Experiment No: 01

Experiment Name: Study and Implementation of DML Commands of SQL with suitable Example.

- Insert
- Delete
- update

Objectives:

1. To Understand and use of Data Manipulation Language to write query for database.
2. To study how to insert, delete and update data in the database.

Theory:

Data Manipulation Language (DML): A data manipulation language users to access or ~~manipulates~~ manipulates data as organized by the appropriate data model.

The types of access are:

i. Retrieval of information stored in the database.

ii. Insertion of new information into the database.

iii. Deletion of information from the database.

iv. Modification of information stored in the database.

Insertion: To insert data into a relation either specify a tuple to be inserted or write a query whose result is a set of tuples to be inserted.

The simplest "insert" statement is a request to insert one tuple. Suppose we want to insert the fact that there is a course CS-437 in the Computer science department with the title "Database systems" and four credit hours in a 'course' table. The statement will be:

```
insert into course
        values ('CS-437', 'Database systems',
                'Comp.sci.', 4);
```

In this example, the values are specified in the order in which the corresponding

attributes are listed in the relation schema.

It also can be written as:

```
insert into course (course-id, title,
                    dept-name, credits)
    values('CS-437', 'Database systems',
                    'Comp. Sci', 4);
```

Deletion:

A delete request is expressed in much the same any as a query. We can delete only whole tuples; we cannot delete values on only particular attributes

SQL expression of delete operation:

```
delete from r
where P;
```

where P represents Predicate and r represents a relation.

The delete statement first finds all tuples t in r for which P(t) is true and then deletes them from r. The where clause can be omitted; in which case all

tuples in r are deleted.

Example:

```
delete from instructor;
```

where 'instructor' is a table name.

update:

In certain situations, we may wish to change a value in a tuple without changing all values in the tuple. For this purpose the update statement can be used.

For example:
Suppose that annual salary increases are being made, and salaries of all instructors are to be increased by 5 percent.
we write:

```
update instructor
set salary = salary * 1.05;
```

Source code:

```
use master
Create database university
--- to use database university
use university
---Create a table named department
Create tabe department (
dept_name varchar(20),
building varchar(15),
budget numeric(15,2),
primary key(dept_name));

insert into department
values('Biology','John',75000)
insert into department
values('CSE','watson',90000)
insert into department
values('Physics','Jonny',85000)
insert into department
values('Chemistry','Priter',100000)
insert into department
values('ICE','Arrshan',65000)
insert into department
values('EEE','Max',95000)
---- to show the department table with
attributes and values
```

```
Select * from department
--- delete one tuple
delete from department where dept_name = 'Biology'
Select * from department
--- update department
update department set budget = budget * 1.05
where budget < 85000
Select * from department
```

Output:

	dept_name	building	budget
1	Biology	Watson	75000.00
2	Chemistry	Painter	100000.00
3	CSE	Taylor	90000.00
4	EEE	Taylor	95000.00
5	ICE	Watson	80000.00
6	Physics	Painter	85.000.00

	dept_name	building	budget
1	Chemistry	Painter	100000.00
2	CSE	Taylor	90000.00
3	EEE	Taylor	95000.00
4	ICE	Watson	80000.00
5	Physics	Painter	85000.00
6			

	dept_name	building	budget
1	Chemistry	Painter	100000.00
2	CSE	Taylor	90000.00
3	EEE	Taylor	95000.00
4	ICE	Watson	87106.23
5	Physics	Painter	85000.00

Experiment No: 02

Experiment Name: Study and Implementation of DDL Commands of SQL with suitable Example.

- Create
- Alter
- Drop

Objectives:

1. To understand and use of Data Definition Language to write query for database.
2. To study how to create, alter and drop table in database.

Theory:

We define SQL relation by using the create table command.

Data Definition Language:

We specify a database schema by a set of definitions expressed by a special language called a data definition language (DDL). The DDL is also used to specify additional properties of the data.

