

Index

SL No	Experiment Name	Page No
01	Study and Implement the following DML Commands of SQL with suitable examples <ul style="list-style-type: none"> • Insert • Delete • Update 	
02	Study and Implement the following DDL Commands of SQL with suitable examples <ul style="list-style-type: none"> • Create • Alter • Drop 	
03	Study and Implement the following DML Commands <ul style="list-style-type: none"> • Select Clause • From Clause • Where Clause 	
04	Study and Implement the following DML Commands <ul style="list-style-type: none"> • Group By and Having Clause • Order By Clause • Create View, Indexing and Procedure Clause 	
05	Study and Implement the following SQL Commands of Join Operations with examples <ul style="list-style-type: none"> • Cartesian Product • Natural Join • Left Outer Join • Right Outer Join • Full Outer Join 	
06	Study and Implement the following Aggregate Function with example <ul style="list-style-type: none"> • Count Function • Max Function • Min Function • Avg Function 	
07	Study and Implement the Triggering System on Database Table using SQL commands with examples.	
08	Study and Implement the SQL Commands to connect MySQL Database with Java or PHP.	

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:

Relational Database Management Systems (RDBMS) play a pivotal role in data management. SQL (Structured Query Language) is a standard language for interacting with RDBMS. DML commands are used to manipulate data within a relational database. In this lab, we focus on three fundamental DML commands: INSERT, DELETE, and UPDATE.

INSERT: The INSERT command allows us to add new data into a table. We can either insert a single record or multiple records at once.

SQL syntax for insert:

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

where:

- table_name is the name of the database table into which the new record will be inserted.
- column1, column2, ... are the names of the columns in the table where the new values will be inserted.
- value1, value2, ... are the values that will be inserted into the columns.

DELETE: The DELETE command enables the removal of records from a table based on specified criteria. It can delete a single record, multiple records, or all records within a table.

SQL syntax for delete:

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

where:

- table_name is the name of the database table from which the records will be deleted.
- condition is an optional expression that specifies the records to be deleted. If no condition is specified, all records in the table will be deleted.

UPDATE: The UPDATE command is used to modify existing records in a table. It can update one or more fields in one or more records based on specific conditions.

SQL syntax for update:

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

where:

- table_name is the name of the database table in which the records will be updated.
- column1, column2, ... are the names of the columns in the table that will be updated.
- value1, value2, ... are the new values that will be inserted into the columns.

- condition is an optional expression that specifies the records to be updated. If no condition is specified, all records in the table will be updated.

Code:

```
-- Creating new database called university
create database university;
use university;
-- Creating new table called students
create table students(
  id int primary key,
  name varchar(100) not null,
  age int not null,
  department varchar(10) not null,
  city varchar(30) not null
);
-- insert data in the students table
INSERT INTO students (id, name, age, department, city)
VALUES
(200616, 'rakib', 24, 'ICE', 'Pabna'),
(200617, 'sabuz', 23, 'ICE', 'Dhaka'),
(200618, 'sumona', 23, 'ICE', 'Naogoan');

-- to view the table
select * from students;
-- update the students table
update students set age=25 where id=200618;
select * from students;
-- delete a row from the students table
delete from students where id=200618;
select * from students;
-- delete table
drop table students;
-- delete database
drop database university;
```

Output:

After inserting the values the table is :

	id	name	age	department	city
1	200616	rakib	24	ICE	Pabna
2	200617	sabuz	23	ICE	Dhaka
3	200618	sumona	23	ICE	Naogoan

After deleting value the table is:

	id	name	age	department	city
1	200616	rakib	24	ICE	Pabna
2	200617	sabuz	23	ICE	Dhaka

After updating the table is

	id	name	age	department	city
1	200616	rakib	24	ICE	Pabna
2	200617	sabuz	23	ICE	Dhaka
3	200618	sumona	25	ICE	Naogoon

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:

Theory:

DDL or Data Definition Language is a subset of SQL used to create, modify, and delete database objects, such as tables, indexes, and constraints. DDL commands are typically executed once to set up the database schema.

CREATE

The CREATE command is used to create new database objects. The syntax for the CREATE TABLE command is as follows:

SQL syntax for CREATE:

```
CREATE TABLE table_name (  
    column_name1 data_type1 [constraints],  
    column_name2 data_type2 [constraints],
```

```
...  
);
```

The `table_name` is the name of the new table. The `column_name` and `data_type` specify the name and data type of each column in the table. The constraints are optional and can be used to define constraints on the data in the table, such as NOT NULL, UNIQUE, and PRIMARY KEY.

ALTER

The ALTER command is used to modify existing database objects. The syntax for the ALTER TABLE command is as follows:

SQL syntax for ALTER:

```
ALTER TABLE table_name  
  ADD column_name data_type [constraints],  
  DROP column_name,  
  MODIFY column_name data_type [constraints],  
  RENAME column_name TO new_column_name;
```

The ADD clause is used to add a new column to the table. The DROP clause is used to drop an existing column from the table. The MODIFY clause is used to modify the data type or constraints on an existing column. The RENAME clause is used to rename an existing column.

DROP

The DROP command is used to delete database objects. The syntax for the DROP TABLE command is as follows:

SQL syntax for DROP:

```
DROP TABLE table_name;
```

The `table_name` is the name of the table to be deleted.

Code:

```
use university;  
-- create new table called employe  
create table employe(  
  id int primary key,  
  name varchar(100) not null  
);  
insert into employe values(101, 'abir');  
insert into employe values(102, 'ahsan');  
select * from employe;  
-- add new column in the employe table  
alter table employe  
  add phone varchar(20);  
insert into employe values(103, 'yasin', '0123456789');  
select * from employe;  
-- delete table  
drop table employe;
```

Output:

Create a table:

	id	name
1	101	abir
2	102	ahsan

Alter a table:

	id	name	phone
1	101	abir	NULL
2	102	ahsan	NULL
3	103	yasin	0123456789

Drop a table

Command(s) completed successfully.

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:

SELECT Clause:

The SELECT clause is used to specify the columns of data to be retrieved from the database table. The basic syntax for the SELECT clause is as follows:

SQL syntax:

```
SELECT column_name1, column_name2, ...  
FROM table_name;
```

The column_name1, column_name2, etc. are the names of the columns to be retrieved. The table_name is the name of the table from which to retrieve the data.

FROM Clause

The FROM clause specifies the table(s) from which to retrieve the data. The basic syntax for the FROM clause is as follows:

SQL syntax:

```
FROM table_name1, table_name2, ...;
```

The table_name1, table_name2, etc. are the names of the tables from which to retrieve the data. You can also use the JOIN keyword to combine data from multiple tables.

WHERE Clause

The WHERE clause is used to filter the data retrieved by the SELECT clause. The basic syntax for the WHERE clause is as follows:

SQL syntax:

```
WHERE condition1 AND condition2 AND ...;
```

The condition1, condition2, etc. are logical expressions that are evaluated to determine whether to include a row in the result set. You can use a variety of operators in the WHERE clause, such as =, <>, >, <, >=, and <=.

Code:

```
use University;
```

```
create table dept(
```

```
    Dept_name varchar(20),  
    Building varchar(20),  
    Budget numeric(12,2),  
    primary key(Dept_name)
```

```
);
```

```
insert into dept
```

```
(Dept_name,Building,Budget)
```

```
values
```

```
('ICE','Engineering building','90000'),
```

```
('CSE','Engineering building','80000'),
```

```
('EEE','Engineering building','90500'),
```

```
('Physics','Science building','50500');
```

```
select * from dept;
```

```
select Building from dept;
```

```
select Dept_name from dept where Building='Engineering building';
```

Output:Select clause:

	Dept_name	Building	Budget
1	CSE	Engineering building	80000.00
2	EEE	Engineering building	90500.00
3	ICE	Engineering building	90000.00
4	Physics	Science building	50500.00

From clause:

	Building
1	Engineering building
2	Engineering building
3	Engineering building
4	Science building

Where clause:

	Dept_name
1	CSE
2	EEE
3	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:

- To understand and implement the data definition language for using the Group by & Having Clause
- To understand and implement the data definition language for using the Order By clause
- To understand and implement the data definition language for using the Create View, Indexing & Procedure Clause

Theory:

Group By & Having Clause

Group By Clause

The GROUP BY clause is used to group the rows in a SQL result set based on the values of one or more columns. The syntax for the GROUP BY clause is as follows:

SQL syntax:

```
GROUP BY column_name1, column_name2, ...;
```

The column_name1, column_name2, etc. are the names of the columns to be used to group the rows. The GROUP BY clause must be used with an aggregate function, such as SUM(), AVG(), COUNT(), or MAX().

Having Clause

The HAVING clause is used to filter the groups created by the GROUP BY clause. The syntax for the HAVING clause is as follows:

SQL syntax:

```
HAVING condition1 AND condition2 AND ...;
```

The condition1, condition2, etc. are logical expressions that are evaluated to determine whether to include a group in the result set. You can use a variety of operators in the HAVING clause, such as =, <>, >, <, >=, and <=.

Order By Clause

The ORDER BY clause is used to sort the rows in a SQL result set based on the values of one or more columns. The syntax for the ORDER BY clause is as follows:

SQL syntax:

```
ORDER BY column_name1 ASC/DESC, column_name2 ASC/DESC, ...;
```

The column_name1, column_name2, etc. are the names of the columns to be used to sort the rows. The ASC and DESC keywords specify whether to sort the rows in ascending or descending order, respectively.

Create View, Indexing & Procedure Clause

Create View

The CREATE VIEW statement is used to create a view, which is a virtual table that is derived from one or more existing tables. Views can be used to simplify complex queries, restrict access to data, and create custom data types.

The syntax for the CREATE VIEW statement is as follows:

SQL syntax:

```
CREATE VIEW view_name AS  
SELECT column_name1, column_name2, ...  
FROM table_name1, table_name2, ...  
WHERE condition1 AND condition2 AND ...;
```

The view_name is the name of the new view. The column_name1, column_name2, etc. are the names of the columns to be included in the view. The table_name1, table_name2, etc. are the names of the tables from which to derive the view. The WHERE clause is optional and can be used to filter the data in the view.

Indexing

An index is a data structure that is used to improve the performance of database queries. Indexes can be created on any column in a table.

The syntax for the CREATE INDEX statement is as follows:

SQL syntax:

```
CREATE INDEX index_name ON table_name (column_name1, column_name2, ...);
```

The index_name is the name of the new index. The table_name is the name of the table on which to create the index. The column_name1, column_name2, etc. are the names of the columns to be included in the index.

Procedure

A stored procedure is a reusable collection of SQL statements that can be executed as a single unit. Stored procedures can be used to encapsulate complex logic and simplify database programming.

The syntax for the CREATE PROCEDURE statement is as follows:

SQL syntax:

```
CREATE PROCEDURE procedure_name (parameter1, parameter2, ...) AS
BEGIN
    SQL statements;
END;
```

The procedure_name is the name of the new procedure. The parameter1, parameter2, etc. are the names of the input and output parameters for the procedure. The BEGIN and END keywords enclose the body of the procedure.

