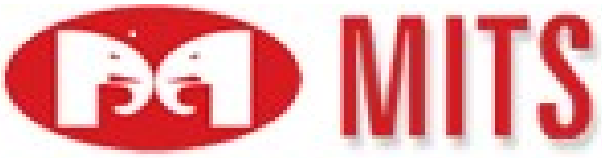

MUTHOOT INSTITUTE OF TECHNOLOGY AND SCIENCE

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

```
1
2   -- DDL Commands - Creating Database, Tables
3
4   -- Create new database
5
6   CREATE DATABASE institution;
7
8   USE institution;
9
10  -- Relations
11
12  -- Consider the following relations:
13
14  -- Student (snum: integer, sname: string, major: string, level:
        string, age: integer)
15
16  -- Class (name: string, meets at: string, room: string, fid:
        integer)
17
18  -- Enrolled (snum: integer, cname: string)
19
20  -- Faculty (fid: integer, fname: string, deptid: integer)
21
22  -- 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)
23
24
25  -- Creating Student table
26
27  CREATE TABLE Student (
28      snum INT,
29      sname VARCHAR(15),
30      major VARCHAR(15),
31      slevel VARCHAR(15),
32      age INT,
33      PRIMARY KEY (snum)
34  );
35
36  DESC Student;
37
38
39  -- Creating Faculty table
40
41  CREATE TABLE Faculty (
42      fid INT,
43      fname VARCHAR(15),
44      deptid INT,
45      PRIMARY KEY (fid)
46  );
47
48  DESC Faculty;
49
50
```

```
51 -- Creating Class table
52
53 CREATE TABLE Class (
54      cname VARCHAR(15),
55      meets_at VARCHAR(15),
56      room VARCHAR(15),
57      fid INT,
58      PRIMARY KEY (cname),
59      FOREIGN KEY (fid) REFERENCES Faculty (fid)
60 );
61
62 DESC Class;
63
64
65 -- Creating Enrolled table
66
67 CREATE TABLE Enrolled (
68      snum INT,
69      cname VARCHAR(15),
70      FOREIGN KEY (snum) REFERENCES Student (snum),
71      FOREIGN KEY (cname) REFERENCES Class (cname)
72 );
73
74 DESC Enrolled;
```

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

```
--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'),
```

```
51                  (6,  'CS5');
```

3 Retrieving information using DQL

```
1
2
3   SELECT * FROM Student;
4
5   SELECT * FROM Faculty;
6
7   SELECT * FROM Class;
8
9   -- Find the names of all Juniors (level = JR) who are enrolled in a
        class taught by Prof. Venkatesan
10
11  SELECT DISTINCT Student.sname
12  FROM Student, Enrolled, Class, Faculty
13  WHERE
14      Student.snum = Enrolled.snum AND
15      Enrolled.cname = Class.cname AND
16      Class.fid = Faculty.fid AND
17      Faculty.fname LIKE '%Venkatesan%' AND
18      Student.slevel = 'JR'
19  ;
20
21  -- Find the names of all classes that either meet in room R128 or
        have five or more Students enrolled.
22
23  SELECT DISTINCT Class.cname
24  FROM Class
25  WHERE
26      Class.room = 'R128' OR
27      Class.cname IN (
28              SELECT Enrolled.cname
29              FROM Enrolled
30              GROUP BY Enrolled.cname
31              HAVING COUNT(*) >= 5
32          )
33  ;
34
35
36
37  -- Find the names of faculty members who teach in every room in
        which some class is taught.
38
39  SELECT DISTINCT Faculty.fname
40  FROM Faculty
41  WHERE NOT EXISTS (
42              SELECT *
43              FROM Class
44              WHERE (Class.room) NOT IN (
45                          SELECT Class.room
46                          FROM Class
47                          WHERE Class.fid = Faculty.fid
48                      )
49          )
50  ;
51
52  -- Find the names of faculty members for whom the combined
```

```
    enrollment of the courses that they teach is less
-- than five.

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

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

DESC Employees;

