K. J. SOMAIYA INSTITUTE OF ENGINEERING AND INFORMATION TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

Subject: Database Management System Lab **Semester:** IVth
Course code: 1UCEL403 **Credits-01** **Practical/week:** 2 hrs/batch

Academic year: 2021-2022

Lab Objectives :

Sr. No.	Objectives
LOb1	To explore design and develop of relational model
LOb2	To present SQL and procedural interfaces to SQL comprehensively
LOb3	To introduce the concepts of transactions and transaction processing

Lab Outcomes (LOs): On successful completion of course, learner will be able to:

Index	Outcomes
LO1	Design ER /EER diagram and convert it to a relational model for the real world application.
LO2	Apply DDL, DML, DCL and TCL commands
LO3	Write simple and complex queries
LO4	Use PL / SQL Constructs.
LO5	Apply triggers and procedures for specific module/task
LO6	Demonstrate the concept of concurrent transactions execution and frontend backend connectivity
LO7	Apply ethical principles like timeliness and adhere to the rules of the laboratory

Mapping of LOs with POs and PSOs:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
LO1			3											3	
LO2			3									2		3	
LO3			3									2		3	
LO4			3									2		3	
LO5			3											3	
LO6				3								3		3	
LO7										3					
LO8								3							
Avg			**3**	**3**				**3**		**3**		**2**		**3**	

Subject Teacher
Dr. Sarita Ambadekar

SOMAIYA

VIDYAVIHAR

K J Somaiya Institute of Engineering and Information Technology

An Autonomous Institute Permanently Affiliated to the University of Mumbai

Accredited by NAAC with 'A' Grade (3.21 CGPA), Approved by AICTE, New Delhi

Department of Computer Engineering

EXPERIMENT LIST WITH OBJECTIVE , OUTCOME AND MAPPING

Lab code: 1UCEL403	**Course Name: Database Management System Lab**

Expt . No:	Experiment Title	Objective	Outcomes	LO mapping
1	Identify the case study and detailed statement of the problem. Design an Entity-Relationship (ER) / Extended Entity Relationship (EER) Model	To understand, analyze and design selected ER and EER diagram.	Students will be able to design ER diagram for the selected case study.	LO1,LO7, LO8
2	Mapping ER/EER to Relational schema model	• Learn and practice data modelling using the entity-relationship and developing database designs. • Understand the concept of mapping of ER and EER model to relational model .	Students will be able to design relational schema model for the given case study.	LO1, LO7, LO8
3	Create a database using Data Definition Language (DDL) and apply integrity constraints for the specified System	To design the database by using SQL commands	Students will be able to develop the data base and create all tables for the same case study.	LO2,LO7, LO8

4	Apply DML Commands for the specified system	• SQL constraints are rules used to limit the type of data that can go into a table. • To maintain the accuracy and integrity of the data inside table.	Students will be able to apply integrity constraints for their database entities.	LO2,LO7,LO8
5	Perform Simple queries, string manipulation operations and aggregate functions.	Understand the use of Structured Query Language (SQL) & to understand various ways to retrieve data using select commands and clauses.	Students will be able to execute SQL queries and verify the results	LO3,LO7,LO8
6	Implement various Join operations.	To understand various ways to retrieve data using nested and complex queries in SQL.	Students will be able to write various join queries in SQL.	LO3,LO7,LO8
7	Perform Nested and Complex queries	To understand various ways to retrieve data using various join operation queries in SQL	Students will be able to write complex queries.	LO2,LO3, LO7,LO8
8	Perform DCL and TCL commands	• To create views on tables. • To create triggers on the database.	Students will be able to write data control language commands and transaction control commands.	LO4,LO7, LO8
9	Implement procedure and functions	To apply normalization techniques to normalize the database.	Students will be able to write function, procedure and cursor on database.	LO4,LO7, LO8
10	Execution of CRUD operations from front end using Database connectivity	To create transactions on database and check for concurrency control.	Students will be able to develop the front end application and make the connectivity with database and execute as a mini project	LO6,LO7, LO8
11	Implementation of Views and Triggers.	To develop application using Java and database (like MySql, PHP)	Students will be able to create views and write	LO6,LO7, LO8

			trigger for perform various tasks.	
12	Implementation and demonstration of Transaction and Concurrency control techniques using locks.		Students will be able to execute various transactions commands.	LO5, LO6

Mrs. Sarita Ambadekar
(Sign of Practical in-charge)

Experiment No: 1

Aim: Identify the case study and detail statement of problem. Design an Entity-Relationship (ER) / Extended Entity-Relationship (EER) Model.

Objective :

- To understand, analyze and design selected ER/EER diagram.
- Learn and practice data modeling using the entity-relationship

Theory: **Case study :** - Company database

Problem Statement:-

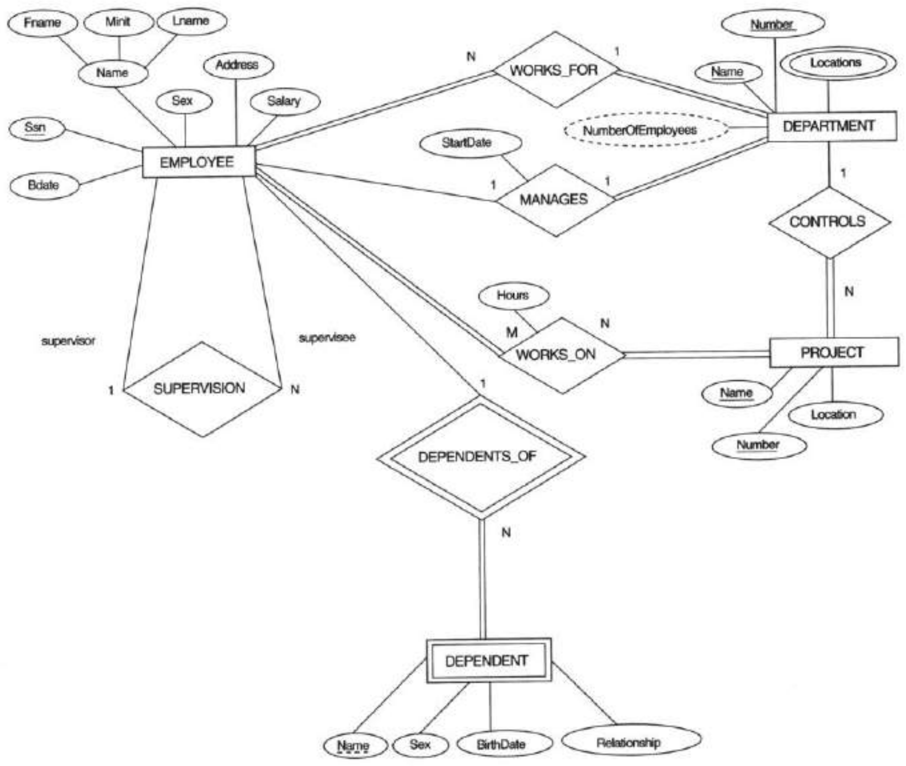
- The company is organized into DEPARTMENTs. Each department has a name, number and an employee who *manages* the department. We keep track of the start date of the department manager.

- Each department *controls* a number of PROJECTs. Each project has a name, number and is located at a single location.

- We store each EMPLOYEE's social security number, address, salary, sex, and birthdate. Each employee *works for* one department but may *work on* several projects. We keep track of the number of hours per week that an employee currently works on each project. We also keep track of the *direct supervisor* of each employee.

- Each employee may *have* a number of DEPENDENTs. For each dependent, we keep track of their name, sex, birthdate, and relationship to employee.

ER DIAGRAM :

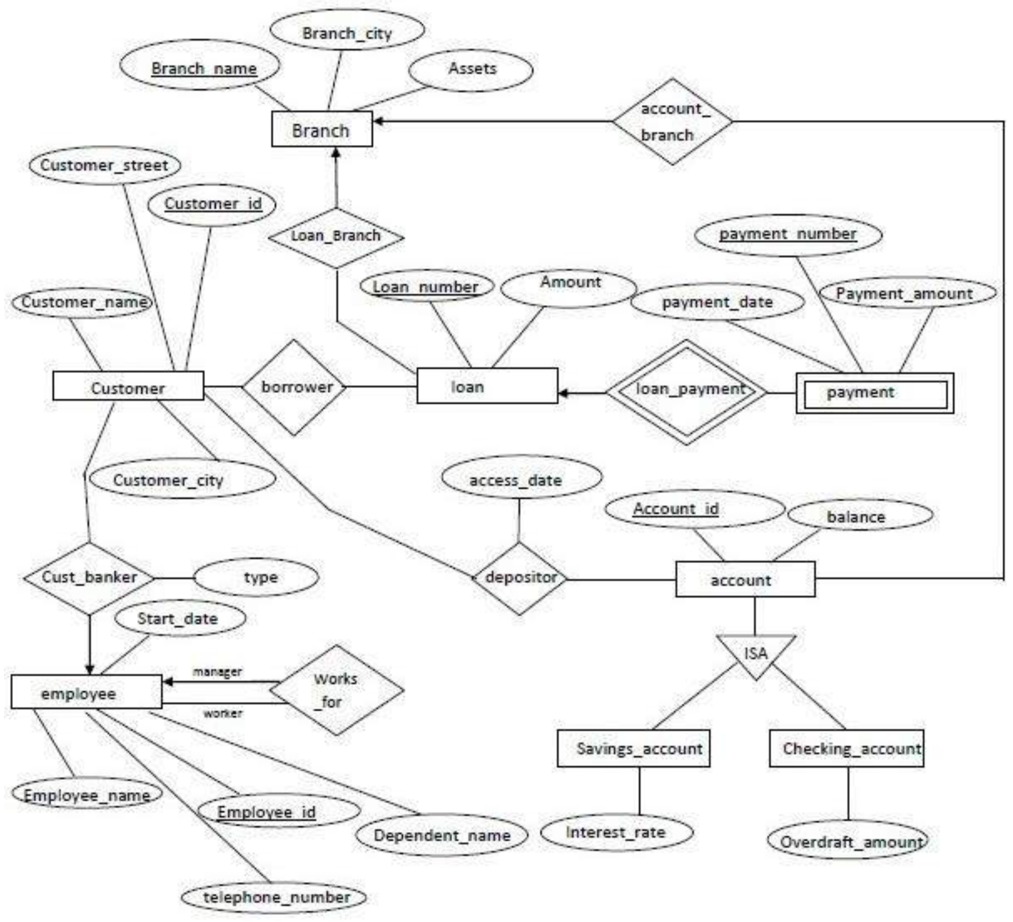


For Students ------ Write statement of problem of your case study

Construct ER / EER model from the problem definition.

For Example :- Banking system

EER model for Banking system:-



Conclusion: Company database (Write your case study name) is designed using data models (ER model)..

Outcome: Students will be able to design ER diagram for the ___________ Write your case study name case study.

Oral Questions:

1. Define DBMS and RDBMS
2. What are the applications of DBMS ?
3. What is the difference between File system and Database system ?
4. What are the notations use in ER and EER diagram ?
5. What do you mean by entity?
6. What are the different types attributes ?
7. Define strong and weak entity?
8. Specify steps to draw ER diagram
9. Define Cardinality .

Experiment No: 2

Aim: To draw ER-EER model and relational model for a given case study.

Objective :

- Learn and practice data modelling using the entity-relationship and developing database designs.
- Understand the LOncept of mapping of ER and EER model to relational model .

Theory:

Relational Database Design by ER- and ERR-to- Relational Mapping

ER-to-Relational Mapping Algorithm

Step 1: Mapping of Regular Entity Types
Step 2: Mapping of Weak Entity Types
Step 3: Mapping of Binary 1:1 Relation Types
Step 4: Mapping of Binary 1:N Relationship Types.
Step 5: Mapping of Binary M:N Relationship Types.
Step 6: Mapping of Multivalued attributes.
Step 7: Mapping of N-ary Relationship Types.

Mapping EER Model Constructs to Relations

Step 8: Options for Mapping Specialization or Generalization.
Step 9: Mapping of Union Types (Categories).

Students will not write the below steps in their theory part instead they will read these 9-steps carefully and write in your own words in terms of steps how you have converted your ER/EER diagram of exp-1 into relational schema.

ER-to-Relational Mapping Algorithm

Step 1: Mapping of Regular Entity Types.

For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E. Choose one of the key attributes of E as the primary key for R. If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R.

Step 2: Mapping of Weak Entity Types

For each weak entity type W in the ER schema with owner entity type E, create a relation R & include all simple attributes (or simple components of composite attributes) of W as attributes of
R. Also, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any.

Step 3: Mapping of Binary 1:1 Relation Types

For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. There are three possible approaches: 1. Foreign Key approach: Choose one of the relations-say S-and include a foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role of S.

Step 4: Mapping of Binary 1:N Relationship Types.

For each regular binary 1:N relationship type R, identify the relation S that represent the participating entity type at the N-side of the relationship type. Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R.
Include any simple attributes of the 1:N relation type as
attributes of S.

Step 5: Mapping of Binary M:N Relationship Types.

For each regular binary M:N relationship type R, create a new relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S. Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S.

Step 6: Mapping of Multivalued attributes.

For each multivalued attribute A, create a new relation R. This relation R will include an attribute corresponding to A, plus the primary key attribute K-as a foreign key in R-of the relation that represents the entity type of relationship type that has A as an attribute. The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.

Step 7: Mapping of N-ary Relationship Types.

For each n-ary relationship type R, where n>2, create a new relationship S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types. Also include any simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes of S.

Step8: Options for Mapping Specialization or Generalization.

Convert each specialization with m subclasses {S1, S2,....,Sm} and generalized superclass C, where the attributes of C are {k,a1,...an} and k is the (primary) key, into relational schemas using one of the four following options:

Option 8A: Multiple relations-Superclass and subclasses
Option 8B: Multiple relations-Subclass relations only
Option 8C: Single relation with one type attribute
Option 8D: Single relation with multiple type attributes

Step 9: Mapping of Union Types

For mapping a category whose defining superclass have different keys, it is customary to specify a new key attribute, called a surrogate key, when creating a relation to correspond to the category.
In the example below we can create a relation OWNER to correspond to the OWNER category and include any attributes of the category in this relation. The primary key of the OWNER relation is the surrogate key, which we called OwnerId.

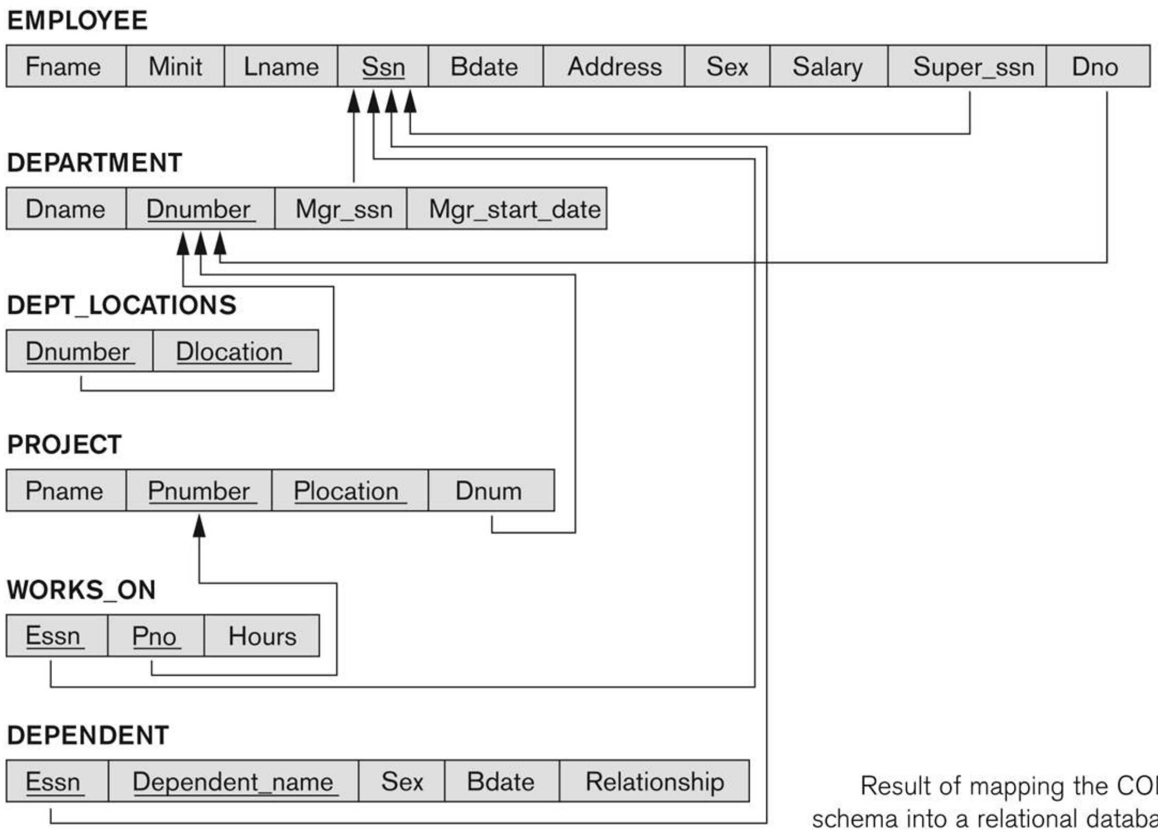


Figure 7.2
Result of mapping the COMPANY ER schema into a relational database schema.

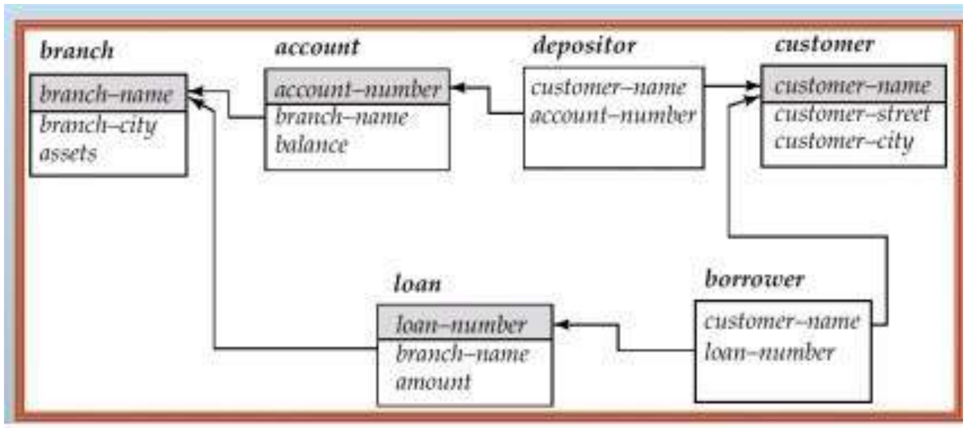


Fig. Result of mapping the Bank ER schema into relational database schema

Conclusion: Relational schema is designed for the _______case study.

Outcome: Students will be able to design relational schema model for the _____ case study.

Oral Questions:

1. What is relational schema?
2. List steps of Mapping from ER and EER to relational schema.
3. Explain steps of mapping from ER-EER to relational schema.

Experiment No: 3

Aim: Create and populate database using Data Definition Language (DDL) and DML commands for your the specified System.

Objective : To design the database by using SQL commands

Software Requirement: MySQL 5.6 on ubantu 16.04

Theory:

Students will explore various datatypes of MySQL.

They will create all tables in their database and also design the relationship among tables by foreign keys .

Insert records in tables and select data from more than one table (DDL and DML commands).

Take printout

To design the given database first understand the Syntax of MySQL commands and data types available

The following is a list of datatypes available in MySQL
Database table contains multiple columns with specific data types such as numeric or string. MySQL provides you many more specific data types than just "numeric" or "string". Each data type in MySQL can be determined by the following characteristics:

2. What kind of value it can represent.
3. The space values take up and whether the values are fixed-length or variable-length.
4. The values of a data type can be indexed.
5. How MySQL compare values of that data types.

➡ Numeric Data Types

TINYINT( ) -128 to 127 normal
0 to 255 UNSIGNED.

SMALLINT( ) -32768 to 32767 normal
0 to 65535 UNSIGNED.

MEDIUMINT( ) -8388608 to 8388607 normal
0 to 16777215 UNSIGNED.

INT( ) -2147483648 to 2147483647 normal
0 to 4294967295 UNSIGNED.

BIGINT( ) -9223372036854775808 to 9223372036854775807 normal
0 to 18446744073709551615 UNSIGNED.

FLOAT A small number with a floating decimal point.

DOUBLE(,) A large number with a floating decimal point.

DECIMAL(,) A **DOUBLE** stored as a string , allowing for a fixed decimal point.
The integer types have an extra option called UNSIGNED. Normally, the integer goes from an negative to

positive value. Using an UNSIGNED command will move that range up so it starts at zero instead of a negative number.

String Data Types

In MySQL, string can hold anything from plain text to binary data such as images and files. String can be compared and searched based on pattern matching by using LIKE clause or regular expression. The table below shows you the string data types in MySQL:

CHAR( ) A fixed section from 0 to 255 characters long.

VARCHAR( ) A variable section from 0 to 255 characters long.

TINYTEXT A string with a maximum length of 255 characters.

TEXT A string with a maximum length of 65535 characters.

BLOB A string with a maximum length of 65535 characters.

MEDIUMTEXT A string with a maximum length of 16777215 characters.

MEDIUMBLOB A string with a maximum length of 16777215 characters.

LONGTEXT A string with a maximum length of 4294967295 characters.

LONGBLOB A string with a maximum length of 4294967295 characters.

CHAR and **VARCHAR** are the most widely used types. **CHAR** is a fixed length string and is mainly used when the data is not going to vary much in it's length. **VARCHAR** is a variable length string and is mainly used when the data may vary in length.

CHAR may be faster for the database to process considering the fields stay the same length down the column. **VARCHAR** may be a bit slower as it calculates each field down the column, but it saves on memory space. Which one to ultimatly use is up to you.

Using both a **CHAR** and **VARCHAR** option in the same table, MySQL will automatically change the **CHAR** into **VARCHAR** for compatability reasons.

BLOB stands for Binary Large OBject. Both **TEXT** and **BLOB** are variable length types that store large amounts of data. They are similar to a larger version of **VARCHAR**. These types can store a large piece of data information, but they are also processed much slower.

Date and Time Data Types

MySQL provides types for date and time and combination of date and time. In addition, MySQL also provide timestamp data type for tracking last change on a record. If you just want to store the year without date and month, you can use YEAR data type. Here is the table which showing MySQL date and type data types:

DATE YYYY-MM-DD.
DATETIME YYYY-MM-DD HH:MM:SS.
TIMESTAMP YYYYMMDDHHMMSS.
TIME HH:MM:SS.

Structure Query Language(SQL) is a database query language used for storing and managing data in Relational DBMS. SQL was the first commercial language introduced for E.F Codd's **Relational** model of database. Today almost all RDBMS(MySql, Oracle, Infomix, Sybase, MS Access) use **SQL** as the standard database query language. SQL is used to perform all types of data operations in RDBMS.

SQL Command

SQL defines following ways to manipulate data stored in an RDBMS.

DDL: Data Definition Language

This includes changes to the structure of the table like creation of table, altering table, deleting a table etc.
All DDL commands are auto-committed. That means it saves all the changes permanently in the database.

Command	**Description**
create	to create new table or database
alter	alters the structure of the existing table
truncate	delete all records from a table
drop	to drop a table
rename	to rename a table

ML: Data Manipulation Language

DML commands are used for manipulating the data stored in the table and not the table itself.
DML commands are not auto-committed. It means changes are not permanent to database, they can be rolled back.

Command	**Description**
insert	to insert a new row

update	to update existing row
delete	to delete a row
select	to retrieve data from a database.

For example

```
mysql> create database playschool;
Query OK, 1 row affected (0.00 sec)

mysql> use playschool;
Database changed

mysql> create table guardian(g_name varchar(40) primary key, emailID varchar(40),
relation_with_child varchar(20));

Query OK, 0 rows affected (0.44 sec)

mysql> create table child
(c_id int(20) primary key,
name varchar(40),
father_name varchar(40),
mother_name varchar(40),
 DOB date, gender char(1),
address varchar(200),
medical_details varchar(100),
guardian_name varchar(40),
foreign key(guardian_name) references guardian(g_name));
Query OK, 0 rows affected (0.45 sec)

mysql> create table fees
(total_fees decimal(10,2),
paid_fees decimal(10,2),
 balance_fees decimal(10,2),
 guardian_name varchar(40),
foreign key(guardian_name) references guardian(g_name));
Query OK, 0 rows affected (0.32 sec)
```

```
mysql> create table handles (staff_id int(20),
child_id int(20),
foreign key(staff_id) references staff(s_ID),
foreign key(child_id) references child(c_id));
Query OK, 0 rows affected (0.33 sec)

 mysql> create table s_ph_no
(staff_id int(10),
ph_no int(10),
foreign key(staff_id) references staff(s_ID));
Query OK, 0 rows affected (0.47 sec)

mysql> create table g_ph_no
(g_name varchar(40),
ph_no int(10),
foreign key(g_name) references guardian(g_name));
Query OK, 0 rows affected (0.30 sec)



mysql> insert into staff values (10,'ritesh',32,'m','sweeper',15000,'SSC');
Query OK, 1 row affected (0.31 sec)

mysql> insert into staff values (32,'rama roy', 40, 'f','teacher',45000,'MA BEd');
Query OK, 1 row affected (0.09 sec)

mysql> insert into guardian values ('shankar sharma','shankar.s@gmail.com','grandfather');
Query OK, 1 row affected (0.04 sec)

mysql> insert into guardian values ('sheetal shah','ss1248@gmail.com','aunt');
Query OK, 1 row affected (0.03 sec)

mysql> insert into child values(146,'sanjana sharma','mayank sharma','kapila sharma','2016-6-
15','f','C/402, Ahmed Nagar, Charai','Egg allergy','shankar sharma');
Query OK, 1 row affected (0.06 sec)

mysql> insert into child values(5,'darshan roy','vikas roy','swati roy','2015-7-7','m','B/903, Motiram
Greens, Strawberry, Teen Haath Naka','asthama','sheetal shah');
Query OK, 1 row affected (0.02 sec)
```

```
mysql> insert into fees values(14000,14000,0,'shankar sharma');
Query OK, 1 row affected (0.03 sec)

mysql> insert into fees values(16000,12000,4000,'sheetal shah');
Query OK, 1 row affected (0.04 sec)

mysql> insert into handles values(32,5);
Query OK, 1 row affected (0.06 sec)

mysql> insert into handles values(32,146);
Query OK, 1 row affected (0.04 sec)

mysql> insert into s_ph_no values(10,9969561538);
ERROR 1264 (22003): Out of range value for column 'ph_no' at row 1

mysql> insert into s_ph_no values(10,25856702);
Query OK, 1 row affected (0.05 sec)

mysql> insert into s_ph_no values(32,2665174);
Query OK, 1 row affected (0.04 sec)

mysql> insert into g_ph_no values('shankar sharma',2895654);
Query OK, 1 row affected (0.04 sec)

mysql> insert into g_ph_no values('sheetal shah',45678910);
Query OK, 1 row affected (0.04 sec)

mysql> select * from staff;
+------+----------+------+--------+----------+----------+---------------+
| s_ID | name     | age  | gender | position | salary   | qualification |
+------+----------+------+--------+----------+----------+---------------+
|   10 | ritesh   |   32 | m      | sweeper  | 15000.00 | SSC           |
|   32 | rama roy |   40 | f      | teacher  | 45000.00 | MA BEd        |
+------+----------+------+--------+----------+----------+---------------+
2 rows in set (0.10 sec)

mysql> select * from child;
+------+----------------+---------------+---------------+------------+--------+----------------------------------------------
-----+-----------------+----------------+
| c_id | name           | father_name   | mother_name   | DOB        | gender | address                                                  |
medical_details | guardian_name  |
+------+----------------+---------------+---------------+------------+--------+---------------------------------------------
-----+-----------------+----------------+
|    5 | darshan roy    | vikas roy     | swati roy     | 2015-07-07 | m      | B/903, Motiram Greens, Strawberry,
Teen Haath Naka | asthama         | sheetal shah   |
|  146 | sanjana sharma | mayank sharma | kapila sharma | 2016-06-15 | f      | C/402, Ahmed Nagar, Charai
| Egg allergy     | shankar sharma |
+------+----------------+---------------+---------------+------------+--------+---------------------------------------------
-----+-----------------+----------------+
2 rows in set (0.00 sec)
```

mysql> **select * from guardian;**

```
+----------------+---------------------+---------------------+
| g_name         | emailID             | relation_with_child |
+----------------+---------------------+---------------------+
| shankar sharma | shankar.s@gmail.com | grandfather         |
| sheetal shah   | ss1248@gmail.com    | aunt                |
+----------------+---------------------+---------------------+
2 rows in set (0.00 sec)
```

```
mysql> select * from fees;
+------------+-----------+--------------+----------------+
| total_fees | paid_fees | balance_fees | guardian_name  |
+------------+-----------+--------------+----------------+
|   14000.00 |  14000.00 |         0.00 | shankar sharma |
|   16000.00 |  12000.00 |      4000.00 | sheetal shah   |
+------------+-----------+--------------+----------------+
2 rows in set (0.00 sec)
```

mysql> **select * from handles;**

```
+----------+----------+
| staff_id | child_id |
+----------+----------+
|       32 |        5 |
|       32 |      146 |
+----------+----------+
2 rows in set (0.00 sec)
```

mysql> **select * from s_ph_no;**

```
+----------+----------+
| staff_id | ph_no    |
+----------+----------+
|       10 | 25856702 |
|       32 |  2665174 |
+----------+----------+
2 rows in set (0.00 sec)

mysql> select * from g_ph_no;
+----------------+----------+
| g_name         | ph_no    |
+----------------+----------+
| shankar sharma |  2895654 |
| sheetal shah   | 45678910 |
+----------------+----------+
2 rows in set (0.00 sec)
mysql> update child set name='shreya sharma' where c_id='146';
Query OK, 1 row affected (0.12 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from child;
+------+---------------+---------------+---------------+------------+--------+-----------------------------------------------
----+-----------------+----------------+
| c_id | name          | father_name   | mother_name   | DOB        | gender | address                                       |
medical_details | guardian_name  |
+------+---------------+---------------+---------------+------------+--------+-----------------------------------------------
----+-----------------+----------------+
|    5 | darshan roy   | vikas roy     | swati roy     | 2015-07-07 | m      | B/903, Motiram Greens, Strawberry,
Teen Haath Naka | asthama         | sheetal shah   |
|  146 | shreya sharma | mayank sharma | kapila sharma | 2016-06-15 | f      | C/402, Ahmed Nagar, Charai
| Egg allergy     | shankar sharma |
+------+---------------+---------------+---------------+------------+--------+-----------------------------------------------
----+-----------------+----------------+
2 rows in set (0.00 sec)

mysql> insert into g_ph_no values('sheetal shah',8754525);
Query OK, 1 row affected (0.04 sec)

mysql> select * from g_ph_no;
+----------------+----------+
| g_name         | ph_no    |
+----------------+----------+
| shankar sharma |  2895654 |
| sheetal shah   | 45678910 |
| sheetal shah   |  8754525 |
+----------------+----------+
3 rows in set (0.00 sec)

mysql> delete from g_ph_no;
Query OK, 3 rows affected (0.05 sec)

mysql> select * from g_ph_no;
Empty set (0.00 sec)

mysql> insert into g_ph_no values('sheetal shah',8754525);
Query OK, 1 row affected (0.03 sec)

mysql> insert into g_ph_no values('shankar sharma',2895654);
Query OK, 1 row affected (0.04 sec)

mysql> select * from g_ph_no;
+----------------+---------+
| g_name         | ph_no   |
+----------------+---------+
| sheetal shah   | 8754525 |
| shankar sharma | 2895654 |
+----------------+---------+
2 rows in set (0.00 sec)

mysql> create view Guardian_Details as select g_name,relation_with_child from guardian;
Query OK, 0 rows affected (0.04 sec)

mysql> select * from Guardian_Details;
+----------------+---------------------+
| g_name         | relation_with_child |
+----------------+---------------------+
| shankar sharma | grandfather         |
| sheetal shah   | aunt                |
+----------------+---------------------+
2 rows in set (0.02 sec)
```

```
mysql> alter table staff add blood_group varchar(10);
Query OK, 0 rows affected (0.89 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select * from staff;
+------+----------+------+--------+----------+----------+---------------+-------------+
| s_ID | name     | age  | gender | position | salary   | qualification | blood_group |
+------+----------+------+--------+----------+----------+---------------+-------------+
|   10 | ritesh   |   32 | m      | sweeper  | 15000.00 | SSC           | NULL        |
|   32 | rama roy |   40 | f      | teacher  | 45000.00 | MA BEd        | NULL        |
+------+----------+------+--------+----------+----------+---------------+-------------+
2 rows in set (0.00 sec)

mysql> show tables;
+----------------------+
| Tables_in_playschool |
+----------------------+
| Guardian_Details     |
| child                |
| fees                 |
| g_ph_no              |
| guardian             |
| handles              |
| s_ph_no              |
| staff                |
+----------------------+
8 rows in set (0.00 sec)

mysql> drop view Guardian_Details;
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
+----------------------+
| Tables_in_playschool |
+----------------------+
| child                |
| fees                 |
| g_ph_no              |
| guardian             |
| handles              |
| s_ph_no              |
| staff                |
+----------------------+
7 rows in set (0.00 sec)
```

Conclusion: The database is design for the given case study by using MySQL commands.

Outcome: Students will be able to develop the data base and create all tables for the ____ case study using SQL commands.

Oral Questions :

1. Define primary key, foreign key.
2. What is the use of referential integrity?
3. Define DDL
4. Define DML
5. What are operations in DDL and DML?

Experiment No: 4

Aim: Apply Integrity constraints for the specified system.

Objective :

- SQL constraints are rules used to limit the type of data that can go into a table.
- To maintain the accuracy and integrity of the data inside table.

Software Requirement: MySQL 5.6 on ubantu 16.04

Theory:

Integrity Constraints

Integrity constraints are used to ensure accuracy and consistency of data in a relational database.
Constraints can be divided into following two types,

- **Column level constraints :** limits only column data

- **Table level constraints :** limits whole table data

Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

SQL CREATE TABLE + CONSTRAINT Syntax

```
CREATE TABLE table_name
(
column_name1 data_type(size) constraint_name,
column_name2 data_type(size) constraint_name,
column_name3 data_type(size) constraint_name,
....
);
```

In SQL, we have the following constraints:

- **NOT NULL -** Indicates that a column cannot store NULL value
- **UNIQUE -** Ensures that each row for a column must have a unique value
- **PRIMARY KEY -** A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have a unique identity which helps to find a particular record in a table more easily and quickly
- **FOREIGN KEY -** Ensure the referential integrity of the data in one table to match values in another table
- **CHECK -** Ensures that the value in a column meets a specific condition
- **DEFAULT -** Specifies a default value for a column

1. NOT NULL Constraint

NOT NULL constraint restricts a column from having a NULL value. Once **NOT NULL** constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value. One important point to note about NOT NULL constraint is that it cannot be defined at table level.

Example using NOT NULL constraint

CREATE table Student(s_id int NOT NULL, Name varchar(60), Age int);

The above query will declare that the **s_id** field of **Student** table will not take NULL value.

2. UNIQUE Constraint

UNIQUE constraint ensures that a field or column will only have unique values. A UNIQUE constraint field will not have duplicate data. UNIQUE constraint can be applied at column level or table level.

Example using UNIQUE constraint when creating a Table (Table Level)

CREATE table Student(s_id int NOT NULL UNIQUE, Name varchar(60), Age int);

The above query will declare that the **s_id** field of **Student** table will only have unique values and wont take NULL value.

Example using UNIQUE constraint after Table is created (Column Level)

ALTER table Student add UNIQUE(s_id);

The above query specifies that **s_id** field of **Student** table will only have unique value.

3. Primary Key Constraint

Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

Example using PRIMARY KEY constraint at Table Level

CREATE table Student (s_id int **PRIMARY KEY**, Name varchar(60) NOT NULL, Age int);

The above command will creates a PRIMARY KEY on the s_id.

Example using PRIMARY KEY constraint at Column Level

ALTER table Student add PRIMARY KEY (s_id);

The above command will creates a PRIMARY KEY on the s_id.

4. Foreign Key Constraint

FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables. To understand FOREIGN KEY, let's see it using two table.

Customer_Detail Table :

c_id	Customer_Name	address
101	Adam	Noida
102	Alex	Delhi
103	Stuart	Rohtak

Order_Detail Table :

Order_id	Order_Name	c_id
10	Order1	101
11	Order2	103
12	Order3	102

In **Customer_Detail** table, c_id is the primary key which is set as foreign key in **Order_Detail** table. The value that is entered in c_id which is set as foreign key in **Order_Detail** table must be present in **Customer_Detail** table where it is set as primary key. This prevents invalid data to be inserted into c_id column of **Order_Detail** table.

Example using FOREIGN KEY constraint at Table Level

CREATE table Order_Detail(order_id int PRIMARY KEY,

order_name varchar(60) NOT NULL,

c_id int FOREIGN KEY REFERENCES Customer_Detail(c_id));

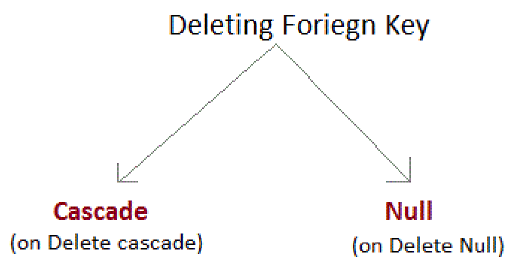
In this query, c_id in table Order_Detail is made as foriegn key, which is a reference of c_id column of Customer_Detail.

Example using FOREIGN KEY constraint at Column Level

ALTER table Order_Detail add FOREIGN KEY (c_id) REFERENCES Customer_Detail(c_id);

Behaviour of Foriegn Key Column on Delete

There are two ways to maintin the integrity of data in Child table, when a particular record is deleted in main table. When two tables are connected with Foriegn key, and certain data in the main table is deleted, for which record exit in child table too, then we must have some mechanism to save the integrity of data in child table.



- **On Delete Cascade :** This will remove the record from child table, if that value of foreign key is deleted from the main table.
- **On Delete Null :** This will set all the values in that record of child table as NULL, for which the value of foreign key is deleted from the main table.

- If we don't use any of the above, then we cannot delete data from the main table for which data in child table exists.

We will get an error if we try to do so.

ERROR: Record in child table exist

5. CHECK Constraint

CHECK constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. Its like condition checking before saving data into a column.

Example using CHECK constraint at Table Level

```
create table Student(s_id   int NOT NULL CHECK(s_id > 0),
                      Name varchar(60) NOT NULL,
                      Age int);
```

The above query will restrict the s_id value to be greater than zero.

Example using CHECK constraint at Column Level

```
ALTER table Student add CHECK(s_id > 0);        .
```

Conclusion:

Various Column and table integrity constraints are used to create tables and database for the case study .

Outcome:

Students will be able to apply integrity constraints for their database entities.

Experiment No: 5

Aim: Perform Simple queries, string manipulation operations.

Objective :

Understand the use of Structured Query Language (SQL)

To understand various ways to retrieve data using select commands and clauses.

Software Requirement: MySQL 5.6 on ubantu 16.04

Theory:

Students will write following theory in their write-ups, ie some explanation and syntax

They will perform following commands on their tables of their case study database and attach the printout.

✓ *Querying the Database – SELECT*

The SELECT command has the following syntax:

-- List all the rows of the specified columns

SELECT column1Name, column2Name, ... FROM tableName

-- List all the rows of ALL columns, * is a wildcard denoting all columns

SELECT * FROM tableName

-- List rows that meet the specified criteria in WHERE clause

SELECT column1Name, column2Name,... FROM tableName WHERE criteria

SELECT * FROM tableName WHERE criteria

The WHERE clause is used to extract only those records that fulfill a specified criterion.

For examples

Comparison Operators

For numbers (INT, DECIMAL, FLOAT), you could use comparison operators: '=' (equal to), '<>' or '!=' (not equal to), '>' (greater than), '<' (less than), '>=' (greater than or equal to), '<=' (less than or equal to), to compare two numbers.

For example, price > 1.0, quantity <= 500.

mysql> SELECT name, price FROM products WHERE price < 1.0;

mysql> SELECT name, quantity FROM products WHERE quantity <= 2000;

CAUTION: Do not compare FLOATs (real numbers) for equality ('=' or '<>'), as they are not precise. On the other hand, DECIMAL are precise.

For strings, you could also use '=', '<>', '>', '<', '>=', '<=' to compare two strings (e.g., productCode = 'PEC').

mysql> SELECT name, price FROM products WHERE productCode = 'PEN';

String Pattern Matching - LIKE and NOT LIKE

For strings, in addition to full matching using operators like '=' and '<>', we can perform pattern matching using operator LIKE (or NOT LIKE) with wildcard characters. The wildcard '_' matches any single character; '%' matches any number of characters (including zero). For example,

- ➡ 'abc%' matches strings beginning with 'abc';
- ➡ '%xyz' matches strings ending with 'xyz';
- ➡ '%aaa%' matches strings containing 'aaa';
- ➡ '___' matches strings containing exactly three characters; and
- ➡ 'a_b%' matches strings beginning with 'a', followed by any single character, followed by 'b', followed by zero or more characters.

For example "name" begins with 'PENCIL'

mysql> SELECT name, price FROM products WHERE name LIKE 'PENCIL%';

Arithmetic Operators

You can perform arithmetic operations on numeric fields using arithmetic operators,as bellow

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
DIV	Integer Division
%	Modulus (Remainder)

Logical Operators - AND, OR, NOT, XOR

You can combine multiple conditions with boolean operators AND, OR, XOR. You can also invert a condition using operator NOT. For examples,

```
mysql> SELECT * FROM products WHERE quantity >= 5000 AND name LIKE 'Pen %';

mysql> SELECT * FROM products WHERE quantity >= 5000 AND price < 1.24 AND name LIKE 'Pen %';
```

IN, NOT IN

You can select from members of a set with IN (or NOT IN) operator.

```
mysql> SELECT * FROM products WHERE name IN ('Pen Red', 'Pen Black');
```

BETWEEN, NOT BETWEEN

To check if the value is within a range, you could use BETWEEN ... AND ... operator. Again, this is easier and clearer than the equivalent AND-OR expression.

```
mysql> SELECT * FROM products

    WHERE (price BETWEEN 1.0 AND 2.0) AND (quantity BETWEEN 1000 AND 2000);
```

IS NULL, IS NOT NULL

NULL is a special value, which represent "no value", "missing value" or "unknown value". You can checking if a column contains NULL by IS NULL or IS NOT NULL. For example,

```
mysql> SELECT * FROM products WHERE productCode IS NULL;

Empty set (0.00 sec)
```

Using comparison operator (such as = or <>) to check for NULL is a mistake - a very common mistake. For example,

```
SELECT * FROM products WHERE productCode = NULL;
```

-- This is a common mistake. NULL cannot be compared.

ORDER BY Clause

The ORDER BY keyword is used to sort the result-set by a specified column.It sort the records in ascending order by default.To sort the records in a descending order, use the DESC keyword.

SELECT ... FROM tableName

WHERE criteria

ORDER BY columnA ASC|DESC, columnB ASC|DESC, ...

If several rows have the same value in columnA, it will be ordered according to columnB, and so on.

-- Order the results by price in descending order

mysql> SELECT * FROM products WHERE name LIKE 'Pen %' ORDER BY price DESC;

-- Order by price in descending order, followed by quantity in ascending (default) order

mysql> SELECT * FROM products WHERE name LIKE 'Pen %' ORDER BY price DESC, quantity;

AS - Alias

You could use the keyword AS to define an alias for an identifier (such as column name, table name). The alias will be used in displaying the name. It can also be used as reference. For example,

mysql> SELECT productID AS ID, productCode AS Code, name AS Description, price AS `Unit Price` FROM products;

DISTINCT

A column may have duplicate values, we could use keyword DISTINCT to select only distinct values. We can also apply DISTINCT to several columns to select distinct combinations of these columns. For examples,

-- With DISTINCT on price

mysql> SELECT DISTINCT price AS `Distinct Price` FROM products;

```
-- DISTINCT combination of price and name

mysql> SELECT DISTINCT price, name FROM products;
```

GROUP BY Clause

The GROUP BY statement is used in conjunction with the aggregate functions(such as COUNT(), AVG(), SUM()) to group the result-set by one or more columns.
Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
```

```
-- Function COUNT(*) returns the number of rows selected
mysql> SELECT COUNT(*) AS `Count` FROM products;
       -- All rows without GROUP BY clause
+-------+
| Count |
+-------+
|     5 |
+-------+

mysql> SELECT productCode, COUNT(*) FROM products GROUP BY productCode;
+-------------+----------+
| productCode | COUNT(*) |
+-------------+----------+
| PEC         |        2 |
| PEN         |        3 |
```

The MAX() function returns the largest value of the selected column. The MIN() function returns the smallest value of the selected column.

```
mysql> SELECT MAX(price), MIN(price), AVG(price), STD(price), SUM(quantity)  FROM products;

mysql> SELECT productCode, MAX(price) AS `Highest Price`, MIN(price) AS `Lowest Price` FROM products

       GROUP BY productCode;
```

HAVING clause

HAVING is similar to WHERE, but it can operate on the GROUP BY aggregate functions; whereas WHERE operates only on columns. The AVG() function returns the average value of a numeric column.

Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value
```

```
mysql> SELECT productCode AS `Product Code`,
         COUNT(*) AS `Count`,
          FROM products
          GROUP BY productCode
          HAVING Count >=3;
```

String Manipulation Opeartions

String Functions

Function	Description
ASCII	Returns the ASCII value for the specific character
CHAR	Returns the character based on the ASCII code
CHARINDEX	Returns the position of a substring in a string
CONCAT	Adds two or more strings together
Concat with +	Adds two or more strings together
CONCAT_WS	Adds two or more strings together with a separator
DATALENGTH	Returns the number of bytes used to represent an expression
DIFFERENCE	Compares two SOUNDEX values, and returns an integer value
FORMAT	Formats a value with the specified format
LEFT	Extracts a number of characters from a string (starting from left)
LEN	Returns the length of a string
LOWER	Converts a string to lower-case

LTRIM	Removes leading spaces from a string
NCHAR	Returns the Unicode character based on the number code
PATINDEX	Returns the position of a pattern in a string
QUOTENAME	Returns a Unicode string with delimiters added to make the string a valid SQL Server delimited identifier
REPLACE	Replaces all occurrences of a substring within a string, with a new substring
REPLICATE	Repeats a string a specified number of times
REVERSE	Reverses a string and returns the result
RIGHT	Extracts a number of characters from a string (starting from right)
RTRIM	Removes trailing spaces from a string
SOUNDEX	Returns a four-character code to evaluate the similarity of two strings
SPACE	Returns a string of the specified number of space characters
STR	Returns a number as string
STUFF	Deletes a part of a string and then inserts another part into the string, starting at a specified position
SUBSTRING	Extracts some characters from a string
TRANSLATE	Returns the string from the first argument after the characters specified in the second argument are translated into the characters specified in the third argument.
TRIM	Removes leading and trailing spaces (or other specified characters) from a string
UNICODE	Returns the Unicode value for the first character of the input expression
UPPER	Converts a string to upper-case

ASCII(str)

Returns the numeric value of the leftmost character of the string str. Returns 0 if str is the empty string. Returns NULL if str is NULL. ASCII() works for 8-bit characters.

```
mysql> SELECT ASCII('2');
        -> 50
mysql> SELECT ASCII(2);
        -> 50
mysql> SELECT ASCII('dx');
        -> 100
```

See also the ORD() function.

BIN(N)

Returns a string representation of the binary value of N, where N is a longlong (BIGINT) number. This is equivalent to CONV(N,10,2). Returns NULL if N is NULL.

```
mysql> SELECT BIN(12);
        -> '1100'
```

BIT_LENGTH(str)

Returns the length of the string str in bits.

```
mysql> SELECT BIT_LENGTH('text');
        -> 32
```

CHAR(N,... [USING charset_name])

CHAR() interprets each argument N as an integer and returns a string consisting of the characters given by the code values of those integers. NULL values are skipped.

```
mysql> SELECT CHAR(77,121,83,81,'76');
        -> 'MySQL'
mysql> SELECT CHAR(77,77.3,'77.3');
        -> 'MMM'
```

CHAR() arguments larger than 255 are converted into multiple result bytes. For example, CHAR(256) is equivalent to CHAR(1,0), and CHAR(256*256) is equivalent to CHAR(1,0,0):

HEX

```
mysql> SELECT HEX(CHAR(1,0)), HEX(CHAR(256));
+----------------+----------------+
| HEX(CHAR(1,0)) | HEX(CHAR(256)) |
+----------------+----------------+
| 0100           | 0100           |
+----------------+----------------+
mysql> SELECT HEX(CHAR(1,0,0)), HEX(CHAR(256*256));
+------------------+--------------------+
| HEX(CHAR(1,0,0)) | HEX(CHAR(256*256)) |
+------------------+--------------------+
| 010000           | 010000             |
+------------------+--------------------+
```

By default, CHAR() returns a binary string. To produce a string in a given character set, use the optional USING clause:

```
mysql> SELECT CHARSET(CHAR(X'65')), CHARSET(CHAR(X'65' USING utf8));
+----------------------+---------------------------------+
| CHARSET(CHAR(X'65')) | CHARSET(CHAR(X'65' USING utf8)) |
+----------------------+---------------------------------+
| binary               | utf8                            |
+----------------------+---------------------------------+
```

If USING is given and the result string is illegal for the given character set, a warning is issued. Also, if strict SQL mode is enabled, the result from CHAR() becomes NULL.

UPPER

To convert the text returned by a SELECT statement to upper case:

```
mysql> select UPPER(prod_name) from product where prod_code=4;
+--------------------------+
| UPPER(prod_name)         |
+--------------------------+
| MICROSOFT 10-20 KEYBOARD |
+--------------------------+
1 row in set (0.00 sec)
```

To know the length of a column value:

```
mysql> SELECT prod_name, LENGTH(prod_name) FROM product where prod_code=4;
+--------------------------+-------------------+
| prod_name                | LENGTH(prod_name) |
+--------------------------+-------------------+
| Microsoft 10-20 Keyboard |                24 |
+--------------------------+-------------------+
1 row in set (0.03 sec)
```

To replace one word with another:

```
mysql> SELECT  REPLACE(prod_name, 'Microsoft', 'Apple') FROM product where prod_code=4;
+------------------------------------------+
| REPLACE(prod_name, 'Microsoft', 'Apple') |
+------------------------------------------+
| Apple 10-20 Keyboard                     |
+-----------------------------------------+
1 row in set (0.00 sec)
```

Conclusion:

Therefore we have executed select command with various predicates and options using Mysql and observed the required results.

Outcome:

Students will be able to execute SQL queries and verify the results.

Oral Questions:

1. How to select few columns instead of all columns?
2. Finding how many rows in tables
3. SQL Query to find second highest salary of Employee
4. SQL Query to find Max Salary from each department.
5. Write SQL Query to display the current date.
6. How do you find all employees which are also manager? .

Experiment No: 6

Aim: Nested queries and complex queries

Objective :

To understand various ways to retrieve data using nested and complex queries in SQL.

Software Requirement: MySQL 5.6 on ubantu 16.04

Theory:

Students will write theory of Nested queries

Perform all types of queries on your own database tables and attach printout.

NESTED QUERIES:-
Nested queries are queries that contain another complete SELECT statements nested within it, that is, in the WHERE clause.
The nested SELECT statement is called an “inner query” or an “inner SELECT.” The main query is called “outer SELECT” or “outer query.”

Many nested queries are equivalent to a simple query using JOIN operation. The use of nested query in this case is to avoid explicit coding of JOIN which is a very expensive database operation and to improve query performance. However, in many cases, the use of nested queries is necessary and cannot be replaced by a JOIN operation.

```
mysql> SELECT s1 FROM t1 WHERE s1 IN  (SELECT s1 FROM t2);
```

```
mysql> SELECT suppliers.name from suppliers
       WHERE suppliers.supplierID
         NOT IN (SELECT DISTINCT supplierID from products_suppliers);
```

A subquery may return a scalar, a single column, a single row, or a table. You can use comparison operator (e.g., '=', '>') on scalar, IN or NOT IN for single row or column, EXISTS or NOT EXIST to test for empty set.

INSERT|UPDATE|DELETE with Subquery

You can also use a subquery with other SQL statements such as INSERT, DELETE, or UPDATE. For example,

```
-- Supplier 'QQ Corp' now supplies 'Pencil 6B'

-- You need to put the SELECT subqueies in parentheses

mysql> INSERT INTO products_suppliers VALUES (
          (SELECT productID  FROM products  WHERE name = 'Pencil 6B'),
          (SELECT supplierID FROM suppliers WHERE name = 'QQ Corp'));

-- Supplier 'QQ Copr' no longer supplies any item

mysql> DELETE FROM products_suppliers
       WHERE supplierID = (SELECT supplierID FROM suppliers WHERE name = 'QQ Corp');
```

```
mysql> select * from employee where dno in(select dept_no from dept);

mysql> select * from emp where salary=(select max(salary) from emp);

mysql> select * from emp where salary=(select min(salary) from emp);

mysql> select *   from customers  where id in (select id  from customers  where salary > 4500);
```

Below Examples are for your reference.

In printout Write English statement what you want to retrieve and then SQL query and output. Like below example.

Q.1) Retrieve email ID of guardian whose balance fees is remaining

mysql> select emailID from guardian where g_name in(select guardian_name from fees where balance_fees!=0);

```
+-----------------------+
| emailID               |
+-----------------------+
| ss1248@gmail.com      |
| kiran.jha@hotmail.com |
| meenal.c@somaiya.edu  |
+-----------------------+
3 rows in set (0.01 sec)
```

Q.2) Retreive total fees of students whose guardians are their grandparents.
mysql> select total_fees from fees where guardian_name in(select g_name from guardian where relation_with_child like "grand%");

```
+------------+
| total_fees |
+------------+
|   12000.00 |
|   14000.00 |
+------------+
2 rows in set (0.02 sec)
```

Q.3) Retreive fee details of students whose guardians are not their parents.
mysql> select * from fees where guardian_name in(select g_name from guardian where relation_with_child!="mother" AND relation_with_child!="father");

```
+------------+-----------+--------------+-------------------+
| total_fees | paid_fees | balance_fees | guardian_name     |
+------------+-----------+--------------+-------------------+
|   12000.00 |  12000.00 |         0.00 | himani gandre     |
|   15000.00 |  14000.00 |      1000.00 | meenal chaturvedi |
|   14000.00 |  14000.00 |         0.00 | shankar sharma    |
|   16000.00 |  12000.00 |      4000.00 | sheetal shah      |
+------------+-----------+--------------+-------------------+
4 rows in set (0.00 sec)
```

Q.4) Retreive guardian's email ID whose child has fees less than Rs. 15000
mysql> select emailID from guardian where g_name in(select guardian_name from fees where total_fees<15000);

```
+-----------------------+
| emailID               |
+-----------------------+
| shankar.s@gmail.com   |
| kiran.jha@hotmail.com |
| himani.g@somaiya.edu  |
+-----------------------+
3 rows in set (0.00 sec)
```

Q.5) Find contact details of guardians who have paid the complete fees of their child

mysql> select G.g_name,G.emailID,P.ph_no from guardian as G,g_ph_no as P where G.g_name in(select guardian_name from fees where balance_fees=0) AND G.g_name=P.g_name;

```
+----------------+----------------------+---------+
| g_name         | emailID              | ph_no   |
+----------------+----------------------+---------+
| shankar sharma | shankar.s@gmail.com  | 2895654 |
| himani gandre  | himani.g@somaiya.edu | 2585106 |
+----------------+----------------------+---------+
2 rows in set (0.00 sec)
```

Q.6) Find name and address of girls whose grandparents are their guardian.
mysql> select name,address from child where guardian_name in (select g_name from guardian where relation_with_child like "grand%") and gender="f";

```
+----------------+----------------------------------+
| name           | address                          |
+----------------+----------------------------------+
| jyotika gandre | E/102, Ravi Estate, Devdaya Nagar |
| shreya sharma  | C/402, Ahmed Nagar, Charai        |
+----------------+----------------------------------+
2 rows in set (0.00 sec)
```

Q.7) Find guardian details of children born in 2015.
mysql> select * from guardian where g_name in(select guardian_name from child where dob like "2015%");

```
+---------------+----------------------+---------------------+
| g_name        | emailID              | relation_with_child |
+---------------+----------------------+---------------------+
| sheetal shah  | ss1248@gmail.com     | aunt                |
| kiran jha     | kiran.jha@hotmail.com | mother              |
| himani gandre | himani.g@somaiya.edu  | grandmother         |
+---------------+----------------------+---------------------+
3 rows in set, 1 warning (0.03 sec)
```

Q.8) Display child details who are taught by the teacher "vrushali"
mysql> select * from child where c_id in(select child_id from handles where staff_id in(select s_ID from staff where name ="vrushali"));

```
+------+--------------------+-------------------+-------------------+------------+--------+-------------------------------------+---------------------+-------------------+
| c_id | name               | father_name       | mother_name       | DOB        | gender | address                             | medical_details     | guardian_name     |
+------+--------------------+-------------------+-------------------+------------+--------+-------------------------------------+---------------------+-------------------+
|   46 | shubham chaturvedi | dinesh chaturvedi | sayali chaturvedi | 2016-12-12 | m      | 165, Dattaniwas, Near Talathi Office | pineapple allergy   | meenal chaturvedi |
|   12 | jyotika gandre     | girish gandre     | girija gandre     | 2015-01-11 | f      | E/102, Ravi Estate, Devdaya Nagar    |
lactose intolerance | himani gandre     |
+------+--------------------+-------------------+-------------------+------------+--------+-------------------------------------+--------
-------------+-------------------+
2 rows in set (0.00 sec)
```

Q.9) Display details of staff who teach children born in 2015
mysql> select * from staff where s_ID in(select staff_id from handles where child_id in(select c_id from child where dob like "2016%"));

```
+------+----------+------+--------+----------+----------+---------------+
| s_ID | name     | age  | gender | position | salary   | qualification |
+------+----------+------+--------+----------+----------+---------------+
|   45 | vrushali |   25 | f      | teacher  | 40000.00 | Diploma, B.Ed |
|   32 | rama roy |   40 | f      | teacher  | 45000.00 | MA BEd        |
+------+----------+------+--------+----------+----------+---------------+
2 rows in set, 1 warning (0.00 sec)
```

Q.10) Display details of children who are taught by teachers that have completed their diploma.
mysql> select * from child where c_id in(select child_id from handles where staff_id in(select s_ID from staff where qualification like "%diploma%"));

```
+------+--------------------+-------------------+-------------------+------------+--------+-------------------------------------+--------
-------------+-------------------+
| c_id | name               | father_name       | mother_name       | DOB        | gender | address                             |
medical_details     | guardian_name     |
+------+--------------------+-------------------+-------------------+------------+--------+-------------------------------------+--------
-------------+-------------------+
|   46 | shubham chaturvedi | dinesh chaturvedi | sayali chaturvedi | 2016-12-12 | m      | 165, Dattaniwas, Near Talathi
Office | pineapple allergy   | meenal chaturvedi |
|   12 | jyotika gandre     | girish gandre     | girija gandre     | 2015-01-11 | f      | E/102, Ravi Estate, Devdaya Nagar    |
lactose intolerance | himani gandre     |
+------+--------------------+-------------------+-------------------+------------+--------+-------------------------------------+--------
-------------+-------------------+
2 rows in set (0.00 sec)
```

Q.11) Retreive child details group by child ID
mysql> select c_id,name from child group by c_id;

```
+------+--------------------+
| c_id | name               |
+------+--------------------+
|    5 | darshan roy        |
|    7 | mehek jha          |
|   12 | jyotika gandre     |
|   46 | shubham chaturvedi |
|  146 | shreya sharma      |
+------+--------------------+
5 rows in set (0.01 sec)
```

Q.12) Retreive medical details of children having ID greater than 10

mysql> select c_id,name,medical_details from child having c_id>10;

```
+------+--------------------+---------------------+
| c_id | name               | medical_details     |
+------+--------------------+---------------------+
|   12 | jyotika gandre     | lactose intolerance |
|   46 | shubham chaturvedi | pineapple allergy   |
|  146 | shreya sharma      | Egg allergy         |
+------+--------------------+---------------------+
3 rows in set (0.00 sec)
```

Q.13) Update: assign students taught by "monica jones" to "rama roy"

mysql> update handles set staff_id = (select s_id from staff where name ="rama roy") where staff_id=(select s_id from staff where name="monica jones");

```
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

mysql> select * from handles;

```
+----------+----------+
| staff_id | child_id |
+----------+----------+
|       32 |        5 |
|       32 |      146 |
|       32 |        5 |
|       45 |       46 |
|       45 |       12 |
+----------+----------+
5 rows in set (0.00 sec)
```

Q.14)Delete: un-assign students taught by the teacher "vrushali"

mysql> delete from handles where staff_id in(select s_ID from staff where name="vrushali");

```
Query OK, 2 rows affected (0.04 sec)
```

mysql> select * from handles;

```
+----------+----------+
| staff_id | child_id |
+----------+----------+
|       32 |        5 |
|       32 |      146 |
|       32 |        5 |
+----------+----------+
3 rows in set (0.00 sec)
```

Q.15)Update salary of staff who are currently teaching some children to Rs. 47000

mysql> update staff set salary=47000 where s_ID in(select distinct staff_id from handles);

```
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

mysql> select * from staff;

```
+------+--------------+------+--------+------------+----------+-----------------+
| s_ID | name         | age  | gender | position   | salary   | qualification   |
+------+--------------+------+--------+------------+----------+-----------------+
|   10 | ritesh       |   32 | m      | sweeper    | 15000.00 | SSC             |
|   25 | monica jones |   35 | f      | teacher    | 47000.00 | MA Honors, B.Ed |
|   32 | rama roy     |   40 | f      | teacher    | 47000.00 | MA BEd          |
|   45 | vrushali     |   25 | f      | teacher    | 40000.00 | Diploma, B.Ed   |
|   73 | milind soman |   35 | m      | accountant | 56000.00 | CA              |
+------+--------------+------+--------+------------+----------+-----------------+
5 rows in set (0.00 sec)
```

Conclusion: Students are able to perform nested queries on their database tables and observe the expected results.

.

Outcome: Students will be able to write complex queries.

Experiment No: 7

Aim: Perform Join operations

Objective :

To understand various ways to retrieve data using various join operation queries in SQL

Software Requirement: MySQL 5.6 on ubantu 16.04

Students will write theory of Join operations in SQL

Perform all types of queries on your own database tables and attach printout.

JOIN :- Return rows when there is at least one match in both tables

➡ CROSS-JOIN :- A cross-join between two tables takes the data from each row in table1 and joins it to the data from each row in table2. .

Syntax:

SELECT <column_name> FROM <table1>, <table2>

mysql> **SELECT * FROM t1,t2;**

EQUI-JOIN OR INNER JOIN :- In the equi-join the comparison we are making between two columns is that they match the same value. We can use this method to select certain fields from both tables and only the correct rows will be joined together.

Syntax:

SELECT <column_name>
FROM <Table1>, <Table2>
WHERE (Table1.column = Table2.column)

mysql> **SELECT * FROM t1,t2 where t1.id = t2.id;**

the following are equivalent:
mysql> **SELECT * FROM t1 INNER JOIN t2 ON t1.id = t2.id;**

mysql> **SELECT * FROM t1 JOIN t2 ON t1.id = t2.id;**

You can use USING clause if the join-columns have the same name

mysql> **SELECT * FROM t1 INNER JOIN t2 USING (id);**

Use WHERE instead of ON

mysql> **SELECT * FROM t1 INNER JOIN t2 WHERE t1.id = t2.id;**

Use "commas" operator to join

mysql> **SELECT * FROM t1, t2 WHERE t1.id = t2.id;**

OUTER JOINS :-INNER JOIN with constrain (ON or USING) produces rows that are found in both tables. On the other hand, OUTER JOIN can produce rows that are in one table, but not in another table.
There are three kinds of OUTER JOINs:

LEFT JOIN: Return all rows from the left table, even if there are no matches in the right table
RIGHT JOIN: Return all rows from the right table, even if there are no matches in the left table
FULL JOIN: Return rows when there is a match in one of the tables
LEFT OUTER JOIN Keyword

The LEFT JOIN keyword returns all rows from the left table (table_name1), even if there are no matches in the right table (table_name2).

:
mysql> **SELECT * FROM t1 LEFT JOIN t2 ON t1.id = t2.id;**

mysql> **SELECT * FROM t1 LEFT OUTER JOIN t2 ON t1.id = t2.id;**

mysql> **SELECT * FROM t1 LEFT JOIN t2 USING (id);** -- join-columns have same name

```
mysql> SELECT t1.id, t1.desc
           FROM t1 LEFT JOIN t2 USING (id)
```

WHERE t2.id IS NULL;

RIGHT OUTER JOIN Keyword

The RIGHT JOIN keyword returns all the rows from the right table (table_name2), even if there are no matches in the left table (table_name1).
mysql> **SELECT * FROM t1 LEFT JOIN t2 ON t1.id = t2.id;**

mysql> **SELECT * FROM t1 RIGHT OUTER JOIN t2 ON t1.id = t2.id;**

mysql> **SELECT * FROM t1 RIGHT JOIN t2 USING (id);** -- join-columns have same name

FULL OUTER JOIN Keyword

The FULL JOIN keyword return rows when there is a match in one of the tables.
SQL FULL JOIN Syntax

mysql> **SELECT * FROM t1 LEFT JOIN t2 ON t1.id = t2.id;**

UNION SELECT * FROM t1 LEFT JOIN t2 ON t1.id = t2.id;

Note: WHERE clause CANNOT be used on OUTER JOIN

example queries.

mysql>select * from employee cross join dept;

OR
mysql> select * from employee inner join dept;

mysql> select * from employee inner join dept on employee.dno=dept.dept_no;

Below Examples are for your reference.

In printout Write English statement what you want to retrieve and then SQL query and output. Like below example.

Q.1) Retrieve email ID of guardian whose balance fees is remaining

```
mysql> select emailID from guardian as g,fees as f where g.g_name=f.guardian_name and balance_fees!=0;
+-----------------------+
| emailID               |
+-----------------------+
| ss1248@gmail.com      |
| kiran.jha@hotmail.com |
| meenal.c@somaiya.edu  |
+-----------------------+
3 rows in set (0.13 sec)
```

Q.2) Retreive total fees of students whose guardians are their grandparents.

```
mysql> select total_fees from fees inner join guardian on fees.guardian_name=guardian.g_name and
relation_with_child like "grand%";
+------------+
| total_fees |
+------------+
|   12000.00 |
|   14000.00 |
+------------+
2 rows in set (0.00 sec)
```

Q.3) Retreive fee details of students whose guardians are not their parents.

```
mysql> select guardian_name,total_fees,paid_fees,balance_fees from guardian join fees on
guardian_name=g_name and relation_with_child!="mother" and relation_with_child!="father";
+------------------+------------+-----------+--------------+
| guardian_name    | total_fees | paid_fees | balance_fees |
+------------------+------------+-----------+--------------+
| himani gandre    |   12000.00 |  12000.00 |         0.00 |
| meenal chaturvedi |   15000.00 |  14000.00 |      1000.00 |
| shankar sharma   |   14000.00 |  14000.00 |         0.00 |
| sheetal shah     |   16000.00 |  12000.00 |      4000.00 |
+------------------+------------+-----------+--------------+
4 rows in set (0.00 sec)
```

Q.4) Retreive guardian's email ID whose child has fees less than Rs. 15000

```
mysql> select emailID from guardian equi join fees where g_name=guardian_name and total_fees<15000;
+-----------------------+
| emailID               |
+-----------------------+
| shankar.s@gmail.com   |
| kiran.jha@hotmail.com |
| himani.g@somaiya.edu  |
+-----------------------+
3 rows in set (0.00 sec)
```

Q.5) Find contact details of guardians who have paid the complete fees of their child
mysql> select g.g_name,emailID,ph_no from guardian as g join g_ph_no using (g_name) inner join fees as f on g.g_name=f.guardian_name and balance_fees=0;

```
+----------------+----------------------+---------+
| g_name         | emailID              | ph_no   |
+----------------+----------------------+---------+
| shankar sharma | shankar.s@gmail.com  | 2895654 |
| himani gandre  | himani.g@somaiya.edu | 2585106 |
+----------------+----------------------+---------+
2 rows in set (0.00 sec)
```

Q.6) Find name and address of girls whose grandparents are their guardian.
mysql> select name,address from child cross join **guardian where guardian_name=g_name and relation_with_child like "grand%" and gender="f";**

```
+----------------+----------------------------------+
| name           | address                          |
+----------------+----------------------------------+
| jyotika gandre | E/102, Ravi Estate, Devdaya Nagar |
| shreya sharma  | C/402, Ahmed Nagar, Charai        |
+----------------+----------------------------------+
2 rows in set (0.15 sec)
```

Q.7) Find guardian details of children born in 2015.
mysql> select g_name,emailID from guardian natural join child where g_name=guardian_name and dob like "2015%";

```
+---------------+-----------------------+
| g_name        | emailID               |
+---------------+-----------------------+
| sheetal shah  | ss1248@gmail.com      |
| kiran jha     | kiran.jha@hotmail.com |
| himani gandre | himani.g@somaiya.edu  |
+---------------+-----------------------+
3 rows in set, 1 warning (0.00 sec)
```

Q.8) Display details of all children who might be taught by the teacher “vrushali”

```
mysql> select c_id,c.name from child as c,staff as s,handles where c_id=child_id and staff_id=s_ID and
s.name="vrushali";
+------+---------------+
| c_id | name          |
+------+---------------+
|    5 | darshan roy   |
|  146 | shreya sharma |
|    7 | mehek jha     |
+------+---------------+
3 rows in set (0.00 sec)
mysql> select c_id,c.name from child as c,staff as s,handles where c_id=child_id and staff_id=s_ID and
s.name="vrushali" group by c_id;
+------+---------------+
| c_id | name          |
+------+---------------+
|    5 | darshan roy   |
|    7 | mehek jha     |
|  146 | shreya sharma |
+------+---------------+
3 rows in set (0.00 sec)
```

Q.9) Display details of staff who teach children born in 2015

```
mysql> select * from staff where s_ID in (select staff_id from handles,child where child_id=c_id and dob like
"2016%");
+------+----------+------+--------+----------+----------+---------------+
| s_ID | name     | age  | gender | position | salary   | qualification |
+------+----------+------+--------+----------+----------+---------------+
|   32 | rama roy |   40 | f      | teacher  | 47000.00 | MA BEd        |
+------+----------+------+--------+----------+----------+---------------+
1 row in set, 1 warning (0.07 sec)
```

Q.10) Display details of children who are taught by teachers that have completed their diploma.

```
mysql> select name,father_name,mother_name from child,handles where c_id=child_id and staff_id in(select
s_ID from staff where qualification like "%diploma%");
+---------------+---------------+---------------+
| name          | father_name   | mother_name   |
+---------------+---------------+---------------+
| darshan roy   | vikas roy     | swati roy     |
| shreya sharma | mayank sharma | kapila sharma |
| mehek jha     | mohit jha     | kiran jha     |
+---------------+---------------+---------------+
3 rows in set (0.00 sec)
```

Q.11) Find details of guardian whose total fees may be greater than or equal to Rs. 15000

mysql> select * from guardian left outer join fees on g_name=guardian_name and total_fees>=15000;

```
+-------------------+----------------------+--------------------+------------+-----------+--------------+-------------------+
| g_name            | emailID              | relation_with_child | total_fees | paid_fees | balance_fees | guardian_name     |
+-------------------+----------------------+--------------------+------------+-----------+--------------+-------------------+
| himani gandre     | himani.g@somaiya.edu  | grandmother        |       NULL |      NULL |         NULL | NULL              |
| kiran jha         | kiran.jha@hotmail.com | mother             |       NULL |      NULL |         NULL | NULL              |
| meenal chaturvedi | meenal.c@somaiya.edu  | sister             |   15000.00 |  14000.00 |      1000.00 | meenal chaturvedi |
| shankar sharma    | shankar.s@gmail.com   | grandfather        |       NULL |      NULL |         NULL | NULL              |
| sheetal shah      | ss1248@gmail.com      | aunt               |   16000.00 |  12000.00 |      4000.00 | sheetal shah      |
+-------------------+----------------------+--------------------+------------+-----------+--------------+-------------------+
5 rows in set (0.00 sec)
```

Q.12) Retreive details of all guardians and medical details of children having grandparents as guardian.
mysql> select * from child right outer join guardian on guardian_name=g_name and relation_with_child like "grand%";

```
+------+----------------+---------------+---------------+------------+--------+----------------------------------+---------------------
+----------------+------------------+----------------------+---------------------+
| c_id | name           | father_name   | mother_name   | DOB        | gender | address                          | medical_details     |
guardian_name  | g_name           | emailID              | relation_with_child |
+------+----------------+---------------+---------------+------------+--------+----------------------------------+---------------------
+----------------+------------------+----------------------+---------------------+
|   12 | jyotika gandre | girish gandre | girija gandre | 2015-01-11 | f      | E/102, Ravi Estate, Devdaya Nagar | lactose
intolerance | himani gandre  | himani gandre     | himani.g@somaiya.edu  | grandmother        |
| NULL | NULL           | NULL          | NULL          | NULL       | NULL   | NULL                             | NULL                |
NULL           | kiran jha        | kiran.jha@hotmail.com | mother             |
| NULL | NULL           | NULL          | NULL          | NULL       | NULL   | NULL                             | NULL                |
NULL           | meenal chaturvedi | meenal.c@somaiya.edu  | sister             |
|  146 | shreya sharma  | mayank sharma | kapila sharma | 2016-06-15 | f      | C/402, Ahmed Nagar, Charai        | Egg
allergy        | shankar sharma | shankar sharma    | shankar.s@gmail.com   | grandfather        |
| NULL | NULL           | NULL          | NULL          | NULL       | NULL   | NULL                             | NULL                |
NULL           | sheetal shah     | ss1248@gmail.com      | aunt               |
+------+----------------+---------------+---------------+------------+--------+----------------------------------+---------------------
+----------------+------------------+----------------------+---------------------+
5 rows in set (0.00 sec)
```

Conclusion: Students are able to perform queries on different types of join commands and obserbed the results..

Outcome: Students will be able to write various join queries in SQL.

Experiment No: 8

Aim: Views and Triggers

Objective :

- To create views on tables.
- To create triggers on the database.

Software Requirement: MySQL 5.6 on Ubuntu 16.04

In MySQL, a trigger is a set of SQL statements that is invoked automatically when a change is made to the data on the associated table. A trigger can be defined to be invoked either BEFORE or AFTER data change by INSERT, UPDATE and DELETE statements. MySQL allows you to define maximum six triggers for each table.

- BEFORE INSERT
- AFTER INSERT
- BEFORE UPDATE
- AFTER UPDATE
- BEFORE DELETE
- AFTER DELETE

When you use a MySQL statement that changes the data but does not use INSERT, DELETE or UPDATE statement, the trigger is not invoked.
For example, the TRUNCATE statement removes the whole data of a table but does not invoke the trigger.

Triggers are defined for a table must have unique name.
Following illustrates the syntax of the CREATE TRIGGER statement:

```
CREATE TRIGGER trigger_name  trigger_time  trigger_event
 ON table_name
 FOR EACH ROW
 BEGIN
 ...
 END
```

- Trigger activation time can be BEFORE or AFTER.
- ➡ Trigger event can be INSERT, UPDATE and DELETE.
- ➡ To define a trigger that is invoked by multiple events, you have to define multiple triggers, one for each event.
- A trigger must be associated with a specific table.

1.The SQL code is placed between BEGIN and END keywords.

6.The OLD keyword refers to the existing record before you change the data and the NEW keyword refers to the new row after you change the data.

Triggers can be viewed as similar to stored procedures in that both consist of procedural logic that is stored at the database level. Stored procedures, however, are not event-drive and are not attached to a specific table as triggers are. Stored procedures are explicitly executed by invoking a CALL to the procedure while triggers are implicitly executed. In addition, triggers can also execute stored procedures.

Example:

Students will write theory in their writeups for Triggers and Views

Students will execute the following commands(Trigger and View both) with different data and observe the result of queries and attach this printout .

```
 sql> create table exam
    -> (rollno int(5) primary key,
    -> name varchar(10),
    -> dbms int(5),
    -> cg int(5),
    -> os int(5))

Query OK, 0 rows affected (0.41 sec)

mysql> create table result
    -> (rollno int (5),
    -> total int(10));
Query OK, 0 rows affected (0.32 sec)

mysql> insert into exam values(10,'sarita',40,50,55);
Query OK, 1 row affected (0.07 sec)

mysql> insert into exam values(11,'sumit',47,66,55);
Query OK, 1 row affected (0.05 sec)

mysql> delimiter //
mysql> create trigger t3 after insert on exam for each row   begin insert into result set
rollno=new.rollno,total=(new.dbms+new.cg+new.os); end//
Query OK, 0 rows affected (0.09 sec)

mysql> select * from exam;
    -> //
+--------+--------+------+------+------+
| rollno | name   | dbms | cg   | os   |
+--------+--------+------+------+----
|     10 | sarita |   40 |   50 |   55 |
|     11 | sumit  |   47 |   66 |   55 |
+--------+--------+------+------+----
2 rows in set (0.00 sec)

mysql> select * from result;
    -> //
Empty set (0.00 sec)

mysql> insert into exam values(13,'meet',60,46,65);
    -> //
Query OK, 1 row affected (0.05 sec)

mysql> select * from exam;
    -> //
+--------+--------+------+------+------+
| rollno | name   | dbms | cg   | os   |
+--------+--------+------+------+------+
|     10 | sarita |   40 |   50 |   55 |
|     11 | sumit  |   47 |   66 |   55 |
|     13 | meet   |   60 |   46 |   65 |
+--------+--------+------+------+------+
3 rows in set (0.00 sec)

mysql> select * from result;
    -> //
+--------+-------+
| rollno | total |
+--------+-------+
|     13 |   171 |
+--------+-------+
1 row in set (0.00 sec)

mysql> update exam set dbms=80 where rollno=10;
    -> //
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> create table backup_exam  (rollno int(5) not null,
    -> name varchar(10),
    -> dbms int(5),
    -> cg int(5),
    -> os int(5));
    -> //
Query OK, 0 rows affected (0.30 sec)
```

```
mysql> create trigger t1 before update on exam for each row   begin insert into backup_exam  set
rollno=old.rollno,dbms=old.dbms,cg=old.cg,os=old.os; end// Query OK, 0 rows affected (0.09 sec)

mysql> select * from backup_exam;
    -> //
Empty set (0.00 sec)

mysql> update exam set dbms=70 where rollno=10;
    -> //
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from backup_exam;
    -> //
+--------+------+------+------+------+
| rollno | name | dbms | cg   | os   |
+--------+------+------+------+------+
|     10 | NULL |   80 |   50 |   55 |
+--------+------+------+------+------+
1 row in set (0.00 sec)

mysql> select * from exam;
    -> //
+--------+--------+------+------+------+
| rollno | name   | dbms | cg   | os   |
+--------+--------+------+------+------+
|     10 | sarita |   70 |   50 |   55 |
|     11 | sumit  |   47 |   66 |   55 |
|     13 | meet   |   60 |   46 |   65 |
+--------+--------+------+------+------+
3 rows in set (0.00 sec)
```

View

A view is a virtual table that contains no physical data. It provide an alternative way to look at the data.

Another way

A VIEW is a virtual table, through which a selective portion of the data from one or more tables can be seen. Views do not contain data of their own. They are used to restrict access to the database or to hide data complexity. A view is stored as a SELECT statement in the database. DML operations on a view like INSERT, UPDATE, DELETE affects the data in the original table upon which the view is based.

The Syntax to create a sql view is

```
CREATE VIEW view_name
AS
SELECT column_list
FROM table_name [WHERE condition];
```

- ***view_name*** is the name of the VIEW.
- The SELECT statement is used to define the columns and rows that you want to display in the view.

For Example: to create a view on the product table the sql query would be like

```
CREATE VIEW view_product
AS
SELECT product_id, product_name
FROM product;
```

Drop SQL VIEW

Once a SQL VIEW has been created, you can drop it with the SQL DROP VIEW Statement.

Syntax

The syntax for the SQL DROP VIEW Statement is:

```
DROP VIEW view_name;
```

Students will execute the following commands with different data and observe the result of queries and attach this printout .

```
mysql> use company;
Database changed

mysql> create table department
    -> (dnum int(10) primary key,
    -> dname varchar(10) not null,
    -> location varchar(20) default 'Mumbai');
Query OK, 0 rows affected (0.50 sec)

mysql> create table employee
    -> (eid int(10) primary key,
    -> ename varchar(10) not null,
    -> address varchar(20) not null,
    -> salary decimal(10,2),
    -> dno int(10),
    -> foreign key (dno) references department(dnum));
Query OK, 0 rows affected (0.31 sec)

mysql> insert into department values(1,'comp','mumbai');
Query OK, 1 row affected (0.08 sec)

mysql> insert into department values(1,'extc','thane');
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
mysql>
mysql> insert into department values(2,'extc','thane');
Query OK, 1 row affected (0.13 sec)

mysql> insert into employee values(101,'sarita','mumbai',50000,2);
Query OK, 1 row affected (0.11 sec)

mysql> insert into employee values(102,'amit','mumbai',670000,2);
Query OK, 1 row affected (0.14 sec)

mysql> insert into employee values(103,'smith','Thane',34000,1);
Query OK, 1 row affected (0.06 sec)


mysql> create view v-emp as select * from employee;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near '-emp
as select * from employee' at line 1

mysql> create view vmp as select * from employee;
Query OK, 0 rows affected (0.13 sec)

mysql> select * from vmp;
+-----+--------+---------+-----------+------+
| eid | ename  | address | salary    | dno  |
+-----+--------+---------+-----------+------+
| 101 | sarita | mumbai  |  50000.00 |    2 |
| 102 | amit   | mumbai  | 670000.00 |    2 |
| 103 | smith  | Thane   |  34000.00 |    1 |
+-----+--------+---------+-----------+------+
3 rows in set (0.13 sec)

mysql> select * from vmp where dno=2;
+-----+--------+---------+-----------+------+
| eid | ename  | address | salary    | dno  |
+-----+--------+---------+-----------+------+
| 101 | sarita | mumbai  |  50000.00 |    2 |
| 102 | amit   | mumbai  | 670000.00 |    2 |
+-----+--------+---------+-----------+------+
2 rows in set (0.08 sec)

mysql> select * from vmp where ename='amit';
+-----+-------+---------+-----------+------+
| eid | ename | address | salary    | dno  |
+-----+-------+---------+-----------+------+
| 102 | amit  | mumbai  | 670000.00 |    2 |
+-----+-------+---------+-----------+------+
1 row in set (0.02 sec)

mysql> insert into vmp values(103,'john','mumbai',6000,1);
ERROR 1062 (23000): Duplicate entry '103' for key 'PRIMARY'
mysql> insert into vmp values(104,'john','mumbai',6000,1);
Query OK, 1 row affected (0.34 sec)

mysql> select * from vmp;
+-----+--------+---------+-----------+------+
| eid | ename  | address | salary    | dno  |
+-----+--------+---------+-----------+------+
| 101 | sarita | mumbai  |  50000.00 |    2 |
| 102 | amit   | mumbai  | 670000.00 |    2 |
| 103 | smith  | Thane   |  34000.00 |    1 |
| 104 | john   | mumbai  |   6000.00 |    1 |
+-----+--------+---------+-----------+------+
4 rows in set (0.00 sec)

mysql> select * from employee;
+-----+--------+---------+-----------+------+
| eid | ename  | address | salary    | dno  |
+-----+--------+---------+-----------+------+
| 101 | sarita | mumbai  |  50000.00 |    2 |
| 102 | amit   | mumbai  | 670000.00 |    2 |
| 103 | smith  | Thane   |  34000.00 |    1 |
| 104 | john   | mumbai  |   6000.00 |    1 |
+-----+--------+---------+-----------+------+
4 rows in set (0.00 sec)

mysql> update employee set address='Navi mumbai' where eid=104;
Query OK, 1 row affected (0.34 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from employee;
+-----+--------+-------------+-----------+------+
| eid | ename  | address     | salary    | dno  |
+-----+--------+-------------+-----------+------+
| 101 | sarita | mumbai      |  50000.00 |    2 |
| 102 | amit   | mumbai      | 670000.00 |    2 |
| 103 | smith  | Thane       |  34000.00 |    1 |
| 104 | john   | Navi mumbai |   6000.00 |    1 |
+-----+--------+-------------+-----------+------+
4 rows in set (0.00 sec)

mysql> select * from vmp;
+-----+--------+-------------+-----------+------+
| eid | ename  | address     | salary    | dno  |
+-----+--------+-------------+-----------+------+
| 101 | sarita | mumbai      |  50000.00 |    2 |
| 102 | amit   | mumbai      | 670000.00 |    2 |
| 103 | smith  | Thane       |  34000.00 |    1 |
| 104 | john   | Navi mumbai |   6000.00 |    1 |
+-----+--------+-------------+-----------+------+
4 rows in set (0.00 sec)

mysql> create view vemp1 as select eid,ename,salary,dname from employee, departm
ent where employee.dno=department.dnum;
Query OK, 0 rows affected (0.13 sec)

mysql> select * from vemp1;
+-----+--------+-----------+-------+
| eid | ename  | salary    | dname |
+-----+--------+-----------+-------+
| 103 | smith  |  34000.00 | comp  |
| 104 | john   |   6000.00 | comp  |
| 101 | sarita |  50000.00 | extc  |
| 102 | amit   | 670000.00 | extc  |
+-----+--------+-----------+-------+
4 rows in set (0.08 sec)

mysql> update vemp1 set salary=55000 where eid=101;
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from vemp1;
+-----+--------+-----------+-------+
| eid | ename  | salary    | dname |
+-----+--------+-----------+-------+
| 103 | smith  |  34000.00 | comp  |
| 104 | john   |   6000.00 | comp  |
| 101 | sarita |  55000.00 | extc  |
| 102 | amit   | 670000.00 | extc  |
+-----+--------+-----------+-------+
4 rows in set (0.00 sec)

mysql> select * from employee;
+-----+--------+-------------+-----------+------+
| eid | ename  | address     | salary    | dno  |
+-----+--------+-------------+-----------+------+
| 101 | sarita | mumbai      |  55000.00 |    2 |
| 102 | amit   | mumbai      | 670000.00 |    2 |
| 103 | smith  | Thane       |  34000.00 |    1 |
| 104 | john   | Navi mumbai |   6000.00 |    1 |
+-----+--------+-------------+-----------+------+
4 rows in set (0.00 sec)

mysql> delete from vemp1 where eid=103;
ERROR 1395 (HY000): Can not delete from join view 'company.vemp1'
mysql> delete from vemp1 where eid=101;
ERROR 1395 (HY000): Can not delete from join view 'company.vemp1'
mysql> update department set dname='account' where dnum=1;
Query OK, 1 row affected (0.06 sec)
```

```
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from employee;
+-----+--------+-------------+-----------+------+
| eid | ename  | address     | salary    | dno  |
+-----+--------+-------------+-----------+------+
| 101 | sarita | mumbai      |  55000.00 |    2 |
| 102 | amit   | mumbai      | 670000.00 |    2 |
| 103 | smith  | Thane       |  34000.00 |    1 |
| 104 | john   | Navi mumbai |   6000.00 |    1 |
+-----+--------+-------------+-----------+------+
4 rows in set (0.00 sec)

mysql> select * from vemp1;
+-----+--------+-----------+---------+
| eid | ename  | salary    | dname   |
+-----+--------+-----------+---------+
| 103 | smith  |  34000.00 | account |
| 104 | john   |   6000.00 | account |
| 101 | sarita |  55000.00 | extc    |
| 102 | amit   | 670000.00 | extc    |
+-----+--------+-----------+---------+
4 rows in set (0.00 sec)

mysql> select * from department;
+------+---------+----------+
| dnum | dname   | location |
+------+---------+----------+
|    1 | account | mumbai   |
|    2 | extc    | thane    |
+------+---------+----------+
2 rows in set (0.00 sec)
```

Conclusion:

- Thus we have learned how to create a trigger in MySQL and how to develop a trigger to audit the changes of the table.

- Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables, or another view.

Outcome:

- Students will be able to write a trigger for the automatically generate derived column values , prevent invalid transactions, enforce complex security authorizations etc.
- Students are able to create views and perform all possible operations on views. Also understand how to restrict access to the data such that a user can see and (sometimes) modify exactly what they need and no more.

Oral Questions:

1. Why do we use views?
2. What is a view in a database management system?
3. Can you update the data in a SQL VIEW?
4. Does the SQL View exist if the table is dropped from the database?
5. Explain Trigger and trigger types?
6. What are triggers? What are they used for?
7. Which statement is used to remove a trigger?
8. Write syntax for Creating triggers and Dropping Triggers along with example for each of them.
9. What are the levels of triggers supported by MySQL?

Experiment No: 10

Aim: Transaction and concurrency control.

Software Requirement: MySQL server 5.7 on Ubantu16.04

Objective: To create transactions on database and check for concurrency control.

Theory:

A *transaction* groups a set of operations into a unit that meets the ACID test:

- ➡ Atomicity: If all the operations succeed, changes are *committed* to the database. If any of the operations fails, the entire transaction is *rolled back*, and no change is made to the database. In other words, there is no partial update.
- ➡ Consistency: A transaction transform the database from one consistent state to another consistent state.
- ➡ Isolation: Changes to a transaction are not visible to another transaction until they are committed.
- ➡ Durability: Committed changes are durable and never lost.

A *atomic transaction* is a set of SQL statements that either ALL succeed or ALL fail. Transaction is important to ensure that there is no *partial* update to the database, given an atomic of SQL statements. Transactions are carried out via COMMIT and ROLLBACK.

· ***Example***

```
mysql> CREATE TABLE accounts (
        name     VARCHAR(30),
        balance  DECIMAL(10,2) );

mysql> INSERT INTO accounts VALUES ('Smith', 1000), ('John', 2000);
mysql> SELECT * FROM accounts;
+-------+---------+
| name  | balance |
+-------+---------+
| Smith  | 1000.00 |
| John | 2000.00 |
+-------+---------+

-- Transfer money from one account to another account
mysql> START TRANSACTION;
mysql> UPDATE accounts SET balance = balance - 100 WHERE name = 'Smith';
mysql> UPDATE accounts SET balance = balance + 100 WHERE name = 'John';
mysql> COMMIT;     -- Commit the transaction and end transaction
mysql> SELECT * FROM accounts;
```

```
+-------+---------+
| name  | balance |
+-------+---------+
| Smith  |  900.00 |
| John | 2100.00 |
+-------+---------+

mysql> START TRANSACTION;
mysql> UPDATE accounts SET balance = balance - 100 WHERE name = 'Smith';
mysql> UPDATE accounts SET balance = balance + 100 WHERE name = 'John';
mysql> ROLLBACK;    -- Discard all changes of this transaction and end Transaction
mysql> SELECT * FROM accounts;
+-------+---------+
| name  | balance |
+-------+---------+
| Paul  |  900.00 |
| Peter | 2100.00 |
+-------+---------+
```

If you start another mysql client and do a SELECT during the transaction (before the commit), you will not see the changes. Once you commit the transaction in first window, then changes can observe in another client window.

Effect of two simultaneous database transaction into data is controlled by using Isolation level. Isolation level is used to separate one database transaction with other.

When a successful transaction is completed, the COMMIT command should be issued so that the changes to all involved tables will take effect.

If a failure occurs, a ROLLBACK command should be issued to return every table referenced in the transaction to its previous state.

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM accounts;
+-------+---------+
| name  | balance |
+-------+---------+
| Smith |  800.00 |
| John  | 2200.00 |
+-------+---------+
2 rows in set (0.00 sec)
```

```
mysql> INSERT INTO accounts VALUES ('Smith', 1000), ('John', 2000);
Query OK, 2 rows affected (0.05 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM accounts;
+-------+---------+
| name  | balance |
+-------+---------+
| Smith |  800.00 |
| John  | 2200.00 |
| Smith | 1000.00 |
| John  | 2000.00 |
+-------+---------+
4 rows in set (0.00 sec)

mysql> delete from accounts;
Query OK, 4 rows affected (0.05 sec)
```

Open another window and type the commands

```
mysql> use company;
Database changed
mysql> SELECT * FROM accounts;
+-------+---------+
| name  | balance |
+-------+---------+
| Smith |  800.00 |
| John  | 2200.00 |
+-------+---------+
2 rows in set (0.00 sec)
```

It is observed that before commit command in first window, in another window, insert and delete command does not update the table.

In fist window type following command

```
mysql> commit;
Query OK, 0 rows affected (0.09 sec)
```

After commit command in first window, if you check in another window following is observed.

```
mysql> SELECT * FROM accounts;
Empty set (0.00 sec)
```

Output: Students will write theory.
Perform above queries and similar queries on your database and attach the printout.

Conclusion: A transaction begins with the first executable SQL statement. A transaction ends when it is committed or rolled back, either explicitly with a COMMIT or ROLLBACK statement or implicitly when a DDL (Create, