STELLA MARYS COLLEGE OF ENGINEERING

(Approved by AICTE, New Delhi, Affiliated to Anna University, Chennai, Accredited by NAAC,NBA)

Aruthenganvilai, Kallukatti Junction, Azhikal Post, Kanyakumari District - 629 202

CS 3481– Data Base Management Systems

Regulation-2021

Computer science and engineering

(SEMESTER-IV)

STELLA MARYS COLLEGE OF ENGINEERING

(Approved by AICTE, New Delhi, Affiliated to Anna University, Chennai, Accredited by NAAC,NBA)

Aruthenganvilai, Kallukatti Junction, Azhikal Post, Kanyakumari District - 629 202

Bonafide Certificate

This is to certify that this is a bonafide record of the work done by

Mr/Ms ..... Of ..... semester B.E.Department of..... Register Number.....

In the .....Laboratory during the academic year 20 - 20 (odd/even).

Faculty-in-charge

(with Name,Date,Designation&Dept)

Head of the department

(with Name,Date, &Dept)

Submitted for the Practical Examination held on .....

Internal Examiner

External Examiner

LIST OF EXPERIMENTS

1. Create a database table, add constraints (primary key, unique, check, Not null), insert rows,update and delete rows using SQL DDL and DML commands.
2. Create a set of tables, add foreign key constraints and incorporate referential integrity.
3. Query the database tables using different 'where' clause conditions and also implementaggregate functions.
4. Query the database tables and explore sub queries and simple join operations.
5. Query the database tables and explore natural, equi and outer joins.
6. Write user defined functions and stored procedures in SQL.
7. Execute complex transactions and realize DCL and TCL commands.
8. Write SQL Triggers for insert, delete, and update operations in a database table.
9. Create View and index for database tables with a large number of records.
10. Create an XML database and validate it using XML schema.
11. Create Document, column and graph based data using NOSQL database tools.
12. Develop a simple GUI based database application and incorporate all the above-mentioned features
13. Case Study using any of the real life database applications from the following list

a) Inventory Management for a EMart Grocery Shop
b) Society Financial Management
c) Cop Friendly App – Eseva
d) Property Management – eMall
e) Star Small and Medium Banking and Finance

- Build Entity Model diagram. The diagram should align with the business and functionalgoals stated in the application.
- Apply Normalization rules in designing the tables in scope.
- Prepared applicable views, triggers (for auditing purposes), functions for enablingenterprise grade features.
- Build PL SQL / Stored Procedures for Complex Functionalities, ex EOD Batch Processing for calculating the EMI for Gold Loan for each eligible Customer.
- Ability to showcase ACID Properties with sample queries with appropriate settings

Course Objectives & Course Outcomes

COURSE OBJECTIVES:

- To learn and implement important commands in SQL.
- To learn the usage of nested and joint queries.
- To understand functions, procedures and procedural extensions of databases.
- To understand design and implementation of typical database applications.
- To be familiar with the use of a front end tool for GUI based application development.

COURSE OUTCOMES:

At the end of this course, the students will be able to:

CO1: Create databases with different types of key constraints.
CO2: Construct simple and complex SQL queries using DML and DCL commands.
CO3: Use advanced features such as stored procedures and triggers and incorporate in GUI basedapplication development.
CO4: Create an XML database and validate with meta-data (XML schema).
CO5: Create and manipulate data using NOSQL database.

CO's- PO's & PSO's MAPPING

CO's	PO's												PSO's		
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3
1	3	3	3	3	-	-	-	-	3	1	3	2	2	3	2
2	2	2	3	2	2	-	-	-	1	2	3	3	2	1	2
3	3	3	2	1	1	-	-	-	1	1	1	3	2	3	3
4	1	3	3	3	1	-	-	-	1	1	3	2	3	1	3
5	3	2	1	1	1	-	-	-	2	2	3	1	3	1	2
AVg.	2	3	2	2	1	-	-	-	2	1	3	2	2	2	2

1 - low, 2 - medium, 3 - high, '-"- no correlation

Department of Computer science and Engineering

POs, PSOs & PEOs

PROGRAMME OUTCOMES (POs):

1.Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one‘s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Programme Specific Outcomes(PSOs)

At the completion of the programme, the students will be able to:

PSO1:

Use data management techniques and algorithmic thinking for Software Design and Development practices.

PSO2:

Develop reliable IT solutions based on the expertise in Distributed Applications Development, Web Designing and Networking for various societal needs and entrepreneurial practices ethically.

PSO3:

Manage multidisciplinary environments effectively through their interpersonal and analytical skills and be responsible members and leaders of the society.

Program Educational Objectives(PEOs)

PEO1:

Graduates will be competent in creating innovative technologies through inter- disciplinary research and comprehensive skills sets that are suitable for the global computing industry.

PEO2:

Graduates will be capable of managing leading positions with a broad understanding of the application of ethics in evolving computer-based solutions for the societal needs.

PEO3:

Graduates will imbibe entrepreneurial qualities and develop their career by upgrading their, communication, analytical and professional skills constantly.

Table of Contents

EX: NO: 1 CREATION OF TABLES (DDL COMMANDS)

AIM:

To execute and verify the Data Definition Language commands.

SQL Command Categories

SQL commands are grouped into four major categories depending on their functionality. They are as follows:

Data Definition Language (DDL)

These SQL commands are used for creating, modifying, and dropping the structure of database objects. The commands are CREATE, ALTER, DROP, RENAME, and TRUNCATE.

Data Manipulation Language (DML)

These SQL commands are used for storing, retrieving, modifying, and deleting data. These commands are SELECT, INSERT, UPDATE, and DELETE.

Transaction Control Language (TCL)

These SQL commands are used for managing changes affecting the data. These commands are COMMIT, ROLLBACK, and SAVEPOINT.

Data Control Language (DCL)

These SQL commands are used for providing security to database objects. These commands are GRANT and REVOKE.

DDL (DATA DEFINITION LANGUAGE)

- CREATE
- ALTER
- DROP
- TRUNCATE
- RENAME

PROCEDURE

STEP 1: Start
STEP 2: Create the table with its essential attributes.
STEP 3: Execute different Commands and extract information from the table.

STEP 4: Stop

SQL COMMANDS

1. COMMAND NAME: **CREATE**
COMMAND DESCRIPTION: **CREATE** command is used to create objects in the database.

CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1<DATATYPE> (SIZE), COLUMN NAME.1 <DATATYPE> (SIZE)……);

Syntax For Create A from an Existing Table With All Fields

CREATE TABLE <TARGET TABLE NAME> AS SELECT * FROM <SOURCE TABLE NAME>;

2. COMMAND NAME: **DROP**
COMMAND DESCRIPTION: **DROP** command is used to delete the object from the database.

Syntax for drop a new column:

ALTER TABLE <TABLE NAME> DROP COLUMN <COLUMN NAME>;

Syntax for drop a table:

Drop table <tablename>;

3. COMMAND NAME: **TRUNCATE**
COMMAND DESCRIPTION: **TRUNCATE** command is used to remove all the records from the table

Syntax truncating the tables.

Truncate table <tablename>;

4. COMMAND NAME: **ALTER**
COMMAND DESCRIPTION: **ALTER** command is used to alter the structure of database.

ALTER <TABLE NAME> MODIFY <COLUMN NAME> <DATATYPE>(SIZE);

Syntax for alter table with multiple column:

SQL > ALTER <TABLE NAME> MODIFY <COLUMN NAME1> <DATATYPE>

(SIZE), MODIFY <COLUMN NAME2> <DATATYPE> (SIZE)…….;

Syntax for add a new column:

SQL> ALTER TABLE <TABLE NAME> ADD (<COLUMN NAME1> <DATA TYPE> <SIZE>,<COLUMN NAME2> <DATA TYPE> <SIZE>,……);

5. COMMAND NAME: **RENAME**
COMMAND DESCRIPTION: **RENAME** command is used to rename the objects.

Syntax For Renaming A table

Rename table <oldname> To <newname>;

Syntax For Renaming A Column

ALTER TABLE tablename **RENAME COLUMN** old column name **TO** new column name;

Data base commands:

Create database: create database <databasename>;

Show database: show databases;

Use: use <databasename>;

Show table: show tables;

Description of a table: desc <tablename>;

QUERY: 01
Q1. Write a query to create a table employee with empno, ename, designation, and salary.

QUERY: 01

SQL>CREATE TABLE EMP (EMPNO INT(4),ENAME VARCHAR(10),DESIGNATION VARCHAR(10),SALARY FLOAT(8,2));

Table created.

QUERY: 02
Q2. Write a query to display the column name and datatype of the table employee.

SQL> DESC EMP;

Name Null? Type
--- -------- ------------
EMPNO NUMBER(4)
ENAME VARCHAR2(10)
DESIGNATIN VARCHAR2(10)
SALARY NUMBER(8,2)
QUERY: 03

Q3. Write a query for create a new table from an existing table with all the fields.

QUERY: 03
SQL> CREATE TABLE EMP1 AS SELECT * FROM EMP;
Table created.
SQL> DESC EMP1
Name Null? Type
--- -------- ------------------
EMPNO NUMBER(4)
ENAME VARCHAR(10)
DESIGNATIN VARCHAR(10)
SALARY NUMBER(8,2)

QUERY: 04
Q4. Write a query to create a table from an existing table with selected fields.

Syntax
SQL> CREATE TABLE <TARGET TABLE NAME> SELECT EMPNO, ENAME FROM <SOURCE TABLE NAME>;

QUERY: 04
SQL> CREATE TABLE EMP2 AS SELECT EMPNO, ENAME FROM EMP;
Table created.

SQL> DESC EMP2;

Name Null? Type
--- -------- ----------------------
EMPNO NUMBER (4)
ENAME VARCHAR(10)

ALTER & MODIFICATION ON TABLE

QUERY: 06
Q6. Write a Query to Alter the column EMPNO NUMBER (4) TO EMPNO NUMBER(6).

QUERY: 06
SQL>ALTER TABLE EMP MODIFY EMPNO NUMBER (6);
Table altered.

SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(6)
ENAME		VARCHAR(10)
DESIGNATIN		VARCHAR(10)
SALARY		NUMBER(8,2)

QUERY: 07

Q7. Write a Query to Alter the table employee with multiple columns (EMPNO, ENAME.)

Syntax for alter table with multiple column:

SQL > ALTER <TABLE NAME> MODIFY <COLUMN NAME1> <DATATYPE> (SIZE), MODIFY <COLUMN NAME2> <DATATYPE> (SIZE)…….;

QUERY: 07
SQL>ALTER TABLE EMP MODIFY EMPNO INT (7),MODIFY ENAME VARCHAR(12));

Table altered.

SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(7)
ENAME		VARCHAR(12)
DESIGNATIN		VARCHAR(10)
SALARY		NUMBER(8,2);

QUERY: 08

Q8. Write a query to add a new column in to employee

QUERY: 08

SQL> ALTER TABLE EMP ADD QUALIFICATION VARCHAR2(6);
Table altered.
SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(7)
ENAME		VARCHAR(12)
DESIGNATIN		VARCHAR(10)
SALARY		NUMBER(8,2)
QUALIFICATION		VARCHAR2(6)

QUERY: 09

Q9. Write a query to add multiple columns in to employee

SQL>ALTER TABLE EMP ADD (DOB DATE, DOJ DATE);

Table altered.

SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(7)
ENAME		VARCHAR(12)
DESIGNATIN		VARCHAR2(10)
SALARY		NUMBER(8,2)
QUALIFICATION		VARCHAR(6)
DOB		DATE
DOJ		DATE

QUERY: 10

Q10. Write the query to change the table name emp as employee

SQL> Rename table emp to employee;

QUERY: 11

Q11. Write the query to change the column name empno to eno of the table employee

SQL> ALTER TABLE employee **RENAME COLUMN** EMPNO **TO** ENO;

SQL> DESC EMPLOYEE;

Name	Null?	Type
ENO		NUMBER(7)
ENAME		VARCHAR(12)
DESIGNATION		VARCHAR(10)
SALARY		NUMBER(8,2)
QUALIFICATION		VARCHAR2(6)

DOB **DATE**
DOJ **DATE**

REMOVE / DROP

QUERY: 12

Q12. Write a query to drop a column from an existing table employee

SQL> ALTER TABLE EMPLOYEE DROP COLUMN DOJ;

SQL> DESC EMP;

Name	Null?	Type
ENO		NUMBER(7)
ENAME		VARCHAR2(12)
DESIGNATIN		VARCHAR2(10)
SALARY		NUMBER(8,2)
QUALIFICATION		VARCHAR2(6)
DOB		DATE

QUERY: 13

Q13. Write a query to truncate table employee

SQL> truncate table employee;

QUERY: 14

Q14. Write a query to drop table employee

SQL> drop table employee;

DML COMMANDS

Data Manipulation Language (DML)

These SQL commands are used for storing, retrieving, modifying, and deleting data. These commands are SELECT, INSERT, UPDATE, and DELETE.