```

```
52  -- Filling Aircraft table with values
53
54  INSERT INTO Aircraft VALUES
55      (1, 'Airbus', 2000),
56      (2, 'Boeing', 700),
57      (3, 'Jet', 550),
58      (4, 'Dreamliner', 5000),
59      (5, 'Boeing', 4500),
60      (6, 'Airbus', 2200)
61  ;
62
63  SELECT * FROM Aircraft;
64
65
66  -- Filling Employees table with values
67
68  INSERT INTO Employees VALUES
69      (162, 'Andrew', 50000),
70      (183, 'Laeddis', 60000),
71      (192, 'Rachel', 70000),
72      (204, 'Solando', 82000),
73      (300, 'Tony', 5000)
74  ;
75
76  SELECT * FROM Employees;
77
78  -- Filling Flights table with values
79
80  INSERT INTO Flights VALUES
81      (1, 'Bengaluru', 'New Delhi', 500, TIMESTAMP '2014-11-4
        09:24:26', TIMESTAMP '2014-11-4 09:24:26', 5000),
82      (2, 'Bengaluru', 'Chennai', 300, TIMESTAMP '2014-11-4 09:24:26'
        , TIMESTAMP '2014-11-4 09:24:26', 3000),
83      (3, 'Trivandrum', 'New Delhi', 800, TIMESTAMP '2014-11-4
        09:24:26', TIMESTAMP '2014-11-4 09:24:26', 6000),
84      (4, 'Bengaluru', 'Frankfurt', 10000, TIMESTAMP '2014-11-4
        09:24:26', TIMESTAMP '2014-11-4 09:24:26', 50000),
85      (5, 'Kolkata', 'New Delhi', 2400, TIMESTAMP '2014-11-4 09:24:26
        ', TIMESTAMP '2014-11-4 09:24:26', 9000),
86      (6, 'Bengaluru', 'Frankfurt', 8000, TIMESTAMP '2014-11-4
        09:24:26', TIMESTAMP '2014-11-4 09:24:26', 40000)
87  ;
88
89  SELECT * FROM Flights;
90
91  -- Relationship
92
93  -- Create relationship Certified between Aircraft and Employees, to
        realise which employees are certified against aircrafts
94
95  -- Certified (eid: integer, aid: integer)
96
97
98  -- Creating Certified table
99
100 CREATE TABLE Certified (
101     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

-- Write each of the following queries in SQL:



-- 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 Certified.eid, MAX(Aircraft.cruisingrange)
FROM Certified, Aircraft
WHERE Aircraft.aid = Certified.aid
GROUP BY Certified.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 Employees.ename
FROM Employees
WHERE Employees.salary < (
            SELECT MIN(Flights.price)
            FROM Flights
```

```
157             WHERE
158                 Flights.ffrom = 'Bengaluru' AND
159                 Flights.tto = 'Frankfurt'
160         )
161 ;
162
163 -- For all aircraft with cruisingrange over 1000 Kms,. Find the
        name of the aircraft and the average salary of all
164 -- pilots certified for this aircraft.
165
166 SELECT Aircraft.aname, AVG (Employees.SALARY)
167 FROM Aircraft, Certified, Employees
168 WHERE
169     Aircraft.aid = Certified.aid AND
170     Certified.eid = Employees.eid AND
171     Aircraft.cruisingrange > 1000
172
173 GROUP BY Aircraft.aid, Aircraft.aname
174
175 ;
176
177 -- Find the names of pilots certified for some Boeing aircraft.
178
179 SELECT DISTINCT Employees.ename
180 FROM Employees, Certified, Aircraft
181 WHERE
182     Employees.eid = Certified.eid AND
183     Aircraft.aid = Certified.aid AND
184     Aircraft.aname = 'Boeing'
185 ;
186
187 -- Find the aids of all aircraft that can be used on routes from
        Bengaluru to New Delhi.
188
189 SELECT DISTINCT Aircraft.aid
190 FROM Aircraft
191 WHERE Aircraft.cruisingrange > (
192                 SELECT MIN(Flights.distance)
193                 FROM Flights
194                 WHERE
195                     Flights.ffrom = 'Bengaluru' AND
196                     Flights.tto = 'New Delhi'
197             )
198 ;
```

5 Creating a database to set various constraints

```
1
2  -- SQL | Constraints
3
4  -- 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.
5
6  -- The available constraints in SQL are:
7
8  --     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.
9  --     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.
10 --     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.
11 --     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.
12 --     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.
13 --     DEFAULT: This constraint specifies a default value for the
       column when no value is specified by the user.
14
15 CREATE TABLE sample_table
16 (
17 column1 data_type(size) constraint_name,
18 column2 data_type(size) constraint_name,
19 column3 data_type(size) constraint_name,
20 ....
21 );
22
23 -- sample_table: Name of the table to be created.
24 -- data_type: Type of data that can be stored in the field.
25 -- constraint_name: Name of the constraint. for example- NOT NULL,
       UNIQUE, PRIMARY KEY etc.
26
27 -- NOT NULL
28 CREATE TABLE Student
29 (
30 ID int(6) NOT NULL,
31 NAME varchar(10) NOT NULL,
32 ADDRESS varchar(20)
33 );
34
35 -- UNIQUE
36 CREATE TABLE Student
37 (
38 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 Rollback, Commit, Savepoint

