MUTHOOT INSTITUTE OF TECHNOLOGY AND SCIENCE

VARIKOLI, ERNAKULAM-682308

DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

CS333 Application Software Development Lab

(AY:2019-2020)

LAB MANUAL

Version number:1.1

Prepared by:

Dr CK Raju

Professor, Department of CSE

VISION

To be recognized as a socially accountable thought leader in applications of computational technologies and a centre of excellence in solving complex and cross-disciplinary problems.

MISSION

- Develop pedagogical practices that help equip students with the ability to self-learn and reason.

- Improve ability to communicate on complex engineering activities and interpersonal skills by developing proper training methods and professional networking.

- Improve problem solving ability by promoting computational thinking and working on cross-disciplinary projects.

- Create and sustain networks in professional, academic and startup environments to stay updated with changes in the field of Computer Science and Engineering.

- Instill social commitment in students and faculty by engaging in spreading computer literacy and other socially relevant services.

- Promote research towards developing insight into complex problems.

Course outcome	
CS333.1	Students should be able to perform basic DDL, DCL, DML, DQL and TCL commands
CS333.2	Students should be able to realise built-in, aggregate functions and conditional queries
CS333.3	Students should be able to implement views, triggers, stored procedures and functions
CS333.4	Students should be able to design and implement database project using forms menus and reports

Experiment	**Course outcome**
1. Creation of Database using DDL	CS333.1
2. Performing DML Comands	CS333.1
3. Retrieving information using DQL	CS333.1
4. Creating relationship within databases	CS333.1
5. Creating a database to set various constraints	CS333.1
6. Practise of TCL commands like Rollback, Commit, Savepoint	CS333.1
7. Performing DCL commands – granting/revoking privileges	CS333.1
8.Creation of Views and Assertions	CS333.3
9. Implementation of built-in functions in RDBMS	CS333.2
10. Implementation of various aggregate functions in SQL	CS333.2
11.Implementation of Order by, Group by and Having clause	CS333.2
12. Implementation of set operations, nested / join queries	CS333.2
13. PL/SQL – Creation of Stored Procedures and Functions	CS333.3
14. Creation of Database Triggers and Cursors	CS333.3
15. Practise of front-end tools with report generation	CS333.4
16. Creating Forms and Menus	CS333.4
17. Mini Project	CS333.4

Contents

1 Creation of Database using DDL

```

-- DDL Commands - Creating Database, Tables

-- Create new database

CREATE DATABASE institution;

USE institution;

-- Relations

-- Consider the following relations:

-- Student (snum: integer, sname: string, major: string, level: string, age: integer)

-- Class (name: string, meets at: string, room: string, fid: integer)

-- Enrolled (snum: integer, cname: string)

-- Faculty (fid: integer, fname: string, deptid: integer)

-- The meaning of these relations is straightforward; for example, Enrolled has one record per student-class pair such that the student is enrolled in the class. Level is a two character code with 4 different values (example: Junior: JR etc)


-- Creating Student table

CREATE TABLE Student (
    snum INT,
    sname VARCHAR(15),
    major VARCHAR(15),
    slevel VARCHAR(15),
    age INT,
    PRIMARY KEY (snum)
);

DESC Student;


-- Creating Faculty table

```

```sql
CREATE TABLE Faculty (
    fid INT,
    fname VARCHAR(15),
    deptid INT,
    PRIMARY KEY (fid)
);

DESC Faculty;


-- Creating Class table

CREATE TABLE Class (
    cname VARCHAR(15),
    meets_at VARCHAR(15),
    room VARCHAR(15),
    fid INT,
    PRIMARY KEY (cname),
    FOREIGN KEY (fid) REFERENCES Faculty (fid)
);

DESC Class;


-- Creating Enrolled table

CREATE TABLE Enrolled (
    snum INT,
    cname VARCHAR(15),
    FOREIGN KEY (snum) REFERENCES Student (snum),
    FOREIGN KEY (cname) REFERENCES Class (cname)
);

DESC Enrolled;
```

2 Performing DML commands like Insertion, Deletion, Modifying, Altering and Updating records based on conditions

```sql
--DML Commands : Insertion, Deletion, Modifying, Altering and Updating Records based on conditions


-- Filling Student table with values

INSERT INTO Student VALUES
        (2, 'Rishabh', 'EC', 'SR', 20),
        (3, 'Aditya', 'CS', 'SR', 20),
        (4, 'Siddharth', 'CS', 'JR', 18),
        (5, 'Amit', 'CS', 'SSR', 22),
        (6, 'Gargi', 'EC', 'SJR', 17)
    ;


-- Filling Faculty table with values

INSERT INTO Faculty VALUES
        (55, 'Prof. Venkatasen', 7),
        (66, 'Prof. Prasad', 7),
        (77, 'Prof. Anupama', 8),
        (88, 'Prof. Poornima', 9),
        (99, 'Prof. Anil', 9)
    ;



-- Filling Class table with values

INSERT INTO Class VALUES
        ('CS1', '12 HR', 'R128', 55),
        ('CS2', '11 HR', 'R138', 66),
        ('CS3', '12 HR', 'R148', 77),
        ('CS4', '11 HR', 'R158', 88),
        ('CS5', '12 HR', 'R168', 99),
        ('CS6', '1 HR', 'R138', 55),
        ('CS7', '2 HR', 'R148', 55),
        ('CS8', '3 HR', 'R158', 55),
        ('CS9', '4 HR', 'R168', 55)
    ;