DML (DATA MANIPULATION LANGUAGE)

- **SELECT**- It is used to retrieve information from the table. It is generally referred to as querying the table.

- **INSERT-** This is used to add one or more rows to a table. The values are separated by commas and the data types char and date are enclosed in apostrophes. The values must be entered in the same order as they are defined.

- **DELETE-** After inserting row in a table we can also delete them if required. The delete command consists of a from clause followed by an optional where clause.

- **UPDATE**- It is used to alter the column values in a table. A single column may be updated or more than one column could be updated.

PROCEDURE:

STEP 1: Start.
STEP 2: Create the table with its essential attributes.
STEP 3: Insert the record into table.
STEP 4: Update the existing records into the table.
STEP 5: Delete the records in to the table.

SQL COMMANDS

1. COMMAND NAME: **INSERT**
COMMAND DESCRIPTION: INSERT command is used to Insert objects in the database.

2. COMMAND NAME: **SELECT**
COMMAND DESCRIPTION: SELECT command is used to SELECT the object from the database.

3. COMMAND NAME: **UPDATE**
COMMAND DESCRIPTION: **UPDATE** command is used to UPDATE the records from the table

4. COMMAND NAME: **DELETE**
COMMAND DESCRIPTION: DELETE command is used to DELETE the Records form the table

INSERT

QUERY: 01
Q1. Write a query to insert the records in to employee.

Syntax for Insert Records in to a table:
SQL > INSERT INTO <TABLE NAME> VALUES< VAL1, 'VAL2',.....);
A(
QUERY: 01

INSERT A RECORD INTO AN EXISTING TABLE:

MYSQL>INSERT INTO EMP VALUES(101,'NAGARAJAN','LECTURER',15000);
MYSQL >INSERT INTO EMP VALUES(102 ,'SARAVANAN',' LECTURER',15000);
MYSQL >INSERT INTO EMP VALUES(103,'PANNERSELVAM',' ASST. PROF,20000);
MYSQL >INSERT INTO EMP VALUES(104,'CHINNI HOD', 'PROF',45000);

1 row created.

SELECT

QUERY: 02

Q3. Write a query to display the records from employee.

Syntax for select Records from the table:

SQL> SELECT * FROM <TABLE NAME>;

QUERY: 02

DISPLAY THE EMP TABLE:

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATIN	SALARY
101	NAGARAJAN	LECTURER	15000
102	SARAVANAN	LECTURER	15000
103	PANNERSELVAM	ASST. PROF	20000
104	CHINNI HOD,	PROF	45000

UPDATE

QUERY: 04
Q1. Write a query to update the records from employee.
Syntax for update Records from the table:

SQL> UPDATE <<TABLE NAME> SET <COLUMNANE>=<VALUE> WHERE <COLUMN NAME=<VALUE>;

QUERY: 04
SQL> UPDATE EMP SET SALARY=16000 WHERE EMPNO=101;
1 row updated.

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATIN	SALARY
101	NAGARAJAN	LECTURER	16000
102	SARAVANAN	LECTURER	15000
103	PANNERSELVAM	ASST. PROF	20000
104	CHINNI HOD,	PROF	45000

UPDATE MULTIPLE COLUMNS

QUERY: 05
Q5. Write a query to update multiple records from employee.

Syntax for update multiple Records from the table:

SQL> UPDATE <<TABLE NAME> SET <COLUMNAME>=<VALUE> WHERE <COLUMN NAME=<VALUE>;

QUERY: 05

SQL>UPDATE EMP SET SALARY = 16000, DESIGNATIN='ASST. PROF' WHERE EMPNO=102;
1 row updated.

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATIN	SALARY
101	NAGARAJAN	LECTURER	16000
102	SARAVANAN	ASST. PROF	15000
103	PANNERSELVAM	ASST. PROF	20000
104	CHINNI HOD,	PROF	45000

DELETE

QUERY: 06
Q5. Write a query to delete records from employee.

Syntax for delete Records from the table:
SQL> DELETE <TABLE NAME> WHERE <COLUMN NAME>=<VALUE>;
QUERY: 06

SQL> DELETE EMP WHERE EMPNO=103;

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATIN	SALARY
101	NAGARAJAN	LECTURER	16000
102	SARAVANAN	ASST. PROF	15000
104	CHINNI HOD,	PROF	45000

Result:

Thus the DDL, DML commands are executed in MySQL and verified successfully.

EX: NO: 3 **CREATION OF TABLES WITH CONSTRAINTS**

AIM:
To execute and verify the SQL commands for adding constraints.

MySQL Constraints

- SQL constraints are used to specify rules for the data in a table.
- Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.
- Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Prevents actions that would destroy links between tables
- CHECK - Ensures that the values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column if no value is specified

PROCEDURE:

STEP 1: Start.
STEP 2: Create the table with its essential attributes.
STEP 3: Add the constraint as a column level and table level
STEP 4: check all the constraints with specified conditions.

Create table1:

```
mysql> create table emp(empno int(3),empname varchar(20),age int(3),deptno
int(3),salary float(7,2),phno int(5));

Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

mysql>desc emp;

```
mysql> +---------+-------------+------+-----+---------+-------+
    -> | Field   | Type        | Null | Key | Default | Extra |
    -> +---------+-------------+------+-----+---------+-------+
    -> | empno   | int         | NO   | PRI | NULL    |       |
    -> | empname | varchar(20) | YES  |     | NULL    |       |
    -> | age     | int         | YES  |     | NULL    |       |
    -> | deptno  | int         | YES  |     | NULL    |       |
    -> | salary  | float(7,2)  | YES  |     | NULL    |       |
    -> | phno    | int         | YES  |     | NULL    |       |
    -> +---------+-------------+------+-----+---------+-------+
    -> 6 rows in set (0.15 sec)
```

PRIMARY KEY

Q1: create the department table with the primary key as a table level constraint.

mysql> create table dept(deptno int(3)primary key,deptname varchar(10));

```
mysql> desc dept;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| deptno   | int         | NO   | PRI | NULL    |       |
| deptname | varchar(10) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
```

Q2: Alter the employee table with the primary key as a column level constraint.

mysql> alter table emp modify empno int primary key;

Query OK, 0 rows affected (0.11 sec)
Records: 0 Duplicates: 0 Warnings: 0

NOT NULL CONSTRAINT:

Q3:Add employee name as a not null constraint using alter command

mysql> alter table emp modify empname varchar(10) not null;

Query OK, 0 rows affected (0.11 sec)
Records: 0 Duplicates: 0 Warnings: 0

CHECK CONSTRAINT:

Q4: **Add Check Constraint For The Column Age**

alter table emp modify age int check(age>=18);

Query OK, 0 rows affected (0.11 sec)
Records: 0 Duplicates: 0 Warnings: 0

DEFAULT CONSTRAINT:

Q5:Set salary column as a default constraint

mysql>alter table emp modify salary float default 1000;

Query OK, 0 rows affected (0.05 sec)
Records: 0 Duplicates: 0 Warnings: 0

UNIQUE CONSTRAINT

Q6: create a unique constraint for the column phone number and check the constraint

alter table emp modify phno int unique;

Query OK, 0 rows affected (0.05 sec)
Records: 0 Duplicates: 0 Warnings: 0

REFERENTIAL CONSTRAINT:FOREIGN KEY)

Q7:

mysql> alter table emp add deptno int(3);

Query OK, 0 rows affected, 1 warning (0.05 sec)
Records: 0 Duplicates: 0 Warnings: 1

mysql> alter table emp add foreign key(deptno) references dept(deptno);

Query OK, 0 rows affected (0.24 sec)
Records: 0 Duplicates: 0 Warnings: 0

Q8) desc emp;

```
mysql> desc emp;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| empno   | int         | NO   | PRI | NULL    |       |
| empname | varchar(10) | NO   |     | NULL    |       |
| age     | int         | YES  |     | NULL    |       |
| salary  | float       | YES  |     | 1000    |       |
| phno    | int         | YES  | UNI | NULL    |       |
| deptno  | int         | YES  | MUL | NULL    |       |
+---------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

foreign key

Q10:

insert into emp values(101,'angel',20,20000,904486322,1);

Query OK, 1 row affected (0.03 sec)

insert into dept values(1,'PROJECT');

Query OK, 1 row affected (0.03 sec)

insert into dept values(2,'DESIGN');

Query OK, 1 row affected (0.03 sec)

insert into dept values(3,'HR');

Query OK, 1 row affected (0.03 sec)

insert into dept values(4,'SOFTWARE');

Query OK, 1 row affected (0.03 sec)

insert into emp values(101,'angel',20,1,20000,904486322);

ERROR 1062 (23000): Duplicate entry '101' for key 'emp.PRIMARY'

primary key

insert into emp values(101,'angana',20,1,30000,904486324);

ERROR 1062 (23000): Duplicate entry '101' for key 'emp.PRIMARY'

insert into emp values(102,’angana’,20,1,30000,904486324);

ERROR 1062 (23000): Duplicate entry '102' for key 'emp.PRIMARY'

check

insert into emp values(103,’anu’,16,2,40000,904486326);

ERROR 3819 (HY000): Check constraint 'emp_chk_1' is violated.

insert into emp values(103,’anu’,26,3,40000,904486326);

ERROR 1062 (23000): Duplicate entry '103' for key 'emp.PRIMARY'

not null

insert into emp values(104,null,21,4,40000,904486332);

ERROR 1048 (23000): Column 'empname' cannot be null

insert into emp values(104,’anju’,21,4,40000,904486332);

ERROR 1062 (23000): Duplicate entry '104' for key 'emp.PRIMARY'

unique

insert into emp values(105,’banu’,21,3,50000,904486332);

ERROR 1062 (23000): Duplicate entry '105' for key 'emp.PRIMARY'

insert into emp values(105,’banu’,21,3,50000,904486334);

ERROR 1062 (23000): Duplicate entry '105' for key 'emp.PRIMARY'

dropping constraints

alter table emp drop primary key;

```
mysql> alter table emp modify empno int primary key;
Query OK, 0 rows affected (0.28 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

alter table emp drop foreign key;

```
mysql> alter table emp modify empno int primary key;
Query OK, 0 rows affected (0.28 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Result:

Thus the MySQL statements for executing constraints are executed successfully.

EX: NO: 3 **WHERE CLAUSE CONDITIONS AND IMPLEMENT AGGREGATE FUNCTIONS**

AIM:

To execute and verify the SQL commands using where clause conditions and implement aggregate functions.

PROCEDURE:

1.Create the table employee with the corresponding fields.
2. Insert the records in the fields.
3. Implement where clause condition and aggregate functions
4. Verify the records and so for.