```
-- 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;
```

```
8
9  START TRANSACTION;
10
11 INSERT INTO TEST VALUES(1);
12
13 SELECT * FROM TEST;
14
15 SAVEPOINT TRAN2;
16
17 INSERT INTO TEST VALUES(2);
18
19 SELECT * FROM TEST;
20
21 ROLLBACK TO TRAN2;
22
23 SELECT * FROM TEST;
24
25 ROLLBACK;
26
27 SELECT * FROM TEST;
28
29 -- Test working of TCL commands on different tables and observe
       changes.
```

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

```
1  -- 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.
2
3  -- 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.
4
5  CREATE USER 'arjun'@'localhost' IDENTIFIED BY 'mypass';
6
7  GRANT ALL ON db1.* TO 'arjun'@'localhost';
8
9  GRANT SELECT ON child TO 'arjun'@'localhost';
10
11 GRANT USAGE ON *.* TO 'arjun'@'localhost' WITH MAX_QUERIES_PER_HOUR
       90;
12
13 -- REVOKE : The REVOKE statement enables system administrators and
       to revoke the privileges from MySQL accounts.
14
15 -- Syntax : REVOKE priv_type [(column_list)] [, priv_type [(
       column_list)]] ... ON [object_type] priv_level FROM user [,
       user] ...
16
17 REVOKE INSERT ON *.* FROM 'arjun'@'localhost';
```

8 Creation of Views and Assertions

```
1  -- VIEWS
2  --    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.
3  --    Views improve security of the database by showing only
       intended data to authorized users. They hide sensitive data.
4  --    Views make life easy as you do not have write complex queries
        time and again.
5  --    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.
6  --Ideally, you should not use VIEWS for updating.
7
8  -- To create a new view in MySQL, you use the CREATE VIEW statement
       . The syntax of creating a view in MySQL is as follows:
9
10 CREATE      [ALGORITHM = {MERGE  | TEMPTABLE | UNDEFINED}] VIEW
       view_name [(column_list)] AS select-statement;
11
12 -- Example
13
14 CREATE VIEW SalePerOrder AS
15     SELECT
16         orderNumber, SUM(quantityOrdered * priceEach) total
17     FROM
18         orderDetails
19     GROUP by orderNumber
20     ORDER BY total DESC;
21
22 -- Examine
23 SHOW TABLES;
24
25 SHOW FULL TABLES;
26
27 -- Observe the differences in both the cases here.
```

9 Implementation of built-in functions in RDBMS