Create:

The following command creates a relation department is in the database:

```
create table department
(dept_name  varchar(20),
 building   varchar(15),
 budget     numeric(12,2),
 primary key (dept_name));
```

The general form of create table command is

```
create table r
    (A1, D1,
     A2, D2,
     - - - - -
     An, Dn,
    (integrity constraint),
    - - - - - -
    (integrity constraint ));
```

where r is the name of the relation, each Ai is the name of an attribute in the schema of relation r and Di is the domain of attribute Ai. That is

Di specifies the type of attribute Ai along with the optional constraints that restrict the set of allowed values for Ai. The semicolon shows the end of the create table statement.

Alter:

We use the alter command to add attribute to an existing relation. All tuples in the relation are assigned null as the value for the new attribute. The form of the alter table command is:

alter table r add A D;

where r is the name of existing relation. A is the name of the attribute to be added and D is the type of the added attribute.

We can drop attributes from a relation by the command:

alter table r drop A;

where r is the name of the existing relation. A is the name of an attribute of the relation.

Drop:

The drop command deletes all information about the dropped relation from the database.

The command is:

```
drop table r;
```

Source code:

```
use master
Create database university
use university
Create table instructor (
ID varchar (20),
name varchar (15) not null,
dept_name varchar (15),
Salary numeric(8,2),
Primary key(ID))
```

```
insert into instructor
values (10101, 'Anwar', 'Physics', 68000)
insert into instructor
values(10121, 'Rakib', 'Music', 85000)
insert into instructor
values(10333, 'Salam', 'Biology' 90000)
insert into instructor
values(14544, 'Kamal', 'Chemistry', 100000)
insert into instructor
values(15353, 'Samad', 'CSE', 95000)
insert into instructor
values(12453, 'Rafiq', 'ICE', 75000)

Select * from instructor
alter table instructor add course_id char(20)
drop table instructor
```

Output:

	ID	name	dept_name	Salary
1	10101	Anwar	physics	68000.00
2	10121	Rakib	Music	85000.00
3	10333	Salam	Biology	90000.00
4	12453	Rafiq	ICE	75000.00
5	14544	kamal	Chemistry	100000.00
6	15353	Samad	CSE	95000.00

	ID	name	dept_name	Salary	Course_id
1	10101	Anwar	physics	68000.00	NULL
2	10121	Rakib	Music	85000.00	NULL
3	10333	Salam	Biology	90000.00	NULL
4	12453	Rafiq	ICE	75000.00	NULL
5	14544	Kamal	Chemistry	100000.00	NULL
6	15353	Samad	CSE	95000.00	NULL

Experiment No: 03

Experiment Name: Study and Implementation of DML commands of

- select clause
- From clause
- Where clause

Objectives:

1. To understand and use of the SQL queries.
2. To study how to implement select, from, where clause in database.

Theory:

The basic structure of an SQL queries consists of three clauses:

- select
- from
- where

A query takes as its input the relations listed in the from clause, operates on

them as specified in the where and select clauses and then produces a relation as the result.

The role of each clause is as follows:

- The select clause is used to list the attributes desired in the result of a query.
- The from clause is a list of the relations to be accessed in the evaluation of the query.
- The where clause is a predicate involving attributes of the relation in the from clause.

A typical SQL query has the form;

select $A_1, A_2 \dots A_n$
from $r_1, r_2 \dots r_m$
where P;

Each A_i represents an attribute and each r_i a relation and P is a predicate.

If the where clause is omitted, the Predicate p is true.

Queries on a single relation

Let us consider an example,

"Find the names of the instructors". Instructor names are found in the instructor relation. So we put that relation in the from clause. The instructors name appears in the name attribute, so we put that in the select clause.

```
select name
from instructor;
```

The result is a relation consisting of a single attribute with the heading name.

Queries on Multiple Relations

Suppose we want to answer the following query:

"Retrieve the names of all instructors along with their department names

and department building name".

To answer the query, each tuple in the instructor relation must be matched with the tuple in the department relation whose dept_name value matches the dept_name value of the instructor tuple.

The query can be written in SQL as -

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

The select clause may contain ~~at~~ arithmetic expressions involving the operations +, -, * and / operating on constants or attribute of tuples.

