Experiment Number: 01

Name of the Experiment: To study and implementation of DML Commands of SQL with Suitable Example:

- Insert.
- Delete.
- Update.

Objectives: To understand and use data manipulation language to write query for database. And to implement *Insertion*, *Deletion* and *Update* operations on a database.

Theory: DML, or Data Manipulation Language, is a subset of SQL (Structured Query Language) used in relational database management systems. DML is responsible for manipulating the data stored in a database. It includes commands and statements for:

Insert: The `INSERT` operation is used to add new records (rows) into a table. It populates a table with fresh data, and the newly inserted data becomes part of the database. Here's an example:

INSERT INTO <table name>

VALUES ('value1', 'value2', 'value3');

Update: The `UPDATE` operation is used to modify existing records within a table. It allows you to change the values of one or more columns in specific rows, typically based on certain conditions.

UPDATE <table name>

SET <value3> = #####

WHERE <primary key> = '#####';

Delete: The `DELETE` operation is used to remove records (rows) from a table. It helps in eliminating unwanted or outdated data, and the table is updated without the deleted records. Example:

DELETE FROM <table name>

WHERE <primary key> = '#####';

Experimental Requirements:

1. Laptop/ Desktop Computer.
2. Microsoft SQL Server application.
3. Microsoft Server Management Studio application.
4. University database.

Source Code:

```
use master
use University
create table instructor(
                    Ins_ID varchar(10),
                    Ins_name varchar(20),
                    Ins_dept varchar(15),
                    Ins_salary numeric(6,0),
                    primary key (Ins_ID))

select * from instructor
insert into instructor
          values('Ex-01','Rahim','Bangla','12345'),
                    ('Ex-02','Karim','English','23451'),
                    ('Ex-03','Ismail','History','34512'),
                    ('Ex-04','Ibrahim','Arabic','45123')
Update instructor
set Ins_salary='99999'
where Ins_ID='Ex-03'

select * from instructor

delete from instructor
where Ins_ID='Ex-01'

select * from instructor
```

Output:

Table (1): Data Inserted successfully in the instructor table.

	Ins_ID	Ins_name	Ins_dept	Ins_salary
1	Ex-01	Rahim	Bangla	12345
2	Ex-02	Karim	English	23451
3	Ex-03	Ismail	History	34512
4	Ex-04	Ibrahim	Arabic	45123

Table (2): Data updated successfully in the instructor table.

	Ins_ID	Ins_name	Ins_dept	Ins_salary
1	Ex-01	Rahim	Bangla	12345
2	Ex-02	Karim	English	23451
3	Ex-03	Ismail	History	99999
4	Ex-04	Ibrahim	Arabic	45123

Table (3): Data deleted successfully from the instructor table.

	Ins_ID	Ins_name	Ins_dept	Ins_salary
1	Ex-02	Karim	English	23451
2	Ex-03	Ismail	History	99999
3	Ex-04	Ibrahim	Arabic	45123

Result and Discussion: Using DML command we have created a database and then have inserted values on the database then we have inserted the rows using the command then has updated some values in the database and finally we have deleted some values from our database and in the output, we can see those result. Because we have fulfilled our objectives so we can say that the experiment went successfully.

Experiment Number: 02

Name of the Experiment: To Study and Implementation of DDL Commands of SQL with Suitable example

- Create.
- Alter.
- Drop.

Objectives: To understand and use data definition language to write query for database. And to implement *create*, *alter* and *drop* operations on a database.

Theory: DDL (Data Definition Language) commands are SQL statements used to manage the structure of a database. They include commands like CREATE (to define tables, indexes, and databases), ALTER (for modifying existing structures), and DROP (for deleting tables or databases). DDL commands are essential for defining and maintaining the database schema, specifying data types, constraints, and relationships. Here are concise definitions for the SQL commands CREATE, ALTER, and DELETE:

CREATE: In SQL, the CREATE command is used to define new database objects, such as tables, indexes, or databases themselves. It specifies the structure and attributes of the object, allowing data to be stored and organized.

ALTER: The ALTER command is used to modify the structure of existing database objects, such as tables or columns. It can add, delete, or modify columns, constraints, or other properties without recreating the entire object.