Code:

```
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),
('12121','Wu','Finance',90000),
('15151','Mozart','Music',40000),
('22222','Einstein','Physics',95000),
('32343','El Said','History',60000),
('33456','Gold','Physics',87000);

select * from instructor;

-- group by clause

select name from instructor group by name;

select dept_name,avg(salary) as AVG
    from instructor group by dept_name;
```

```
select dept_name,count(*) as cout from instructor group by dept_name;
```

```
-- having clause
```

```
select dept_name,avg(salary) as AVG
```

```
from instructor group by dept_name having avg(salary)>70000;
```

```
-- 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 faculty;
```

```
-- index
```

```
create index dept_inx on instructor(dept_name);
```

```
-- procedure
```

```
DELIMITER $$
```

```
create procedure get_instructor_info (in salary numeric(8,2))
```

```
begin
```

```
select * from instructor where instructor.salary >= salary;
```

```
end
```

```
$$
```

```
DELIMITER ;
```

```
call get_instructor_info(65000);
```

Output:

	ID	name	dept_name	salary
▶	10101	Srinivasan	Comp.Sci	65000.00
	12121	Wu	Finance	90000.00
	15151	Mozart	Music	40000.00
	22222	Einstein	Physics	95000.00
	32343	EI Said	History	60000.00
	33456	Gold	Physics	87000.00
*	NULL	NULL	NULL	NULL

	dept_name	AVG
▶	Comp.Sci	65000.000000
	Finance	90000.000000
	History	60000.000000
	Music	40000.000000
	Physics	91000.000000

	dept_name	AVG
▶	Finance	90000.000000
	Physics	91000.000000

	ID	name	dept_name
▶	10101	Srinivasan	Comp.Sci
	12121	Wu	Finance
	15151	Mozart	Music
	22222	Einstein	Physics
	32343	EI Said	History
	33456	Gold	Physics

	ID	name	dept_name	salary
▶	15151	Mozart	Music	40000.00
	32343	EI Said	History	60000.00
	10101	Srinivasan	Comp.Sci	65000.00
	33456	Gold	Physics	87000.00
	12121	Wu	Finance	90000.00
	22222	Einstein	Physics	95000.00
*	NULL	NULL	NULL	NULL

	ID	name	dept_name	salary
▶	10101	Srinivasan	Comp.Sci	65000.00
	12121	Wu	Finance	90000.00
	22222	Einstein	Physics	95000.00
	33456	Gold	Physics	87000.00

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:

Join Operations in SQL:

Join operations are used to combine data from two or more tables based on a common field between them. There are four main types of join operations in SQL:

Cartesian Product

A Cartesian product is the simplest type of join operation. It returns all possible combinations of rows from the two tables, regardless of whether there is a match between the common field.

Syntax:

SQL

```
SELECT * FROM table1, table2;
```

Natural Join

A natural join combines two tables based on all common columns between them.

Syntax:

SQL

```
SELECT * FROM table1 NATURAL JOIN table2;
```

Left Outer Join

A left outer join returns all rows from the left table, even if there is no matching row in the right table.

Syntax:

SQL

```
SELECT * FROM table1 LEFT JOIN table2 ON table1.column_name =  
table2.column_name;
```

Right Outer Join

A right outer join returns all rows from the right table, even if there is no matching row in the left table.

Syntax:

SQL

```
SELECT * FROM table1 RIGHT JOIN table2 ON table1.column_name =  
table2.column_name;
```

Full Outer Join

A full outer join returns all rows from both tables, even if there is no matching row in the other table.

Syntax:

SQL

```
SELECT * FROM table1 FULL JOIN table2 ON table1.column_name =  
table2.column_name;
```

Code:

```
use university;
```

```
create table instruct(
```

```
    ID varchar(20) primary key,
```

```
    name varchar(20) not null,
```

```
    dept_name varchar(20),
```

```

    salary numeric(8,2)
);
insert into instruct
values
('10101','Srinivasan','CSE',65000),
('12121','Wu','Finance',90000),
('15151','Mozart','Music',40000),
('22222','Einstein','Physics',95000),
('32343','El Said','History',60000),
('33456','Gold','Physics',87000);
select * from instruct;
create table DEPARTMENT(
    dept_name varchar(20) primary key,
    Building_name varchar(20),
    Budget numeric(12,2)
);
insert into DEPARTMENT
(dept_name,Building_name,Budget) values
('ICE','Engineering building',90000),
('CSE','Engineering building',80000),
('EEE','Engineering building',90500),
('Physics','Science building',50500),
('Social Work','Arts building',30500);
select * from DEPARTMENT;
-- cartesian product
select Building_name,salary from DEPARTMENT,instruct where
DEPARTMENT.dept_name=instruct.dept_name;
-- join operation
select ID,name,budget from DEPARTMENT join instruct on
DEPARTMENT.dept_name=instruct.dept_name;
-- left outer join