For example,

```
Select ID, salary * 1.1
from instructor;
```

Source Code:

```
use university
Select dept_name from instructor;
Select name from instructor
where dept_name = 'Physics';
```

Output:

	dept_name
1	Physics
2	Music
3	Biology
4	ICE
5	Chemistry
6	CSE

	name
1	Anwar

Experiment No: 04

Experiment Name: Study and Implementation of DML Commands of

- Group By and Having clause
- Order By clause
- Create view, Indexing and Procedure clause

Objectives:

1. To understand and use of the SQL queries.
2. To study how to implement group by, having, order by clause in SQL code.
3. To create view, indexing and procedure clause in database.

Theory:

There are circumstances where we would like to apply aggregate function, in these case we use group by and having clause.

The aggregate functions are :

- Average : avg
- Maximum : max
- Minimum : min
- Total : sum
- Count : count

Group by clause

In some cases we apply aggregate function not only to a single set of tuples, but also to a group of sets of tuples; we specify this in SQL using group by clause. The attribute or attributes given in the group by clause are placed in one group.

Example:

Find the average salary in each department

Query:

```
Select dept_name, avg(salary) as avg_salary
from instructor
group by dept_name;
```

Having clause:

It is useful to state a condition that applies to groups rather than to tuples.

For example, we want to see only these departments where the average salary of the instructors is more than 42000. This condition does not apply to a single tuple, rather it applies to each group constructed by the group by clause. To express such a query, we use having clause.

we express this query:

```
select dept-name, avg(salary) as avg-salary
from instructor
group by dept-name
having avg(salary) > 42000;
```

Order by clause:

SQL offers the users to some control over the order in which tuples in a relation are displayed.

The 'order By' clause causes the tuples in the result of a query to appear in sorted order.

For example:

```
Select *
from instructor
Order by salary desc, name asc;
```

By default, the order By clause lists items in ascending order. To specify the sort order, we may specify 'desc' for descending order, "asc" for ascending order.

Create view:

We define a view in SQL by using the create view command. To define a view, we must give the view a name and must state the query that computes the view.

The form of the create view command is:

Create view v as <query expression>;

where <query expression> is any legal query expression. The view name is represented by v.

Indexing:

An index on an attribute of a relation is a data structure that allows the database system to find those tuples in the relation that have a specified value for that attribute efficiently without scanning through all the tuples of the relation.

We create an index with the create index command, which takes the form:

create index <index_name> on <relation_name>
(<attribute_list>);

Create index dept_name on instructor
(dept_name);

Procedure:

A stored procedure is a set of SQL statements with an assigned name, which are stored in a relational database management system as a group. So it can be reused and shared by multiple programs.

Syntax:

```
CREATE RROCEDURE Procedure_name
AS
BEGIN
SQL QUEURY
END
EXEC procedure_name;
```

Source code:

```
use university
select dept_name, avg(salary) as avg_salary
from instructor
group by dept_name;

select dept_name, avg(salary) as avg_salary
from instructor
group by dept_name
having avg(salary)>75000
```

```
Select * from instructor order by salary
                                  desc, name asc;

Create view faculty as
select ID, name, dept_name
from instructor

CREATE PROCEDURE instruct_proc
AS
BEGIN
Select name as authors_name from instructor
where ID = '10121'
END
EXEC instruct_proc
Select * from instructor
```

Output:

	dept_name	avg_salary
1	Biology	90000.000000
2	Chemistry	100000.000000
3	CSE	95000.000000
4	ICE	75000.000000
5	Music	85000.000000
6	physics	68000.000000

	dept_name	avg_salary
1	Biology	90000.000000
2	Chemistry	100000.000000
3	CSE	95000.000000
4	Music	85000.000000

	ID	name	dept_name	Salary	Course_id
1	14544	Kamal	Chemistry	100000.00	NULL
2	15353	Samad	CSE	95000.00	NULL
3	10333	Salam	Biology	90000.00	NULL
4	10121	Rakib	Music	85000.00	NULL
5	12453	Rafiq	ICE	75000.00	NULL
6	10101	Anwar	Physics	68000.00	NULL

	Authors_name
1	Rakib

Experiment No: 05

Experiment Name: study and Implementation of SQL Commands of Join operations with example.

