TAKSHASHILA UNIVERSITY

Tamil Nadu State Private University

(Established under Tamil Nadu Private Universities Act 2019
& Recognized by UGC u/s 2(f) of the UGC Act, 1956)
Ongur, Tindivanam Taluk, Villupuram District, Tamil Nadu - 604305

TAKSHASHILA
UNIVERSITY

FACULTY OF SCIENCES

SCHOOL OF COMPUTER SCIENCE

STUDENT RECORD

REGISTER NUMBER : _____

NAME OF THE STUDENT : _____

PROGRAM NAME : _____

YEAR / SEMESTER : _____

COURSE CODE : _____

COURSE NAME : _____

TAKSHASHILA UNIVERSITY

Tamil Nadu State Private University

(Established under Tamil Nadu Private Universities Act 2019
& Recognized by UGC u/s 2(f) of the UGC Act, 1956)
Ongur, Tindivanam Taluk, Villupuram District, Tamil Nadu – 604305

FACULTY OF SCIENCES

School of Computer Science

Bonafide Certificate

Register Number	

This is to certify that this is a Bonafide Record of practical work done by Mr./Ms. ..... a Student of ..... in School of Computer Science, has successfully completed the..... Laboratory during the academic year .....

Signature of Subject Staff *Signature of School In-charge*

Submitted the University Practical Examination held on …….

Signature Internal Examiner Signature External Examiner

INDEX

EXERCISE 1: CREATE TABLES: STUDENTS, COURSES, ENROLLMENTS. INSERT AT LEAST 5 SAMPLE RECORDS IN EACH TABLE.

AIM:

To design a relational database schema using Data Definition Language (DDL) and populate the tables with sample data using Data Manipulation Language (DML) to establish a fundamental Student Information System.

PROGRAM:

```
-- Create the Students table
CREATE TABLE Students
(
   StudentID INT PRIMARY
   KEY, FirstName
   VARCHAR(50) NOT NULL,
   LastName VARCHAR(50)
   NOT NULL,
   DateOfBirth DATE,
   Email VARCHAR(100) UNIQUE
);

-- Create the Courses table
CREATE TABLE Courses (
   CourseID INT PRIMARY KEY,
   CourseName VARCHAR(100)
   NOT NULL,
   CourseCode VARCHAR(10) UNIQUE NOT NULL,
   Credits INT
);
```

```
-- Create the Enrollments
table CREATE TABLE
Enrollments (
   EnrollmentID INT
   PRIMARY KEY, StudentID
   INT NOT NULL, CourseID
   INT NOT NULL,
   EnrollmentDate DATE NOT
   NULL, Grade CHAR(1),
   FOREIGN KEY (StudentID) REFERENCES
   Students(StudentID), FOREIGN KEY (CourseID)
   REFERENCES Courses(CourseID)
);

-- Insert sample records into Students table
INSERT INTO Students (StudentID, FirstName, LastName, DateOfBirth, Email)
VALUES (1, 'Alice', 'Smith', '2003-05-15', 'alice.smith@example.com'),
(2, 'Bob', 'Johnson', '2002-11-22', 'bob.j@example.com'),
(3, 'Charlie', 'Brown', '2004-01-08', 'charlie.b@example.com'),
(4, 'Diana', 'Prince', '2003-09-30', 'diana.p@example.com'),
(5, 'Ethan', 'Hunt', '2002-07-12', 'ethan.h@example.com');

-- Insert sample records into Courses table
INSERT INTO Courses (CourseID, CourseName, CourseCode, Credits)
VALUES (101, 'Introduction to Computer Science', 'CS101', 3),
(102, 'Calculus I', 'MA101', 4),
(103, 'English Literature', 'ENGL201', 3),
(104, 'Database Management', 'DB301', 3),
(105, 'Linear Algebra', 'MA201', 4);
```

```
-- Insert sample records into Enrollments table

INSERT INTO Enrollments (EnrollmentID, StudentID, CourseID,
EnrollmentDate, Grade) VALUES

(1001, 1, 101, '2024-09-01', 'A'),

(1002, 2, 102, '2024-09-01', 'B'),

(1003, 3, 101, '2024-09-05', 'C'),

(1004, 4, 103, '2024-09-01', 'A'),

(1005, 5, 104, '2024-09-10', 'B');
```

OUTPUT :

select * from students;

StudentID	FirstName	LastName	DateOfBirth	Email
1	Alice	Smith	2003-05-15	alice.smith@example.com
2	Bob	Johnson	2002-11-22	bob.j@example.com
3	Charlie	Brown	2004-01-08	charlie.b@example.com
4	Diana	Prince	2003-09-30	diana.p@example.com
5	Ethan	Hunt	2002-07-12	ethan.h@example.com

select * from courses;

CourseID	CourseName	CourseCode	Credits
101	Introduction to Computer Science	CS101	3
102	Calculus I	MA101	4
103	English Literature	ENGL201	3
104	Database Management	DB301	3
105	Linear Algebra	MA201	4

select * from enrollments;

EnrollmentID	StudentID	CourseID	EnrollmentDate	Grade
1001	1	101	2024-09-01	A
1002	2	102	2024-09-01	B
1003	3	101	2024-09-05	C
1004	4	103	2024-09-01	A
1005	5	104	2024-09-10	B

RESULT :

Hence the above program is verified and executed successfully.

EXERCISE 2 :GIVEN A SINGLE TABLE WITH STUDENT AND COURSE DATA, SPLIT IT INTO 3NF.

AIM:

To apply the principles of database normalization by decomposing an unnormalized dataset into Third Normal Form (3NF) to minimize data redundancy and ensure referential integrity.

PROGRAM:

```
CREATE TABLE
 StudentCourseData
 ( StudentID INT PRIMARY
 KEY,
 StudentName
 VARCHAR(255),
 CourseID INT,
 CourseName
 VARCHAR(255),
 CourseInstructor
 VARCHAR(255),
 InstructorEmail
 VARCHAR(255)
);
```

1. Create the Studentstable

```
CREATE TABLE
 Students ( StudentID
 INT PRIMARY KEY,
 StudentName VARCHAR(255)
);
```

2. Create the Instructorstable

```
CREATE TABLE Instructors (
   InstructorEmail VARCHAR(255)
   PRIMARY KEY, InstructorName
   VARCHAR(255)
);
```

3. Create the Coursestable

```
CREATE TABLE
   Courses ( CourseID
   INT PRIMARY KEY,
   CourseName
   VARCHAR(255),
   InstructorEmail
   VARCHAR(255),
   FOREIGN KEY (InstructorEmail) REFERENCES Instructors(InstructorEmail)
);

INSERT INTO Courses (CourseID, CourseName,
InstructorEmail) SELECT DISTINCT CourseID,
CourseName, InstructorEmail  FROM
StudentCourseData;
```

4. Create the StudentCoursesjunction table:

```
CREATE TABLE
   StudentCourses
   ( StudentID INT,
   CourseID INT,
   PRIMARY KEY (StudentID, CourseID),
```

```
    FOREIGN KEY (StudentID) REFERENCES
    Students(StudentID), FOREIGN KEY (CourseID)
    REFERENCES Courses(CourseID)
);
INSERT INTO StudentCourses (StudentID, CourseID) SELECT DISTINCT StudentID,
CourseID FROM StudentCourseDat
```

OUTPUT :

STUDENTS TABLE

StudentID	StudentName
1	Alice Smith
2	Bob Johnson
3	Charlie Brown

INSTRUCTORS TABLE

InstructorEmail	InstructorName
jdoe@example.c om	Dr. John Doe
jsmith@example.com	Dr. Jane Smith
edavis@example.com	Dr. Emily Davis

COURSES TABLE

CourseID	CourseName	InstructorEmail
101	Database Systems	jdoe@example.com
102	Operating Systems	jsmith@example.com
103	Computer Networks	edavis@example.com

STUDENTCOURSES TABLE (JUNCTION)

StudentID	**CourseID**
1	101
2	102
3	101
1	103

RESULT :

Hence the above program is verified and executed successfully

EXERCISE 3 : CREATE TABLES: STUDENTS, COURSES, ENROLLMENTS. WRITE QUERIES TO: DISPLAY ALL STUDENTS, SHOW COURSES ENROLLEDBY A PARTICULAR STUDENT.

AIM:

To demonstrate proficiency in SQL data retrieval by utilizing SELECT statements to filter records, performing JOIN operations across multiple tables, and applying aggregate functions (like COUNT) for data analysis.

PROGRAM:

```
-- Create the Students
table CREATE TABLE
Students (
   StudentID INT PRIMARY
   KEY, FirstName
   VARCHAR(50) NOT NULL,
   LastName VARCHAR(50)
   NOT NULL,
   DateOfBirth DATE,
   Email VARCHAR(100) UNIQUE
);

-- Create the Courses
table CREATE TABLE
Courses (
   CourseID INT PRIMARY KEY,
   CourseName VARCHAR(100)
   NOT NULL,
   CourseCode VARCHAR(10) UNIQUE NOT NULL,
   Credits INT);
```

```
-- Create the Enrollments table
CREATE TABLE
  Enrollments
  ( EnrollmentID INT
  PRIMARY KEY,
  StudentID INT NOT
  NULL, CourseID INT
  NOT NULL,
  EnrollmentDate DATE NOT
  NULL, Grade CHAR(1),
  FOREIGN KEY (StudentID) REFERENCES
  Students(StudentID), FOREIGN KEY (CourseID)
  REFERENCES Courses(CourseID)
);

-- Insert sample records into Students table
INSERT INTO Students (StudentID, FirstName, LastName, DateOfBirth,
Email) VALUES (1, 'Alice', 'Smith', '2003-05-15',
'alice.smith@example.com'),
(2, 'Bob', 'Johnson', '2002-11-22', 'bob.j@example.com'),
(3, 'Charlie', 'Brown', '2004-01-08', 'charlie.b@example.com'),
(4, 'Diana', 'Prince', '2003-09-30', 'diana.p@example.com'),
(5, 'Ethan', 'Hunt', '2002-07-12', 'ethan.h@example.com');

-- Insert sample records into Courses table
INSERT INTO Courses (CourseID, CourseName, CourseCode,
Credits) VALUES (101, 'Introduction to Computer Science', 'CS101',
3),
```

(102, 'Calculus I', 'MA101', 4),

(103, 'English Literature', 'ENGL201', 3),

(104, 'Database Management', 'DB301', 3),

(105, 'Linear Algebra', 'MA201', 4);

-- Insert sample records into Enrollments table

INSERT INTO Enrollments (EnrollmentID, StudentID, CourseID, EnrollmentDate, Grade) VALUES

(1001, 1, 101, '2024-09-01', 'A'),

(1002, 2, 102, '2024-09-01', 'B'),

(1003, 3, 101, '2024-09-05', 'C'),

(1004, 4, 103, '2024-09-01', 'A'),

(1005, 5, 104, '2024-09-10', 'B');

A) Display all students

SELECT *FROM Students;

B) Show courses enrolled by a particular student (e.g., StudentID = 1)

SELECT s.StudentID,

s.StudentName,

c.CourseID,

c.CourseName

FROM Students s

JOIN StudentCourses sc ON s.StudentID =

sc.StudentID JOIN Courses c ON

sc.CourseID = c.CourseID WHERE s.StudentID

= 1;

C) Count number of students in each course

```
SELECT c.CourseID,
    c.CourseName,
    COUNT(sc.StudentID) AS
StudentCount FROM Courses c
LEFT JOIN StudentCourses sc ON c.CourseID =
sc.CourseID GROUP BY c.CourseID, c.CourseName;
```

OUTPUT:

All Students

StudentID	Name
1	Alice
2	Bob
3	Charlie
4	Diana
5	Eve

Courses by Student ID=1 (Alice)

CourseID	CourseName
101	Math101
102	Physics101

Students per Course

CourseID	CourseName	StudentCount
101	Math101	2
102	Physics101	1
103	Chemistry101	1

RESULT:

Hence the above program is verified and executed successfully.

EXERCISE 4:CREATE TABLES:STUDENT,COURSE,ENROLLMENT.DIVIDE STUDENT DATA BY BRANCHES AND STORE THEM IN TWO SEPARATE TABLES

AIM:

To implement horizontal fragmentation (partitioning) by splitting a global relation into disjoint subsets based on a specific predicate (e.g., Branch) to optimize query performance and manageability.

PROGRAM:

```
-- Create the     COURSE table
CREATE TABLE COURSE (
   course_id INT PRIMARY KEY,
   course_name VARCHAR(100),
   credits INT
);
-- Create the STUDENT table (General table before division by branch)
CREATE TABLE STUDENT (
   student_id INT PRIMARY KEY,
   name VARCHAR(100),
   branch VARCHAR(50),
   email VARCHAR(100)
);
```

```
-- Create the ENROLLMENT table
CREATE TABLE ENROLLMENT (
    enrollment_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    enrollment_date DATE,
    FOREIGN KEY (student_id) REFERENCES STUDENT(student_id),
    FOREIGN KEY (course_id) REFERENCES COURSE(course_id)
);
-- Create a table for students in the Engineering branch
CREATE TABLE STUDENT_ENGINEERING (
    student_id INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100)
);

-- Create a table for students in the Science branch
CREATE TABLE STUDENT_SCIENCE (
    student_id INT PRIMARY KEY,
    name VARCHAR(100),
```

```
    email VARCHAR(100)
);

-- Insert students in Engineering branch into STUDENT_ENGINEERING table
INSERT INTO STUDENT_ENGINEERING (student_id, name, email)
SELECT student_id, name, email
FROM STUDENT
WHERE branch = 'Engineering';


-- Insert students in Science branch into STUDENT_SCIENCE table
INSERT INTO STUDENT_SCIENCE (student_id, name, email)
SELECT student_id, name, email
FROM STUDENT
WHERE branch = 'Science';
```

OUTPUT:

Branches Table

branch_id	branch_name
111	tindivanam
222	villupuram
333	gingee
444	chennai
555	pondy
666	kerala
777	andhra

Students111 Table

student_id	student_name	branch_id
A1	Arun	111
B1	Anbu	222
C1	Ajay	333
D1	Deva	444
E1	Raj	111
F1	Ragu	666
G1	Aravind	777

JOIN Query Result: Student Names with Branch Names

student_name	branch_name
arun	tindivanam
anbu	villupuram
ajay	gingee
deva	chennai
raj	tindivanam
ragu	kerala
aravind	andhra

.

RESULT:

Hence the above program is verified and executed successfully.

EXERCISE 5 : WRITE A SIMPLE SELECT QUERY ASSUMING TABLES ARE STORED IN DIFFERENT LOCATIONS.

AIM:

To simulate a distributed database environment and demonstrate location transparency by querying data logically integrated but physically distributed across different storage locations.

PROGRAM:

```
mysql> create database college;
Query OK, 1 row affected (0.30 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| anbu               |
| bca                |
| college            |
| company_db         |
| dsp                |
| information_schema |
| mysql              |
| performance_schema |
| pooja              |
| sakila             |
| sys                |
```

```
| world              |
+--------------------+
12 rows in set (0.34 sec)

mysql> connect
college; Connection id:
                        14
Current database:
college

mysql> CREATE TABLE Persons (
    ->     PersonID int,
    ->     LastName varchar(255),
    ->     FirstName varchar(255),
    ->     Address varchar(255),
    ->     City varchar(255)
    -> );
Query OK, 0 rows affected (0.64 sec)
mysql> describe Persons;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| PersonID  | int          | YES  |     | NULL    |       |
| LastName  | varchar(255) | YES  |     | NULL    |       |
| FirstName | varchar(255) | YES  |     | NULL    |       |
| Address   | varchar(255) | YES  |     | NULL    |       |
| City      | varchar(255) | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
```

5 rows in set (0.11 sec)

mysql> insert into Persons values(111,'arun','kumar','Tindivanam','tn');
Query OK, 1 row affected (0.17 sec)

mysql> insert into Persons values(222,'rajan','babu','pondy','tn');
Query OK, 1 row affected (0.84 sec)

mysql> insert into Persons values(333,'deva','kumar','villupuram','tn'); Query
OK, 1 row affected (0.04 sec)

mysql> select * from Persons;

```
+----------+----------+-----------+-----------+------+
| PersonID | LastName | FirstName | Address   | City |
+----------+----------+-----------+-----------+------+
|      111 | arun     | kumar     | Tindivanam | tn   |
|      222 | rajan    | babu      | pondy      | tn   |
|      333 | deva     | kumar     | villupuram | tn   |
+----------+----------+-----------+-----------+------+
```

3 rows in set (0.00 sec)

mysql> create database student;
Query OK, 1 row affected (0.07 sec)

mysql> connect
student; Connection
id: 15

Current database: student

```
mysql> create table student1(id int,gender varchar(20),dob date,mark int,dept varchar(20));
Query OK, 0 rows affected (0.33 sec)

mysql> describe student1;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int         | YES  |     | NULL    |       |
| gender | varchar(20) | YES  |     | NULL    |       |
| dob    | date        | YES  |     | NULL    |       |
| mark   | int         | YES  |     | NULL    |       |
| dept   | varchar(20) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> insert into student1 values(333,'male','2000-05-10',120,'cs');
Query OK, 1 row affected (0.04 sec)

mysql> insert into student1 values(222,'male','2001-06-11',150,'cs');
Query OK, 1 row affected (0.09 sec)

mysql> insert into student1 values(111,'male','2002-07-12',110,'cs');
Query OK, 1 row affected (0.04 sec)
```

```
mysql> insert into student1 values(444,'male','2003-08-01',100,'cs');
Query OK, 1 row affected (0.06 sec)

mysql> select * from student1;
+------+--------+------------+------+------+
| id   | gender | dob        | mark | dept |
+------+--------+------------+------+------+
|  333 | male   | 2000-05-10 |  120 | cs   |
|  222 | male   | 2001-06-11 |  150 | cs   |
|  111 | male   | 2002-07-12 |  110 | cs   |
|  444 | male   | 2003-08-01 |  100 | cs   |
+------+--------+------------+------+------+
4 rows in set (0.00 sec)
mysql> select PersonID ,address from college.Persons;
+----------+-------------+
| PersonID | address     |
+----------+-------------+
|      111 | Tindivanam  |
|      222 | pondy       |
|      333 | villupuram  |
+----------+-------------+
3 rows in set (0.00 sec)

mysql> select PersonID ,mark,genter from student.student1; ERROR
1054 (42S22): Unknown column 'PersonID' in 'field list'
```

```
mysql> select id ,mark,genter from student.student1;
ERROR 1054 (42S22): Unknown column 'genter' in 'field list' mysql>
select id ,mark,gender from student.student1;
+------+------+--------+
| id   | mark | gender |
+------+------+--------+
|  333 |  120 | male   |
|  222 |  150 | male   |
|  111 |  110 | male   |
|  444 |  100 | male   |
+------+------+--------+
4 rows in set (0.00 sec)

mysql> select * from college.Persons inner join student.student1 on
PersonID=id;
+----------+----------+-----------+-----------+------+------+--------+------------+------+------+
| PersonID | LastName | FirstName | Address   | City | id   | gender | dob        | mark | dept |
+----------+----------+-----------+-----------+------+------+--------+------------+------+------+
|      333 | deva     | kumar     | villupuram | tn   |  333 | male   | 2000-05-10 |  120 | cs   |
|      222 | rajan    | babu      | pondy      | tn   |  222 | male   | 2001-06-11 |  150 | cs   |
|      111 | arun     | kumar     | Tindivanam | tn   |  111 | male   | 2002-07-12 |  110 | cs   |
```

RESULT:

Hence the above program is verified and executed successfully.

EXERCISE 6: WRITE SQL STATEMENTS TO DEMONSTRATE: A TRANSACTION USING BEGIN, COMMIT, ROLLBACK, A SITUATION THAT CAUSES DIRTY READ AND RESOLVE IT USING ISOLATION LEVELS

AIM:

To enforce the ACID properties of database transactions using Transaction Control Language (TCL) commands and to manage concurrency by adjusting transaction isolation levels to prevent anomalies such as Dirty Reads.

PROGRAM:

```
create table accounts(
accountID int primary key,
balance int
);
insert into accounts values(1,1000);
begin;
update accounts
set balance=balance-200
where accountID=1;
commit;
begin;
update accounts
set balance=balance-500
where accountID=1;
rollback;
dirty read scenario and its solution
transaction A
set transaction isolation level read uncommitted;
begin;
update accounts
set balance=balance-500
```

where accountID=1;

not committed yet

transaction B

set transaction isolation level read uncommitted;

select balance

from accounts

where accountID=1;

rollback;

begin;

update accounts

set balance=500

where accountID=1;

transaction B

set transaction isolation level read committed;

select balance

from accounts

where accountID=1;

accountID	balance
1	800

RESULT:

Hence the above program is verified and executed successfully

EXERCISE 7 - CREATE A TABLE WITH PLACE NAME, LATITUDE, AND LONGITUDE.WRITE A QUERY TO FIND ALL PLACES IN A CERTAIN AREA (DUMMY CONDITIONS).

AIM:

To implement horizontal fragmentation (partitioning) by splitting a global relation into disjoint subsets based on a specific predicate (e.g., Branch) to optimize query performance and manageability.

PROGRAM:

```
create table
places( placeID int
primary key,placename
varchar(100), latitude
decimal(10,6),
longitude decimal(10,6)
);
insert into places(placeID,placename,latitude,longitude)values
(1,'chennai',13.0827,80.2707),
(2,'bangalore',12.9716,77.5946),
(3,'hyderabad',17.3850,78.4867),
(4,'pune',18.5204,73.8567),
(5,'mumbai',19.0760,72.8777);

select placename,latitude,longitude from places where latitude between 12 and 18 and
longitude between 75 and 80;
```

OUTPUT

Places in Area (lat 12-18, long 75-80)

placename	latitude	longitude
chennai	13.0827	80.2707
bangalore	12.9716	77.5946

RESULT:

Hence the above program is verified and executed successfully

EXERCISE 8 :CREATE A TABLE WITH EMPLOYEE, STARTDATE, AND ENDDATE. QUERY ALL CURRENT EMPLOYEE (ENDDATE IS NULL).

AIM:

To identify all current (active) employees by selecting records where the EndDate is NULL**,** indicating that the employee is still working in the organization.

PROGRAM:

```
CREATE TABLE Employ(
EmployeeID INT PRIMARY KEY,
EmployeeName VARCHAR(100),
StartDate DATE,
EndDate DATE
);
INSERT INTO Employ VALUES (1, 'Arun', DATE'2022-01-10', NULL);
INSERT INTO Employ VALUES (2, 'Divya', DATE'2021-06-15', DATE'2023-12-31');
INSERT INTO Employ VALUES (3, 'karthik', DATE'2023-03-01', NULL);
INSERT INTO Employ VALUES (4, 'Meena', DATE'2022-08-20', DATE'2022-11-30');
SELECT * FROM Employ;
SELECT EmployeeID, EmployeeName, StartDate
FROM Employ
WHERE EndDate IS NULL;
```

```
1|Arun|DATE2022-01-10|
3|karthik|DATE2023-03-01|
```

RESULT:

Hence the above program is verified and executed successfully.

EXERCISE 9 : CREATE AN EMPLOYEES TABLE WITH MANAGERID. WRITE A QUERY TO FIND ALL SUBORDINATES OF A MANAGER USING WITH RECURSIVE

AIM:

To retrieve all employees who work under a given manager**,** including direct and indirect subordinates**,** by using a recursive SQL query **(**WITH RECURSIVE**)** on a self-referencing EMPLOYEES table.

PROGRAM:

```
create table employees(
employeeID INT PRIMARY
KEY, employeename
VARCHAR(50), managerID
INT,
FOREIGN KEY (managerID) REFERENCES employees(employeeID)
);

INSERT INTO employees (employeeID, employeename, managerID)
VALUES (1, 'rahul(manager)', NULL),
(2, 'priya', 1),
(3, 'karan', 1),
(4, 'sneha', 2),
(5, 'vijay', 2),
(6, 'arun', 3);

WITH RECURSIVE subordinates AS (

SELECT employeeID, employeename,
```

```
    managerID FROM employees
    WHERE managerID = 1

    UNION ALL

    SELECT e.employeeID, e.employeename,
    e.managerID FROM employees e
    INNER JOIN subordinates s
        ON e.managerID = s.employeeID
)

SELECT * FROM subordinates;

employeeID int primary
key, employeename
varchar(50), managerID
int,
foreignkey(managerID)references  employees(employeeID)
);
insert into employees(employeeID,employeename,managerID) values
(1,'rahul(manager)',null),
(2,'priya',1),
(3,'karan',1),
(4,'sneha',2),
(5,'vijay',2),
(6,'arun',3);
with recursive suborinates as(
select
employeeID,employeename,managerI
D from employees
```

```
where
managerID=1
union all
select

e.employeeID,e.employeename,e.managerI
D from employees e

inner join sunordinates son e.managerID=s.employeeID)

select * from subordinates;
```