Create table 2:

mysql> create table emp(empid int primary key,empname varchar(20),age int,salary int,deptno int);
Query OK, 1 row affected (0.06 sec)

mysql> insert into emp values(101,'angel',20,20000,1);
Query OK, 1 row affected (0.06 sec)

mysql> insert into emp values(102,'angena',20,30000,1);
Query OK, 1 row affected (0.02 sec)

mysql> insert into emp values(103,'anu',26,40000,2);
Query OK, 1 row affected (0.02 sec)

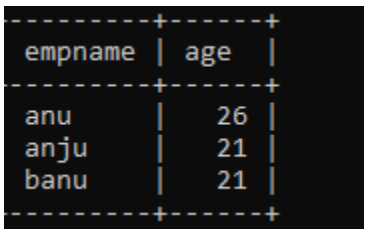
mysql> insert into emp values(104,'anju',21,40000,3);
Query OK, 1 row affected (0.01 sec)

mysql> insert into emp values(105,'banu',21,50000,4);
Query OK, 1 row affected (0.02 sec)

```
mysql> select * from emp;
+-------+---------+------+--------+--------+
| EMPID | EMPNAME | AGE  | SALARY | deptno |
+-------+---------+------+--------+--------+
|   101 | angel   |   20 |  20000 |      1 |
|   102 | angena  |   20 |  30000 |      1 |
|   103 | anu     |   26 |  40000 |      2 |
|   104 | anju    |   21 |  40000 |      3 |
|   105 | banu    |   21 |  50000 |      4 |
+-------+---------+------+--------+--------+
5 rows in set (0.00 sec)
```

Q1. Select the employee who all re above age 21.

mysql> select empname, age from emp where age>=21;

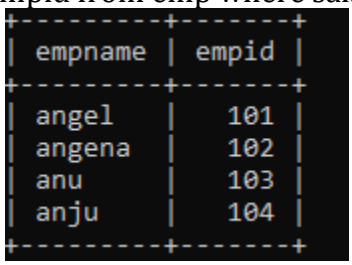


```
+---------+------+
| empname | age  |
+---------+------+
| anu     |   26 |
| anju    |   21 |
| banu    |   21 |
+---------+------+
```

3 rows in set (0.01 sec)

Q2: Display employee name those having salary between 20000 and 40000

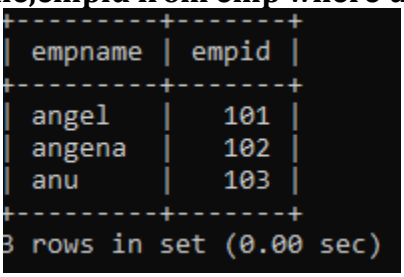
mysql> select empname,empid from emp where salary between 20000 and 40000;



```
+---------+-------+
| empname | empid |
+---------+-------+
| angel   |   101 |
| angena  |   102 |
| anu     |   103 |
| anju    |   104 |
+---------+-------+
```

Q3:Display the employee details who are in department 1 and 2

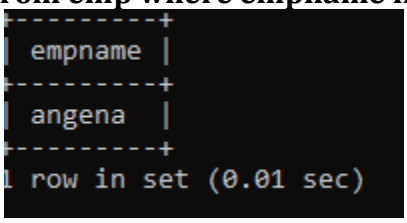
mysql> select empname,empid from emp where deptno in(1,2);



```
+---------+-------+
| empname | empid |
+---------+-------+
| angel   |   101 |
| angena  |   102 |
| anu     |   103 |
+---------+-------+
3 rows in set (0.00 sec)
```

Q4: Display the employee names that ends with letter a.

mysql> select empname from emp where empname like '%a';



```
+---------+
| empname |
+---------+
| angena  |
+---------+
1 row in set (0.01 sec)
```

Q5: Display the employee names having letter a.

mysql> select empname,empid from emp where empname like '%a%';

```
+---------+-------+
| empname | empid |
+---------+-------+
| angel   |   101 |
| angena  |   102 |
| anu     |   103 |
| anju    |   104 |
| banu    |   105 |
+---------+-------+
5 rows in set (0.00 sec)
```

Q6: Display the employee names starts with letter a.

mysql> select empname,empid from emp where empname like 'a%';

```
+---------+-------+
| empname | empid |
+---------+-------+
| angel   |   101 |
| angena  |   102 |
| anu     |   103 |
| anju    |   104 |
+---------+-------+
4 rows in set (0.00 sec)
```

Q8: display the employee details having employee id greater than 103 and department number 4.

mysql> select empid,deptno,empname from emp where empid>103 and deptno=4;

```
+-------+--------+---------+
| empid | deptno | empname |
+-------+--------+---------+
|   105 |      4 | banu    |
+-------+--------+---------+
1 row in set (0.00 sec)
```

Q10: display the employee details that are not in 102 and 104

mysql> select empname,empid from emp where empid not in(102,104);

```
+---------+-------+
| empname | empid |
+---------+-------+
| angel   |   101 |
| anu     |   103 |
| banu    |   105 |
+---------+-------+
3 rows in set (0.00 sec)
```

Aggregate Functions:

Create table 2:

create table works(empid int,companyname varchar2(20),location varchar(20),salary int(5),fk_id foreign key(empid) references emp(empid));
Query OK, 0 rows affected (0.19 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> insert into works values(101,'infosis','chennai',10000);
Query OK, 1 row affected (0.06 sec)

mysql> insert into works values(102,'wipro','bangalore',20000);
Query OK, 1 row affected (0.03 sec)

mysql> insert into works values(103,'agile','nagercoil',30000);
Query OK, 1 row affected (0.02 sec)

mysql> insert into works values(104,'accenture','chennai',40000);
Query OK, 1 row affected (0.03 sec)

mysql> select * from works;

```
+-------+-------------+-----------+--------+
| empid | companyname | location  | salary |
+-------+-------------+-----------+--------+
|   101 | infosis     | chennai   |  10000 |
|   102 | wipro       | bangalore |  20000 |
|   103 | agile       | nagercoil |  30000 |
|   104 | accenture   | chennai   |  40000 |
+-------+-------------+-----------+--------+
4 rows in set (0.00 sec)
```

Q3:Display the company name which one having more than one employee.
mysql> insert into works values(105,'wipro','bangalore',30000);
Query OK, 1 row affected (0.03 sec)

mysql> select companyname,count(empid) as no_of_emp from works group by companyname having count(empid)>1;

```
+-------------+------------+
| companyname | no_of_emp  |
+-------------+------------+
| wipro       |          2 |
+-------------+------------+
1 row in set (0.00 sec)
```

Q4:Display the employee name having maximum salary

```
mysql> select max(salary) as maximum from emp;
+----------+
| maximum |
+----------+
|   50000 |
+----------+
```

Q5: Display the employee name having maximum salary

```
mysql> select min(salary) as minimum from emp;
+---------+
| minimum |
+---------+
|  20000 |
+---------+
```

Q6: Display the average salary of employee

```
select avg(salary) as average from emp;
```

```
mysql> select avg(salary) as average from emp;
+------------+
| average    |
+------------+
| 36000.0000 |
+------------+
```

Q7: Display the employee name who is earning maximum salary

```
mysql> select empid,empname from emp where salary=(select max(salary)from
emp);
+-------+---------+
| empid | empname |
+-------+---------+
|  105 | banu    |
+-------+---------+
```

Q8: Display the sum of salary of each department

```
mysql> select deptno,sum(salary)as total from emp group by deptno;
+--------+-------+
| deptno | total |
+--------+-------+
|      1 | 50000 |
|      2 | 40000 |
|      3 | 40000 |
|      4 | 50000 |
```

Q9: Display the sum of salary of each company.

select companyname,sum(salary) as total from works group by companyname;

mysql> select companyname,sum(salary) as total from works group by companyname;

```
+-------------+-------+
| companyname | total |
+-------------+-------+
| infosis     | 10000 |
| wipro       | 50000 |
| agile       | 30000 |
| accenture   | 40000 |
```

Q10: Display company name which one is having less sum of salary compared to others

mysql> select companyname from works group by companyname having sum(salary)<=all(select sum(salary)from works group by companyname);

```
+-------------+
| companyname |
+-------------+
| infosis     |
+-------------+
```

mysql> insert into works values(105,'accenture','chennai',null);
Query OK, 1 row affected (0.05 sec)

Q11: Insert the null record

mysql> select * from works where salary is null;

```
mysql> select * from works where salary is null;
+-------+-------------+----------+--------+
| empid | companyname | location | salary |
+-------+-------------+----------+--------+
|   105 | accenture   | chennai  |   NULL |
+-------+-------------+----------+--------+
1 row in set (0.00 sec)
```

Q12: Display the not null record from the table.

select * from works where salary is not null;

```
mysql> select * from works where salary is not null;
+-------+-------------+-----------+--------+
| empid | companyname | location  | salary |
+-------+-------------+-----------+--------+
|   101 | infosis     | chennai   |  10000 |
|   102 | wipro       | bangalore |  20000 |
|   103 | agile       | nagercoil |  30000 |
|   104 | accenture   | chennai   |  40000 |
|   105 | wipro       | bangalore |  30000 |
+-------+-------------+-----------+--------+
5 rows in set (0.00 sec)
```

order by clause:

Q13: Display the employee in decending order

mysql> select * from emp order by empname desc;

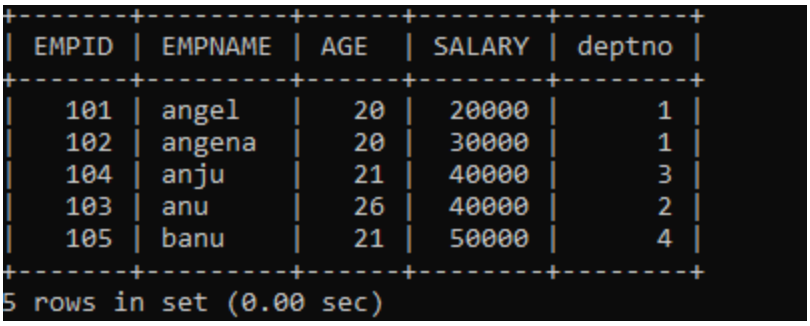
```
mysql> select * from emp order by empname desc;
+-------+---------+------+--------+--------+
| EMPID | EMPNAME | AGE  | SALARY | deptno |
+-------+---------+------+--------+--------+
|   105 | banu    |   21 |  50000 |      4 |
|   103 | anu     |   26 |  40000 |      2 |
|   104 | anju    |   21 |  40000 |      3 |
|   102 | angena  |   20 |  30000 |      1 |
|   101 | angel   |   20 |  20000 |      1 |
+-------+---------+------+--------+--------+
5 rows in set (0.01 sec)
```

Q15: Display the employee in decending order

mysql> select * from emp order by empname asc;

```
mysql> select * from emp order by empname asc;
+-------+---------+------+--------+--------+
| EMPID | EMPNAME | AGE  | SALARY | deptno |
+-------+---------+------+--------+--------+
|   101 | angel   |   20 |  20000 |      1 |
|   102 | angena  |   20 |  30000 |      1 |
|   104 | anju    |   21 |  40000 |      3 |
|   103 | anu     |   26 |  40000 |      2 |
|   105 | banu    |   21 |  50000 |      4 |
+-------+---------+------+--------+--------+
5 rows in set (0.00 sec)
```

mysql> select * from emp order by empname;



EMPID	EMPNAME	AGE	SALARY	deptno
101	angel	20	20000	1
102	angena	20	30000	1
104	anju	21	40000	3
103	anu	26	40000	2
105	banu	21	50000	4

5 rows in set (0.00 sec)

Result:

Thus the where clause conditions using MySQL statements are verified and executed successfully.

EX: NO: 4 **SIMPLE JOIN OPERATIONS**

AIM:

To Query the Database Table and explore Subqueries and simple Join Operations.

PERFORM THE FOLLOWING:

1. Create table 1 and table 2.
2. Perform sub queries using where, from and select clauses.
3. Perform single row and multiple rows sub queries.
4. Perform nested sub queries.
5. Perform simple join operations.

SUB-QUERY:

A sub-query is a SQL query nested inside a larger query. A Sub Query can also be called a Nested/Inner Query.

The sub queries used with:

1. SELECT CLAUSE
2. FROM CLAUSE
3. WHERE CLAUSE

CREATE TABLE 1:

mysql> CREATE TABLE EMP(EMPID INTEGER,EMPNAME VARCHAR(20),AGE INTEGER,SALARY INTEGER);

Query OK, 0 rows affected (0.55 sec)

mysql> INSERT INTO EMP VALUES(1,'ASHI',16,10000);

Query OK, 1 row affected (0.15 sec)

mysql> INSERT INTO EMP VALUES(2,'ANI',18,20000);

Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO EMP VALUES(3,'BISMI',17,15000);

mysql> SELECT * FROM EMP;

+-------+---------+------+--------+

| EMPID | EMPNAME | AGE | SALARY |

+-------+---------+------+--------+

| 1 | ASHI | 16 | 10000 |

| 2 | ANI | 18 | 20000 |

| 3 | BISMI | 17 | 15000 |

+-------+---------+------+--------+

CREATE TABLE 2:

mysql> CREATE TABLE REPORT(EMPID INTEGER,AGE INTEGER,SALARY INTEGER);

Query OK, 0 rows affected (0.08 sec)

mysql> INSERT INTO REPORT VALUES(3,17,15000);

Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO REPORT VALUES(1,16,10000);

Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO REPORT VALUES(4,16,20000);

Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM REPORT;

+-------+------+--------+

| EMPID | AGE | SALARY |

+-------+------+--------+

| 3 | 17 | 15000 |

| 1 | 16 | 10000 |

| 4 | 16 | 20000 |

+-------+------+--------+

3 rows in set (0.00 sec)

SUB-QUERY USING Where Clause:

A sub-query in a WHERE clause can be used to qualify a column against a set of rows.

SYNTAX:

SELECT* FROM table_name1 WHERE column_name1 IN(SELECT column_name1 FROM table_name2WHERE condition);

mysql> SELECT * FROM REPORT WHERE EMPID IN(SELECT EMPID FROM EMP WHERE EMPNAME='BISMI');

+-------+------+--------+

| EMPID | AGE | SALARY |

+-------+------+--------+

| 3 | 17 | 15000 |

+-------+------+--------+

1 row in set (0.08 sec)

SUB-QUERY USING FROM CLAUSE:

FROM clause can be used to specify a sub-query expression in SQL. The relation produced by the sub-queryis then used as a new relation on which the outer query is applied.

SYNTAX:

SELECT a.column_name1, b.column_name2 FROM table_name1 a, (SELECT b.column_name2, function(variable) as b.column_name FROM table_name GROUP BYcolumn_name)b Where Condition;

```
mysql> SELECT A.EMPNAME,B.AGE FROM EMP A,(SELECT
EMPID,AVG(AGE)AS AGE FROM REPORT GROUP BY EMPID)B WHERE
A.EMPID=B.EMPID;

+---------+---------+

| EMPNAME | AGE     |

+---------+---------+

| ASHI    | 16.0000 |

| BISMI   | 17.0000 |

+---------+---------+

2 rows in set (0.03 sec)

mysql> SELECT * FROM EMP WHERE SALARY=(SELECT MIN(SALARY) FROM
EMP);

+-------+---------+------+--------+

| EMPID | EMPNAME | AGE  | SALARY |

+-------+---------+------+--------+

|     1 | ASHI    |   16 |  10000 |

+-------+---------+------+--------+

1 row in set (0.00 sec)
```

SINGLE ROW SUB QUERY USING HAVING CLAUSE:

The HAVING clause is used to filter out groups of records. Because it becomes very useful in filtering onaggregate values such as averages, summations, and count.

SYNTAX:

SELECT column_name1,function(column_name2) FROM table_name1 GROUP BY column_name1 HAVINGfunction (column_name2) > (SELECTfunction (column_name2) FROM table_name1 WHERE condition);

mysql> SELECT EMPID,MAX(SALARY) FROM EMP GROUP BY EMPID HAVING MAX(SALARY)>(SELECT MAX(SALARY) FROM EMP WHERE EMPID=1);

```
+-------+-------------+
| EMPID | MAX(SALARY) |
+-------+-------------+
|     2 |       20000 |
|     3 |       15000 |
+-------+-------------+
```

2 rows in set (0.01 sec)

JOINS:

JOIN USING USING CLAUSE:

The USING clause specifies which columns to test for equality when two tables are joined.

SYNTAX:

SELECT*FROM table_name1 JOIN table_name2 USING (common_column name1);

mysql> SELECT * FROM EMP JOIN REPORT USING(EMPID);

```
+-------+---------+------+--------+------+--------+
| EMPID | EMPNAME | AGE  | SALARY | AGE  | SALARY |
+-------+---------+------+--------+------+--------+
|     3 | BISMI   |   17 |  15000 |   17 |  15000 |
|     1 | ASHI    |   16 |  10000 |   16 |  10000 |
+-------+---------+------+--------+------+--------+
```

JOIN USING ON CLAUSE:

ON clause can be used to join columns that have different names. Use the ON clause to specify conditions orspecify columns to join.

SYNTAX:

SELECT*FROM table_name1 JOIN table_name2 ON (condition_using_common_column_name);

mysql> SELECT A.EMPNAME,A.EMPID,B.AGE,B.SALARY FROM EMP A JOIN REPORT B ON(A.EMPID=B.EMPID);

```
+---------+-------+------+--------+
| EMPNAME | EMPID | AGE  | SALARY |
+---------+-------+------+--------+
| BISMI   |     3 |   17 |  15000 |
| ASHI    |     1 |   16 |  10000 |
+---------+-------+------+--------+
```

2 rows in set (0.00 sec)

Result:

Thus the where clause conditions using MySQL statements are verified and executed successfully.

EX: NO: 5 **NATURAL, EQUI AND OUTER JOIN OPERATIONS**

AIM:

To Query the Database Table and explore natural, equi and outer join operations.

PERFORM THE FOLLOWING:

1. Create table 1 and table 2.
2. Perform sub queries using natural and outer join operations
3. Analyze the difference of each queries.
4. Report the answers.

CREATE TABLE 1:

```
mysql> create table employee(ename varchar(20),ecity varchar(20),eno int(10));

Query OK, 0 rows affected, 1 warning (0.10 sec)
```

CREATE TABLE 2:

```
mysql> create table salary(eno int(10),dname varchar(20),esal int(10));

Query OK, 0 rows affected, 2 warnings (0.10 sec)

mysql> desc employee;

+-------+-------------+------+-----+---------+-------+

| Field | Type        | Null | Key | Default | Extra |

+-------+-------------+------+-----+---------+-------+

| ename | varchar(20) | YES  |     | NULL    |       |

| ecity | varchar(20) | YES  |     | NULL    |       |

| eno   | int         | YES  |     | NULL    |       |

+-------+-------------+------+-----+---------+-------+

mysql> desc salary;

+-------+-------------+------+-----+---------+-------+

| Field | Type        | Null | Key | Default | Extra |
```

```
+-------+-------------+------+-----+---------+-------+
| eno   | int         | YES  |     | NULL    |       |
| dname | varchar(20) | YES  |     | NULL    |       |
| esal  | int         | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
mysql> insert into employee values('Ajay','Chennai',11);
Query OK, 1 row affected (0.06 sec)
mysql> insert into employee values('Vijay','Banglore',12);
Query OK, 1 row affected (0.04 sec)
mysql> insert into employee values('Sujay','Chennai',13);
Query OK, 1 row affected (0.03 sec)
mysql> insert into employee values('Jay','Madurai',14);
Query OK, 1 row affected (0.04 sec)
mysql> select* from employee;
+-------+----------+------+
| ename | ecity    | eno  |
+-------+----------+------+
| Ajay  | Chennai  |   11 |
| Vijay | Banglore |   12 |
| Sujay | Chennai  |   13 |
| Jay   | Madurai  |   14 |
mysql> insert into salary values(11,'IT',20000);
Quey OK, 1 row affected (0.03 sec)
mysql> insert into salary values(12,'CSE',20020);
Query OK, 1 row affected (0.03 sec)
```

```
mysql> insert into salary values(13,'IT',20050);
Query OK, 1 row affected (0.03 sec)
mysql> insert into salary values(14,'CSE',20000);
Query OK, 1 row affected (0.03 sec)
mysql> select* from salary;
+------+-------+-------+
| eno  | dname | esal  |
+------+-------+-------+
|   11 | IT    | 20000 |
|   12 | CSE   | 20020 |
|   13 | IT    | 20050 |
|   14 | CSE   | 20000 |
+------+-------+-------+
```

NATURAL JOIN OPERATIONS

A natural join is a type of join operation that creates an implicit join by combining tables based on columns with the same name and data type

- There is no need to specify the column names to join.
- The resultant table always contains unique columns.
- It is possible to perform a natural join on more than two tables.
- Do not use the ON clause

Syntax:

SELECT [column_names*] FROM table_name1 NATURAL JOIN table_name2;

```
mysql> select * from employee natural join salary;
+------+-------+----------+-------+-------+
| eno  | ename | ecity    | dname | esal  |
```

+------+-------+----------+-------+-------+

| 11 | Ajay | Chennai | IT | 20000 |

| 12 | Vijay | Banglore | CSE | 20020 |

| 13 | Sujay | Chennai | IT | 20050 |

| 14 | Jay | Madurai | CSE | 20000 |

+------+-------+----------+-------+-------+

4 rows in set (0.00 sec)

CROSS JOIN OPERATION:

MySQL CROSS JOIN is used to combine all possibilities of the two or more tables and returns the result that contains every row from all contributing tables. The CROSS JOIN is also known as CARTESIAN JOIN, which provides the Cartesian product of all associated tables.

Syntax:

SELECT column-lists FROM table1 CROSS JOIN table2;

mysql> select * from employee join salary;

+-------+----------+------+------+-------+-------+

| ename | ecity | eno | eno | dname | esal |

+-------+----------+------+------+-------+-------+

| Jay | Madurai | 14 | 11 | IT | 20000 |

| Sujay | Chennai | 13 | 11 | IT | 20000 |

| Vijay | Banglore | 12 | 11 | IT | 20000 |

| Ajay | Chennai | 11 | 11 | IT | 20000 |

| Jay | Madurai | 14 | 12 | CSE | 20020 |

| Sujay | Chennai | 13 | 12 | CSE | 20020 |

| Vijay | Banglore | 12 | 12 | CSE | 20020 |

| Ajay | Chennai | 11 | 12 | CSE | 20020 |

| Jay | Madurai | 14 | 13 | IT | 20050 |

| Sujay | Chennai | 13 | 13 | IT | 20050 |

| Vijay | Banglore | 12 | 13 | IT | 20050 |

| Ajay | Chennai | 11 | 13 | IT | 20050 |

| Jay | Madurai | 14 | 14 | CSE | 20000 |

| Sujay | Chennai | 13 | 14 | CSE | 20000 |

| Vijay | Banglore | 12 | 14 | CSE | 20000 |

| Ajay | Chennai | 11 | 14 | CSE | 20000 |

+-------+----------+------+------+-------+-------+

INNER JOIN OPERATION:

The MySQL Inner Join is used to returns only those results from the tables that match the specified condition and hides other rows and columns. MySQL assumes it as a default Join, so it is optional to use the Inner Join keyword with the query.

Syntax:

SELECT columns FROM table1 INNER JOIN table2 ON condition1
INNER JOIN table3 ON condition2;

mysql> select * from employee inner join salary on employee.eno=salary.eno;;

+-------+----------+------+------+-------+-------+

| ename | ecity | eno | eno | dname | esal |

+-------+----------+------+------+-------+-------+

| Ajay | Chennai | 11 | 11 | IT | 20000 |

| Vijay | Banglore | 12 | 12 | CSE | 20020 |

| Sujay | Chennai | 13 | 13 | IT | 20050 |

| Jay | Madurai | 14 | 14 | CSE | 20000 |

LEFT OUTER JOIN OPERATION:

The LEFT JOIN returns all the rows from the table on the left even if no matching rows have been found in the table on the right. Where no matches have been found in the table on the right, NULL is returned.

Syntax:

SELECT column-lists FROM table1 Left outer join table2;

mysql> select * from employee left outer join salary on employee.eno=salary.eno;

```
+-------+----------+------+------+-------+-------+
| ename | ecity    | eno  | eno  | dname | esal  |
+-------+----------+------+------+-------+-------+
| Ajay  | Chennai  |   11 |   11 | IT    | 20000 |
| Vijay | Banglore |   12 |   12 | CSE   | 20020 |
| Sujay | Chennai  |   13 |   13 | IT    | 20050 |
| Jay   | Madurai  |   14 |   14 | CSE   | 20000 |
```

RIGHT OUTER JOIN OPERATION:

RIGHT JOIN is obviously the opposite of LEFT JOIN. The RIGHT JOIN returns all the columns from the table on the right even if no matching rows have been found in the table on the left. Where no matches have been found in the table on the left, NULL is returned.

Syntax:

SELECT column-lists FROM table1 Right outer join table2;

```
mysql> select * from employee right outer join salary on employee.eno=salary.eno;
+-------+----------+------+------+-------+-------+
| ename | ecity    | eno  | eno  | dname | esal  |
+-------+----------+------+------+-------+-------+
| Ajay  | Chennai  |   11 |   11 | IT    | 20000 |
| Vijay | Banglore |   12 |   12 | CSE   | 20020 |
| Sujay | Chennai  |   13 |   13 | IT    | 20050 |
| Jay   | Madurai  |   14 |   14 | CSE   | 20000 |
+-------+----------+------+------+-------+-------+
4 rows in set (0.00 sec)
```

Result:

Thus the MYSQL commands to execute the natural, equi and outer join operations are executed and verified successfully.

EX: NO: 6 **PROCEDURES AND USER DEFINED FUNCTIONS**

AIM:

To create procedures and user defined functions using PL/SQL block.

PERFORM THE FOLLOWING:

1. Create tables.
2. Create procedures and functions.
3. Call procedures and functions to perform listed operations
4. Report the answers.

PROCEDURE:

A procedure (often called a stored procedure) is a **collection of pre-compiled SQL statements** stored inside the database. It is a subroutine or a subprogram in the regular computing language. **A procedure always contains a name, parameter lists, and SQL statements**.

Syntax:

```
CREATE PROCEDURE ProcedureName
BEGIN
    SQL Statements
END
```

1. Write a pl/sql program to find the sum &avg marks of all the student using procedures.

STUD				
RollNo	Name	M1	M2	M3

Create Table1:

mysql> create table stud(rollno int,name varchar(),m1 int,m2 int,m3 int);

INSERT VALUES INTO THE TABLE:

insert into stud values(1,'abi',50,60,70);

insert into stud values(2,'bob',70,60,50);

insert into stud values(3,'tom',80,60,70);

Create Procedure:

```
mysql> delimiter &&
mysql> create procedure mycalc()
   -> begin
   -> select name,m1+m2+m3 as total,(m1+m2+m3)/3 as aver from stud;
   -> end
   -> &&
mysql> delimiter ;
        ->&&
```

Call Procedure:

```
mysql> call mycalc();
        ->&&
```

Out PUT:

NAME	Total	Average
Abi	180	60
Bob	180	60
tom	210	70

2. Write a pl/sql program to find the product of 3 numbers in a procedure using in & out parameter.

   Create Procedure:

```
mysql> delimiter &&
mysql> create procedure myproduct(n1 int,n2 int,n3 int,out result int)
   -> begin
   -> set result=n1*n2*n3;
   -> end
```

-> &&

mysql> delimiter ;

->&&

mysql> call myproduct(3,4,3,@ans);

->&&

mysql> select @ans;

->&&

Output:

@ans
36

USER DEFINED FUNCTIONS

The function which is defined by the user is called a user-defined function. MySQL user-defined functions may or may not have parameters that are optional, but it always returns a single value that is mandatory. The returned value which is return by the MySQL Function can be of any valid MySQL data type.

```
DELIMITER $$

CREATE FUNCTION Function_Name(
    Parameter_1 DataType,
    Parameter_2 DataType,
    Parameter_n DataType,
)
RETURNS Return_Datatype
[NOT] DETERMINISTIC
BEGIN
        Function Body
        Return Return_Value
END $$

DELIMITER ;
```

1. Write a pl/sql program to calculate age in user defined function

Create Table:

mysql> create table employee(empid int primary key,name varchar(50),salary int,dob date);

Query OK, 0 rows affected (1.61 sec)

Inserting record:

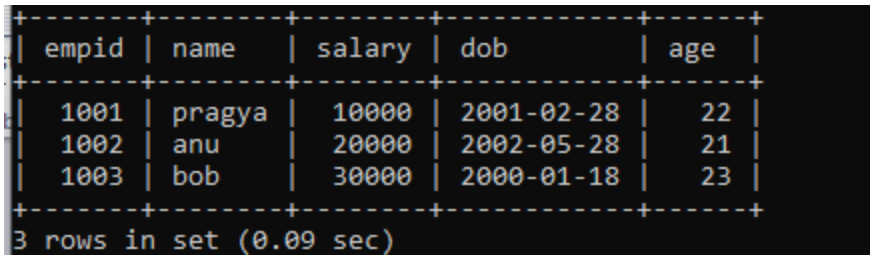
```
mysql> insert into employee values(1001,'pragya',10000,'2001-02-28');
Query OK, 1 row affected (0.12 sec)
mysql> insert into employee values(1002,'anu',20000,'2002-05-28');
Query OK, 1 row affected (0.08 sec)
mysql> insert into employee values(1003,'bob',30000,'2000-01-18');
Query OK, 1 row affected (0.03 sec)
```

Create Function:

```
mysql> DELIMITER &&
mysql> CREATE FUNCTION Func_Calculate_Age(Age date)
    -> RETURNS INT DETERMINISTIC
    -> BEGIN
    -> DECLARE TodayDate DATE;
    -> SELECT CURRENT_DATE() INTO TodayDate;
    -> RETURN YEAR(TodayDate)-YEAR(Age);
    -> END
    -> &&
Query OK, 0 rows affected (0.03 sec)
```

Calling Function:

```
mysql> select empid,name,salary,dob,func_calculate_age(dob)as age from employee;
    -> &&
```



empid	name	salary	dob	age
1001	pragya	10000	2001-02-28	22
1002	anu	20000	2002-05-28	21
1003	bob	30000	2000-01-18	23

3 rows in set (0.09 sec)

Result:

Thus the procedures user defined function were created and executed successfully.

EX: NO: 7 **DCL AND TCL COMMANDS**

AIM:

To create and execute complex transactions and realize DCL and TCL commands.

PERFORM THE FOLLOWING:

1. Create table.
2. Perform DCL and TCL commands.
3. Execute different user privileges.
4. Report the answers.

DCL (Data Control Language):

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

- **GRANT:** This command gives users access privileges to the database.

  GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_ USER;

- **REVOKE:** This command withdraws the user's access privileges given by using the GRANT command.

  REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

TCL (Transaction Control Language):

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: success or failure.

COMMIT: Commits a Transaction.

COMMIT;

ROLLBACK: Rollbacks a transaction in case of any error occurs.

ROLLBACK to SAVEPOINT_NAME;

SAVEPOINT: Sets a save point within a transaction.

SAVEPOINT SAVEPOINT_NAME;

TCL Exercises:

```
mysql> select * from employee;
+-------+--------+--------+------------+
| empid | name   | salary | dob        |
+-------+--------+--------+------------+
|  1001 | pragya |  10000 | 2001-02-28 |
|  1002 | anu    |  20000 | 2002-05-28 |
|  1003 | bob    |  30000 | 2000-01-18 |
+-------+--------+--------+------------+
3 rows in set (0.17 sec)
```

START TRANSACTION:

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
```

CREATE SAVEPOINT:

```
mysql> savepoint s1;
Query OK, 0 rows affected (0.01 sec)

mysql> insert into employee values(1004,'vini',40000,'1999-06-13');
Query OK, 1 row affected (0.10 sec)
mysql> select * from employee;
+-------+--------+--------+------------+
| empid | name   | salary | dob        |
+-------+--------+--------+------------+
|  1001 | pragya |  10000 | 2001-02-28 |
|  1002 | anu    |  20000 | 2002-05-28 |
|  1003 | bob    |  30000 | 2000-01-18 |
|  1004 | vini   |  40000 | 1999-06-13 |
+-------+--------+--------+------------+
```

4 rows in set (0.17 sec)

CREATE updated SAVEPOINT:

mysql> savepoint upd;
Query OK, 0 rows affected (0.00 sec)

UPDATE TABLE

mysql> update employee set name='anupriya' where empid=1002;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> select * from employee;

```
+-------+----------+--------+------------+
| empid | name     | salary | dob        |
+-------+----------+--------+------------+
|  1001 | pragya   |  10000 | 2001-02-28 |
|  1002 | anupriya |  20000 | 2002-05-28 |
|  1003 | bob      |  30000 | 2000-01-18 |
|  1004 | vini     |  40000 | 1999-06-13 |
+-------+----------+--------+------------+
```

4 rows in set (0.00 sec)

CREATE delete SAVEPOINT:

mysql> savepoint del;
Query OK, 0 rows affected (0.00 sec)

DELETE ROWS FROM TABLE:

mysql> delete from employee where empid=1003;

Query OK, 1 row affected (0.01 sec)

```
mysql> select * from employee;
+-------+----------+--------+------------+
| empid | name     | salary | dob        |
+-------+----------+--------+------------+
|  1001 | pragya   |  10000 | 2001-02-28 |
|  1002 | anupriya |  20000 | 2002-05-28 |
|  1004 | vini     |  40000 | 1999-06-13 |
+-------+----------+--------+------------+
3 rows in set (0.00 sec)
```

ROLLBACK TO DELETED SAVEPOINT:

```
mysql> rollback to del;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from employee;
+-------+----------+--------+------------+
| empid | name     | salary | dob        |
+-------+----------+--------+------------+
|  1001 | pragya   |  10000 | 2001-02-28 |
|  1002 | anupriya |  20000 | 2002-05-28 |
|  1003 | bob      |  30000 | 2000-01-18 |
|  1004 | vini     |  40000 | 1999-06-13 |
+-------+----------+--------+------------+
4 rows in set (0.00 sec)
```

ROLLBACK TO UPDATE SAVEPOINT:

```
mysql> rollback to upd;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select * from employee;
+-------+--------+--------+------------+
| empid | name   | salary | dob        |
+-------+--------+--------+------------+
|  1001 | pragya |  10000 | 2001-02-28 |
|  1002 | anu    |  20000 | 2002-05-28 |
|  1003 | bob    |  30000 | 2000-01-18 |
|  1004 | vini   |  40000 | 1999-06-13 |
+-------+--------+--------+------------+
```

ROLLBACK TO S1 SAVEPOINT:

```
mysql> rollback to s1;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from employee;
+-------+--------+--------+------------+
| empid | name   | salary | dob        |
+-------+--------+--------+------------+
|  1001 | pragya |  10000 | 2001-02-28 |
|  1002 | anu    |  20000 | 2002-05-28 |
|  1003 | bob    |  30000 | 2000-01-18 |
+-------+--------+--------+------------+
3 rows in set (0.17 sec)
```

COMMIT TRANSACTION:

```
mysql> commit;
Query OK, 0 rows affected (0.16 sec)
```

DCL (Data Control Language):

CREATE DATABASE:

mysql>create database employ;
Query ok..

CURRENT USER:

mysql> **select user();**

```
+----------------+
| user()         |
+----------------+
| root@localhost |
+----------------+
```

1 row in set (0.28 sec)

mysql> **create user 'ajay'@'localhost' identified by 'ajay';**
Query OK, 0 rows affected (1.85 sec)

TO DISPLAY ALL USERS

mysql> **select user from mysql.user;**

```
+------------------+
| user             |
+------------------+
| ajay             |
| mysql.infoschema |
| mysql.session    |
| mysql.sys        |
| root             |
+------------------+
```

5 rows in set (0.03 sec)

mysql> **grant insert on EMPLOY.emp to 'ajay'@'localhost';**
Query OK, 0 rows affected (0.07 sec)

mysql> **grant update(eid) on EMPLOY.emp to 'ajay'@'localhost';**
Query OK, 0 rows affected (0.06 sec)

mysql> **show grants for 'ajay'@'localhost';**

```
+-------------------------------------------------------------+
| Grants for ajay@localhost                                   |
+-------------------------------------------------------------+
| GRANT SELECT ON *.* TO `ajay`@`localhost`                   |
| GRANT INSERT, UPDATE (`eid`) ON `rr`.`emp` TO `ajay`@`localhost` |
+-------------------------------------------------------------+
```

2 rows in set (0.00 sec)

LOGIN TO AJAY

mysql> **system mysql -u ajay –p**
Enter password: **** (Type:ajay)
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 8.0.32 MySQL Community Server - GPL

```
mysql> select user();
+----------------+
| user()         |
+----------------+
| ajay@localhost |
+----------------+
1 row in set (0.00 sec)

mysql> use employ;
Database changed
mysql> select * from emp;
+-----+--------+-----------+
| eid | efname | job       |
+-----+--------+-----------+
| 101 | hari   | SeniorEng |
| 102 | Rahul  | Editor    |
+-----+--------+-----------+
2 rows in set (0.02 sec)

mysql> insert into emp values(103,'Gina','Researcher');
Query OK, 1 row affected (0.05 sec)

mysql> update emp set eid=104 where efname='Rahul';
Query OK, 1 row affected (0.08 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update emp set efname='lisha' where  eid=101;        //Not Granted
ERROR 1143 (42000): UPDATE command denied to user 'ajay'@'localhost' for column
'efname' in table 'emp'

mysql> system mysql -u root –p                              //Login to Root
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 8.0.32 MySQL Community Server - GPL
mysql> select user();
+----------------+
| user()         |
```

```
+----------------+
| root@localhost |
+----------------+
1 row in set (0.00 sec)
```

Revoke Permission from Ajay user

```
mysql> revoke update(eid) on employ.emp from 'ajay'@'localhost';
Query OK, 0 rows affected (0.08 sec)

mysql> revoke insert on rr.emp from 'ajay'@'localhost';
Query OK, 0 rows affected (0.04 sec)

mysql> show grants for 'ajay'@'localhost';
+-----------------------------------------+
| Grants for ajay@localhost               |
+-----------------------------------------+
| GRANT SELECT ON *.* TO `ajay`@`localhost` |
+-----------------------------------------+
1 row in set (0.00 sec)
mysql> system mysql -u ajay –p                    //Login to Ajay User
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 8.0.32 MySQL Community Server - GPL

mysql> select user();
+----------------+
| user()         |
+----------------+
| ajay@localhost |
+----------------+
1 row in set (0.00 sec)

mysql> use rr;
Database changed
```

INSERT PERMISSION REVOKED

```
mysql> insert into emp values(108,'naresh','juneng');
ERROR 1142 (42000): INSERT command denied to user 'ajay'@'localhost' for table
'emp'
mysql> system mysql -u root –p
```

//Login to Root User

