

# INDEX

**Experiment 1:** Study and Implementation of DML Commands of SQL with Suitable Examples-

- Insert
- Delete
- Update

**Experiment 2:** Study and Implementation of DDL Commands of SQL with Suitable Examples-

- Create
- Alter
- Drop

**Experiment 3:** Study and Implementation of DML Commands of

- Select Clause
- From Clause
- Where Clause

**Experiment 4:** Study and Implementation of DML Commands of

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

**Experiment 5:** Study and Implementation of SQL Commands of Join Operations with Examples-

- Cartesian Product
- Natural Join
- Left Outer Join
- Right Outer Join
- Full Outer Join

**Experiment 6:** Study and Implementation of Aggregate Function with Examples-

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

**Experiment 7:** Study and Implementation of Triggering System on Database Table Using SQL Commands with Examples.

**Experiment 8:** Study and Implementation of SQL Commands to Connect MySQL Database with Java or PHP.

## Experiment No: 01

Name of Experiment: 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 is a language that enables users to access or manipulate 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.

(a) Insertion: To insert data into a relation we 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 -

is a course CS-437 in the computer science department with the title "Database systems" and four credit hours in a 'courses' table. The statement will be:

insert into course

values ('CS-437', Database Systems, 'com.Sci', 4)

(b) Deletion: A delete request is expressed in much the same way as a query. We can delete only whole tuples; we cannot delete values on only particular attributes.

SQL expression of delete operation:

delete from  $\text{r}$  where  $P$ ;

Here,  $P$  represents predicate and  $\text{r}$  represents a relation.

A delete command operates on only one relation. If we want to delete tuples from several relations, we must use one delete command for each relation.

Example:-

delete from instructor;

where 'instructor' is a table name.

(c) 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 instruction are to be increased by 5 percent we write.

update instructor set salary = salary \* 1.05 .

### Source code:

use master

create database University-190609

-- To use the University-190609 database

use University-190609

-- Create a table named department which has 3 attributes

create table department (

dept-name varchar (20),

building. varchar (15),

budget numeric (12,2),

primary key (dept-name));

insert into department values ('Biology', 'Watson', 9000)

insert into department values ('Comp. Sci', 'Taylor', 10000)

insert into department values ('Elec. Eng', 'Taylor', 85000)

insert into department values ('Finance', 'Pointer', 82000)

insert into department values ('History', 'Pointer', 50000)

insert into department values ('music', 'Packard', 80000)

insert into department values ('physics', 'Watson', 70000)

-- To show the department table with attributes and values

select \* from department;

-- To delete one tuple

delete from department where dept-name = 'Biology';

-- To update department

update department set budget = budget \* 1.05 where budget <

85000;

## Output:

	dept_name	building	budget
1	Biology	Watson	90000.00
2	Comp.Sci	Taylor	100000.00
3	Ebc.Eng.	Taylor	85000.00
4	Finance	Painter	120000.00
5	History	Painter	75000.00
6	MUSIC	Packard	60000.00
7	physics	Watson	70000.00

	dept-name	building	budget
1	Com.Sci	Taylor	100000.00
2	Ebc.Eng	Taylor	85000.00
3	Finance	Painter	120000.00
4	History	Painter	75000.00
5	MUSIC	Packard	60000.00
6	physics	Watson	70000.00

	dept-name	building	budget
1	Com.Sci	Taylor	100000.00
2	Ebc.Eng	Taylor	85000.00
3	Finance	Painter	120000.00
4	History	Painter	72500.00
5	MUSIC	Packard	84000.00
6	physics	Watson	73500.00

Experiment No: 02

Name of the experiment: Study and Implementation of DDL command 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:

Data Definition Language: We specify a database schema by a set of definition 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 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));
```

Where  $r$  is the name of the relation, each  $A_i$  is the name of an attribute in the schema of relation  $r$  and  $D_i$  is the domain of attribute  $A_i$ .

### Drop:

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

The command is : `drop table r;`

### Alter:

We use the alter command to add attribute to an existing relation. All tuples in the relation are assigned null or 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 an 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.

Source Code:

use muster

create database university

use university

-- Create a table named instructor

create table instructor (

ID varchar (5),

name varchar (20) not null,

dept-name varchar (20),

salary numeric (6,2),

primary key (ID));

insert into instructor values ('10101', 'Srinivasan', 'Com.Scii', 65000)

insert into instructor values ('12121', 'Wu', 'Finance', 90000)

insert into instructor values ('15151', 'Mozart', 'music', 40000)

insert into instructor values ('22222', 'Einstein', 'physics', 65000)

insert into instructor values ('32343', 'EI Said', 'History', 60000)

insert into instructor values ('33456', 'Gold', 'Physics', 82000)

select \* from instructor

alter table instructor add course-no char(20);

drop table instructor

## Output:

	ID	name	dept-name	Salary
1	10101	Srinivasan	Comp. Sci.	65000.00
2	12121	Wu	Finance	90000.00
3	15151	Mozart	MUSIC	40000.00
4	22222	Einstein	Physics	9500.00
5	32343	E1 said	History	60000.00
6	33456	Gcold	physics	87000.00

	ID	name	dept-name	Salary	Course-no
1	10101	Srinivasan	Comp. Sci	65000.00	NULL
2	12121	Wu	Finance	90000.00	NULL
3	15151	Mozart	MUSIC	40000.00	NULL
4	22222	Einsten	Physics	95000.00	NULL
5	32343	E1 said	History	60000.00	NULL
6	33456	Gcold	physics	87000.00	NULL

Experiment No: 03

Name of the Experiment: 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 query consists of three clause :

- (i) select
- (ii) from
- (iii) where

A query takes as its input the relations listed in the from clause, operation, 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:

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

A typical SQL query has the form:

Select A<sub>1</sub>, A<sub>2</sub>...A<sub>n</sub> from r<sub>1</sub>, r<sub>2</sub>...r<sub>m</sub> where P;

Each A<sub>i</sub> represents an attribute and each r<sub>i</sub> a relation and P is a predicate. If the where clause is omitted the predicate P is true.

Queries on a 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;

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

Source code:

use University

Select dept-name from instructor;

Select name from instructor where dept-name='physics';

Output:

	dept-name
1	Comp.Science
2	Finance
3	MUSIC
4	physics
5	History
6	physics

	name
1	Einstein
2	Grold

Experiment No: 04

Name of the Experiment: 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 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 function are:

- (i) Average : avg
- (ii) Maximum: max
- (iii) Minimum: min
- (iv) Total : sum
- (v) 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

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 those departments where the average salary of the instructors is more than 42000.

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 the tuples in the result of a query to appear in sorted order.

For example,

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

### 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>;`

### 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 dept-index 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 PROCEDURE ProcedureName
AS
BEGIN
SQL QUERY
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) > 42000;

select \* from instructor order by salary desc, name asc;

create view faculty as select ID, name, dept-name from  
instructor;

create index dept-index on instructor(dept-name);

CREATE PROCEDURE instruct\_Proc

AS

BEGIN

select name as authors\_name from instructor where ID = 15151

END

EXEC instruct\_Proc

select \* from instructor;

Output:

	dept-name	avg-Salary
1	Comp. Sci	65000.000000
2	Finance	90000.000000
3	History	60000.000000
4	MUSIC	40000.000000
5	physics	91000.000000

	dept-name	avg-Salary
1	Comp. Sci	65000.000000
2	Finance	90000.000000
3	History	60000.000000
4	physics	91000.000000

	ID	name	deptname	Salary
1	22222	Einstein	Physics	95000.00
2	12121	Wu	Finance	90000.00
3	33456	Grold	Physics	87000.00
4	10101	Srinivasan	Comp. sci	65000.00
5	32343	El said	History	60000.00
6	15151	Mozart	MUSIC	40000.00

	authors-name
1	Mozart

Experiment No: 05

Name of the experiment: Study and Implementation of SQL commands of join operations with Examples -

- Cartesian Product
- Right outer-join
- Natural join
- Full outer join
- Left 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 operations that allow the programmer to write some queries in a more natural way 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  $\bowtie$ .

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

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<sub>1</sub>, A<sub>2</sub>... A<sub>n</sub> from r<sub>1</sub> natural join r<sub>2</sub> natural  
join ... natural join r<sub>m</sub> 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<sub>1</sub> natural left outer join r<sub>2</sub>;

(ii) The right-outer join: The right-outer join preserves tuples only in the relation named after the right outer join operation.

Example - Select A<sub>1</sub>, A<sub>2</sub>,... A<sub>n</sub> from r<sub>1</sub> natural right outer  
join r<sub>2</sub>;

### (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.

```
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 instructors;
```

```
select * from departments;
```

#### -- Cartesian Product

```
select building, departments.dept-name, salary from
departments, instructors where departments.dept-name = instructors.
dept-name;
```

#### -- join operation

```
select salary, building from departments join instructors. dept-name
=departments.dept-name;
```

#### -- left outer join

```
select * from departments left outer join instructors on
departments.dept-name = instructor.dept-name;
```

-- Right outer join

select \* from instructors right outer join departments on  
departments.dept-name = instructors.dept-name;

-- Full Outer join

select \* from instructors full outer join department on  
departments.dept-name = instructors.dept-name;

Output :-

Cartesian Product

	building	dept-name	Salary
1	Taylor	Comp.Sei	65000.00
2	Painter	Finance	90000.00
3	Packard	MUSIC	40000.00
4	Watson	physics	95000.00
5	Painter	History	60000.00
6	Watson	physics	87000.00

Join operation

	Salary	building
1	65000.00	Taylor
2	90000.00	Painter
3	40000.00	Packard
4	95000.00	Watson
5	60000.00	Painter
6	87000.00	Watson

left outer join:

	dept-name	building	budget	ID	name	dept-name	salary
1	Biology	Watson	90000.00	NULL	NULL	NULL	NULL
2	CompSci	Taylor	100000.00	10101	Srinivasan	CompSci	65000.00
3	Elec. Eng	Taylor	85000.00	NULL	NULL	NULL	NULL
4	Finance	Painter	120000.00	12121	Wu	Finance	90000.00
5	History	Painter	70000.00	32343	El said	History	60000.00
6	MUSIC	Packard	80000.00	15151	Mozart	MUSIC	40000.00
7	physics	Watson	70000.00	22222	Einstein	physics	95000.00
8	physics	Watson	70000.00	33456	Gold	physics	87000.00

right outer join:

	ID	name	dept-name	salary	dept-name	building	budget
1	NULL	NULL	NULL	NULL	Biology	Watson	90000.00
2	10101	Srinivasan	CompSci	65000.00	CompSci	Taylor	100000.00
3	NULL	NULL	NULL	NULL	Elec. Eng	Taylor	85000.00
4	12121	Wu	Finance	90000.00	Finance	Painter	120000.00
5	32343	El said	History	60000.00	History	Painter	70000.00
6	15151	Mozart	MUSIC	40000.00	MUSIC	Packard	80000.00
7	22222	Einstein	physics	95000.00	physics	Watson	70000.00
8	33456	Gold	physics	87000.00	physics	Watson	70000.00

full outer join:

	ID	name	dept-name	salary	dept-name	building	budget
1	10101	Srinivasan	CompSci	65000.00	CompSci	Taylor	100000.00
2	12121	Wu	Finance	90000.00	Finance	Painter	120000.00
3	15151	Mozart	MUSIC	40000.00	MUSIC	Packard	80000.00
4	22222	Einstein	physics	95000.00	physics	Watson	70000.00
5	32343	El said	History	60000.00	History	Painter	70000.00
6	33456	Gold	physics	87000.00	physics	Watson	70000.00
7	NULL	NULL	NULL	NULL	Biology	Watson	90000.00
8	NULL	NULL	NULL	NULL	Elec. Eng	Taylor	85000.00

Experiment No: 06

Name of the experiment: Study and Implementation of Aggregate Function with example.

- Count Function
- Min Function
- Max 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() Functions in SQL on database.

Theory:

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

SQL offers five standard built-in aggregate functions.

- |                 |                 |
|-----------------|-----------------|
| • Average : avg | • maximum : max |
| • Minimum : min | • 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 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 is SQL is COUNT (\*).

Example- To find the number of tuples in the instructor relation we can write,

select count (\*) from instructor;

There are some cases where we must eliminate duplicates before computing an aggregate function.

For example :

select count (distinct 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 instructors;

Select max (Salary) as max-Salary from instructors;

Select min (Salary) as min-Salary from instructors;

Select avg (Salary) as avg-Salary from instructors;

Output:

Count-ID	
1	6

Max-Salary	
1	95000.00

Min-Salary	
1	40000.00

Avg-Salary	
1	72833.333333

→ 6 ↘

Experiment No: 08

Name of the experiment: Study and implementation of Triggering system on Database Table using SQL commands with examples.

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 execute 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 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 storetable
use storetable
create table customer (
cusl_id char(5) Primary key check (cusl_id like ('[CS][0-9]
[0-9][0-9]')),
cusl_fname char(12) NOT NULL,
cusl_lname varchar(12),
cusl_address TEXT);