- Cartesian product
- Natural join
- Left ~~or~~ outer join
- Right outer join
- Full outer join

Objectives:

1. To understand and use of the cartesian Product in SQL queries on database.
2. To study the commands of Join operations and their implementation on database.

Theory:

Join operation:

Join operation that allow the programmer to write some queries in a more natural way and to express some queries that are difficult to do with only the cartesian product.

The goal of creating a join condition is that it helps to combine the data from two or more DBMS tables.
It is denoted by ⋈.

The Natural Join:

The natural join operation operates on two relations and produces a relation as the result. Unlike the Cartesian product of two relations which concatenates each tuple of the first relation with every tuple of the second natural join considers only those pairs of tuples with the same value on those attributes that appear in the schemas of both relations.

For example:
We write the query for all students in the university who have taken some course, find their names and the course ID of all courses they took as:

```
Select name, Course_id
from student natural join takes;
```

Syntax:

Select $A_1, A_2 \dots A_n$
from r_1 natural join r_2 natural join ---
natural join r_m
where P;

Outer join:

The outer join operation works in a manner similar to the join operations, but it preserves those tuples that would be lost in a join by creating tuples in the result containing null values.

There are three forms of outer join:

i) The Left outer join -

The left outer join preserves tuples only in the relation named before the left outer join operation.

Example:

Select *
from r_1 natural left outer join r_2;

ii) The right outer join-

The right outer join preserves tuples only in the relation named after the right outer join operation.

Example:

```
Select A1, A2--- An
from r1 natural right outer join r2;
```

iii) The full outer join-

The full outer join preserves tuples in both relations it is the union of a left outer join and the corresponding right outer join.

Example:

```
Select *
from r1 natural full outer join r2;
```

Cartesian product:

The cartesian product operation allow us to combine information from any two relation.

Source code:

```
use university
Select * from instructor
select * from department

---Cartesian product
select building, department.dept_name, salar
from department, instructor
where department.dept_name = instructor.dept_nam
--- join operation
select building, salary
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 instructor right outer join
                                department on
department.dept_name = instructor.dept_name
--- full outer join
select * from instructor full outer join department
                                                on
department.dept_name = instructor.dept_name
```

Output:

Cartesian product

	building	dept-name	Salary
1	Painter	Physics	68000.00
2	Watson	Biology	90000.00
3	Watson	ICE	75000.00
4	painter	Chemistry	100000.00
5	Taylor	CSE	95000.00

Join operation:

	building	Salary
1	Painter	68000.00
2	Watson	90000.00
3	Watson	75000.00
4	painter	100000.00
5	Taylor	95000.00

left outer join:

	dept_name	building	budget	ID	name	dept_name	Salary
1	Biology	Watson	75000.00	10333	Salam	Biology	90000.00
2	Chemistry	Painter	100000.00	14544	Kamal	Chemistry	100000.00
3	CSE	Taylor	90000.00	15353	Samad	CSE	95000.00
4	EEE	Taylor	95000.00	NULL	NULL	NULL	NULL
5	ICE	Watson	87106.23	12453	Rafiq	ICE	75000.00
6	Physics	Painter	85000.00	10101	Anwar	physics	68000.00

right outer join:

	ID	name	dept_name	Salary	dept_name	building	budget
1	10333	Salam	Biology	90000.00	Biology	Watson	75000.00
2	14544	Kamal	Chemistry	100000.00	Chemistry	Painter	100000.00
3	15353	Samad	CSE	95000.00	CSE	Taylor	90000.00
4	NULL	NULL	NULL	NULL	EEE	Taylor	95000.00
5	12453	Rafiq	ICE	75000.00	ICE	Watson	87106.23
6	10101	Anwar	physics	68000.00	physics	Painter	85000.00

Full outer join:

	ID	name	dept_name	salary	dept_name	building	budget
1	10101	Anwar	physics	68000.00	physics	Painter	85000.00
2	10121	Rakib	Music	85000.00	NULL	NULL	NULL
3	10333	Salam	Biology	90000.00	Biology	watson	75000.00
4	12453	Rafiq	ICE	75000.00	ICE	watson	87106.23
5	14544	Kamal	Chemistry	100000.00	Chemistry	Painter	100000.00
6	15353	Samad	CSE	95000.00	CSE	Taylor	90000.00
7	NULL	NULL	NULL	NULL	EEE	Taylor	95000.00

Experiment No: 06

Experiment Name: Study and Implementation of Aggregate Function with Example.