DELETE: The DELETE command is employed to remove data rows from a table, effectively erasing specific records while keeping the table's structure intact. It is often used to eliminate unwanted or outdated data from a database.

Experimental Requirements:

1. Laptop/ Desktop Computer.
2. Microsoft SQL Server application.
3. Microsoft Server Management Studio application.
4. University database.

Source Code:

```
create table student(
                stu_ID varchar(10),
                stu_name varchar(20),
                stu_dept varchar(15),
```

```
                    stu_credic numeric(6,0),
                    primary key (stu_ID))
insert into student
        values('01','Allison','Bangla','12345'),
                    ('02','Karim Benzema','English','23451'),
                    ('03','Ronaldo','History','34512'),
                    ('10','Neymar','Arabic','45123'),
                     ('11','Messi','Arabic','40123'),
                      ('12','Vini Vr','Arabic','45123'),
                            ('99','PAPON','BCB','0')
select * from student

alter table student
drop column stu_credic
select * from student

drop table student
SELECT NAME FROM SYSOBJECTS WHERE XTYPE='U'
```

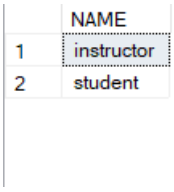
Output:

Table (1): Student table data before altering in university database.

	stu_ID	stu_name	stu_dept	stu_credic
1	01	Allison	Bangla	12345
2	02	Karim Benzema	English	23451
3	03	Ronaldo	History	34512
4	10	Neymar	Arabic	45123
5	11	Messi	Arabic	40123
6	12	Vini Vr	Arabic	45123
7	99	PAPON	BCB	0

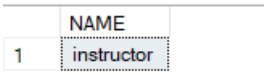
Table (2): Student table data after altering database.

	stu_ID	stu_name	stu_dept
1	01	Allison	Bangla
2	02	Karim Benzema	English
3	03	Ronaldo	History
4	10	Neymar	Arabic
5	11	Messi	Arabic
6	12	Vini Vr	Arabic
7	99	PAPON	BCB



	NAME
1	instructor
2	student

Figure (1): Before performing drop table.



	NAME
1	instructor

Figure (2): After performing drop table.

Result and Discussion: Using DDL command first we have created a database and inside the database we have created a table. Then inserted some data in the table and we can see that we can easily insert those data in the table that means the database and table created successfully. Then we altered the table, we deleted one attribute from the table and that is the alter operation, after altering the table we can see and compare the output of before and after. Finally, we dropped a table from the database and as a result we can see before there were two databases in the table and after dropping one database only one database remains. If we drop the database, then the database will be deleted we can perform that as well. So, we can say from the output that we have completed the experiment successfully.

Experiment Number: 03

Name of the Experiment: To Study and Implementation of DML Commands of

- Select Clause
- From Clause
- Where Clause

Objectives: To understand and use data manipulation language to write query for database. And to implement *select*, *from* and *where* clause on a database.

Theory:

The SELECT statement is used to specify which columns or expressions you want to retrieve from one or more tables in a database. We can select specific columns by listing their names after the SELECT keyword. For example, `SELECT column1, column2` would retrieve data from columns named column1 and column2. We can also use wildcard characters like '*' to select all columns from a table. For example, `SELECT *` retrieves all columns in the specified table. We can perform calculations or apply functions on the selected columns to transform the data before retrieval. For example, `SELECT column1 + column2` would return the sum of values in column1 and column2.

The FROM clause is crucial as it specifies the source tables from which you want to retrieve data. After the FROM keyword, we specify the name of the table or tables from which we want to retrieve data. We can also use JOIN operations to combine data from multiple tables into a single result set. Joins help you link data between related tables by specifying the relationships between them.

The WHERE clause is often used in conjunction with the SELECT statement to filter the results based on specific conditions. In the WHERE clause, we can define conditions that the data must meet to be included in the result set. These conditions can include equality checks, range comparisons, logical operators, and more. For example, we can use a WHERE clause to retrieve all rows where a particular column has a specific value, such as `WHERE column1 = 'value'`.We can also use logical operators like AND and OR to combine multiple conditions, enabling you to create complex filters. Here's a simple example of a SQL query that incorporates all these elements:

SELECT column1, column2

FROM table_name

WHERE column3 = 'value' AND column4 > 100;This query selects data from the specified columns in the "table_name" table, but only includes rows where "column3" has the value

'value' and "column4" is greater than 100. The result will be a subset of data that meets these criteria.

Experimental Requirements:

1. Laptop/ Desktop Computer.
2. Microsoft SQL Server application.
1. Microsoft Server Management Studio application.
2. University Database

Source Code:

```
select * from instructor
select Ins_name,Ins_dept
from instructor
where Ins_dept='English'
```

Output:

Table (1): Showing all data from Student table.

	Ins_ID	Ins_name	Ins_dept	Ins_salary
1	Ex-02	Karim	English	23451
2	Ex-03	Ismail	History	99999
3	Ex-04	Ibrahim	Arabic	45123

Table (2): Showing data using select clause from Student table.

	Ins_name	Ins_dept
1	Karim	English
2	Ismail	History
3	Ibrahim	Arabic

Table (3): Showing data using select clause then where clause from Student table.

	Ins_name	Ins_dept
1	Karim	English

Resul and Discussion: Using select clause we first have shown all the tuples of the student table from database, then we have shown only student name, department name and session from student table using the from clause then finally we have shown the student's name their roll and session from department of English using the 'where clause' . From the output we can say that our SQL command is accurate, that's why we have a perfect output. So, we can say that the experiment went successfully.

Experiment Number: 04

Name of the Experiment: To Study and Implementation of DML Commands of

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

Objectives:

1. To understand and use data manipulation language to write query for database.
2. To implement group by & having, order by and create view, indexing & procedure clause on a database.

Theory:

GROUP BY Clause: The GROUP BY clause is used to group rows from a table based on the values in one or more columns. It divides the result set into distinct groups, where each group shares the same values in the specified column(s). This is especially useful when you want to perform aggregate functions (e.g., SUM, COUNT, AVG) on data within each group.

HAVING Clause: The HAVING clause is used in conjunction with the GROUP BY clause to filter the grouped results. It specifies conditions that the groups must meet. Only the groups that satisfy these conditions are included in the result. HAVING allows you to filter based on aggregated values, such as only selecting groups with a certain total or average value.

ORDER BY Clause: The ORDER BY clause is used to sort the result set of a query in either ascending (ASC) or descending (DESC) order based on one or more columns. This helps arrange the output data in a specific sequence, making it easier to analyze or present. You can sort data based on multiple columns, which allows for more complex sorting criteria.

Create View: A SQL view is a virtual table created by a query. It doesn't store data itself but provides a way to simplify complex queries by storing them as views. Views can be queried like regular tables, and they dynamically represent specific subsets of data from one or more tables. Views are particularly useful for encapsulating and abstracting complex query logic.

Indexing: Indexes are database structures used to optimize data retrieval. They work like an index in a book, allowing the database management system to quickly locate and retrieve rows of data based on indexed columns. Indexing is crucial for improving the performance of data retrieval operations, especially in large databases.

Procedure Clause: SQL procedures, often referred to as stored procedures, are precompiled SQL statements and commands that are saved in the database. They can be executed on demand. Stored procedures allow you to encapsulate and reuse a series of SQL statements as a single unit. This enhances code organization, security, and reusability, making it easier to manage and maintain complex database logic.

These additional SQL features expand the capabilities of SQL beyond basic data retrieval and manipulation, providing tools for more advanced data management, optimization, and code organization in a relational database system.

Experimental Requirements:

1. Laptop/ Desktop Computer.
2. Microsoft SQL Server application.
3. Microsoft Server Management Studio application.

Source Code:

```
select * from instructor
select count(ID) as Teachers,dept_name
from instructor
group by dept_name
having avg(salary)>72000
select ID,name,dept_name,salary
from instructor
order by salary ASC
select ID,name,dept_name,salary
from instructor
where dept_name='Finance'
order by salary ASC

create view Teacher as
select name,dept_name
from instructor
where salary>80000
select * from Teacher
create index teacher on instructor(name)
/* speed up data retrieval for queries that involve the "Ins_name" column*/
CREATE PROCEDURE get_teacher_by_dept
   @tech_dept VARCHAR(30)
AS
BEGIN
   SELECT ID, name
   FROM instructor
   WHERE dept_name = @tech_dept;
END
EXEC get_teacher_by_dept @tech_dept = 'Finance'
```

OUTPUT:

Table (1): Instructor table after applying the 'group by' clause.

	Teachers	dept_name
1	1	Biology
2	3	Comp. Sci.
3	1	Elec. Eng.
4	2	Finance
5	2	History
6	1	Music
7	2	Physics

Table (2): Instructor table after applying the 'group by and having' clause.

	Teachers	dept_name
1	1	Biology
2	3	Comp. Sci.
3	1	Elec. Eng.
4	2	Finance
5	2	Physics

Table (3): Instructor table after applying the 'order by' clause on salary Ascending.

	ID	name	dept_name	salary
1	15151	Mozart	Music	43600.00
2	32343	El Said	History	65400.00
3	58583	Califieri	History	67580.00
4	10101	Srinivasan	Comp. Sci.	77226.50
5	76766	Crick	Biology	78480.00
6	98345	Kim	Elec. Eng.	87200.00
7	76543	Singh	Finance	87200.00
8	45565	Katz	Comp. Sci.	89107.50
9	33456	Gold	Physics	94830.00
10	12121	Wu	Finance	98100.00
11	22222	Einstein	Physics	103550.00
12	83821	Brandt	Comp. Sci.	109305.20

Table (4): Instructor table after applying the 'Create View' clause.

	name	dept_name
1	Wu	Finance
2	Einstein	Physics
3	Gold	Physics
4	Katz	Comp. Sci.
5	Singh	Finance
6	Brandt	Comp. Sci.
7	Kim	Elec. Eng.

Table (5): Instructor table after applying the ‘Procedure’ clause.

	ID	name
1	12121	Wu
2	76543	Singh

Resul and Discussion: When utilizing the GROUP BY clause, it's imperative to incorporate aggregate functions; otherwise, it won't function. The HAVING clause can operate independently, but when used alongside GROUP BY, it necessitates the use of aggregate functions as well. In the ORDER BY clause, you have the flexibility to arrange data in either ascending or descending order. Creating a view is akin to forging a fresh table from an existing one, selecting specific attributes. Indexing doesn't display results; instead, it enhances data retrieval speed for designated attributes. The Procedure clause resembles a function in which varying inputs yield distinct outputs, as previously defined.

Experiment Number: 05

Name of the Experiment: To 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:

1. To understand the concept of the Join Operations.
2. To implement Cartesian Product, Natural Join, Left Outer Join, Right Outer Join and Full Outer Join on a database.

Theory: A Cartesian product, both a mathematical concept and a database operation, involves the combination of every element from one set with every element from another set to create an extensive set of all possible pairings. For example, when merging a set of three colors with a set of two sizes, this results in a total of six unique color-size combinations. While this approach is foundational in mathematical theory and can be applied in databases, it has the potential to exponentially increase data volume, often leading to unwieldy and impractical results. As a result, in database query design, it is generally avoided to maintain data manageability and query efficiency.

In the realm of Structured Query Language (SQL), join operations are crucial for integrating data from multiple tables. These operations enable the retrieval and presentation of data while establishing relationships and connections between records in different tables. There are several common types of join operations:

INNER JOIN: This type of join returns only the rows where there is a match in both tables, based on the specified join condition. It combines data from two tables where the values in the specified columns are equal.

LEFT JOIN (or LEFT OUTER JOIN): In a left join, all rows from the left table are returned, along with the matching rows from the right table. If there is no match in the right table, NULL values are returned for the columns of the right table.

RIGHT JOIN (or RIGHT OUTER JOIN): A right join, conversely, returns all rows from the right table and the matching rows from the left table. Rows from the left table that do not find a match in the right table result in NULL values for the left table's columns.

FULL JOIN (or FULL OUTER JOIN): A full join returns all rows when there is a match in either the left or right table. This join type combines data from both tables, including unmatched rows with NULL values for missing matches.

Join operations, particularly INNER JOIN, are fundamental in enabling complex data analysis and allowing database systems to present data in a structured and meaningful way, making them an essential component of modern relational database management.

Experimental Requirements:

1. Laptop/ Desktop Computer.
2. Microsoft SQL Server application.
3. Microsoft SQL Server Management Studio application.
4.

Source Code:

```
select  ID,name,
 building,budget
 from instructor
 cross join department

  select  ID,name,
 building,budget
 from instructor
 inner join department
 on instructor.dept_name=department.dept_name

 select  ID,name,
 building,budget
 from instructor
 left join department
 on instructor.dept_name=department.dept_name

 select  ID,name,
 building,budget
 from instructor
 right join department
 on instructor.dept_name=department.dept_name



 select  ID,name,
 building,budget
 from instructor
 full outer join department
 on instructor.dept_name=department.dept_name
```

OUTPUT:

Table (1): Table after applying the Cartesian Product operation.

	ID	name	building	budget
69	76543	Singh	Painter	120000.00
70	76766	Crick	Painter	120000.00
71	83821	Brandt	Painter	120000.00
72	98345	Kim	Painter	120000.00
73	10101	Srinivasan	Painter	50000.00
74	12121	Wu	Painter	50000.00
75	15151	Mozart	Painter	50000.00
76	22222	Einstein	Painter	50000.00
77	32343	El Said	Painter	50000.00
78	33456	Gold	Painter	50000.00
79	45565	Katz	Painter	50000.00
80	58583	Califieri	Painter	50000.00
81	76543	Singh	Painter	50000.00
82	76766	Crick	Painter	50000.00
83	83821	Brandt	Painter	50000.00
84	98345	Kim	Painter	50000.00

Table (2): Table after applying the Inner Join operation.

	ID	name	building	budget
1	10101	Srinivasan	Taylor	100000.00
2	12121	Wu	Painter	120000.00
3	15151	Mozart	Packard	80000.00
4	22222	Einstein	Watson	70000.00
5	32343	El Said	Painter	50000.00
6	33456	Gold	Watson	70000.00
7	45565	Katz	Taylor	100000.00
8	58583	Califieri	Painter	50000.00
9	76543	Singh	Painter	120000.00
10	76766	Crick	Watson	90000.00
11	83821	Brandt	Taylor	100000.00
12	98345	Kim	Taylor	85000.00

Table (3): Table after applying the left outer join operation.

	ID	name	building	budget
1	10101	Srinivasan	Taylor	100000.00
2	12121	Wu	Painter	120000.00
3	15151	Mozart	Packard	80000.00
4	22222	Einstein	Watson	70000.00
5	32343	El Said	Painter	50000.00
6	33456	Gold	Watson	70000.00
7	45565	Katz	Taylor	100000.00
8	58583	Califieri	Painter	50000.00
9	76543	Singh	Painter	120000.00
10	76766	Crick	Watson	90000.00
11	83821	Brandt	Taylor	100000.00
12	98345	Kim	Taylor	85000.00

Table (4): Table after applying the right outer join operation.

	ID	name	building	budget
1	76766	Crick	Watson	90000.00
2	NULL	NULL	Sayeeda Khan	1065369.00
3	10101	Srinivasan	Taylor	100000.00
4	45565	Katz	Taylor	100000.00
5	83821	Brandt	Taylor	100000.00
6	NULL	NULL	Sayeeda Khan	1065369.00
7	98345	Kim	Taylor	85000.00
8	12121	Wu	Painter	120000.00
9	76543	Singh	Painter	120000.00
10	32343	El Said	Painter	50000.00
11	58583	Califieri	Painter	50000.00
12	NULL	NULL	Sayeeda Khan	1065369.00
13	NULL	NULL	Sayeeda Khan	1065369.00
14	NULL	NULL	Sayeeda Khan	1065369.00
15	15151	Mozart	Packard	80000.00
16	22222	Einstein	Watson	70000.00

Table (5): Table after applying the full outer join operation.

	ID	name	building	budget
1	10101	Srinivasan	Taylor	100000.00
2	12121	Wu	Painter	120000.00
3	15151	Mozart	Packard	80000.00
4	22222	Einstein	Watson	70000.00
5	32343	El Said	Painter	50000.00
6	33456	Gold	Watson	70000.00
7	45565	Katz	Taylor	100000.00
8	58583	Califieri	Painter	50000.00
9	76543	Singh	Painter	120000.00
10	76766	Crick	Watson	90000.00
11	83821	Brandt	Taylor	100000.00
12	98345	Kim	Taylor	85000.00
13	NULL	NULL	Sayeeda Khan	1065369.00

Result and Discussion: In the first table, the large size is attributed to the Cartesian product, where each tuple in the "student" table is combined with every tuple in the "instructor" table, resulting in an expansive dataset.

In the second table, only tuples with matching department values for both students and instructors are included, effectively filtering the results for relevant matches.

The third table showcases a left outer join, preserving all values from the left table and displaying NULL where there's no corresponding value in the right table.

The final table demonstrates a right outer join, akin to the left outer join but with all values from the right table included. In this case, the "student" table is the left table, and the "instructor" table is the right table. The successful outcome of this experiment is evident from the output.

Experiment Number: 06

Name of the Experiment: To Study and Implementation of Aggregate Function with Example:

- Count Function
- Max Function
- Min Function
- Avg Function

Objectives:

1. To understand the concept of the aggregate function.
2. To understand the working process of *count*, *max, min* and *average* function.

Theory: Aggregate functions in SQL are indispensable for summarizing data in a database table, performing calculations on groups of rows, and yielding single results. These functions, including COUNT (for row counting), SUM (for numeric column totals), AVG (for average value computation), MAX (for maximum value determination), and MIN (for minimum value extraction), offer valuable insights into large datasets. Here are brief definitions for some of these key aggregate functions:

COUNT: The COUNT function assesses the number of rows meeting specific conditions in a dataset. It is frequently used to tally row occurrences in a column, aiding in dataset size evaluation and the quantification of records satisfying particular criteria.

AVERAGE (AVG): AVG computes the arithmetic mean of values in a numeric column, providing an assessment of the typical value within a dataset.

MINIMUM (MIN): MIN identifies the smallest value in a dataset for a specified column, assisting in the determination of the dataset's minimum or starting point.

MAXIMUM (MAX): MAX locates the largest value in a dataset for a specified column, facilitating the identification of the dataset's maximum or peak value. These aggregate functions are crucial for data analysis and decision-making, enabling the extraction of valuable insights from complex datasets.

Experimental Requirements:

1. Laptop/ Desktop Computer.
2. Microsoft SQL Server application.
3. Microsoft SQL Server Management Studio application.

Source Code:

```
select * from instructor

select dept_name,count(ID) as Department_Teachers
from instructor
group by dept_name

select min(name) as TeacherMinname,dept_name
from instructor
group by  dept_name

select max(name) as TeacherMaxname,dept_name
from instructor
group by  dept_name

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

Output:

Table (1): Showing all data from instructor table in database.

	ID	name	dept_name	salary
1	10101	Srinivasan	Comp. Sci.	77226.50
2	12121	Wu	Finance	98100.00
3	15151	Mozart	Music	43600.00
4	22222	Einstein	Physics	103550.00
5	32343	El Said	History	65400.00
6	33456	Gold	Physics	94830.00
7	45565	Katz	Comp. Sci.	89107.50
8	58583	Califieri	History	67580.00
9	76543	Singh	Finance	87200.00
10	76766	Crick	Biology	78480.00
11	83821	Brandt	Comp. Sci.	109305.20
12	98345	Kim	Elec. Eng.	87200.00

Table (2): Table After applying the 'count' function.

	dept_name	Department_Teachers
1	Biology	1
2	Comp. Sci.	3
3	Elec. Eng.	1
4	Finance	2
5	History	2
6	Music	1
7	Physics	2

Table (3): Table after applying the ‘min’ function.

	TeacherMinname	dept_name
1	Crick	Biology
2	Brandt	Comp. Sci.
3	Kim	Elec. Eng.
4	Singh	Finance
5	Califieri	History
6	Mozart	Music
7	Einstein	Physics

Table (4): Table after applying the ‘max’ function.

	TeacherMaxname	dept_name
1	Crick	Biology
2	Srinivasan	Comp. Sci.
3	Kim	Elec. Eng.
4	Wu	Finance
5	El Said	History
6	Mozart	Music
7	Gold	Physics

Table (5): Table after applying the ‘avg’ function.

	DepartmentAVG_salary	dept_name
1	78480.000000	Biology
2	91879.733333	Comp. Sci.
3	87200.000000	Elec. Eng.
4	92650.000000	Finance
5	66490.000000	History
6	43600.000000	Music
7	99190.000000	Physics

Result and Discussion: In the initial step, we established an "instructor" table featuring an attribute named "Salary," which is of numeric type. It's important to note that the "average()" function is exclusively designed for numeric data. Other aggregate functions can handle a broader range of data types.

The "count" function, for instance, is versatile and can be applied to various data types. It's used in conjunction with the "group by" function, which categorizes the total number of instructors in each department. This enables us to ascertain the count of instructors within specific department groupings.

The "min" function, when dealing with numeric data, identifies the smallest number, whereas, with varchar data, it pinpoints the name beginning with the minimum alphabet (from 'a' to 'z'). The results are further grouped by department name. In contrast, the "max" function locates the name starting with the maximum alphabet and also groups results by department.

The "Average" function, being exclusive to numeric data, calculates the average salary within each department. This approach provides valuable insights into departmental salary averages, enhancing our understanding of the dataset.

Experiment Number: 08

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

Objectives: To understand the concept of SQL Commands to Connect MySQL Database with Java or PHP.

Theory: Connecting a MySQL database with PHP involves using PHP's MySQLi or PDO extensions, which facilitate the interaction between your PHP script and the MySQL database server. To establish a connection, specific database credentials, including the server's hostname or IP address, a username, and a password, are required. These credentials authenticate your PHP script with the database.

Once the connection is established, you can send SQL queries to the database to perform various operations, including data retrieval, insertion, updates, and deletions. The results of these queries can be processed within your PHP script for purposes such as displaying data on a web page or performing calculations.

Proper error handling is crucial for managing potential issues with database connections or query execution. Additionally, it's important to close the database connection when it's no longer needed to free up server resources. This process enables the creation of dynamic, data-driven web applications that can interact with and retrieve data from a MySQL database.

Experimental Requirements:

1. Laptop/ Desktop Computer.
2. Microsoft SQL Server application.
3. Microsoft SQL Server Management Studio application.

Source Code:

```
<?php
$connect=mysqli_connect("localhost","root","","student");

if(isset($_POST["insert"])){
   $id =$_POST["id"];
   $name=$_POST["name"];
   $sess=$_POST["session"];
   $phone=$_POST["ph_number"];
   $city=$_POST["city"];
   ;
```

```
    $insert="insert into Semester_Reg(ID,Name,Session,Phone_No,City)
     values('$id','$name','$sess', '$phone', '$city')";
    $result=mysqli_query($connect,$insert);
    if($result==1){
       echo"Successfully insert a record!";
    }else{
       echo"Unsucess";}
}if(isset($_POST["delete"])){
    $id =$_POST["id"];
    $name=$_POST["name"];
    $sess=$_POST["session"];
    $delete="delete from `semester_reg` where ID='$id'
    and Name='$name' and Session='$sess'";
    $result=mysqli_query($connect,$delete);
    if($result==1){
       echo"Successfully delete your record!";
    }else{
       echo"Unsucess";}
} if(isset($_POST["update"])){
    $id =$_POST["id"];
    $name=$_POST["name"];
    $sess=$_POST["session"];
    $phone=$_POST["ph_number"];
    $city=$_POST["city"];
    $insert="update semester_reg  set Name='$name',Session='$sess',Phone_No='$phone',
     City='$city' where ID='$id'";
    $result=mysqli_query($connect,$insert);
    if($result==1){
       echo"Successfully updated your record!";
    }else{
       echo"Unsucess";}
}if(isset($_POST["select"])){
    $query="SELECT * FROM semester_reg";
    $result=mysqli_query($connect,$query);
    if($result==true){
       echo "All  Registered Students List <br>";
    echo "<table cellpadding=10 border='1'>
    <tr>
       <th>ID</th>
       <th>Name</th>
       <th>Session</th>
       <th>Phone Number</th>
       <th>City</th>