```
1
2
3   -- Verify and explain each of the built-in functions listed herein.
4   -- Search another 10 such built-in functions not listed here
5
6   SELECT UCASE(NAME) FROM Students;
7
8   SELECT MID(column_name,start,length) AS some_name FROM table_name;
9   -- specifying length is optional here, and start signifies start
        position ( starting from 1 )
10
11  SELECT LENGTH(NAME) FROM Students;
12
13  SELECT ROUND(MARKS,0) FROM table_name;
14
15  SELECT NAME, FORMAT(Now(),'YYYY-MM-DD') AS Date FROM Students;
16
17  SELECT SUM(ISNULL(Salary, 10000) AS Salary FROM Employee;
18
19  SELECT ABS(-243.5);
20
21  SELECT COS(30);
22
23  SELECT GREATEST(30, 2, 36, 81, 125);
24
25  SELECT LOG(2);
26
27  SELECT MOD(18, 4);
28
29  SELECT TRUNCATE(7.53635, 2);
30
31  SELECT CURRENT_DATE();
32
33  SELECT DAYOFMONTH('2018-07-16');
34
35  SELECT HOUR("2018-07-16 09:34:00");
36
37  SELECT MAKEDATE(2009, 138);
38
39  SELECT CHARACTER_LENGTH('geeks for geeks');
40
41  SELECT CONCAT_WS('_', 'geeks', 'for', 'geeks');
42
43  SELECT LOWER('GEEKSFORGEEKS.ORG');
44
45  SELECT STRCMP('google.com', 'geeksforgeeks.com');
46
47  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

```
1
2  -- Relations
3
4  -- The following tables are maintained by a book dealer.
5
6  -- AUTHOR (author-id:int, name:string, city:string, country:string)
7
8  -- PUBLISHER (publisher-id:int, name:string, city:string, country:
       string)
9
10 -- CATALOG (book-id:int, title:string, author-id:int, publisher-id:
       int, category-id:int, year:int, price:int)
11
12 -- CATEGORY (category-id:int, description:string)
13
14 -- ORDER-DETAILS (order-no:int, book-id:int, quantity:int)
15
16 -- Creating Author table
17
18 CREATE TABLE Author (
19      authorid INT,
20      name VARCHAR(30),
21      city VARCHAR(30),
22      country VARCHAR(30),
23      PRIMARY KEY (authorid)
24 );
25
26 DESC Author;
27
28
29 -- Creating Publisher table
30
31 CREATE TABLE Publisher (
32      publisherid INT,
33      name VARCHAR(30),
34      city VARCHAR(30),
35      country VARCHAR(30),
36      PRIMARY KEY (publisherid)
37 );
38
39 DESC Publisher;
40
41
42 -- Creating BookCategory table
43
44 CREATE TABLE BookCategory (
45      categoryid INT ,
46      description VARCHAR(30),
47      PRIMARY KEY (categoryid)
48 );
49
50 DESC BookCategory;
51
```

```
52
53  -- Creating Catalog table
54
55  CREATE TABLE Catalog (
56      bookid INT,
57      title VARCHAR(30),
58      authorid INT,
59      publisherid INT,
60      categoryid INT,
61      yearofpublish INT,
62      price INT,
63      PRIMARY KEY (bookid),
64      FOREIGN KEY (authorid) REFERENCES Author (authorid),
65      FOREIGN KEY (publisherid) REFERENCES Publisher (publisherid),
66      FOREIGN KEY (categoryid) REFERENCES BookCategory (categoryid)
67  );
68
69  DESC Catalog;
70
71
72  -- Creating OrderDetails table
73
74  CREATE TABLE OrderDetails (
75      orderno INT,
76      bookid INT,
77      quantity INT,
78      PRIMARY KEY (orderno, bookid),
79      FOREIGN KEY (bookid) REFERENCES Catalog(bookid)
80  );
81
82  DESC OrderDetails;
83
84  -- Filling Author table with values
85
86  INSERT INTO Author VALUES
87      (1, 'NAVATHE', 'ARLINGTON', 'USA'),
88      (2, 'RAGHU RAMAKRISHNAN', 'CALIFORNIA', 'USA'),
89      (3, 'DHAMDHERE', 'MUMBAI', 'INDIA'),
90      (4, 'BJARNE', 'NEW JERSY', 'USA'),
91      (5, 'TANENBAUM', 'AMSTERDAM', 'NETHERLAND')
92  ;
93
94  SELECT * FROM Author;
95
96
97  -- Filling Publisher table with values
98
99  INSERT INTO Publisher VALUES
100     (1, 'JOHN WILEY', 'NEW YORK', 'USA'),
101     (2, 'PEARSON', 'BANGALORE', 'INDIA'),
102     (3, 'O REILLY', 'NEW JERSY', 'USA'),
103     (4, 'TMH', 'CALCUTTA', 'INDIA'),
104     (5, 'JOHN WILEY', 'NEW DELHI', 'INDIA')
105 ;
106
107 SELECT * FROM Publisher;
108
```

```
109
110 -- Filling BookCategory table with values
111
112 INSERT INTO BookCategory VALUES
113     (1, 'DATABASE MANAGEMENT'),
114     (2, 'OPERATING SYSTEMS'),
115     (3, 'C++'),
116     (4, 'COMPUTER NETWORKS'),
117     (5, 'C')
118 ;
119
120 SELECT * FROM BookCategory;
121
122
123 -- Filling Catalog table with values
124
125 INSERT INTO Catalog VALUES
126     (1, 'FUNDAMENTALS OF DBMS', 1, 2, 1, 2004, 500),
127     (2, 'PRINCIPLES OF DBMS', 2, 1, 1, 2004, 400),
128     (3, 'OPERATING SYSTEMS', 3, 4, 2, 2004, 200),
129     (4, 'C++ BIBLE', 4, 5, 3, 2003, 500),
130     (5, 'COMPUTER NETWORKS', 5, 3, 4, 2002, 250),
131     (6, 'FUNDAMENTALS OF C', 1, 2, 5, 2004, 700),
132     (7, 'OPERATING SYSTEMS 2', 3, 2, 2, 2001, 600)
133 ;
134
135 SELECT * FROM Catalog;
136
137
138 -- Filling OrderDetails table with values
139
140 INSERT INTO OrderDetails VALUES
141     (1, 1, 1),
142     (2, 2, 1),
143     (3, 3, 1),
144     (4, 4, 1),
145     (5, 5, 1),
146     (6, 6, 7),
147     (7, 7, 9)
148 ;
149
150 SELECT * FROM OrderDetails;
151
152
153 -- Queries
154
155 -- Give the details of the authors who have 2 or more books in the
        catalog and the price of the books is greater
156 -- than the average price of the books in the catalog and the year
        of publication is after 2000.
157
158 SELECT *
159 FROM Author A
160 WHERE EXISTS (
161         SELECT A1.authorid, COUNT(A1.authorid)
162         FROM Author A1, Catalog C
163         WHERE
```

```
164                 A1.authorid = C.authorid AND
165                 A.authorid = A1.authorid AND
166                 C.yearofpublish > 2000 AND
167                 C.price > (
168                         SELECT AVG(price)
169                         FROM Catalog
170                     )
171             GROUP BY A1.authorid
172             HAVING COUNT(A1.authorid) >= 2
173         )
174 ;
175
176 -- Find the author of the book which has maximum sales.
177
178 SELECT DISTINCT A.NAME
179 FROM Author A, Catalog C, OrderDetails ODM
180 WHERE
181     A.authorid = C.authorid AND
182     ODM.bookid = C.bookid AND
183     EXISTS (
184                 SELECT OD.bookid, SUM(OD.quantity)
185                 FROM OrderDetails OD
186                 WHERE OD.bookid = ODM.bookid
187                 GROUP BY bookid
188                 HAVING SUM(OD.quantity) >= ALL (
189                                     SELECT SUM(quantity)
190                                     FROM OrderDetails
191                                     GROUP BY bookid
192                                     )
193             )
194 ;
195
196
197 -- Demonstrate how you increase the price of books published by a
        specific publisher by 10%.
198
199 UPDATE  Catalog
200 SET price = (1.1) * price
201 WHERE authorid = (
202                 SELECT authorid
203                 FROM Author
204                 WHERE name = 'NAVATHE'
205             )
206 ;
```