```

insert customer values ('C0001', 'Rahatul', 'Rabbi', 'Balmonirhat')  
 create table Items (  
 item\_id char(5) primary key check (item\_id like ('[P][0-9]  
 [0-9][0-9]')),  
 item\_name char (12),  
 item\_category char (10),  
 item\_price float (12) check (item\_price >= 0),  
 item\_qty int check (item\_qty >= 0),  
 item\_sold datetime default getdate());  
 Select \* from Items;  
 insert Items values ('P0001', 'Jamuna', 'Laptop', 125.5, 26,  
 '10-2-2023');  
 insert Items values ('P0003', 'Realme', 'Phone', 80.5, 30, '15-14-2023')  
 insert Items values ('P0004', 'HP', 'Laptop', 130.5, 20, '2-18-2023')  
 create table transact (  
 tran\_id char (8) check (tran\_id like ('[T][0-9][0-9][0-9]  
 [0-9][0-9][0-9][0-9]')),  
 item\_id char (5) foreign key references Items (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 Items 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\_qon = item\_qon - @amount  
where item\_id = @item\_id

ELSE

Update Items set item\_qon = item\_qon + @amount  
where item\_id = @item\_id

END

Insert transact values ('T00000001', 'P0003', 'CO01', 'S',  
5, '3-12-2023')

select \* from transact;

select \* from Items;

Output:

Before triggering 'Items' table

	item_id	item_name	Item_Catagory	Item_Price	item_qty	item_last_sold
1	P0001	Jamuna	Laptop	125.5	26	2023-10-02
2	P0003	Realme	Phone	80.5	30	2023-05-14
3	P0004	HP	Laptop	130.5	20	2023-02-18

Before triggering 'transact' table

tran-id	item-id	custid	tran-type	tran-quantity	tran-date

After triggering 'transact' table

	tranid	itemid	cuslid	transtype	tran-quantity	tran-date
1	T000001	P0003	C0001	S	5	2023-3-12

After triggering 'Items' table

	Item_id	item_name	item_catagory	item_price	item_qty	item_last_sold
1	P0001	Jamuna	Laptop	125.5	26	2023-10-02
2	P0003	Realme	Phone	80.5	25	2023-05-14
3	P0004	HP	Laptop	130.5	20	2023-02-18

### Experiment No: 08

Name of the Experiment: Study and implementation of SQL commands to Connect MySQL Database with java or PHP.

#### Objectives:

- (i) To understand about SQL commands.
- (ii) To learn how to create a database in XAMPP.MYSQL.
- (iii) To learn how to create a table in database.
- (iv) To learn how to connect database with PHP.

Theory MySQL is an open source relational database management system. It is commonly used in web application to store and manage data.

PHP is a popular server side scripting language used to create dynamic web page. The combination of MySQL and PHP provides an efficient way to manage data in web application.

Source code:connect.sql :

```

create database student
create table student-list (
stu-id varchar(12) primary key,
stu-name text(12),
stu-gpa varchar(6));
insert into student-list (stu-id, stu-name, stu-gpa) values
('190609', 'MD RAHATUL', '3.75');

```

index.php

## &lt;?php

```

$connect = mysqli_connect ('localhost', 'root',
                           '', 'student');

```

```

if ($connect) {

```

```

    echo "Connect successfully";

```

```

} else {

```

```

    echo "</br>";

```

```

    die("Database is not connected");
}

```

```
?>
```