```
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 16
```

Server version: 8.0.32 MySQL Community Server - GPL
mysql> select user();
+----------------+
| user() |
+----------------+
| root@localhost |
+----------------+
1 row in set (0.00 sec)

//To Grant all on Ajay user
mysql>mysql> grant all privileges on *.* to 'ajay'@'localhost' with grant option;
Query OK, 0 rows affected (0.23 sec)

//To Change pwd for ajay user

mysql> alter user 'ajay'@'localhost' identified by 'abcd';
Query OK, 0 rows affected (0.05 sec)

mysql> system mysql -u ajay -p
Enter password: **** (Type abcd)

//To Lock ajay user
mysql> alter user 'ajay'@'localhost' account lock;
Query OK, 0 rows affected (0.04 sec)

mysql> system mysql -u ajay -p;
Enter password: ****
ERROR 3118 (HY000): Access denied for user 'ajay'@'localhost'. Account is locked.
//To Unlock ajay user

mysql> alter user 'ajay'@'localhost' account unlock;
Query OK, 0 rows affected (0.03 sec)

mysql> system mysql -u ajay -p
Enter password: ****

mysql> system mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.

mysql> select user();
+----------------+
| user() |
+----------------+
| root@localhost |
+----------------+

1 row in set (0.00 sec)

```
mysql> select user, host from mysql.user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| ajay             | localhost |
| mysql.infoschema | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
| root             | localhost |
+------------------+-----------+
5 rows in set (0.01 sec)
```

//To Drop user

```
mysql> drop user 'ajay'@'localhost';
Query OK, 0 rows affected (0.19 sec)

mysql> select user, host from mysql.user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| mysql.infoschema | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
| root             | localhost |
+------------------+-----------+
4 rows in set (0.00 sec)
```

–

Result:

Thus the TCL and DCL commands were executed and verified successfully.

EX: NO: 8 **TRIGGERS**

AIM:

To develop and execute a Trigger for Before and After update, Delete, Insert operations on a table

Trigger:

A Trigger is a stored procedure that defines an action that the database automatically takes when some database-related event such as Insert, Update or Delete occur.

Types Of Triggers:

The various types of triggers are as follows

Before: Before triggers are fired before the DML statement is actually executed.
After: After triggers are fired after the DML statement has finished execution.
For each row: It specifies that the trigger fires once per row.
For each statement: This is the default trigger that is invoked. It specifies that the trigger fires once per statement.

Variables used in triggers

- :new
- :old

These two variables retain the new and old values of the column updated in the database. The values in these variables can be used in the database triggers for data manipulation.

Snytax:

CREATE TRIGGER <trigger name> <trigger time > <trigger event> ON <table name> FOR EACH ROW <trigger body>;

Using MySQL Triggers

Every trigger associated with a table has a unique name and function based on two factors:

1. **Time**. **BEFORE** or **AFTER** a specific row event.

2. **Event**. **INSERT**, **UPDATE** or **DELETE**.

Delete Triggers

To delete a trigger, use the **DROP TRIGGER** statement:

DROP TRIGGER<trigger name>;

Alternatively, use:

DROP TRIGGER IF EXISTS<trigger name>;

Procedure:

1. Create a table called *person* with *name* and *age* for columns.

```
CREATE TABLE person (name varchar(45), age int);
```

Insert sample data into the table:

```
INSERT INTO person VALUES ('Matthew', 25), ('Mark', 20);
```

Select the table to see the result:

```
SELECT * FROM person;
```

```
mysql> CREATE TABLE person (name varchar(45), age int);
Query OK, 0 rows affected (0.14 sec)

mysql> INSERT INTO person VALUES ('Matthew', 25), ('Mark', 20);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM person;
+---------+------+
| name    | age  |
+---------+------+
| Matthew |   25 |
| Mark    |   20 |
+---------+------+
2 rows in set (0.00 sec)
```

2. Create a table called *average_age* with a column called *average*:

```
CREATE TABLE average_age (average double);
```

Insert the average age value into the table:

```
INSERT INTO average_age SELECT AVG(age) FROM person;
```

Select the table to see the result:

```
SELECT * FROM average_age;
```

```
mysql> CREATE TABLE average_age (average double);
Query OK, 0 rows affected (0.20 sec)

mysql> INSERT INTO average_age SELECT AVG(age) FROM person;
Query OK, 1 row affected (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM average_age;
+---------+
| average |
+---------+
|    22.5 |
+---------+
1 row in set (0.00 sec)
```

3. Create a table called *person_archive* with *name*, *age*, and *time* columns:

```
CREATE TABLE person_archive (
name varchar(45),
age int,
time timestamp DEFAULT NOW());
```

```
mysql> CREATE TABLE person_archive (
    -> name varchar(45),
    -> age int,
    -> time timestamp DEFAULT NOW());
Query OK, 0 rows affected (0.28 sec)
```

Note: The function **NOW()** records the current time.

Create a BEFORE INSERT Trigger

To create a **BEFORE INSERT** trigger, use:

```
CREATE TRIGGER <trigger name> BEFORE INSERT
ON <table name>
FOR EACH ROW
<trigger body>;
```

The **BEFORE INSERT** trigger gives control over data modification before committing into a database table..

BEFORE INSERT Trigger Example

Create a **BEFORE INSERT** trigger to check the age value before inserting data into the *person* table:

```
delimiter //
CREATE TRIGGER person_bi BEFORE INSERT
ON person
FOR EACH ROW
IF NEW.age < 18 THEN
SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'Person must be older than 18.';
END IF; //
delimiter ;
```

```
mysql> delimiter //
mysql> CREATE TRIGGER person_bi BEFORE INSERT
    -> ON person
    -> FOR EACH ROW
    -> IF NEW.age < 18 THEN
    -> SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'Person must be older than 18.';
    -> END IF; //
Query OK, 0 rows affected (0.17 sec)

mysql> delimiter ;
```

Inserting data activates the trigger and checks the value of *age* before committing the information:

```
INSERT INTO person VALUES ('John', 14);
```

```
mysql> INSERT INTO person VALUES ('John', 14);
ERROR 1644 (50001): Person must be older than 18.
```

The console displays the descriptive error message. The data does not insert into the table because of the failed trigger check.

Create an AFTER INSERT Trigger

Create an **AFTER INSERT** trigger with:

```
CREATE TRIGGER <trigger name> AFTER INSERT
ON <table name>
FOR EACH ROW
<trigger body>;
```

The **AFTER INSERT** trigger is useful when the entered row generates a value needed to update another table.

AFTER INSERT Trigger Example

Inserting a new row into the *person* table does not automatically update the average in the *average_age* table. Create an **AFTER INSERT** trigger on the *person* table to update the *average_age* table after insert:

```
delimiter //
CREATE TRIGGER person_ai AFTER INSERT
ON person
FOR EACH ROW
UPDATE average_age SET average = (SELECT AVG(age) FROM person); //
delimiter ;
```

```
mysql> delimiter //
mysql> CREATE TRIGGER person_ai AFTER INSERT
    -> ON person
    -> FOR EACH ROW
    -> UPDATE average_age SET average = (SELECT AVG(age) FROM person); //
Query OK, 0 rows affected (0.30 sec)

mysql> delimiter ;
```

Inserting a new row into the *person* table activates the trigger:

INSERT INTO person VALUES ('John', 19);

```
mysql> INSERT INTO person VALUES ('John', 19);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM person;
+---------+------+
| name    | age  |
+---------+------+
| Matthew |   25 |
| Mark    |   20 |
| John    |   19 |
+---------+------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM average_age;
+--------------+
| average      |
+--------------+
| 21.333333333 |
+--------------+
1 row in set (0.00 sec)
```

The data successfully commits to the *person* table and updates the *average_age* table with the correct average value.

Create a BEFORE UPDATE Trigger

Make a **BEFORE UPDATE** trigger with:

CREATE TRIGGER <trigger name> BEFORE UPDATE
ON <table name>
FOR EACH ROW

```
<trigger body>;
```

The **BEFORE UPDATE** triggers go together with the **BEFORE INSERT** triggers. If any restrictions exist before inserting data, the limits should be there before updating as well.

BEFORE UPDATE Trigger Example

If there is an age restriction for the *person* table before inserting data, the age restriction should also exist before updating information. Without the **BEFORE UPDATE** trigger, the age check trigger is easy to avoid. Nothing restricts editing to a faulty value.

Add a **BEFORE UPDATE** trigger to the *person* table with the same body as the **BEFORE INSERT** trigger:

```
delimiter //
CREATE TRIGGER person_bu BEFORE UPDATE
ON person
FOR EACH ROW
IF NEW.age < 18 THEN
SIGNAL SQLSTATE '50002' SET MESSAGE_TEXT = 'Person must be older than 18.';
END IF; //
delimiter ;
```

```
mysql> delimiter ;
mysql> delimiter //
mysql> CREATE TRIGGER person_bu BEFORE UPDATE
    -> ON person
    -> FOR EACH ROW
    -> IF NEW.age < 18 THEN
    -> SIGNAL SQLSTATE '50002' SET MESSAGE_TEXT = 'Person must be older than 18';
    -> END IF; //
Query OK, 0 rows affected (0.36 sec)

mysql> delimiter ;
```

Updating an existing value activates the trigger check:

```
UPDATE person SET age = 17 WHERE name = 'John';
```

```
mysql> UPDATE person SET age = 17 WHERE name = 'John';
ERROR 1644 (50001): Person must be over the age of 18.
```

Updating the *age* to a value less than 18 displays the error message, and the information does not update.

Create an AFTER UPDATE Trigger

Use the following code block to create an **AFTER UPDATE** trigger:

```
CREATE TRIGGER <trigger name> AFTER UPDATE
ON <table name>
FOR EACH ROW
<trigger body>;
```

The **AFTER UPDATE** trigger helps keep track of committed changes to data. Most often, any changes after inserting information also happen after updating data.

AFTER UPDATE Trigger Example

Any successful updates to the *age* data in the table *person* should also update the intermediate average value calculated in the *average_age* table.

Create an **AFTER UPDATE** trigger to update the *average_age* table after updating a row in the *person* table:

```
delimiter //
CREATE TRIGGER person_au AFTER UPDATE
ON person
FOR EACH ROW
UPDATE average_age SET average = (SELECT AVG(age) FROM person); //
delimiter ;
```

```
mysql> delimiter //
mysql> CREATE TRIGGER person_au AFTER UPDATE
    -> ON person
    -> FOR EACH ROW
    -> UPDATE average_age SET average = (SELECT AVG(age) FROM person); //
Query OK, 0 rows affected (0.93 sec)

mysql> delimiter ;
```

Updating existing data changes the value in the *person* table:

```
UPDATE person SET age = 21 WHERE name = 'John';
```

```
mysql> UPDATE person SET age = 21 WHERE name = 'John';
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM person;
+---------+------+
| name    | age  |
+---------+------+
| Matthew |   25 |
| Mark    |   20 |
| John    |   21 |
+---------+------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM average_age;
+---------+
| average |
+---------+
|      22 |
+---------+
1 row in set (0.00 sec)
```

Updating the table *person* also updates the average in the *average_age* table.

Create a BEFORE DELETE Trigger

To create a **BEFORE DELETE** trigger, use:

```
CREATE TRIGGER <trigger name> BEFORE DELETE
ON <table name>
FOR EACH ROW
<trigger body>;
```

The **BEFORE DELETE** trigger is essential for security reasons. If a parent table has any children attached, the trigger helps block deletion and prevents orphaned tables. The trigger also allows archiving data before deletion.

BEFORE DELETE Trigger Example

Archive deleted data by creating a **BEFORE DELETE** trigger on the table *person* and insert the values into the *person_archive* table:

```
delimiter //
CREATE TRIGGER person_bd BEFORE DELETE
ON person
FOR EACH ROW
INSERT INTO person_archive (name, age)
VALUES (OLD.name, OLD.age); //
delimiter ;
```

```
mysql> delimiter //
mysql> CREATE TRIGGER person_bd BEFORE DELETE
    -> ON person
    -> FOR EACH ROW
    -> INSERT INTO person_archive (name, age)
    -> VALUES (OLD.name, OLD.age); //
Query OK, 0 rows affected (0.33 sec)

mysql> delimiter ;
```

Deleting data from the table *person* archives the data into the *person_archive* table before deleting:

```
DELETE FROM person WHERE name = 'John';
```

```
mysql> DELETE FROM person WHERE name = 'John';
Query OK, 1 row affected (0.03 sec)

mysql> SELECT * FROM person;
+---------+------+
| name    | age  |
+---------+------+
| Matthew |   25 |
| Mark    |   20 |
+---------+------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM person_archive;
+------+-----+---------------------+
| name | age | time                |
+------+-----+---------------------+
| John |  21 | 2021-04-09 09:37:57 |
+------+-----+---------------------+
1 row in set (0.00 sec)
```

Inserting the value back into the *person* table keeps the log of the deleted data in the *person_archive* table:

```
INSERT INTO person VALUES ('John', 21);
```

```
mysql> INSERT INTO person(name, age) VALUES ('John', 21);
Query OK, 1 row affected (0.14 sec)

mysql> SELECT * FROM person;
+---------+------+
| name    | age  |
+---------+------+
| Matthew |   25 |
| Mark    |   20 |
| John    |   21 |
+---------+------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM person_archive;
+------+-----+---------------------+
| name | age | time                |
+------+-----+---------------------+
| John |  21 | 2021-04-09 09:37:57 |
+------+-----+---------------------+
1 row in set (0.00 sec)
```

The **BEFORE DELETE** trigger is useful for logging any table change attempts.

Create an AFTER DELETE Trigger

Make an **AFTER DELETE** trigger with:

```
CREATE TRIGGER <trigger name> AFTER DELETE
ON <table name>
FOR EACH ROW
<trigger body>;
```

The **AFTER DELETE** triggers maintain information updates that require the data row to disappear before making the updates.

AFTER DELETE Trigger Example

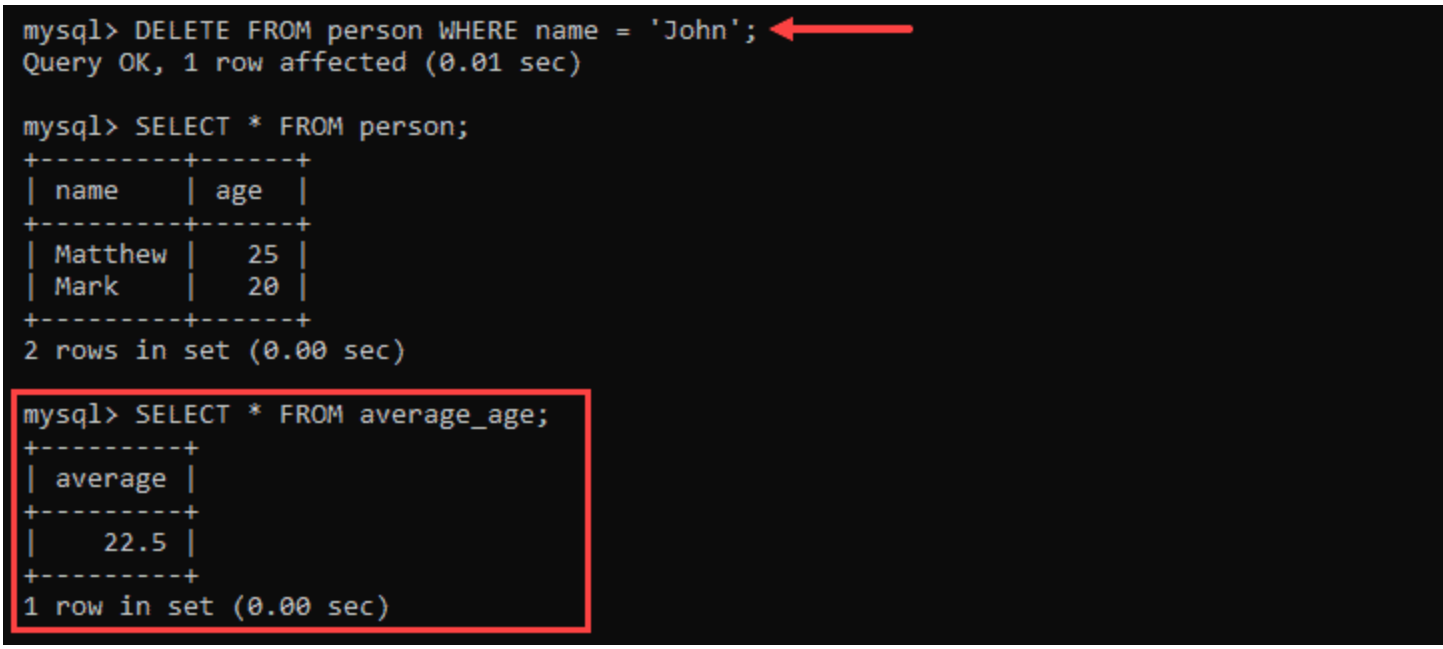
Create an **AFTER DELETE** trigger on the table *person* to update the *average_age* table with the new information:

```
delimiter //
CREATE TRIGGER person_ad AFTER DELETE
ON person
FOR EACH ROW
UPDATE average_age SET average = (SELECT AVG(person.age) FROM person); //
delimiter ;
```

```
mysql> delimiter //
mysql> CREATE TRIGGER person_ad AFTER DELETE
    -> ON person
    -> FOR EACH ROW
    -> UPDATE average_age SET average = (SELECT AVG(age) FROM person); //
Query OK, 0 rows affected (0.25 sec)

mysql> delimiter ;
```

Deleting a record from the table *person* updates the *average_age* table with the new average:



Without the **AFTER DELETE** trigger, the information does not update automatically.

Result:

Thus the Trigger for Before and After update, Delete, Insert operations were executed successfully.

.

EX: NO: 9 **VIEWS AND INDEXES**

AIM:

To create view and index for database tables with a large number of records.

.

VIEWS

OBJECTIVE:

- Views Helps to encapsulate complex query and make it reusable.
- Provides user security on each view - it depends on your data policy security.
- Using view to convert units - if you have a financial data in US currency, you can
- Create view to convert them into Euro for viewing in Euro currency.

PROCEDURE

STEP 1: Start
STEP 2: Create the table with its essential attributes.
STEP 3: Insert attribute values into the table.
STEP 4: Create the view from the above created table.
STEP 5: Execute different Commands and extract information from the View.
STEP 6: Stop

SQL COMMANDS

1. COMMAND NAME: **CREATE VIEW**
COMMAND DESCRIPTION: **CREATE VIEW** command is used to define a view.

2. COMMAND NAME: **INSERT IN VIEW**
COMMAND DESCRIPTION: **INSERT** command is used to insert a new row into the view.

3. COMMAND NAME: **DELETE IN VIEW**
COMMAND DESCRIPTION: **DELETE** command is used to delete a row from the view.

4. COMMAND NAME: **UPDATE OF VIEW**
COMMAND DESCRIPTION: **UPDATE** command is used to change a value in a tuple without changing all values in the tuple.

5. COMMAND NAME: **DROP OF VIEW**
COMMAND DESCRIPTION: **DROP** command is used to drop the view table

COMMANDS EXECUTION

CREATION OF TABLE

mysql> create table employee(empid integer,emp_name varchar(20),deptname varchar(10),dept_no integer,DOJ date);
Query OK, 0 rows affected (0.39 sec)
5 rows in set (0.00 sec)

DESCRIPTION OF TABLE:

```
mysql> DESC EMPLOYEE;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| emp_name  | varchar(20) | YES  |     | NULL    |       |
| emp_id    | int         | YES  |     | NULL    |       |
| dept_name | varchar(10) | YES  |     | NULL    |       |
| dept_no   | int         | YES  |     | NULL    |       |
| DOJ       | date        | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
```

DISPLAY VIEW:
mysql>SELECT * FROM EMPLOYEE;

```
+----------+--------+------------+---------+------------+
| emp_name | emp_id | dept_name  | dept_no | DOJ        |
+----------+--------+------------+---------+------------+
| ravi     |    124 | project    |       3 | 2020-05-03 |
| vijay    |    110 | developing |       2 | 2020-07-23 |
| jay      |    112 | HR         |       1 | 2018-07-01 |
+----------+--------+------------+---------+------------+
3 rows in set (0.00 sec)
```

SYNTAX FOR CREATION OF VIEW

MYSQL> CREATE <VIEW> <VIEW NAME> AS SELECT<COLUMN_NAME_1>, <COLUMN_NAME_2> FROM <TABLE NAME>;

CREATION OF VIEW

mysql> create view empview as select emp_name,empid,deptname,dept_no from employee;
Query OK, 0 rows affected (0.09 sec)

DESCRIPTION OF VIEW

MYSQL> DESC EMPVIEW;

```
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| emp_name  | varchar(20) | YES  |     | NULL    |       |
| emp_id    | int         | YES  |     | NULL    |       |
| dept_name | varchar(10) | YES  |     | NULL    |       |
| dept_no   | int         | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

DISPLAY VIEW:
SQL> SELECT * FROM EMPVIEW;

```
+----------+--------+------------+---------+
| emp_name | emp_id | dept_name  | dept_no |
+----------+--------+------------+---------+
| ravi     |    124 | project    |       3 |
| vijay    |    110 | developing |       2 |
| jay      |    112 | HR         |       1 |
+----------+--------+------------+---------+
3 rows in set (0.00 sec)
```

INSERTION INTO VIEW

INSERT STATEMENT:
SYNTAX:

MYSQL>INSERT INTO <VIEW_NAME> (COLUMN NAME1,...)VALUES(VALUE1,....);

mysql> INSERT INTO EMPVIEW VALUES('SRI',120,'HR',1);
Query OK, 1 row affected (0.04 sec)
SQL> SELECT * FROM EMPVIEW;

```
+----------+--------+------------+---------+
| emp_name | emp_id | dept_name  | dept_no |
+----------+--------+------------+---------+
| ravi     |    124 | project    |       3 |
| vijay    |    110 | developing |       2 |
| jay      |    112 | HR         |       1 |
| SRI      |    120 | HR         |       1 |
+----------+--------+------------+---------+
4 rows in set (0.00 sec)
```

DELETION OF VIEW:

DELETE STATEMENT:

SYNTAX:

SQL> DELETE <VIEW_NMAE>WHERE <COLUMN NMAE> =’VALUE’;
mysql> DELETE FROM EMPVIEW WHERE EMP_NAME='SRI';
Query OK, 1 row affected (0.30 sec)
SQL> SELECT * FROM EMPVIEW;

```
+----------+--------+------------+---------+
| emp_name | emp_id | dept_name  | dept_no |
+----------+--------+------------+---------+
| ravi     |    124 | project    |       3 |
| vijay    |    110 | developing |       2 |
| jay      |    112 | HR         |       1 |
+----------+--------+------------+---------+
3 rows in set (0.01 sec)
```

UPDATE STATEMENT:

SYNTAX:

MYSQL>UPDATE <VIEW_NAME> SET< COLUMN NAME> = <COLUMN NAME> <VIEW> WHERE <COLUMNNAME>=VALUE;

mysql> UPDATE EMPVIEW SET EMP_NAME='SRIVIJAY' WHERE EMP_ID=110;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1 Changed: 1 Warnings: 0

```
mysql> SELECT * FROM EMPVIEW;
+----------+--------+------------+---------+
| emp_name | emp_id | dept_name  | dept_no |
+----------+--------+------------+---------+
| ravi     |    124 | project    |       3 |
| SRIVIJAY |    110 | developing |       2 |
| jay      |    112 | HR         |       1 |
+----------+--------+------------+---------+
3 rows in set (0.00 sec)
```

DROP A VIEW:

SYNTAX:
MYSQL> DROP VIEW <VIEW_NAME>

EXAMPLE:

mysql> DROP VIEW EMPVIEW;

Query OK, 0 rows affected (0.08 sec)

INDEX

- The CREATE INDEX statement is used to create indexes in tables.
- Indexes allow the database application to find data fast; without reading the whole table.
- An index can be created in a table to find data more quickly and efficiently.
- The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So you should only create indexes on columns (and tables) that will be frequently searched against.

