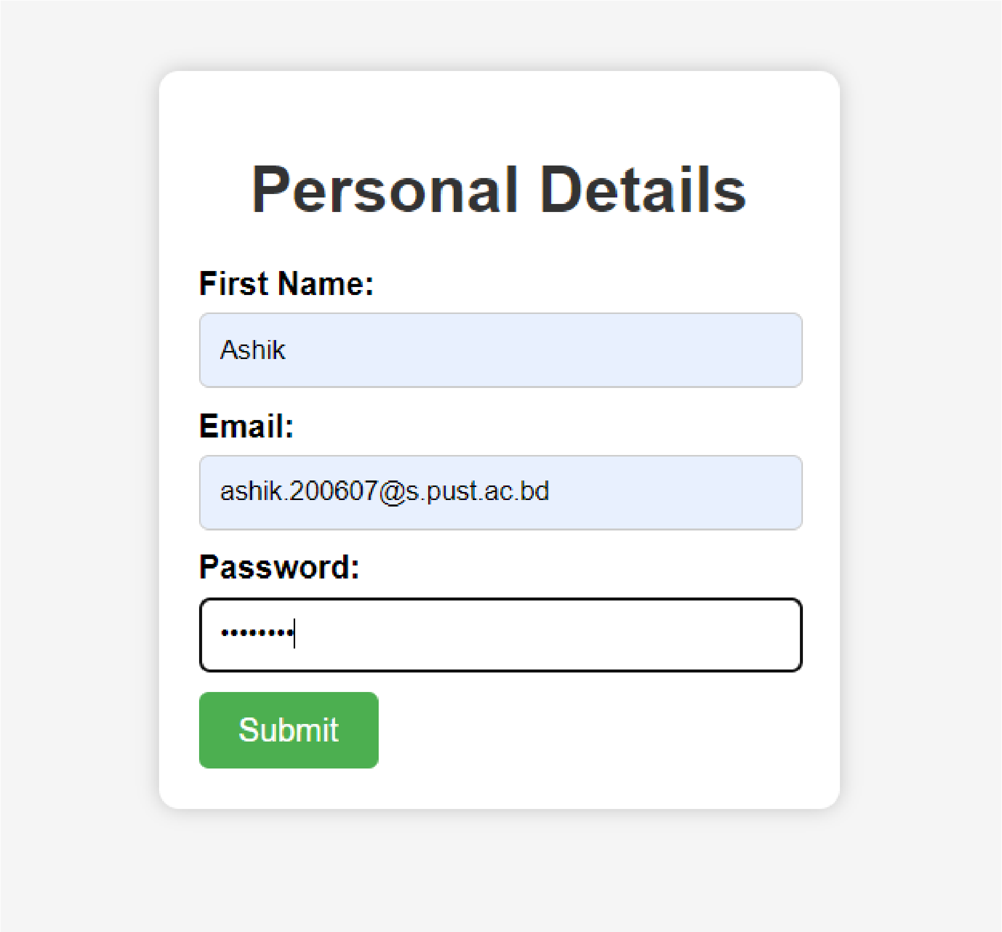
Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Insert Form</title>
   <style>
      body {
         background-color: #f5f5f5;
         font-family: Arial, sans-serif;
         display: flex;
         justify-content: center;
```

```
        .form-container {
            background-color: #ffffff;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0px 0px 10px 0px rgba(0,0,0,0.2);
            width: 300px;
        }
        input[type="submit"]:hover {
            background-color: #45a049;
        }
    </style>
</head>
<body>
    <div class="form-container">
        <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" required>
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" placeholder="Enter valid email" required>
            <label for="password">Password:</label>
            <input type="password" id="password" name="password" placeholder="Enter 6 digit password" required>
            <input type="submit" name="submit" value="Submit">
        </form>
    </div>
</body>
</html>


<?php
$base = mysqli_connect('localhost', 'root', ", 'insert');
if(isset($_POST['submit'])){
    $name = mysqli_real_escape_string($base, $_POST['name']);
    $email = mysqli_real_escape_string($base, $_POST['email']);
    $password = mysqli_real_escape_string($base, $_POST['password']);

    // Hash the password for security
    $hashed_password = password_hash($password, PASSWORD_DEFAULT);

    $sql = "INSERT INTO insertform (name, email, password) VALUES ('$name', '$email',
'$hashed_password')";
    if(mysqli_query($base, $sql)){
        echo "Inserted successfully";
    }
    else{
        echo "Insertion failed: " . mysqli_error($base); // Log the error message for debugging
    }
}

mysqli_close($base); // Close the connection after use
?>
```

Output:

php form



Personal Details

First Name:

Ashik

Email:

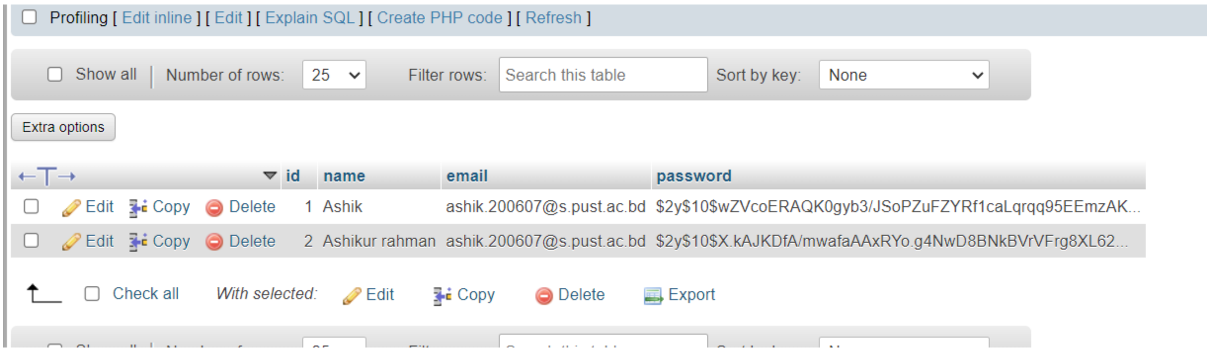
ashik.200607@s.pust.ac.bd

Password:

••••••••

Mysq

l database



□ Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

□ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

	id	name	email	password
□ Edit Copy Delete	1	Ashik	ashik.200607@s.pust.ac.bd	$2y$10$wZVcoERAQK0gyb3/JSoPZuFZYRf1caLqrqq95EEmzAK...
□ Edit Copy Delete	2	Ashikur rahman	ashik.200607@s.pust.ac.bd	$2y$10$X.kAJKDfA/mwafaAAxRYo.g4NwD8BNkBVrVFrg8XL62...