```

```

-- Filling Enrolled table with values

INSERT INTO Enrolled VALUES
          (2, 'CS1'),
          (3, 'CS2'),
          (4, 'CS1'),
          (5, 'CS4'),
          (6, 'CS5');
```

3 Retrieving information using DQL

```sql


SELECT * FROM Student;

SELECT * FROM Faculty;

SELECT * FROM Class;

-- Find the names of all Juniors (level = JR) who are enrolled in a class taught by Prof. Venkatesan

SELECT DISTINCT S.sname
FROM Student S, Enrolled E, Class C, Faculty F
WHERE
    S.snum = E.snum AND
    E.cname = C.cname AND
    C.fid = F.fid AND
    F.fname LIKE '%Venkatesan%' AND
    S.slevel = 'JR'
;

-- Find the names of all classes that either meet in room R128 or have five or more Students enrolled.

SELECT DISTINCT C.cname
FROM Class C
WHERE
    C.room = 'R128' OR
    C.cname IN (
            SELECT E.cname
            FROM Enrolled E
            GROUP BY E.cname
            HAVING COUNT(*) >= 5
        )
;

-- Find the names of all students who are enrolled in two classes that meet at the same time.

SELECT DISTINCT S.sname
FROM Student S
WHERE S.snum IN (
            SELECT E1.snum
            FROM Enrolled E1, Enrolled E2, Class C1, Class C2
            WHERE
                E1.snum = E2.snum AND
                E1.cname <> E2.cname AND
                E1.cname = C1.cname AND
                E2.cname = C2.cname AND
                C1.meets_at = C2.meets_at
        )
;

-- Find the names of faculty members who teach in every room in which some class is taught.

SELECT DISTINCT F.fname
FROM Faculty F
WHERE NOT EXISTS (
            SELECT *
            FROM Class C
            WHERE (C.room) NOT IN (
                        SELECT C1.room
                        FROM Class C1
                        WHERE C1.fid = F.fid
                    )
        )
;

-- Find the names of faculty members for whom the combined enrollment of the courses that they teach is less
-- than five.

SELECT DISTINCT F.fname
FROM Faculty F
WHERE 5 > (
        SELECT COUNT(E.snum)
        FROM Class C, Enrolled E
        WHERE
            C.cname = E.cname AND
            C.fid = F.fid
    )
;
```

4 Creating relationship within databases

```
-- Relations

-- The following relations keep track of airline flight information:

-- Flights (no: integer, from: string, to: string, distance: integer, Departs: time, arrives: time, price: real)

-- Aircraft (aid: integer, aname: string, cruisingrange: integer)

-- Employees (eid: integer, ename: string, salary: integer)

-- Note that the Employees relation describes pilots and other kinds of employees as well; Every pilot is certified for some aircraft, and only pilots are certified to fly.

-- Creating Flights table

CREATE TABLE Flights (
    flno INT,
    ffrom VARCHAR(15),
    tto VARCHAR(15),
    distance INTEGER,
    departs TIMESTAMP,
    arrives TIMESTAMP,
    price REAL,
    PRIMARY KEY (flno)
);

DESC Flights;


-- Creating Aircraft table

CREATE TABLE Aircraft (
    aid INT,
    aname VARCHAR(15),
    cruisingrange INT,
    PRIMARY KEY (aid)
);

DESC Aircraft;


-- Creating Employees table