12 Implementation of set operations, nested queries and Join queries

```
1
2  -- SET Operations
3
4  -- UNION Operator
5
6  -- 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:
7
8  DROP TABLE IF EXISTS t1;
9  DROP TABLE IF EXISTS t2;
10
11 CREATE TABLE t1 (
12     id INT PRIMARY KEY
13 );
14
15 CREATE TABLE t2 (
16     id INT PRIMARY KEY
17 );
18
19 INSERT INTO t1 VALUES (1),(2),(3);
20 INSERT INTO t2 VALUES (2),(3),(4);
21
22
23 SELECT id FROM t1 UNION SELECT id FROM t2;
24
25 -- INTERSECT Operator
26
27 -- MySQL does not support the INTERSECT operator. However, you can
       simulate the INTERSECT operator.
28
29 CREATE TABLE t1 ( id INT PRIMARY KEY );
30
31 CREATE TABLE t2 LIKE t1;
32
33 INSERT INTO t1(id) VALUES(1),(2),(3);
34
35 INSERT INTO t2(id) VALUES(2),(3),(4);
36
37 -- INTERSECT simulation
38 SELECT DISTINCT     id  FROM t1    INNER JOIN t2 USING(id);
39
40
41 -- MINUS Simulation
42
43  -- MySQL does not support MINUS operator. However, you can use the
       MySQL join to simulate it.
44
45 CREATE TABLE t1 (     id INT PRIMARY KEY);
46
47 CREATE TABLE t2 (     id INT PRIMARY KEY);
48
49 INSERT INTO t1 VALUES (1),(2),(3);
```

```
50  INSERT INTO t2 VALUES (2),(3),(4);
51
52
53  -- MINUS Simulation
54  SELECT column_list FROM table_1 LEFT JOIN table_2 ON
        join_predicate WHERE table_2.id IS NULL;
55
56
57  -- Consider the following database of student enrollment in courses
        & books adopted for each course.
58
59  -- STUDENT (regno: string, name: string, major: string, bdate:date)
60
61  -- COURSE (course #:int, cname:string, dept:string)
62
63  -- ENROLL ( regno:string, course#:int, sem:int, marks:int)
64
65  -- BOOK_ADOPTION (course# :int, sem:int, book-ISBN:int)
66
67  -- TEXT (book-ISBN:int, book-title:string, publisher:string, author
        :string)
68
69  -- Creating Student table
70
71  CREATE TABLE Student (
72      regno VARCHAR(30),
73      sname VARCHAR(30),
74      major VARCHAR(30),
75      bdate DATE,
76      PRIMARY KEY (regno)
77  );
78
79  DESC Student;
80
81
82  -- Creating Course table
83
84  CREATE TABLE Course (
85      course INT,
86      cname VARCHAR(30),
87      dept VARCHAR(30),
88      PRIMARY KEY (course)
89  );
90
91  DESC Course;
92
93
94  -- Creating Enroll table
95
96  CREATE TABLE Enroll (
97      regno VARCHAR(30),
98      course INT,
99      sem INT,
100     marks INT,
101     PRIMARY KEY (regno, course, sem),
102     FOREIGN KEY (regno) REFERENCES Student(regno),
103     FOREIGN KEY (course) REFERENCES Course(course));
```

```
104
105 DESC Enroll;
106
107
108 -- Creating Text table
109
110 CREATE TABLE Text (
111     bookisbn INT,
112     booktitle VARCHAR(30),
113     publisher VARCHAR(30),
114     author VARCHAR(30),
115     PRIMARY KEY (bookisbn)
116 );
117
118 DESC Text;
119
120
121 -- Creating BookAdoption table
122
123 CREATE TABLE BookAdoption (
124     course INT,
125     sem INT,
126     bookisbn INT,
127     PRIMARY KEY (course, sem, bookisbn),
128     FOREIGN KEY (course) REFERENCES Course (course),
129     FOREIGN KEY (bookisbn) REFERENCES Text (bookisbn)
130 );
131
132 DESC BookAdoption;
133
134 -- Filling Student table with values
135
136 INSERT INTO Student VALUES
137     ('1DS16CS735', 'Rishabh', 'DBMS', '1994-06-24'),
138     ('1DS16CS747', 'Siddharth', 'ADA', '1993-11-9'),
139     ('1DS16CS701', 'Aditya', 'GTC', '1994-04-28'),
140     ('1DS16CS703', 'Amit', 'SE', '1993-10-7'),
141     ('1DS16CS730', 'Gargi', 'DS', '1993-09-12')
142 ;
143
144 SELECT * FROM Student;
145
146
147 -- Filling Course table with values
148
149 INSERT INTO Course VALUES
150     (1, 'DBMS', 'CS'),
151     (2, 'ADA', 'CS'),
152     (3, 'GTC', 'TC'),
153     (4, 'SE', 'EE'),
154     (5, 'DS', 'EC'),
155     (6, 'DS', 'CS')
156 ;
157
158 SELECT * FROM Course;
159
160
```

```
-- 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),
     ('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