```

```

select * from DEPARTMENT left outer join instruct on
DEPARTMENT.dept_name=instruct.dept_name;
-- right outer join
select * from DEPARTMENT right outer join instruct on
DEPARTMENT.dept_name=instruct.dept_name;
-- full outer join
select * from DEPARTMENT left join instruct on
DEPARTMENT.dept_name=instruct.dept_name
union
select * from DEPARTMENT right join instruct on
DEPARTMENT.dept_name=instruct.dept_name;

```

Output:

---cartesian product

	Building_name	salary
▶	Engineering building	65000.00
	Science building	95000.00
	Science building	87000.00

----join product

	ID	name	budget
▶	10101	Srinivasan	80000.00
	22222	Einstein	50500.00
	33456	Gold	50500.00

--- outer join

	dept_name	Building_name	Budget	ID	name	dept_name	salary
▶	CSE	Engineering building	80000.00	10101	Srinivasan	CSE	65000.00
	EEE	Engineering building	90500.00	NULL	NULL	NULL	NULL
	ICE	Engineering building	90000.00	NULL	NULL	NULL	NULL
	Physics	Science building	50500.00	33456	Gold	Physics	87000.00
	Physics	Science building	50500.00	22222	Einstein	Physics	95000.00
	Social Work	Arts building	30500.00	NULL	NULL	NULL	NULL

---full outer join

	dept_name	Building_name	Budget	ID	name	dept_name	salary
▶	CSE	Engineering building	80000.00	10101	Srinivasan	CSE	65000.00
	EEE	Engineering building	90500.00	NULL	NULL	NULL	NULL
	ICE	Engineering building	90000.00	NULL	NULL	NULL	NULL
	Physics	Science building	50500.00	33456	Gold	Physics	87000.00
	Physics	Science building	50500.00	22222	Einstein	Physics	95000.00
	Social Work	Arts building	30500.00	NULL	NULL	NULL	NULL
	NULL	NULL	NULL	12121	Wu	Finance	90000.00
	NULL	NULL	NULL	15151	Mozart	Music	40000.00
	NULL	NULL	NULL	32343	EI Said	History	60000.00

Experiment No:06

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

Objectives:

- To understand the different issues in the design and implementation of a database system
- To apply and implement the Aggregate Function

Theory:

Aggregate functions are used to perform calculations on multiple values in a SQL column and return a single value. Some common aggregate functions include:

- COUNT(): Counts the number of rows in a column or the number of non-NULL values in a column.
- MAX(): Returns the largest value in a column.
- MIN(): Returns the smallest value in a column.
- AVG(): Calculates the average of the values in a column.

Aggregate functions can be used with the GROUP BY clause to group the data and perform calculations on each group.

Code:

```
use university;

select * from instruct;

-- count

select count(ID) as cnt_ID from instruct where dept_name='Physics';

-- max

select max(salary) as maxSalary from instruct;

-- min

select min(salary) as minSalary from instruct;
```



```
-- avg
```

```
select avg(salary) as avg_salary from instruct;
```

Output :

	cnt_ID		maxSalary		minSalary		avg_salary
▶	2	▶	95000.00	▶	40000.00	▶	72833.333333