```

```sql
CREATE TABLE Employees (
    eid INT,
    ename VARCHAR(15),
    salary REAL,
    PRIMARY KEY (eid)
);

DESC Employees;

-- Filling Aircraft table with values

INSERT INTO Aircraft VALUES
    (1, 'Airbus', 2000),
    (2, 'Boeing', 700),
    (3, 'Jet', 550),
    (4, 'Dreamliner', 5000),
    (5, 'Boeing', 4500),
    (6, 'Airbus', 2200)
;

SELECT * FROM Aircraft;


-- Filling Employees table with values

INSERT INTO Employees VALUES
    (162, 'Andrew', 50000),
    (183, 'Laeddis', 60000),
    (192, 'Rachel', 70000),
    (204, 'Solando', 82000),
    (300, 'Tony', 5000)
;

SELECT * FROM Employees;

-- Filling Flights table with values

INSERT INTO Flights VALUES
    (1, 'Bengaluru', 'New Delhi', 500, TIMESTAMP '2014-11-4 09:24:26', TIMESTAMP '2014-11-4 09:24:26', 5000),
    (2, 'Bengaluru', 'Chennai', 300, TIMESTAMP '2014-11-4 09:24:26', TIMESTAMP '2014-11-4 09:24:26', 3000),
    (3, 'Trivandrum', 'New Delhi', 800, TIMESTAMP '2014-11-4 09:24:26', TIMESTAMP '2014-11-4 09:24:26', 6000),
    (4, 'Bengaluru', 'Frankfurt', 10000, TIMESTAMP '2014-11-4 09:24:26', TIMESTAMP '2014-11-4 09:24:26', 50000),
    (5, 'Kolkata', 'New Delhi', 2400, TIMESTAMP '2014-11-4 09:24:26', TIMESTAMP '2014-11-4 09:24:26', 9000),
    (6, 'Bengaluru', 'Frankfurt', 8000, TIMESTAMP '2014-11-4
    09:24:26', TIMESTAMP '2014-11-4 09:24:26', 40000)
;

SELECT * FROM Flights;

-- Relationship

-- Create relationship Certified between Aircraft and Employees, to realise which employees are certified against aircrafts

-- Certified (eid: integer, aid: integer)


-- Creating Certified table

CREATE TABLE Certified (
     eid INT,
     aid INT,
     PRIMARY KEY (eid, aid),
     FOREIGN KEY (eid) REFERENCES Employees (eid),
     FOREIGN KEY (aid) REFERENCES Aircraft (aid)
);

DESC Certified;

-- Filling Certified table with values

INSERT INTO Certified VALUES
     (162, 2),
     (162, 4),
     (162, 5),
     (162, 6),
     (183, 1),
     (183, 3),
     (183, 5),
     (192, 2),
     (192, 3),
     (192, 5),
     (192, 6),
     (204, 6),
     (204, 1),
     (204, 3),
     (300, 3)
;

SELECT * FROM Certified;


-- Queries
```

```sql

-- Write each of the following queries in SQL:


-- Find the names of Aircraft such that all pilots certified to operate them have salaries more than Rs.80,000.

SELECT DISTINCT A.aname
FROM Aircraft A
WHERE A.aid IN (
                SELECT C.aid
                FROM Certified C, Employees E
                WHERE
                     C.eid = E.eid AND
                     NOT EXISTS (
                                SELECT *
                                FROM Employees E1
                                WHERE
                                     E1.eid = E.eid AND
                                     E1.salary <= 80000
                            )
            )
;

-- For each pilot who is certified for more than three aircrafts, find the eid and the maximum cruisingrange of
-- the aircraft for which she or he is certified.

SELECT C.eid, MAX(A.cruisingrange)
FROM Certified C, Aircraft A
WHERE A.aid = C.aid
GROUP BY C.eid
HAVING COUNT(*) > 3

;

-- Find the names of pilots whose salary is less than the price of the cheapest route from Bengaluru to Frankfurt.

SELECT E.ename
FROM Employees E
WHERE E.salary < (
                SELECT MIN(F.price)
                FROM Flights F
                WHERE
                     F.ffrom = 'Bengaluru' AND
                     F.tto = 'Frankfurt'
            )
;
```

```sql

-- For all aircraft with cruisingrange over 1000 Kms,. Find the name of the aircraft and the average salary of all
-- pilots certified for this aircraft.

SELECT A.aname, AVG (E.SALARY)
FROM Aircraft A, Certified C, Employees E
WHERE
    A.aid = C.aid AND
    C.eid = E.eid AND
    A.cruisingrange > 1000

GROUP BY A.aid , A.aname

;

-- Find the names of pilots certified for some Boeing aircraft.

SELECT DISTINCT E.ename
FROM Employees E, Certified C, Aircraft A
WHERE
    E.eid = C.eid AND
    A.aid = C.aid AND
    A.aname = 'Boeing'
;

-- Find the aids of all aircraft that can be used on routes from Bengaluru to New Delhi.

SELECT DISTINCT A.aid
FROM Aircraft A
WHERE A.cruisingrange > (
                SELECT MIN(F.distance)
                FROM Flights F
                WHERE
                    F.ffrom = 'Bengaluru' AND
                    F.tto = 'New Delhi'
            )
;
```

5 Creating a database to set various constraints

```sql

-- SQL | Constraints

-- Constraints are the rules that we can apply on the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints.

-- The available constraints in SQL are:

--     NOT NULL: This constraint tells that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more.
--     UNIQUE: This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated.
--     PRIMARY KEY: A primary key is a field which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as primary key.
--     FOREIGN KEY: A Foreign key is a field which can uniquely identify each row in a another table. And this constraint is used to specify a field as Foreign key.
--     CHECK: This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition.
--     DEFAULT: This constraint specifies a default value for the column when no value is specified by the user.

CREATE TABLE sample_table
(
column1 data_type(size) constraint_name,
column2 data_type(size) constraint_name,
column3 data_type(size) constraint_name,
....
);

-- sample_table: Name of the table to be created.
-- data_type: Type of data that can be stored in the field.
-- constraint_name: Name of the constraint. for example- NOT NULL , UNIQUE, PRIMARY KEY etc.

-- NOT NULL
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
ADDRESS varchar(20)
);

-- UNIQUE
CREATE TABLE Student
(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20)
);

-- PRIMARY KEY
CREATE TABLE Student
(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20),
PRIMARY KEY(ID)
);

-- FOREIGN KEY
CREATE TABLE Orders
(
O_ID int NOT NULL,
ORDER_NO int NOT NULL,
C_ID int,
PRIMARY KEY (O_ID),
FOREIGN KEY (C_ID) REFERENCES Customers(C_ID)
);

-- CHECK
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int NOT NULL CHECK (AGE >= 18)
);

-- DEFAULT
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int DEFAULT 18
);
```

6 Practise of SQL TCL commands like Roll-back, Commit, Savepoint

```sql
-- A Transaction is a collection of statements between specific client and server. These transactions can be controlled efficiently by using MySQL TCL (Transaction Control Language). Transaction Control Statements are

--     Commit
--     Roll-back
--     Save points

CREATE TABLE test(test_id INT NOT NULL PRIMARY KEY) ENGINE=InnoDB;

START TRANSACTION;

INSERT INTO TEST VALUES(1);

SELECT * FROM TEST;

SAVEPOINT TRAN2;

INSERT INTO TEST VALUES(2);

SELECT * FROM TEST;

ROLLBACK TO TRAN2;

SELECT * FROM TEST;

ROLLBACK;

SELECT * FROM TEST;

-- Test working of TCL commands on different tables and observe changes.
```

7 Practise of SQL DCL commands for granting and revoking user privileges

```
-- DCL- DCL is the abstract of Data Control Language. Data
    Control Language includes commands such as GRANT, and
    concerns with rights, permissions and other controls of the
    database system. DCL is used to grant / revoke permissions on
     databases and their contents.

-- GRANT :It provides the  user's access privileges to the
    database. In the MySQL database offers both the administrator
     and user a great extent of the control options. By the
    administration side of the process includes the possibility
    for the administrators to control certain user privileges
    over the MySQL server by restricting their access to an
    entire the database or ust limiting permissions for a
    specific table.

CREATE USER 'arjun'@'localhost' IDENTIFIED BY 'mypass';

GRANT ALL ON db1.* TO 'arjun'@'localhost';

GRANT SELECT ON child TO 'arjun'@'localhost';

GRANT USAGE ON *.* TO 'arjun'@'localhost' WITH
    MAX_QUERIES_PER_HOUR 90;

-- REVOKE : The REVOKE statement enables system administrators
    and to revoke the privileges from MySQL accounts.

-- Syntax : REVOKE priv_type [(column_list)] [, priv_type [(
    column_list)]] ... ON [object_type] priv_level FROM user [,
    user] ...

REVOKE INSERT ON *.* FROM 'arjun'@'localhost';
```

8 Creation of Views and Assertions

```
-- VIEWS
--    Views are virtual tables; they do not contain the data that is returned. The data is stored in the tables referenced in the SELECT statement.
--    Views improve security of the database by showing only intended data to authorized users. They hide sensitive data.
--    Views make life easy as you do not have write complex queries time and again.
--    It's possible to use INSERT, UPDATE  and DELETE on a VIEW. These operations will change the underlying tables of the VIEW.  The only consideration is that VIEW should contain all NOT NULL columns of the tables it references.
--Ideally, you should not use VIEWS for updating.

-- To create a new view in MySQL, you use the CREATE VIEW statement. The syntax of creating a view in MySQL is as follows:

CREATE      [ALGORITHM = {MERGE  | TEMPTABLE | UNDEFINED}] VIEW view_name [(column_list)] AS select-statement;

-- Example

CREATE VIEW SalePerOrder AS
    SELECT
        orderNumber, SUM(quantityOrdered * priceEach) total
    FROM
        orderDetails
    GROUP by orderNumber
    ORDER BY total DESC;

-- Examine
SHOW TABLES;

SHOW FULL TABLES;

-- Observe the differences in both the cases here.
```

9 Implementation of built-in functions in RDBMS

```sql


-- Verify and explain each of the built-in functions listed herein.
-- Search another 10 such built-in functions not listed here

SELECT UCASE(NAME) FROM Students;

SELECT MID(column_name,start,length) AS some_name FROM table_name;
-- specifying length is optional here, and start signifies start position ( starting from 1 )

SELECT LENGTH(NAME) FROM Students;

SELECT ROUND(MARKS,0) FROM table_name;

SELECT NAME, FORMAT(Now(),'YYYY-MM-DD') AS Date FROM Students;

SELECT SUM(ISNULL(Salary, 10000) AS Salary FROM Employee;

SELECT ABS(-243.5);

SELECT COS(30);

SELECT GREATEST(30, 2, 36, 81, 125);

SELECT LOG(2);

SELECT MOD(18, 4);

SELECT TRUNCATE(7.53635, 2);

SELECT CURRENT_DATE();

SELECT DAYOFMONTH('2018-07-16');

SELECT HOUR("2018-07-16 09:34:00");

SELECT MAKEDATE(2009, 138);

SELECT CHARACTER_LENGTH('geeks for geeks');

SELECT CONCAT_WS('_', 'geeks', 'for', 'geeks');

SELECT LOWER('GEEKSFORGEEKS.ORG');

SELECT STRCMP('google.com', 'geeksforgeeks.com');

SELECT SUBSTRING_INDEX('www.geeksforgeeks.org', '.', 1);
```

