Experiment no: 01

Experiment name: Study and Implementation of DML Commands of SQL with Suitable Example.

- Insert
- Delete
- Update

Theory:

Data Manipulation Language (DML) commands in SQL are used to manage data records stored within the database tables. They do not deal with changes to database objects and their structure. The most common DML commands are INSERT, UPDATE, and DELETE.

INSERT

The INSERT command is used to insert new data records into a database table. The syntax for the INSERT command is as follows:

SQL

```
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 is used to remove data records from a database table. The syntax for the DELETE command is as follows:

SQL

```
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 data records in a database table. The syntax for the UPDATE command is as follows:

SQL

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

where:

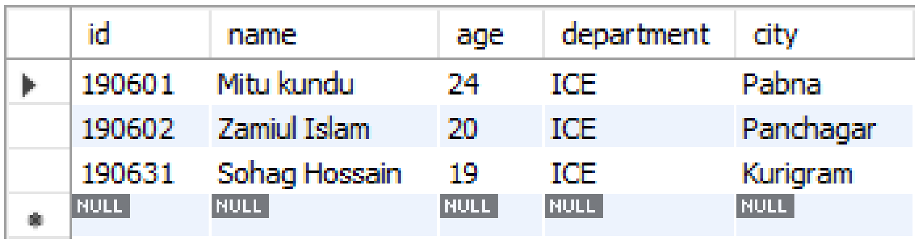
- table_name is the name of the database table in which the records will be updated.

- 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.

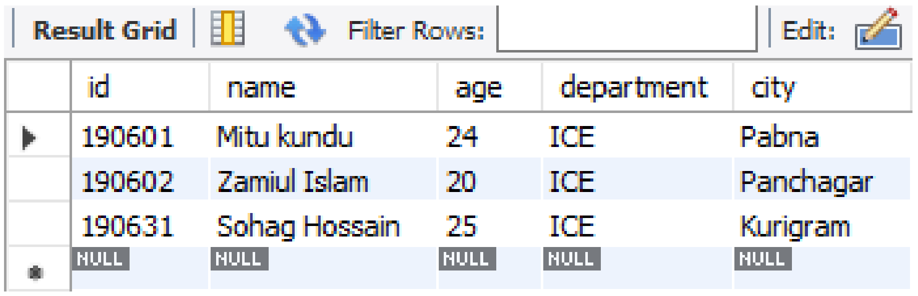
SQL Code:

```
-- 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
(190601, "Mitu kundu", 24, "ICE", "Pabna"),
(190602, "Zamiul Islam", 20, "ICE", "Panchagar"),
(190631, "Sohag Hossain", 19, "ICE", "Kurigram");
-- to view the table
select * from students;
-- update the students table
update students set age=25 where id=190631;
select * from students;
-- delete a row from the students table
delete from students where id=190601;
select * from students;
-- delete table
drop table students;
-- delete database
drop database university;
```

Output:

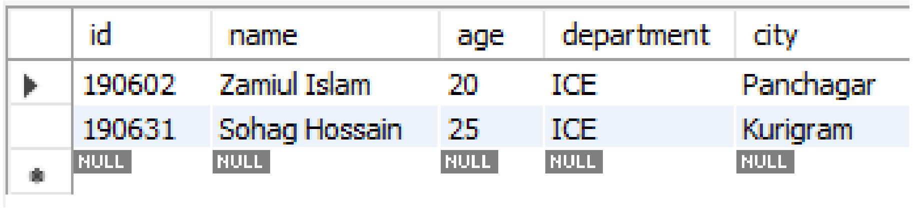


id	name	age	department	city
190601	Mitu kundu	24	ICE	Pabna
190602	Zamiul Islam	20	ICE	Panchagar
190631	Sohag Hossain	19	ICE	Kurigram
NULL	NULL	NULL	NULL	NULL



Result Grid Filter Rows: Edit:

id	name	age	department	city
190601	Mitu kundu	24	ICE	Pabna
190602	Zamiul Islam	20	ICE	Panchagar
190631	Sohag Hossain	25	ICE	Kurigram
NULL	NULL	NULL	NULL	NULL



id	name	age	department	city
190602	Zamiul Islam	20	ICE	Panchagar
190631	Sohag Hossain	25	ICE	Kurigram
NULL	NULL	NULL	NULL	NULL

Experiment no: 02

Experiment name: Study and Implementation of DDL Commands of SQL with Suitable Example

- Create
- Alter
- Drop

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

```
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

```
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

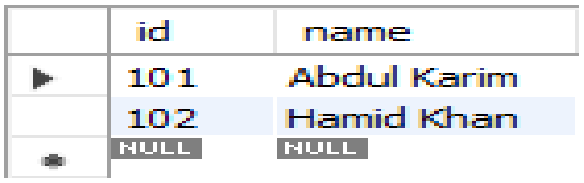
```
DROP TABLE table_name;
```

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

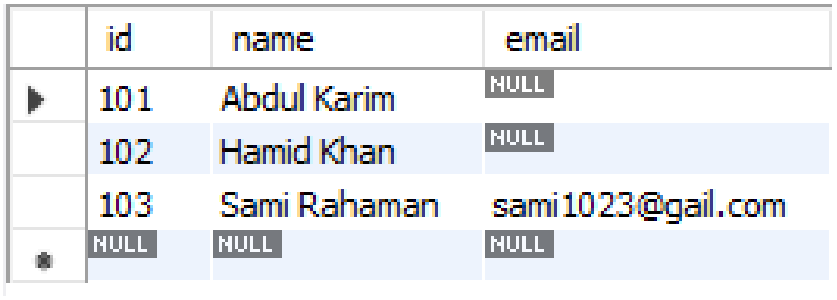
SQL Code:

```
use university;
-- create new table called employe
create table employe(
   id int primary key,
   name varchar(100) not null
);
insert into employe values(101, "Abdul Karim");
insert into employe values(102, "Hamid Khan");
select * from employe;
-- add new column in the employe table
alter table employe
     add email varchar(100);
insert into employe values(103, "Sami Rahaman", "sami1023@gail.com");
select * from employe;
-- delete table
drop table employe;
```

Output:



	id	name
▶	101	Abdul Karim
	102	Hamid Khan
●	NULL	NULL



	id	name	email
▶	101	Abdul Karim	NULL
	102	Hamid Khan	NULL
	103	Sami Rahaman	sami1023@gail.com
●	NULL	NULL	NULL

Experiment no: 03

Experiment name: Study and Implementation of DML Commands of

- Select Clause
- From Clause
- Where Clause

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

```
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

```
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

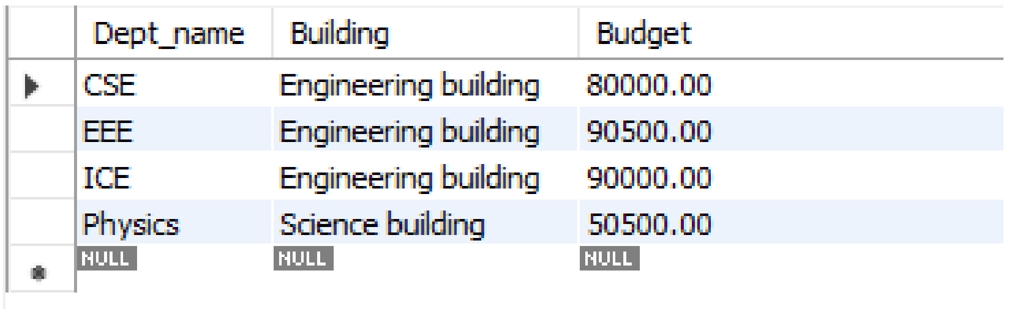
```
WHERE condition1 AND condition2 AND ...;
```

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 <=.

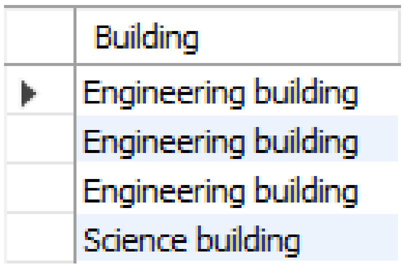
SQL Code:

```
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';
```

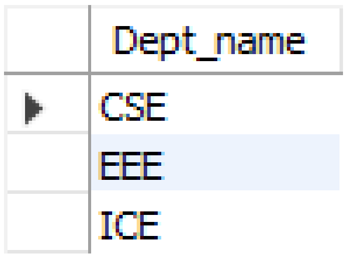
Output:



Dept_name	Building	Budget
CSE	Engineering building	80000.00
EEE	Engineering building	90500.00
ICE	Engineering building	90000.00
Physics	Science building	50500.00
NULL	NULL	NULL



Building
Engineering building
Engineering building
Engineering building
Science building



Dept_name
CSE
EEE
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

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

```
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

```
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

```
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

```
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

```
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

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

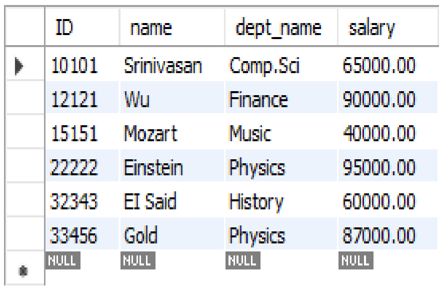
The procedure_name is the name of the new procedure. The parameter1, parameter2, etc. are the names of the input and output parameters for the procedure. The BEGIN and END keywords enclose the body of the procedure.

SQL Code:

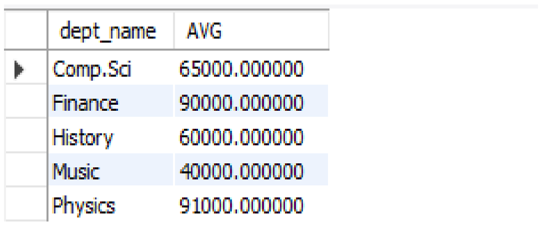
```
use university;
create table instructor(

        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);
```

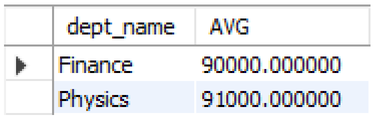
Output:



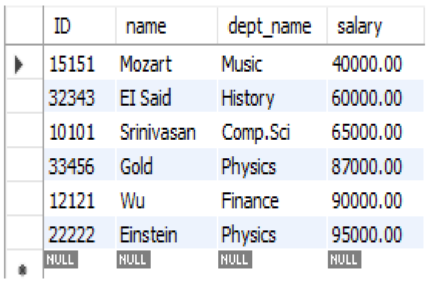
ID	name	dept_name	salary
10101	Srinivasan	Comp.Sci	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
NULL	NULL	NULL	NULL



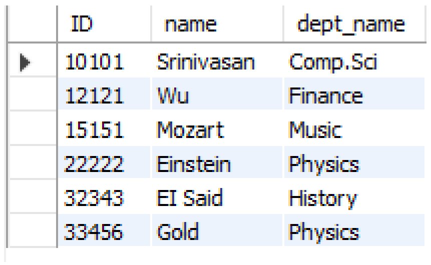
dept_name	AVG
Comp.Sci	65000.000000
Finance	90000.000000
History	60000.000000
Music	40000.000000
Physics	91000.000000



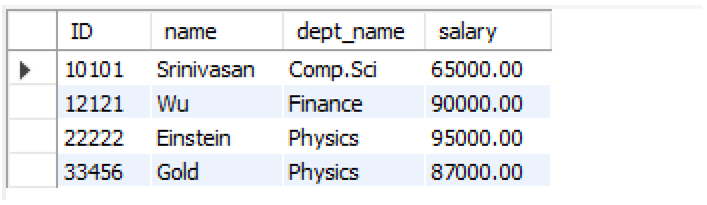
dept_name	AVG
Finance	90000.000000
Physics	91000.000000



ID	name	dept_name	salary
15151	Mozart	Music	40000.00
32343	El 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
10101	Srinivasan	Comp.Sci
12121	Wu	Finance
15151	Mozart	Music
22222	Einstein	Physics
32343	El Said	History
33456	Gold	Physics



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

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

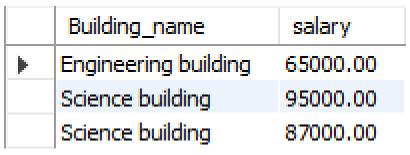
```
SELECT * FROM table1 FULL JOIN table2 ON table1.column_name = table2.column_name;
```

SQL Code:

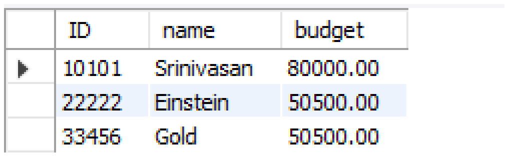
```
use university;
create table instruct(
        ID varchar(20) primary key,
        name varchar(20) not null,
        dept_name varchar(20),
        salary numeric(8,2)
);

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;
```

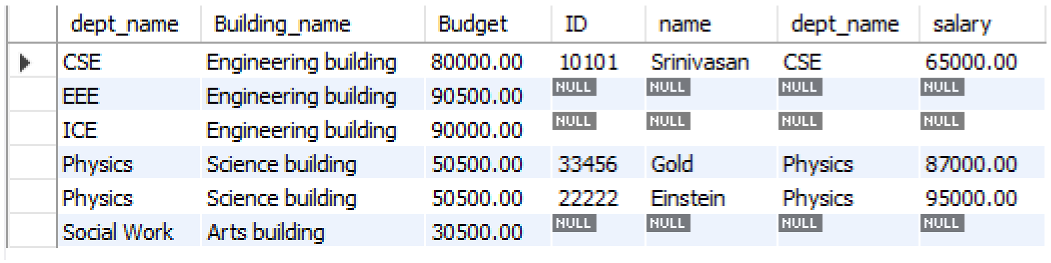
Output:



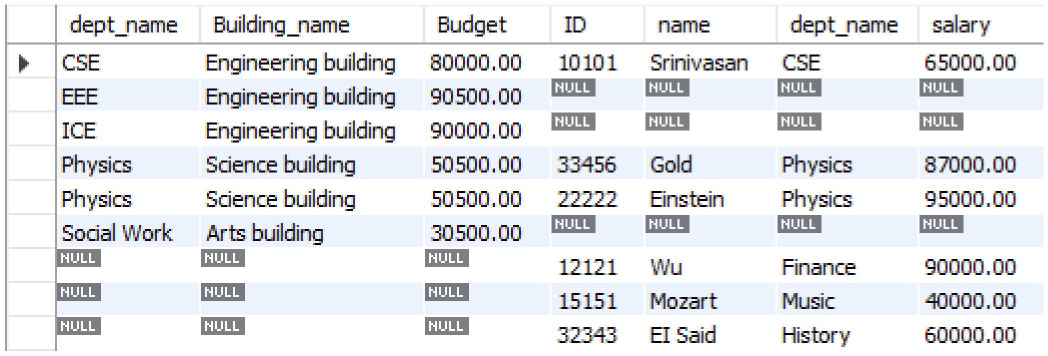
Building_name	salary
Engineering building	65000.00
Science building	95000.00
Science building	87000.00



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



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



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

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:

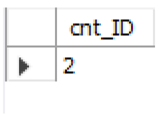
- COUNT(): Counts the number of rows in a column or the number of non-NULL values in a column.
- MAX(): Returns the largest value in a column.
- MIN(): Returns the smallest value in a column.
- AVG(): Calculates the average of the values in a column.

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

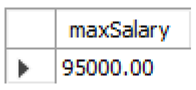
SQL Code:

```
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;
```

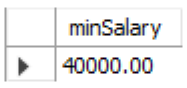
Output:



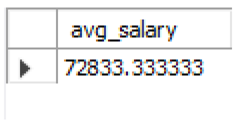
```
cnt_ID
2
```



```
maxSalary
95000.00
```



```
minSalary
40000.00
```



```
avg_salary
72833.333333
```

Problem no: 07

Problem name: Study and Implementation of Triggering System on Database Table Using SQL Commands with Example.

Theory:

A database trigger is a programmed procedure that is automatically executed when a specific event occurs on a database table, such as an INSERT, UPDATE, or DELETE statement. Triggers can be used to enforce data integrity, automate tasks, and audit changes to data.

To create a trigger in SQL, you use the CREATE TRIGGER statement. The syntax for the CREATE TRIGGER statement is as follows:

SQL

```
CREATE TRIGGER trigger_name
ON table_name
FOR {INSERT | UPDATE | DELETE}
[AS]
trigger_body;
```

The trigger_name is the name of the trigger. The table_name is the name of the table on which the trigger is created. The FOR clause specifies the event that will trigger the trigger. The trigger_body is the code that will be executed when the trigger is triggered.

Triggers can be either simple or complex. Simple triggers typically perform a single task, such as updating a column in another table. Complex triggers can perform multiple tasks, such as logging changes to data or sending an email notification.

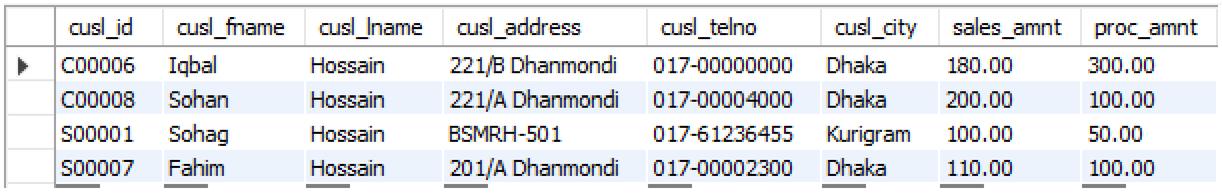
SQL Code:

```
create database shop;
use shop;

CREATE TABLE CustomerAndSuppliers (
   cusl_id CHAR(6) PRIMARY KEY CHECK (cusl_id REGEXP '^[CS][0-9]{5}$'),
   cusl_fname CHAR(15) NOT NULL,
   cusl_lname VARCHAR(15),
   cusl_address TEXT,
   cusl_telno CHAR(12) CHECK (cusl_telno REGEXP '^[0-9]{3}-[0-9]{8}$'),
   cusl_city CHAR(12) DEFAULT 'Rajshahi',
   sales_amnt DECIMAL(10, 2) CHECK (sales_amnt >= 0),
   proc_amnt DECIMAL(10, 2) CHECK (proc_amnt >= 0)
);

insert into CustomerAndSuppliers
(cusl_id, cusl_fname, cusl_lname, cusl_address, cusl_telno, cusl_city, sales_amnt, proc_amnt)
values
('S00001','Sohag','Hossain','BSMRH-501','017-61236455','Kurigram',100,50),
('C00006','Iqbal','Hossain','221/B Dhanmondi','017-00000000', 'Dhaka', 100,200),
('S00007','Fahim','Hossain','201/A Dhanmondi','017-00002300', 'Dhaka', 50,100),
('C00008','Sohan','Hossain','221/A Dhanmondi','017-00004000', 'Dhaka', 200,100);

create table Item(
	item_id char(6) primary key check (item_id regexp ('^[P][0-9]{5}$')),
	item_name char(12),
	item_catagory char(10),
	item_price decimal(10, 2) check(item_price>=0),
	item_qoh int check(item_qoh>=0),
	item_last_sold datetime  default current_timestamp
);

insert into Item
(item_id, item_name, item_catagory, item_price, item_qoh)
values
('P00001', 'i-phone12', 'phone', 50, 1),
('P00002', 'i-phone14', 'phone', 10, 3),
('P00003', 'i-phone13', 'phone', 30, 2),
('P00004', 'i-phone15', 'phone', 40, 1);


create table Transactions(
	tran_id char(10) primary key check (tran_id regexp ('^[T][0-9]{9}$')),
	item_id char(6),
	cusl_id char(6),
	tran_type char(1),
	tran_quality int check (tran_quality>=0),
	tran_date datetime default current_timestamp,
   foreign key(item_id) references item(item_id),
   foreign key(cusl_id) references CustomerAndSuppliers(cusl_id)
);

DELIMITER //
CREATE TRIGGER trg_update_item
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN
```

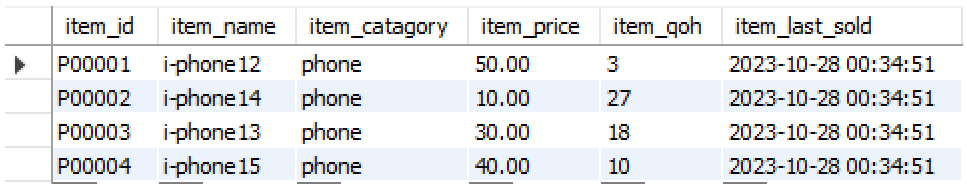
```
   DECLARE price DECIMAL(10, 2);

   SELECT item_price INTO price FROM Item WHERE item_id = new.item_id;

   IF new.tran_type = 'S' THEN
      UPDATE Item SET item_qoh = item_qoh - NEW.tran_quality WHERE item_id = new.item_id;
      UPDATE CustomerAndSuppliers SET sales_amnt = sales_amnt + (price * new.tran_quality) WHERE
cusl_id = new.cusl_id;
   ELSE
      UPDATE Item SET item_qoh = item_qoh + NEW.tran_quality WHERE item_id = new.item_id;
      UPDATE CustomerAndSuppliers SET proc_amnt = proc_amnt + (price * new.tran_quality) WHERE
cusl_id = new.cusl_id;
   END IF;
END;
//
DELIMITER ;
insert Transactions(tran_id ,item_id,cusl_id,tran_type,tran_quality)
values('T000000001','P00002','C00006','S',1);

insert Transactions(tran_id ,item_id,cusl_id,tran_type,tran_quality)
values('T000000002','P00001','S00007','P',1);
select * from Transactions;
select * from CustomerAndSuppliers;
select * from Item;
```

Output:

Before Transaction:

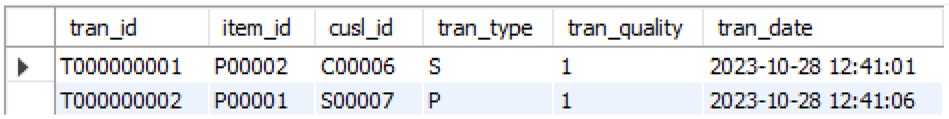


cusl_id	cusl_fname	cusl_lname	cusl_address	cusl_telno	cusl_city	sales_amnt	proc_amnt
C00006	Iqbal	Hossain	221/B Dhanmondi	017-00000000	Dhaka	180.00	300.00
C00008	Sohan	Hossain	221/A Dhanmondi	017-00004000	Dhaka	200.00	100.00
S00001	Sohag	Hossain	BSMRH-501	017-61236455	Kurigram	100.00	50.00
S00007	Fahim	Hossain	201/A Dhanmondi	017-00002300	Dhaka	110.00	100.00

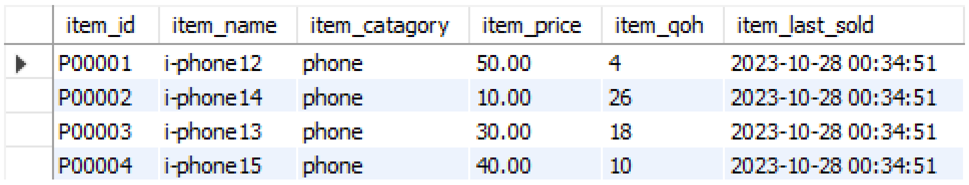


item_id	item_name	item_catagory	item_price	item_qoh	item_last_sold
P00001	i-phone12	phone	50.00	3	2023-10-28 00:34:51
P00002	i-phone14	phone	10.00	27	2023-10-28 00:34:51
P00003	i-phone13	phone	30.00	18	2023-10-28 00:34:51
P00004	i-phone15	phone	40.00	10	2023-10-28 00:34:51

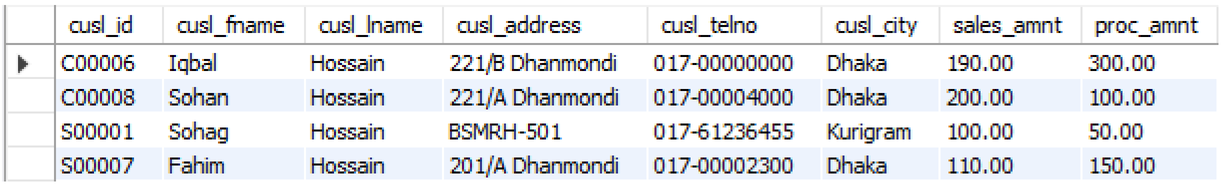
After Transaction:



tran_id	item_id	cusl_id	tran_type	tran_quality	tran_date
T000000001	P00002	C00006	S	1	2023-10-28 12:41:01
T000000002	P00001	S00007	P	1	2023-10-28 12:41:06



item_id	item_name	item_catagory	item_price	item_qoh	item_last_sold
P00001	i-phone12	phone	50.00	4	2023-10-28 00:34:51
P00002	i-phone14	phone	10.00	26	2023-10-28 00:34:51
P00003	i-phone13	phone	30.00	18	2023-10-28 00:34:51
P00004	i-phone15	phone	40.00	10	2023-10-28 00:34:51



cusl_id	cusl_fname	cusl_lname	cusl_address	cusl_telno	cusl_city	sales_amnt	proc_amnt
C00006	Iqbal	Hossain	221/B Dhanmondi	017-00000000	Dhaka	190.00	300.00
C00008	Sohan	Hossain	221/A Dhanmondi	017-00004000	Dhaka	200.00	100.00
S00001	Sohag	Hossain	BSMRH-501	017-61236455	Kurigram	100.00	50.00
S00007	Fahim	Hossain	201/A Dhanmondi	017-00002300	Dhaka	110.00	150.00