OUTPUT:

Manager Subordinates (Manager ID=1)

Employees Table

employeeID	employeename	managerID
1	rahul(manager)	NULL
2	priya	1
3	karan	1
4	sneha	2
5	vijay	2
6	arun	3

WITH RECURSIVE Subordinates Result

employeeID	employeename	managerID
2	priya	1
3	karan	1
4	sneha	2
5	vijay	2
6	arun	3

RESULT:

Hence the above program is verified and executed successfully

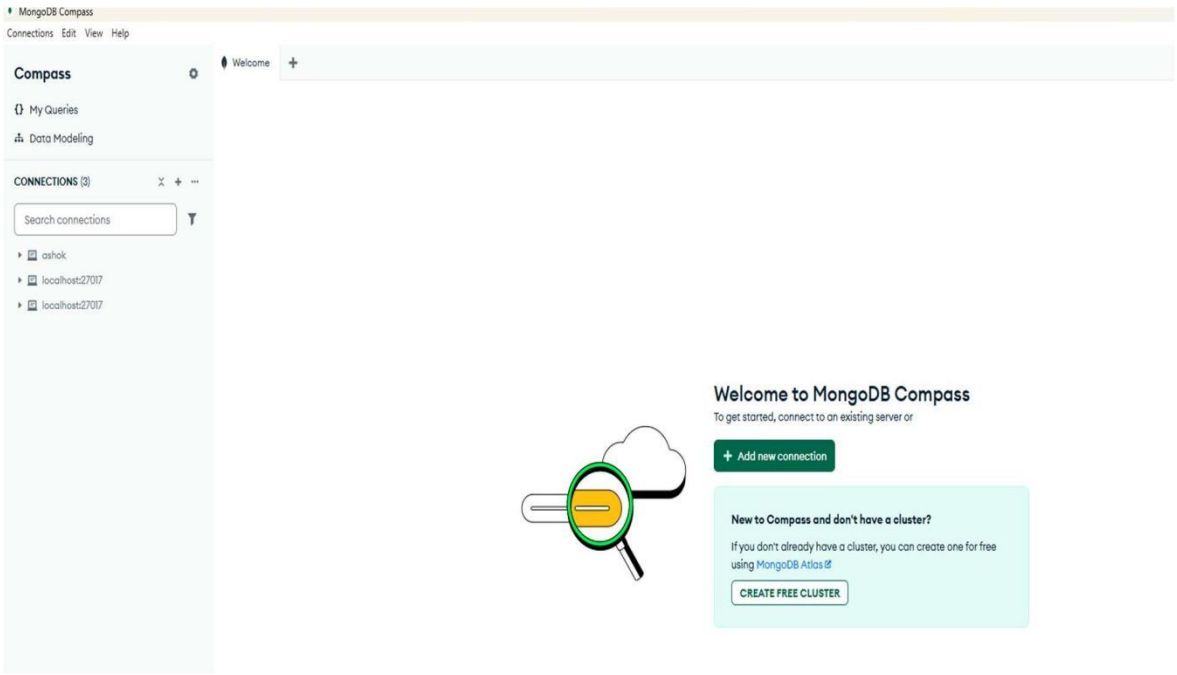
EXERCISE 10 : NOSQL DOCUMENT STORAGE (MONGODB)

AIM:

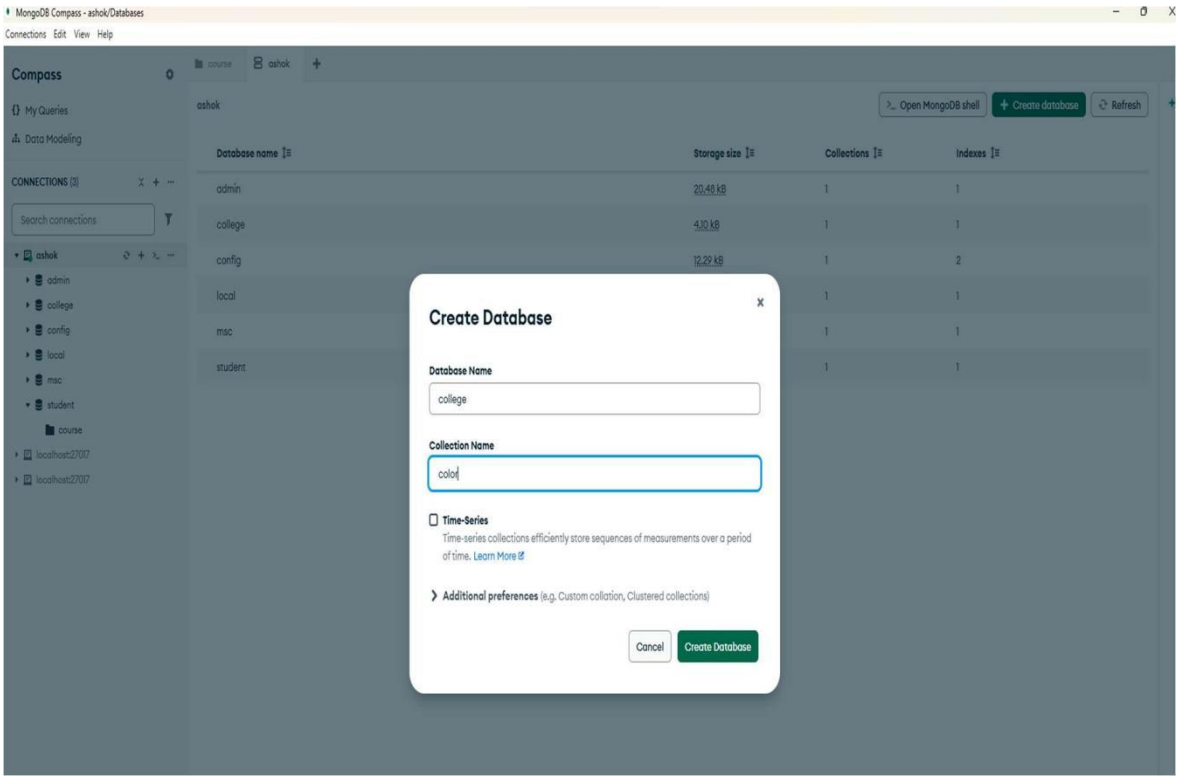
To introduce NoSQL document-oriented database concepts by creating collections and inserting semi-structured BSON documents using MongoDB.

PROGRAM:

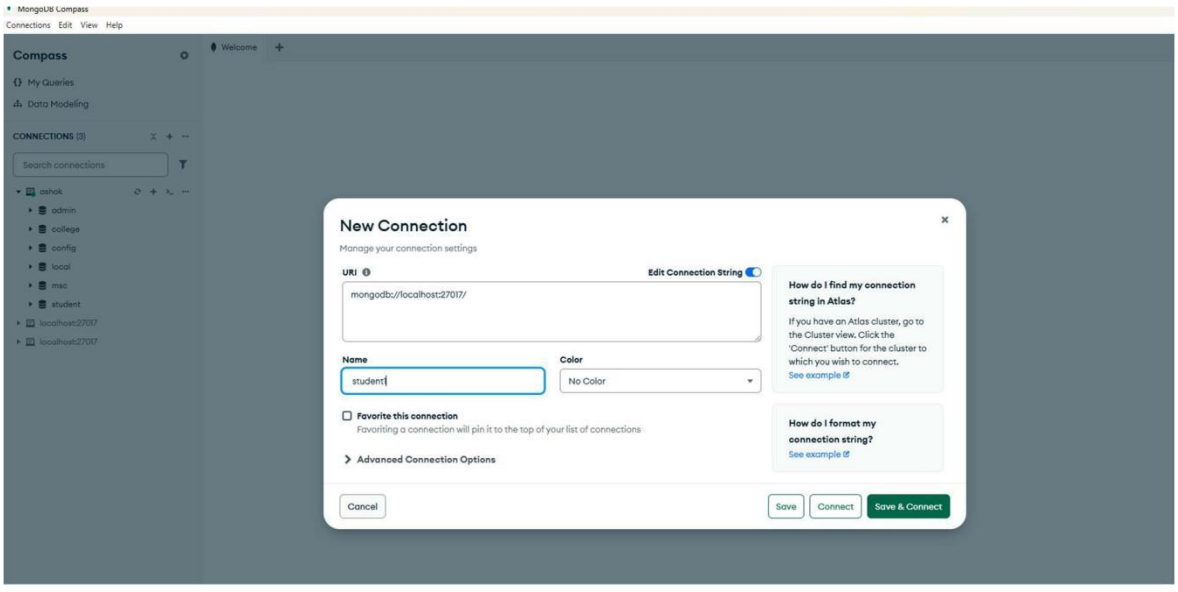
Step 1: Start MongoDB



Step 2: Select (or Create) a Database



Step 3: Choose (or Create) a Collection



Step 4: Insert a Single Document

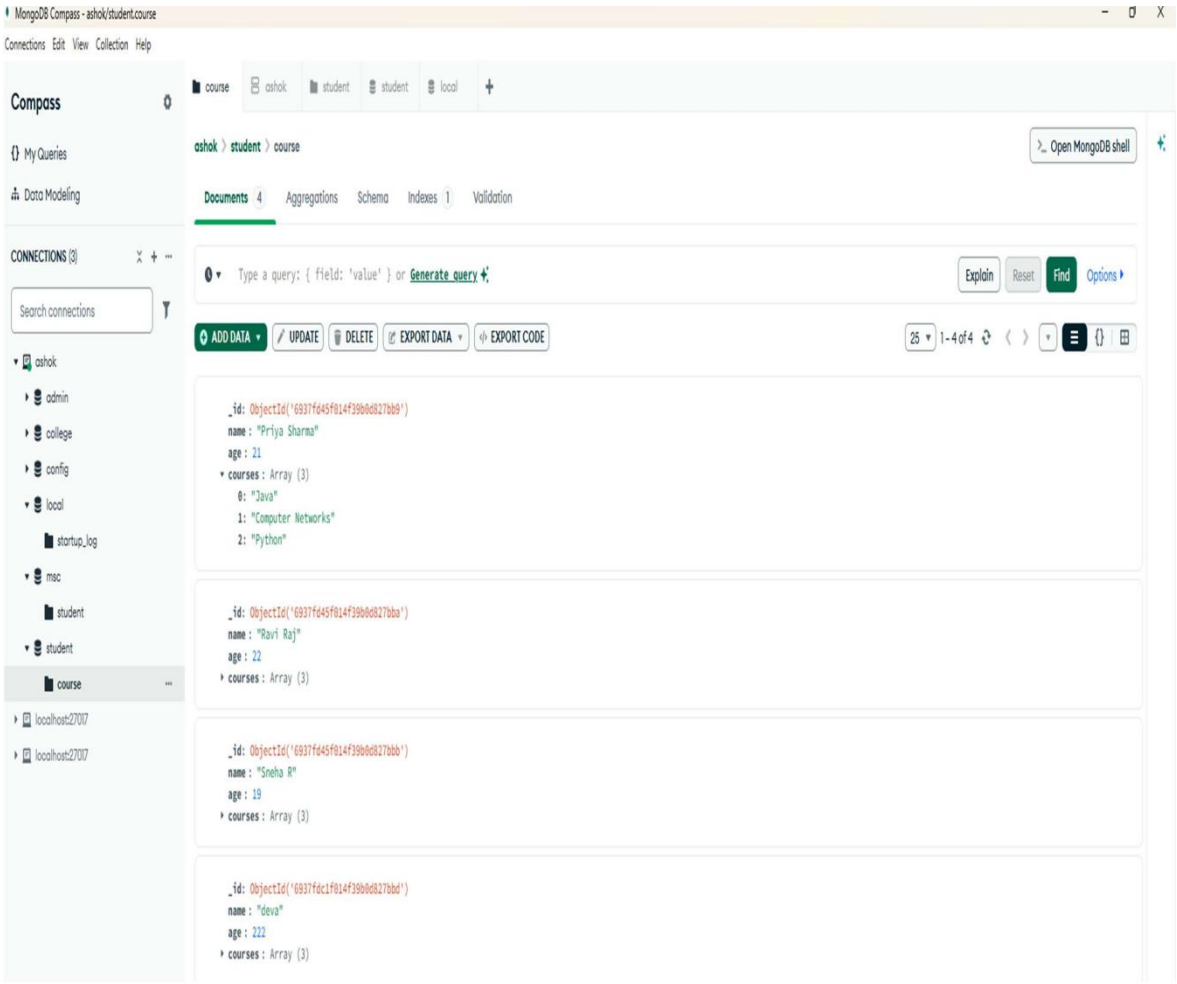
ADD data insert document open window

```
b.students.insertOn
 e({ name: "Ravi",
 age: 21,
 course: "Computer Science"
})
```

Step 5: Insert Multiple Documents

```
db.student1.insertMany([
 {
  name:
  "Ravi", age:
  21,
  course: "Computer Science"
 },
 {
  name:
  "Priya",
  age: 22,
  course: "Information Technology"
 },
 {
  name:
  "Arun", age:
  20,
  course: "Data Science"
```

```
  }
])
```


MongoDB Compass - ashok/student.course
Connections Edit View Collection Help
Compass
My Queries
Data Modeling
CONNECTIONS (3)
Search connections
ashok
admin
college
config
local
startup_log
msc
student
student
course
localhost:27017
localhost:27017
course
ashok
student
student
local
ashok 〉 student 〉 course
Open MongoDB shell
Documents 4
Aggregations
Schema
Indexes 1
Validation
Type a query: { field: 'value' } or Generate query
Explain
Reset
Find
Options
ADD DATA
UPDATE
DELETE
EXPORT DATA
EXPORT CODE
25
1-4 of 4
_id: ObjectId('6937fd45f814f39b0d827bb9')
name : "Priya Sharma"
age : 21
courses : Array (3)
0: "Java"
1: "Computer Networks"
2: "Python"
_id: ObjectId('6937fd45f814f39b0d827bba')
name : "Ravi Raj"
age : 22
courses : Array (3)
_id: ObjectId('6937fd45f814f39b0d827bbb')
name : "Sneha R"
age : 19
courses : Array (3)
_id: ObjectId('6937fdc1f814f39b0d827bbd')
name : "deva"
age : 222
courses : Array (3)

RESULT:

Hence the above program is verified and executed successfully.