```
-- 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);

    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));
```

```
47 create table contacts_audit (contact_id integer, deleted_date date,
       deleted_by varchar(20));
48
49 -- Trigger defintion
50
51 delimiter //
52 create trigger contacts_after_delete
53             after delete
54             on contacts for each row
55             begin
56
57                DECLARE vUser varchar(50);
58
59                -- Find username of person performing the DELETE into
       table
60                SELECT USER() into vUser;
61
62                -- Insert record into audit table
63                INSERT into contacts_audit
64                ( contact_id,
65                  deleted_date,
66                  deleted_by)
67                VALUES
68                ( OLD.contact_id,
69                  SYSDATE(),
70                  vUser );
71             end; //
72
73 -- activating trigger
74 delimiter;
75 insert into contacts values (1, "Newton", "Isaac",
76                              str_to_date ("19-08-1985", "%d-%m-%Y")
       ,
77                              str_to_date ("23-07-2018", "%d-%m-%Y")
       , "xyz");
78 delete from contacts where first_name="Isaac";
79 insert into contacts values (1, "Newton", "Isaac",
80                              str_to_date ("19-08-1985", "%d-%m-%Y")
       ,
81                              str_to_date ("23-07-2018", "%d-%m-%Y")
       , "xyz");
82 delete from contacts where first_name="Isaac";
83
84 -- verify activation of trigger
85 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
```