CREATE INDEX

Syntax

Creates an index on a table. Duplicate values are allowed:

CREATE INDEX index_name ON table_name (column_name);

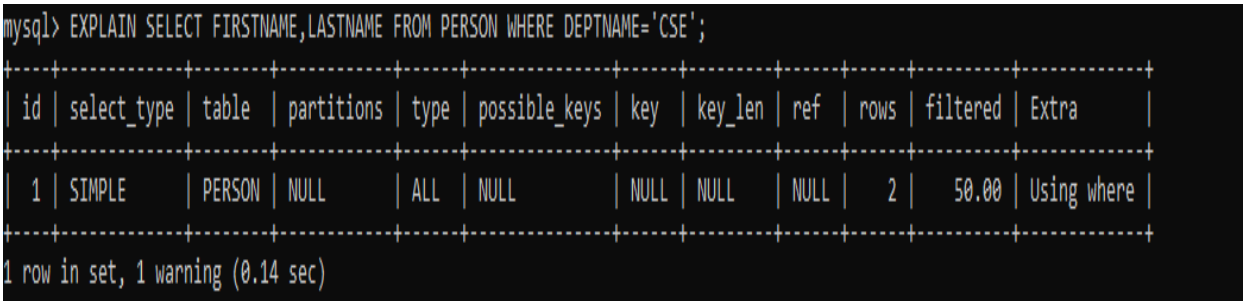
mysql> CREATE TABLE PERSON(FIRSTNAME VARCHAR(20),LASTNAME VARCHAR(20),DEPTNO INTEGER,DEPTNAME VARCHAR(20));
Query OK, 0 rows affected (0.21 sec)

mysql> CREATE INDEX IDX_PNAME ON PERSON(LASTNAME,FIRSTNAME);
Query OK, 0 rows affected (0.30 sec)
Records: 0 Duplicates: 0 Warnings: 0

To access the query plan for the SELECT query, execute the following:

After executing the above statement, the index is created successfully. Now, run the below statement to see how MySQL internally performs this query.

mysql> EXPLAIN SELECT FIRSTNAME,LASTNAME FROM PERSON WHERE DEPTNAME='CSE';



SHOW INDEXES:

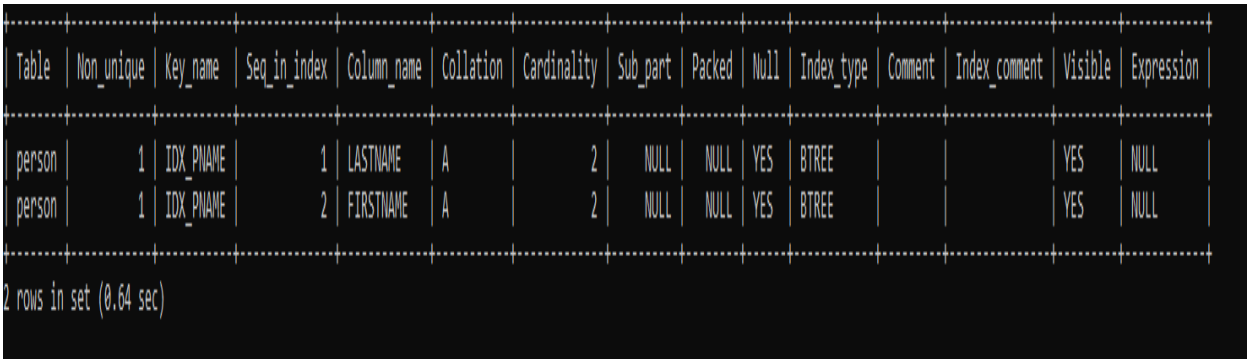


Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
person	1	IDX_PNAME	1	LASTNAME	A	2	NULL	NULL	YES	BTREE			YES	NULL
person	1	IDX_PNAME	2	FIRSTNAME	A	2	NULL	NULL	YES	BTREE			YES	NULL

2 rows in set (0.04 sec)

DROPPING INDEXES

mysql> DROP INDEX IDX_PNAME ON PERSON;

Query OK, 0 rows affected (0.12 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> show indexes from person;

Empty set (0.00 sec)

Result:

Thus the views and indexes created on person table was executed successfully.

EX: NO: 10 CREATION OF XML DATABASE AND VALIDATION USING XML SCHEMA

AIM:

To Create a XML database and validation using XML schema.

Procedure:

1. Search liquid xml validator from google.

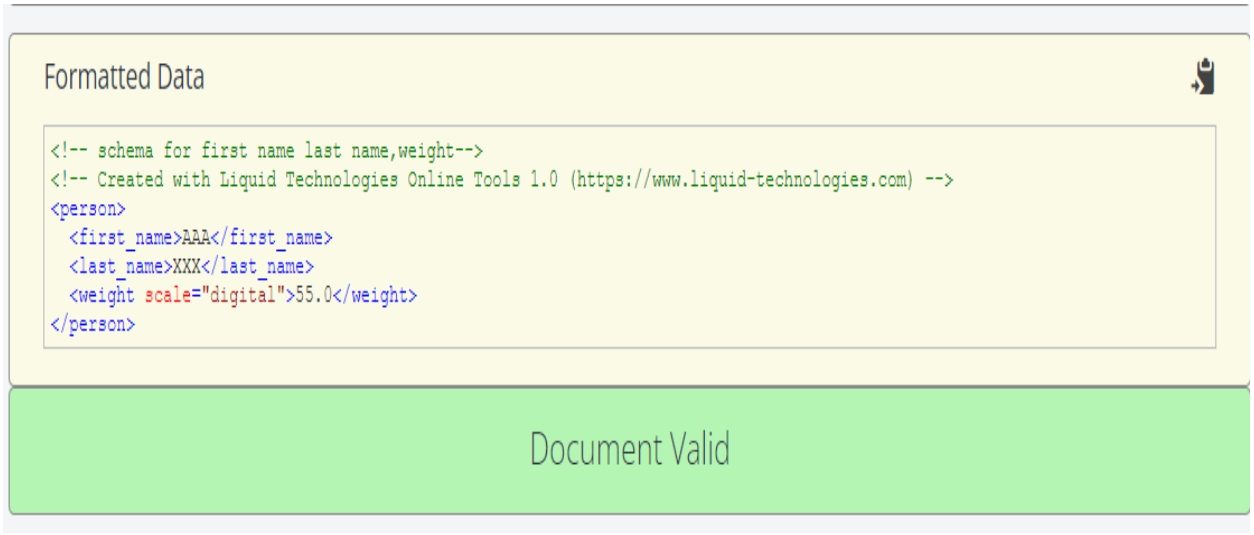
```
  <!-- schema for first name last name,weight-->
<person>
  <first_name>AAA</first_name>
  <last_name>XXX</last_name>
  <weight scale="digital">55.0</weight>
</person>
```

```
<?xml version="1.0">
<xs:schema xmlns:xs="http://www.w3.org/2001/XML Schema">
  <xs:element name="person">
            <xs:complexType>
                    <xs:sequence>
                            <xs:element name="first_name" type="xs:string/>
                            <xs:element name="last_name" type="xs:string/>
                            <xs:element name="weight">
                                    <xs:complexType>
                                    <xs:simpleContent>
                                            <xs:extension base="xs:float">
                                                    <xs:attribute name="scale"
type="xs:string"/>
                                            </xs:extension>
                                    </xs:simpleContent>
                            </xs:complexType>
                            </xs:element>
                    </xs:sequence>
            </xs:complexType>
  </xs:element>
```

</xs:schema>

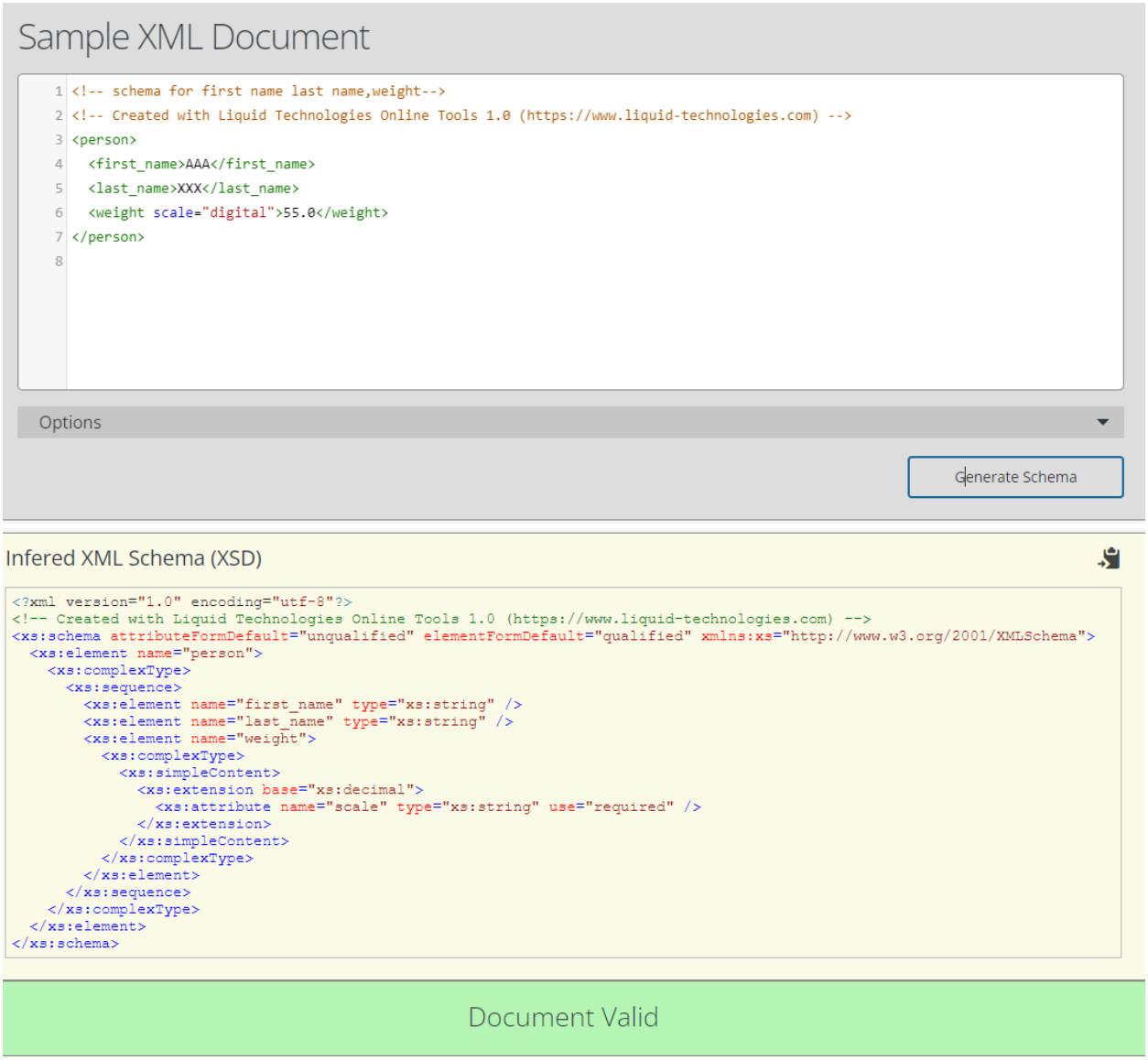
2. Open xml validator link from google.

2.Select xml formatter.



3. Select XML TO XSD

Copy the formatted text there.

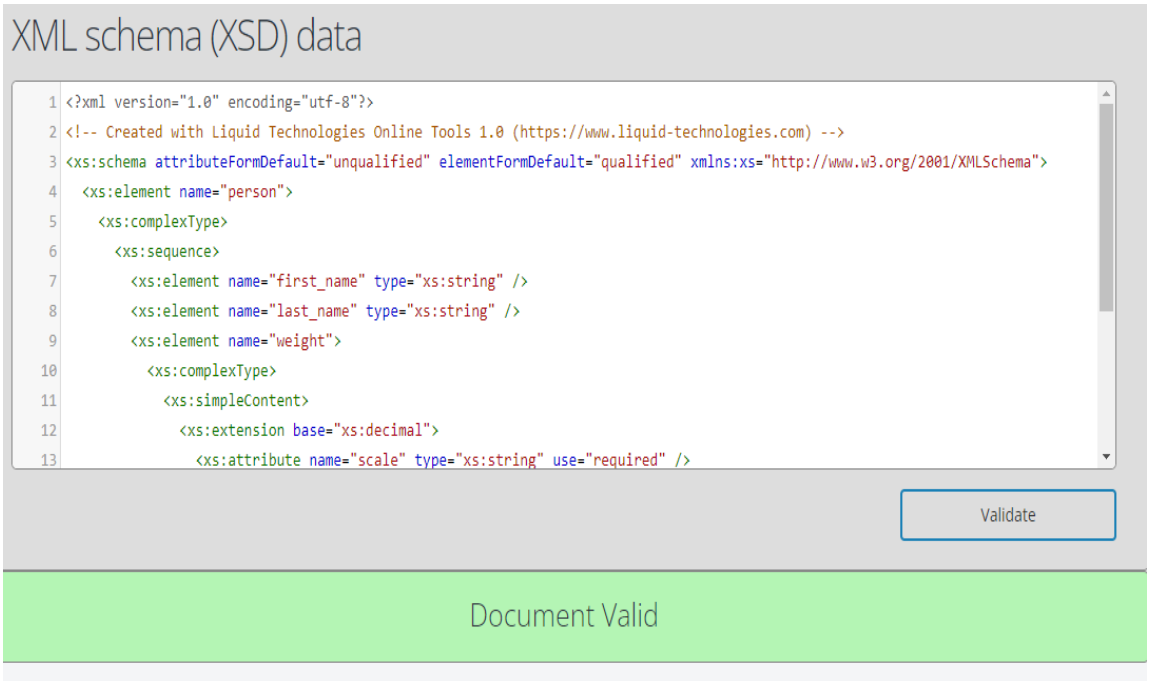


4. SELECT XML VALIDATOR XSD

Copy the generated text.

5. **Last step select** XML Validator (XSD)

6. Copy first few lines of code in first box and the generated code in the second box.



RESULT:

Thus the creation of XML database and validation using XML schema was executed successfully.