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,'King Kong','Physics',95000)
insert into instructor values(12121,'Nora Fatehi','Finance',90000)
insert into instructor values(32343,'Ravindra Nath Tagore','History',60000)
insert into instructor values(45565,'Katz','CSE',75000)
insert into instructor values(98345,'Kim','EEE',80000)
insert into instructor values(98346,'Kesa','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 trigger
create trigger ins_trigger
on instructor
after insert
as begin
    print'The trigger 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 trigger
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;

```

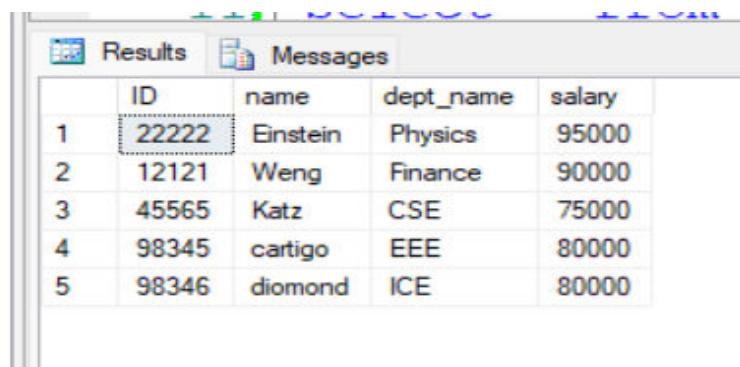
```

DELETE FROM instructor WHERE ID = 32343;
select * from backup_del

```

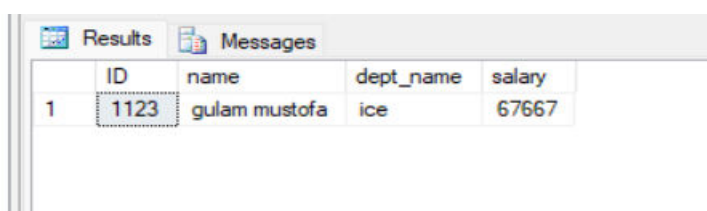
Output:

The original table instructor is:



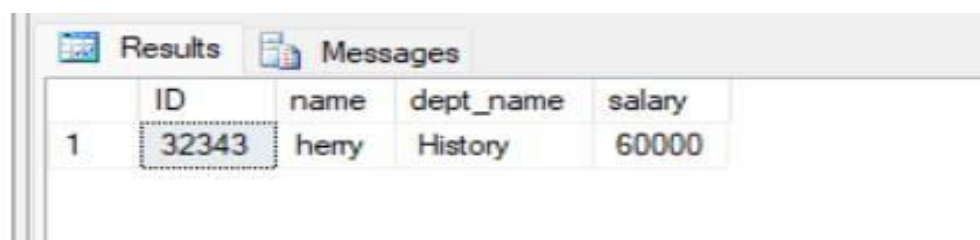
	ID	name	dept_name	salary
1	22222	Einstein	Physics	95000
2	12121	Weng	Finance	90000
3	45565	Katz	CSE	75000
4	98345	cartigo	EEE	80000
5	98346	diomond	ICE	80000

After inserting one element the inserted table



	ID	name	dept_name	salary
1	1123	gulam mustofa	ice	67667

After deleted tuple from instructor table the backup table is :



	ID	name	dept_name	salary
1	32343	herry	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="password">Password : </label>
    <input type="password" id="password" name="password" placeholder="Enter 6 digit
password"><br>
```

```
<input type="submit" name="submit" value="submit">
</form>
</body>
</html>
```

Output:

php form:

Personal Details	
First Name :	<input type="text" value="MD Rakibul Hossain"/>
Email :	<input type="text" value="mdrakibulhossain7869@gmail.com"/>
Password :	<input type="password" value="....."/>
<input type="submit" value="submit"/>	

Mysql database:

<input type="checkbox"/> Show all	Number of rows: 25	Filter rows: <input type="text" value="Search this table"/>
Extra options		
name	email	password
MD.Rakibul Hossain	mdrakibulhossain7869@gmail.com	123456
gulam mustofa	ridubai12@gmail.com	123678
<input type="checkbox"/> Show all	Number of rows: 25	Filter rows: <input type="text" value="Search this table"/>