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

Objectives:

1. To understand and use of the aggregate function on database.
2. To study how to implement count(), max(), min(), avg() function in SQL on database.

Theory:

Aggregate functions are functions that take a collection of values as input and return a single value.

SQL ~~offers~~ offers five standard built-in aggregate functions.

- Average : avg
- Minimum : Min
- Maximum : Max
- Total : sum
- Count : count

The input to sum and avg must be a collection of numbers but the other operations can operate on collections of nonnumeric data types, such as string as well.

Avg() : This avg() function returns the average value of some collection of numbers in a data table.

Example :

```
select avg(salary) as ~~avg~~ average_salary
from instructor
where dept_name = "physics";
```

Min() : This min() function returns the minimum value from some collection of numbers under an attribute in a relation.

Example:

```
Select min(salary) as minimum_salary
from instructor
where dept_name = "physics";
```

This query returns the minimum salary of a physics department.

max(): max() function returns the maximum value from some collection of numeric values under an attribute in a relation.

Example:

```
select max(salary) as maximum_Salary
from instructor
where dept_name = "physics";
```

This query returns the maximum salary of a physics department.

Count(): Count() function returns number of tuples in a relation usually.

The notation for this function in SQL is Count(*).

Example:

To find the number of tuples in the instructor relation.

```
select count(*)
from instructor;
```

There are some cases where we must eleminate duplicates ~~det~~ before computing an aggregate function.

For example:

```
select count(distrinct ID)
from teaches
where semester='spring' and year=2018;
```

This returns the total number of instructors who teach a course in the Spring 2018 semester.

Source code:

```
use university
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 :

	count_ID
1	6

	max_salary
1	100000.00

	min_salary
1	68000.00

	avg_salary
1	85500.000000

Experiment No: 07

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

Objectives:

1. To understand the triggering system on Database table.
2. To understand how trigger can be used for automatically updating a table when an insert/update/delete statement takes place in another table.

Theory:

Trigger is a special type of procedure which is attached to a table and is only executed when an Insert, update or Delete occurs on that table. One has to specify the

modification actions that fire the trigger when it is created.

Unlike stored procedures, trigger can not be explicitly executed.

Once we enter a trigger into the database, the database system takes on the responsibility of executing it whenever the specified event occurs and the corresponding condition is satisfied.

Trigger cannot usually perform updates outside the database. Triggers can be used to implement certain integrity constraints that cannot be specified using the constraint mechanism of SQL.

Syntax:

```
CREATE TRIGGER trigger_name on table_name
FOR INSERT | DELETE | UPDATE
AS
BEGIN
"TRIGGER BODY"
END
```

where create trigger creates or replaces an existing trigger with the trigger_name.

On table_name: This specifies the name of the table associated with the trigger.

FOR INSERT| DELETE | UPDATE : This specifies the DML operation.

Trigger_body : This provides the operation to be performed as trigger is fired.

Source code:

```
Create database ~~sto~~ storetable
use storetable
Create table customer(
cust_id char(5) primary key check(cust_id
like('[cs][0-9][0-9][0-9][0-9]')),
cust_fname char(12) NOT NULL,
cust_lname varchar(12),
cust_address TEXT);
insert Customer values('c0001', 'Farziha',
                       'Mahzabin', 'Sherpur')
```

```
Create table Items (
item_id char(5) primary key check(item_id
like('[P][0-9][0-9][0-9][0-9]')),
item_name char(12),
item_category char(10),
item_price float(12) check(item_price>=0),
item_qoh int check(item_qoh>=0),
item_last_sold datetime default getdate())

insert into Items
values('P0001','Jamuna','laptop',125.5,26,'10-02-2023')
insert into Items
values('P0002','Realme','phone',80.5,30,'5-14-2023')
insert into Items
values('P0003','Hp','laptop',130.5,20,'2-18-2023')
Select * from items

Create table transact(
tran_id char(10) check(tran_id like('[T][0-9][0-9]
[0-9][0-9][0-9][0-9][0-9][0-9]')),
item_id char(5) foreign key references
Item(item_id),
cust_id char(5) foreign key references
customer(cust_id),
```

```
tran_type char(1),
tran_quantity int check(tran_quantity>0),
tran_date datetime default getdate())
Select * from transact

---- Trigger
Create trigger test on Item FOR INSERT
AS
BEGIN
Declare @item_id char(5), @amount char(12),
                    @tran_type char(1)
select @item_id=item_id,@amount = tran_quantity,
@tran_type= tran_type from inserted
IF(@tran_type ='S')
   update Items set item_qoh = item_qoh_@amount
where item_id =@item_id
EISE
update Items set item_qoh=item_qoh +
@amount where item_id =@item_id
END
insert transact values ('T00000001', 'P0003,'
                'C0001', 'S', 5, '3-01-2023)
```

```
Select * from transact
Select * from Items
```

output :

Before triggering "Items" table

	item_id	item_name	item_Category	item_price	item_qoh	item_last_sold
1	P0001	Jamuna	laptop	125.5	26	10-02-2023
2	P0002	Realme	phone	80.5	30	5-14-2023
3	P0003	Hp	laptop	130.5	20	2-18-2023

Before triggering "transact" table

tran_id	item_id	cust_id	tran_type	tran_quantity	tran_date

After triggering "transact" table

	tran_id	item_id	cust_id	tran_type	tran_quantity	tran_date
1	T00000001	P0003	C0001	S	5	3-01-2023

After triggering "Items" table

	item_id	item_name	item_Category	item_price	item_qoh	item_last_sold
1	P0001	Jamuna	laptop	125.5	26	10-02-2023
2	P0002	Realme	phone	80.5	25	5-14-2023
3	P0003	Hp	laptop	130.5	20	2-18-2023

Experiment No : 08

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

Objectives:

1. To understand and use the SQL queries on database.
2. To Study the SQL commands to connect MySQL Database with Java or PHP.

Theory:

To connect Java application with the MySQL database, we need to follow 5 following steps.

In this example we are using MySQL as the database. So we need to know following informations for the mysql database:

1. Driver class: The driver class for the mysql database is com.mysql.jdbc.Driver.

2. Connection URL: The connection URL for the mysql database is jdbc:mysql://localhost:3306/sonoo where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IPaddress, 3306 is the port number and sonoo is the database name. We may use any database, in such case we need to replace the sonoo with our database name.

3. Username: The default username for the mysql database is root.

4. Password: It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Source Code :

```
import java.sql.*;
public class TestSQLJava
{
        Connection con;
        TestSQLJava()
        {
            Create Connection();
            //showData();
            //showCustomData("172-32-1176");
            // updateData("172-32-1176","CA");
        }
    Void createConnection()
    {
        try
        {
            //Class.forName("Sun.Jdbc.Odbc.OdbcO
                        JdbcOdbcDriver");
            Con=DriverManager.getConnection
            ("Jdbc:Odbc:TestData");}
```

```
catch(Exception exp){ sys system.out.print
        ("No Connection with DB"+exp);}
}
Void showData()
{
    try
    {
        Statement st=con.createStatement();
        ResultSet rs = st.executeQuery
                ("select* from Authors");
        System.out.println("Author ID:\
                tAuthor last name:");
        while(rs.next())
        System.out.println(rs.getString(1)+
        "\t"+rs.getString(2));
    }
    Catch(Exception exp){ System.out.print
    ("Probleb with query result"+exp);}
}
```

```
void updateData(String au_id, String state)
{
    try
    {
        preparedStatement pst =
        con.prepareStatement("update
        authors set state=? where au_id=?");
        pst.setString(1, state);
        pst.setString(2, au_id);
        pst.executeUpdate();
    }
    Catch(Exception exp) { System.out.print
    ("Probleb with query result"+exp); }
}
public static void main(String args[])
{
    new TestSQLJava();
}
}
```