10 Implementation of various aggregate functions in SQL

```sql
-- Aggregate functions:
-- These functions are used to do operations from the values of the column and a single value is returned.

--     AVG()
--     COUNT()
--     FIRST()
--     LAST()
--     MAX()
--     MIN()
--     SUM()


-- Examples:
SELECT AVG(MARKS) AS AvgMarks FROM Students;

SELECT COUNT(column_name) FROM table_name;

SELECT COUNT(DISTINCT AGE) AS NumStudents FROM Students;

SELECT FIRST(column_name) FROM table_name;

SELECT LAST(column_name) FROM table_name;

SELECT MAX(MARKS) AS MaxMarks FROM Students;

SELECT MIN(AGE) AS MinAge FROM Students;

SELECT SUM(MARKS) AS TotalMarks FROM Students;
```

11 Implementation of Order By, Group By and Having Clause

```sql

-- Relations

-- The following tables are maintained by a book dealer.

-- AUTHOR (author-id:int, name:string, city:string, country:string)

-- PUBLISHER (publisher-id:int, name:string, city:string, country:string)

-- CATALOG (book-id:int, title:string, author-id:int, publisher-id:int, category-id:int, year:int, price:int)

-- CATEGORY (category-id:int, description:string)

-- ORDER-DETAILS (order-no:int, book-id:int, quantity:int)

-- Creating Author table

CREATE TABLE Author (
    authorid INT,
    name VARCHAR(30),
    city VARCHAR(30),
    country VARCHAR(30),
    PRIMARY KEY (authorid)
);

DESC Author;


-- Creating Publisher table

CREATE TABLE Publisher (
    publisherid INT,
    name VARCHAR(30),
    city VARCHAR(30),
    country VARCHAR(30),
    PRIMARY KEY (publisherid)
);

DESC Publisher;


-- Creating BookCategory table
```

```

CREATE TABLE BookCategory (
    categoryid INT ,
    description VARCHAR(30),
    PRIMARY KEY (categoryid)
);

DESC BookCategory;


-- Creating Catalog table

CREATE TABLE Catalog (
    bookid INT,
    title VARCHAR(30),
    authorid INT,
    publisherid INT,
    categoryid INT,
    yearofpublish INT,
    price INT,
    PRIMARY KEY (bookid),
    FOREIGN KEY (authorid) REFERENCES Author (authorid),
    FOREIGN KEY (publisherid) REFERENCES Publisher (publisherid),
    FOREIGN KEY (categoryid) REFERENCES BookCategory (categoryid)
);

DESC Catalog;


-- Creating OrderDetails table

CREATE TABLE OrderDetails (
    orderno INT,
    bookid INT,
    quantity INT,
    PRIMARY KEY (orderno, bookid),
    FOREIGN KEY (bookid) REFERENCES Catalog(bookid)
);

DESC OrderDetails;

-- Filling Author table with values

INSERT INTO Author VALUES
    (1, 'NAVATHE', 'ARLINGTON', 'USA'),
    (2, 'RAGHU RAMAKRISHNAN', 'CALIFORNIA', 'USA'),
    (3, 'DHAMDHERE', 'MUMBAI', 'INDIA'),
    (4, 'BJARNE', 'NEW JERSY', 'USA'),
    (5, 'TANENBAUM', 'AMSTERDAM','NETHERLAND')
```

```sql
;

SELECT * FROM Author;


-- Filling Publisher table with values

INSERT INTO Publisher VALUES
     (1, 'JOHN WILEY', 'NEW YORK', 'USA'),
     (2, 'PEARSON', 'BANGALORE', 'INDIA'),
     (3, 'O REILLY', 'NEW JERSY', 'USA'),
     (4, 'TMH', 'CALCUTTA', 'INDIA'),
     (5, 'JOHN WILEY', 'NEW DELHI', 'INDIA')
;

SELECT * FROM Publisher;


-- Filling BookCategory table with values

INSERT INTO BookCategory VALUES
     (1, 'DATABASE MANAGEMENT'),
     (2, 'OPERATING SYSTEMS'),
     (3, 'C++'),
     (4, 'COMPUTER NETWORKS'),
     (5, 'C')
;

SELECT * FROM BookCategory;


-- Filling Catalog table with values

INSERT INTO Catalog VALUES
     (1, 'FUNDAMENTALS OF DBMS', 1, 2, 1, 2004, 500),
     (2, 'PRINCIPLES OF DBMS', 2, 1, 1, 2004, 400),
     (3, 'OPERATING SYSTEMS', 3, 4, 2, 2004, 200),
     (4, 'C++ BIBLE', 4, 5, 3, 2003, 500),
     (5, 'COMPUTER NETWORKS', 5, 3, 4, 2002, 250),
     (6, 'FUNDAMENTALS OF C', 1, 2, 5, 2004, 700),
     (7, 'OPERATING SYSTEMS 2', 3, 2, 2, 2001, 600)
;

SELECT * FROM Catalog;


-- Filling OrderDetails table with values

INSERT INTO OrderDetails VALUES
```

```sql
    (1, 1, 1),
    (2, 2, 1),
    (3, 3, 1),
    (4, 4, 1),
    (5, 5, 1),
    (6, 6, 7),
    (7, 7, 9)
;

SELECT * FROM OrderDetails;


-- Queries

-- Give the details of the authors who have 2 or more books in the catalog and the price of the books is greater
-- than the average price of the books in the catalog and the year of publication is after 2000.

SELECT *
FROM Author A
WHERE EXISTS (
        SELECT A1.authorid, COUNT(A1.authorid)
        FROM Author A1, Catalog C
        WHERE
            A1.authorid = C.authorid AND
            A.authorid = A1.authorid AND
            C.yearofpublish > 2000 AND
            C.price > (
                    SELECT AVG(price)
                    FROM Catalog
                )
        GROUP BY A1.authorid
        HAVING COUNT(A1.authorid) >= 2
    )
;

-- Find the author of the book which has maximum sales.

SELECT DISTINCT A.NAME
FROM Author A, Catalog C, OrderDetails ODM
WHERE
    A.authorid = C.authorid AND
    ODM.bookid = C.bookid AND
    EXISTS (
            SELECT OD.bookid, SUM(OD.quantity)
            FROM OrderDetails OD
            WHERE OD.bookid = ODM.bookid
            GROUP BY bookid
            HAVING SUM(OD.quantity) >= ALL (
                            SELECT SUM(quantity)
                            FROM OrderDetails
                            GROUP BY bookid
                            )
        )
;


-- Demonstrate how you increase the price of books published by a specific publisher by 10%.

UPDATE  Catalog
SET price = (1.1) * price
WHERE authorid = (
            SELECT authorid
            FROM Author
            WHERE name = 'NAVATHE'
        )
;
```

12 Implementation of set operations, nested queries and Join queries

```

-- SET Operations

-- UNION Operator

-- UNION operator allows you to combine two or more result sets of queries into a single result set. The following illustrates the syntax of the UNION operator:

DROP TABLE IF EXISTS t1;
DROP TABLE IF EXISTS t2;

CREATE TABLE t1 (
    id INT PRIMARY KEY
);

CREATE TABLE t2 (
    id INT PRIMARY KEY
);

INSERT INTO t1 VALUES (1),(2),(3);
INSERT INTO t2 VALUES (2),(3),(4);


SELECT id FROM t1 UNION SELECT id FROM t2;

-- INTERSECT Operator

-- MySQL does not support the INTERSECT operator. However, you can simulate the INTERSECT operator.

CREATE TABLE t1 ( id INT PRIMARY KEY );

CREATE TABLE t2 LIKE t1;

INSERT INTO t1(id) VALUES(1),(2),(3);

INSERT INTO t2(id) VALUES(2),(3),(4);

-- INTERSECT simulation
SELECT DISTINCT    id   FROM t1    INNER JOIN t2 USING(id);


-- MINUS Simulation

```

```sql
 -- MySQL does not support MINUS operator. However, you can use the MySQL join to simulate it.

CREATE TABLE t1 (     id INT PRIMARY KEY);

CREATE TABLE t2 (     id INT PRIMARY KEY);

INSERT INTO t1 VALUES (1),(2),(3);
INSERT INTO t2 VALUES (2),(3),(4);


-- MINUS Simulation
SELECT   column_list   FROM    table_1   LEFT JOIN table_2 ON join_predicate WHERE  table_2.id IS NULL;


-- Consider the following database of student enrollment in courses & books adopted for each course.

-- STUDENT (regno: string, name: string, major: string, bdate: date)

-- COURSE (course #:int, cname:string, dept:string)

-- ENROLL ( regno:string, course#:int, sem:int, marks:int)

-- BOOK_ADOPTION (course# :int, sem:int, book-ISBN:int)

-- TEXT (book-ISBN:int, book-title:string, publisher:string, author:string)

-- Creating Student table

CREATE TABLE Student (
     regno VARCHAR(30),
     sname VARCHAR(30),
     major VARCHAR(30),
     bdate DATE,
    PRIMARY KEY (regno)
);

DESC Student;


-- Creating Course table

CREATE TABLE Course (
     course INT,
     cname VARCHAR(30),
    dept VARCHAR(30),
    PRIMARY KEY (course)
);

DESC Course;


-- Creating Enroll table

CREATE TABLE Enroll (
    regno VARCHAR(30),
    course INT,
    sem INT,
    marks INT,
    PRIMARY KEY (regno, course, sem),
    FOREIGN KEY (regno) REFERENCES Student(regno),
    FOREIGN KEY (course) REFERENCES Course(course));

DESC Enroll;


-- Creating Text table

CREATE TABLE Text (
    bookisbn INT,
    booktitle VARCHAR(30),
    publisher VARCHAR(30),
    author VARCHAR(30),
    PRIMARY KEY (bookisbn)
);

DESC Text;


-- Creating BookAdoption table

CREATE TABLE BookAdoption (
    course INT,
    sem INT,
    bookisbn INT,
    PRIMARY KEY (course, sem, bookisbn),
    FOREIGN KEY (course) REFERENCES Course (course),
    FOREIGN KEY (bookisbn) REFERENCES Text (bookisbn)
);

DESC BookAdoption;

-- Filling Student table with values

```

```
INSERT INTO Student VALUES
    ('1DS16CS735', 'Rishabh', 'DBMS', '1994-06-24'),
    ('1DS16CS747', 'Siddharth', 'ADA', '1993-11-9'),
    ('1DS16CS701', 'Aditya', 'GTC', '1994-04-28'),
    ('1DS16CS703', 'Amit', 'SE', '1993-10-7'),
    ('1DS16CS730', 'Gargi', 'DS', '1993-09-12')
;

SELECT * FROM Student;


-- Filling Course table with values

INSERT INTO Course VALUES
    (1, 'DBMS', 'CS'),
    (2, 'ADA', 'CS'),
    (3, 'GTC', 'TC'),
    (4, 'SE', 'EE'),
    (5, 'DS', 'EC'),
    (6, 'DS', 'CS')
;

SELECT * FROM Course;


-- Filling Text table with values

INSERT INTO Text VALUES
    (1, 'FUNDAMENTALS OF DBMS', 'PEARSON', 'RAMEZ ELMASRI'),
    (2, 'DESGIN OF ALGORITHMS','UNIVERSITY PRESS', 'SAHNI'),
    (3, 'GRAPH THEORY', 'PRISM', 'DSC'),
    (4, 'SE BIBLE','PEARSON', 'MEENA'),
    (5, 'POWER OF JAVA', 'SUN', 'JAMES GOSLING'),
    (6, 'POWER OF C', 'JOHN WILEY', 'DENNISRITCHIE'),
    (7, 'CORMEN ALGORITHMS', 'PEARSON', 'CLRS'),
    (8, 'INTRODUCTION TO C++', 'JOHN WILEY', 'HERBERT SHIELD'),
    (9, 'DATABASE', 'JOHN WILEY', 'SHAMKANT'),
    (10, 'ENGG MATH', 'PRISM', 'KSC')
;
SELECT * FROM Text;



-- Filling Enroll table with values

INSERT INTO Enroll VALUES
    ('1DS16CS735', 1, 5, 98),
    ('1DS16CS747', 2, 3, 88),
    ('1DS16CS701', 3, 5, 91),
```

```sql
    ('1DS16CS703', 4, 5, 76),
    ('1DS16CS730', 5, 5, 49)
;

SELECT * FROM Enroll;



-- Filling BookAdoption table with values

INSERT INTO BookAdoption VALUES
    (1, 5, 1),
    (1, 4, 4),
    (2, 3, 2),
    (3, 5, 3),
    (4, 5, 4),
    (5, 5, 5),
    (6, 4, 6),
    (6, 4, 7),
    (6, 4, 8)
;

SELECT * FROM BookAdoption;


-- Queries

-- Demonstrate how you add a new text book to the database and make this book be adopted by some
-- department.

INSERT INTO Text VALUES (11, 'DATABASE FUNDAMENTALS', 'TATA MCGRAW HILL', 'SCHIELD');
INSERT INTO BookAdoption VALUES (1, 3, 11);

-- Produce a list of text books (include Course #, Book-ISBN, Book-title) in the alphabetical order for courses
-- offered by the CS department that use more than two books.

SELECT C.course, T.bookisbn, T.booktitle
FROM Course C, BookAdoption BA, Text T
WHERE
    C.course = BA.course AND
    BA.bookisbn = T.bookisbn AND
    C.dept = 'CS' AND
    EXISTS (
            SELECT *
            FROM BookAdoption BA1
            WHERE BA1.course = C.course
            GROUP BY BA1.course
            HAVING COUNT(BA1.course) > 2
        )
ORDER BY T.booktitle

;

-- List any department that has all its adopted books published by a specific publisher.

SELECT C.dept, T.booktitle, T.publisher
FROM Course C, Text T, BookAdoption BA
WHERE
    C.course = BA.course AND
    T.bookisbn = BA.bookisbn AND
    T.publisher = 'PEARSON'AND
    T.publisher = ALL (
                SELECT T1.publisher
                FROM Course C1, BookAdoption BA1, Text T1
                WHERE
                    BA1.bookisbn = T1.bookisbn AND
                    BA1.course = C1.course AND
                    C.dept = C1.dept
            )
;
```

13 Implementation of various control structures using PL SQL - Creation of Procedures and Functions

```sql
-- MySQL stored procedure parameters

-- The parameters make the stored procedure more flexible and useful. In MySQL, a parameter has one of three modes: IN,OUT, or INOUT.

 --   IN is the default mode. When you define an IN parameter in a stored procedure, the calling program has to pass an argument to the stored procedure. In addition, the value of an IN parameter is protected. It means that even the value of the IN parameter is changed inside the stored procedure, its original value is retained after the stored procedure ends. In other words, the stored procedure only works on the copy of the IN parameter.

DELIMITER //
CREATE PROCEDURE GetOfficeByCountry(IN countryName VARCHAR(255))
 BEGIN
 SELECT *
 FROM offices
 WHERE country = countryName;
 END //
DELIMITER ;



CALL GetOfficeByCountry('USA');

--    the value of an OUT parameter can be changed inside the stored procedure and its new value is passed back to the calling program. Notice that the stored procedure cannot access the initial value of the OUT parameter when it starts.

DELIMITER $$
CREATE PROCEDURE CountOrderByStatus(
 IN orderStatus VARCHAR(25),
 OUT total INT)
BEGIN
 SELECT count(orderNumber)
 INTO total
 FROM orders
 WHERE status = orderStatus;
END$$
```

```
DELIMITER ;

CALL CountOrderByStatus('Shipped',@total);
SELECT @total;


--    an INOUT  parameter is a combination of IN  and OUT parameters. It means that the calling program may pass the argument, and the stored procedure can modify the INOUT parameter, and pass the new value back to the calling program.


DELIMITER $$
CREATE PROCEDURE set_counter(INOUT count INT(4),IN inc INT(4))
BEGIN
 SET count = count + inc;
END$$
DELIMITER ;

SET @counter = 1;
CALL set_counter(@counter,1);
CALL set_counter(@counter,1);
CALL set_counter(@counter,5);
SELECT @counter;

-- FUNCTION

-- A stored function is a special kind stored program that returns a single value. You use stored functions to encapsulate common formulas or business rules that are reusable among SQL statements or stored programs.

--  Different from a stored procedure, you can use a stored function in SQL statements wherever an expression is used. This helps improve the readability and maintainability of the procedural code.


--  Example

DELIMITER $$

CREATE FUNCTION CustomerLevel(p_creditLimit double) RETURNS VARCHAR(10)
    DETERMINISTIC
BEGIN
    DECLARE lvl varchar(10);

```

```sql
    IF p_creditLimit > 50000 THEN
  SET lvl = 'PLATINUM';
    ELSEIF (p_creditLimit <= 50000 AND p_creditLimit >= 10000) THEN
        SET lvl = 'GOLD';
    ELSEIF p_creditLimit < 10000 THEN
        SET lvl = 'SILVER';
    END IF;

 RETURN (lvl);
END

-- How to invoke

SELECT customerName, CustomerLevel(creditLimit) FROM customers ORDER BY customerName;
```

14 Creation of Database Triggers and Cursors

```
-- A MySQL trigger is a stored program (with queries) which is executed automatically to respond to a specific event such as insertion, updation or deletion occurring in a table.

-- There are 6 different types of triggers in MySQL:
-- Before UPDATE trigger
-- After UPDATE trigger
-- Before INSERT trigger
-- After INSERT trigger
-- Before DELETE trigger
-- After DELETE trigger

-- Example for before-update trigger
create table customer (acc_no integer primary key,
                              cust_name varchar(20),
                               avail_balance decimal);

create table mini_statement (acc_no integer,
                              avail_balance decimal,
                     foreign key(acc_no) references customer(acc_no) on delete cascade);

insert into customer values (1000, "Fanny", 7000);
insert into customer values (1001, "Peter", 12000);

-- Trigger definition
delimiter //
create trigger update_cus
     before update on customer
     for each row
     begin
     insert into mini_statement values (old.acc_no, old.avail_balance);
     end; //

-- making updates to activate trigger

delimiter;
update customer set avail_balance = avail_balance + 3000 where acc_no = 1001;
update customer set avail_balance = avail_balance + 3000 where acc_no = 1000;

-- verify whether trigger activated or not
select *from mini_statement;

```

```

-- example for AFTER-DELETE trigger
create table contacts (contact_id int (11) NOT NULL AUTO_INCREMENT,
                            last_name VARCHAR (30) NOT NULL, first_name VARCHAR (25),
                            birthday DATE, created_date DATE, created_by VARCHAR (30),
                            CONSTRAINT contacts_pk PRIMARY KEY (contact_id));
create table contacts_audit (contact_id integer, deleted_date date, deleted_by varchar(20));

-- Trigger defintion

delimiter //
create trigger contacts_after_delete
           after delete
           on contacts for each row
           begin

              DECLARE vUser varchar(50);

              -- Find username of person performing the DELETE into table
              SELECT USER() into vUser;

              -- Insert record into audit table
              INSERT into contacts_audit
              ( contact_id,
                deleted_date,
                deleted_by)
              VALUES
              ( OLD.contact_id,
                SYSDATE(),
                vUser );
           end; //

-- activating trigger
delimiter;
insert into contacts values (1, "Newton", "Isaac",
                             str_to_date ("19-08-1985", "%d-%m-%Y"),
                             str_to_date ("23-07-2018", "%d-%m-%Y"), "xyz");
delete from contacts where first_name="Isaac";
insert into contacts values (1, "Newton", "Isaac",
                             str_to_date ("19-08-1985", "%d-%m-%Y"),
                              str_to_date ("23-07-2018", "%d-%m-%Y
   "), "xyz");
delete from contacts where first_name="Isaac";

-- verify activation of trigger
select *from contacts_audit;
```

15 Practise various front-end tools with report generation

```
-- Self-learning exercise on report generation
```

16 Creating Forms and Menus

```
-- Self-learning exercise on creating forms and menus
-- Try with PHP
```

17 Mini Project - Application Development using Oracle or MySQL using Database Connectivity

```
-- Student group project
-- Select any topic for project from the list
--(a) Inventory Control System
--(b) Material Requirement Processing
--(c) Hospital Management System
--(d) Railway Reservation System
--(e) Personal Information System
--(f) Web-based User Identification System
--(g) Timetable Management System
--(h) Hotel Management System
```