    </tr>";
    if(mysqli_num_rows($result) > 0){
       while($row = mysqli_fetch_array($result)){
          echo "<tr>";
```

```
            echo "<td style='color:black'>" . $row['ID'] ."</td>";
            echo "<td style='color:black'>" . $row['Name'] . "</td>";
            echo "<td style='color:black'>" . $row['Session'] . "</td>";
            echo "<td style='color:black'>" . $row['Phone_No'] . "</td>";
            echo "<td style='color:black'>" . $row['City'] . "</td>";
            echo "</tr>";
        }echo "</table>";
    }
    } else{
        echo "No record found!";
    }
}
//end of show data

?>
<!DOCTYPE html>
<html>
<head>
    <title>Student Registration Form</title>
    <style type="text/css">
        body {
            text-align: center;
            font-family: Cambria, Cochin, Georgia, Times,
            'Times New Roman', serif, sans-serif;
            background-color: antiquewhite;
            padding: 10px;
        }h2 {
            font-size: 30px;
            margin-top: 20px;
            background-color:aquamarine;
            text-align: center;
        }table {
            margin: auto;
            font-size: 20px;
            border-collapse: collapse;
            background-color: blanchedalmond;
        }th, td {
            padding: 10px;
        }th {
            text-align: center;
            vertical-align: middle;
             }input[type="text"] {
            font-size: 20px;
            width: 100%;
            padding: 5px;
        }input[type="radio"] {
            margin-right: 15px;
        }label {
```

```
            color: crimson;
        }
    </style>
</head>
<body>
    <h2>Student's Registration Form</h2>
    <form method="post" action="">
        <table border="1">
            <tr>
                <th>ID</th>
                <td><input type="text" name="id" required></td>
            </tr>
            <tr>
                <th>Name</th>
                <td><input type="text" name="name" required></td>
            </tr>
            <tr>
                <th>Session</th>
                <td><input type="text" name="session" required></td>
            </tr>
            <tr>
                <th>Phone Number</th>
                <td><input type="text" name="ph_number"></td>
            </tr>
            <tr>
                <th>City</th>
                <td><input type="text" name="city" value=""></td>
            </tr>

            <tr>
                <td colspan="2" class="button-container">
                    <input type="submit" name="insert" value="Insert">
                    <input type="submit" name="select" value="Show">
                    <input type="submit" name="delete" value="Delete">
                    <input type="submit" name="update" value="Update">
                </td>
            </tr>
        </table>
        <br>
        <label style="color: red">N.B.</label>
        1. To delete a record, please enter your ID, Name, and Session.<br>
        2. You can update all information except the ID Number.<br>
        3. To show all records, enter your ID, Name, and Session.
    </form>
</body>
</html>
```

SQL code

```
create database student
create table Semester_Reg(
        Name varchar (20),
        Email varchar (25),
        Password char(11),
        Phone varchar(20),
        primary key(Phone))
```

Output:

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	Name	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
2	Email	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
3	Password	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
4	Phone	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More

Figure (1): "Student_Reg" table in "student" database in the XAMPP server.

Figure (2): HTML form, from here we will collect the PHP values.

	ID	Name	Session	Phone_No	City
Edit Copy Delete	12345	ASDDGG	2018-2019	017003344	Kushtia
Edit Copy Delete	190612	Arif	2018-2019	01700331456	Dinajput
Edit Copy Delete	200615	Gulam Mostafa	2019-2020	01700975999	Jamalpur
Edit Copy Delete	200618	Ibne Shihab	2019-2000	01517840907	Rajshahi
Edit Copy Delete	200627	Sharif Rahman	2019-2020	01755555666	Rajshahi

Figure (3): Inserted value "Student_Reg" table in "student" database in the XAMPP server via PHP.

Result and Discussion: In our experiment, we successfully collected values from an HTML form and transmitted these values to the corresponding PHP file using the POST method. The PHP file then processed and stored these values in our database, which was created using SQL. Subsequently, we observed that data submitted via the HTML form is correctly stored in the database. This successful process demonstrates the functionality of our data collection and storage system, indicating that our experiment was a success. It showcases the effective integration of HTML forms, PHP for server-side processing, and SQL for database management, ultimately achieving the desired outcome of data capture and storage.