Jagannath Gupta Institute of Engineering and Technology

LAB MANUAL
4CS4-22: Database Management System Lab

Experiment-1
Design a Database and create required tables. For e.g. Bank, College Database
Solution-

```
CREATE DATABASE Bank;

BEGIN TRANSACTION;


/* Create a table called NAMES */

CREATE TABLE STUDENT(S_Id integer PRIMARY KEY, S_Name text, S_CITY);


/* Create few records in this table */

INSERT INTO STUDENT VALUES(1,'Tom','AJMER');

INSERT INTO STUDENT VALUES(2,'Lucy','KOTA');

INSERT INTO STUDENT VALUES(3,'Frank','JAIPUR');

INSERT INTO STUDENT VALUES(4,'Jane','DELHI');

INSERT INTO STUDENT VALUES(5,'Robert','MUMBAI');

COMMIT;


/* Display all the records from the table */

SELECT * FROM STUDENT;
```

Experiment-2

Apply the constraints like Primary Key, Foreign key, NOT NULL to the tables.

```
CREATE TABLE DEPT ( DNAME VARCHAR(10) NOT NULL,
 DNUMBER INTEGER NOT NULL,
MGRSSN CHAR(9), MGRSTARTDATE CHAR(9),
 PRIMARY KEY (DNUMBER), UNIQUE (DNAME),
FOREIGN KEY (MGRSSN) REFERENCES EMP(SSN));
```

PROGRAM-3
Write a SQL statement for implementing ALTER,UPDATE and DELETE.

```
BEGIN TRANSACTION;

/* Create a table called NAMES */
CREATE TABLE student(
sid int,
sname varchar(255),
scity varchar(255),
sage int
);

/* insert data into table */
INSERT INTO student VALUES(101,"AMIT","AJMER",26);
insert into student values(102,"manish","kota",35);
insert into student values(103,"ankit","jaipur",40);
```

/* show the table */

```
select *
from student;
```

/* Alter table */

```
ALTER TABLE student
add sweight int;
```

/* insert data into table */

```
INSERT INTO student VALUES(101,"AMIT","AJMER",26,90);
insert into student values(102,"manish","kota",35,100);
insert into student values(103,"ankit","jaipur",40,75);
```

/* show the table */

```
select *
from student;
```

/* UPDATE table */

```
UPDATE student
set sname='RAM',scity='newyork'
 where sid='101';
```

/* show the table */

```
select *

from student;

/*DELETE IS USED delete particular Record*/

DELETE from student

where sid='102';

 /* show the table */

select *

from student;
```

Experiment-4

Write the queries to implement the joins.

Solution:-

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

Consider the two tables below:

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

The simplest Join is INNER JOIN.

INNER JOIN: The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

INNER JOIN table2
```

```
ON table1.matching_column = table2.matching_column;
```

table1: First table.
table2: Second table
matching_column: Column common to both the tables.

This query will show the names and age of students enrolled in different courses.

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE FROM Student

INNER JOIN StudentCourse

ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

utput:

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

Experiment-5

Write the query for implementing the following functions: MAX (), MIN (), AVG ()

Solution

Students-Table

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

AVG(): It returns average value after calculating from values in a numeric column.
Syntax:

```
SELECT AVG(column_name) FROM table_name;
```

Queries:

1. Computing average marks of students.
2. SELECT AVG(MARKS) AS AvgMarks FROM Students;

Output:

AvgMarks
80

MAX(): The MAX() function returns the maximum value of the selected column.
Syntax:

```
SELECT MAX(column_name) FROM table_name;
```

Queries:

1. Fetching maximum marks among students from the Students table.
2. SELECT MAX(MARKS) AS MaxMarks FROM Students;

Output:

MaxMarks
95

MIN(): The MIN() function returns the minimum value of the selected column.
Syntax:

```
SELECT MIN(column_name) FROM table_name;
```

Queries:

1. Fetching minimum marks among students from the Students table.
2. SELECT MIN(MARKS) AS MinMarks FROM Students;

Output:

MinMarks
50

Experiment-6
Write the query to create the views.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

The following SQL creates a view that shows all customers from Brazil:

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

We can query the view above as follows:

```
SELECT * FROM [Brazil Customers];
```

Experiment-7
Perform the queries for triggers.
Trigger: A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.
Syntax:

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
```

```
on [table_name]
[for each row]
[trigger_body]
```

Given Student Report Database, in which student marks assessment is recorded. In such schema, create a trigger so that the total and average of specified marks is automatically inserted whenever a record is insert.

Here, as trigger will invoke before record is inserted so, BEFORE Tag can be used.

```
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| tid   | int(4)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
| subj1 | int(2)      | YES  |     | NULL    |                |
| subj2 | int(2)      | YES  |     | NULL    |                |
| subj3 | int(2)      | YES  |     | NULL    |                |
| total | int(3)      | YES  |     | NULL    |                |
| per   | int(3)      | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
```

SQL Trigger to problem statement.

```
create trigger stud_marks
before INSERT
on
Student
for each row
set Student.total = Student.subj1 + Student.subj2 + Student.subj3, Student.per
```

Experiment-8
Perform the following operation for demonstrating the insertion , updation and deletion.

Solution:-

we will insert 2 rows into the students table, one for each student:

```
INSERT INTO Students(StudentId, StudentName, DepartmentId, DateOfBirth)
        VALUES(11, 'Ahmad', 4, '1997-10-12');

INSERT INTO Students VALUES(12, 'Aly', 4, '1996-10-12');
```

In the following UPDATE statement, we will update the DepartmentId for the Student with StudentId = 6 to be 3:

```
UPDATE Students
SET DepartmentId = 3
WHERE StudentId = 6;
```

In the following statement, we will delete two students with StudentId 11 and 12:

```
DELETE FROM Students WHERE StudentId = 11 OR StudentId = 12;
```

Experiment-9
Using the referential integrity constraints

Another example of Referential Integrity is Employee and Department relationship. If we have dept_id as foreign key in Employee table than by using referential integrity constraints we can avoid creating Employee without department or non existing department.

In short Referential Integrity makes primary key foreign key relationship viable. Let's first create Employee and Department table with primary key, foreign key and referential Integrity constraints.

```
CREATE TABLE Department (dept_id INT NOT NULL,
                dept_name VARCHAR(256),
                PRIMARY KEY (dept_id)) ENGINE=INNODB;

CREATE TABLE Employee (emp_id INT NOT NULL,
                emp_name VARCHAR(256),
                dept_id INT,
```

```
        FOREIGN KEY (dept_id) REFERENCES Department(dept_id)
        ON DELETE CASCADE) ENGINE=INNODB;
```

Above SQL statements will create both Department and Employee table. dept_id is now foreign key in Employee table.

In this SQL, while creating foreign key we have specified ON DELETE clause which tells, what needs to done when a record from parent table is deleted. CASCADE referential action allows to **delete or update all matching rows from child table**, after deleting a record in parent table. This way Refrential Integrity preserve data integrity of relationship.

Experiment-10
Write the query for creating the users and their role.

Solution:- standard SQL syntax for creating how to create users,

```
CREATE USER username IDENTIFIED BY password
IDENTIFIED WITH auth_plugin
```

CREATE ROLE creates a set of privileges which may be assigned to users of a database. Once a role is assigned to a user, (s)he gets all the Privileges of that role. By creating and granting roles, best means of database security can be practiced.

SQL Syntax:

```
CREATE ROLE role_name [WITH ADMIN {CURRENT_USER | CURRENT_ROLE}]
```

Parameters:

Name	Description
role_name	A name to identify the role.

Explanation:

With the above syntax, a role with role_name is created and immediately assigned to the current user or the currently active role is passed on to other users. The default usage is WITH ADMIN CURRENT_USER.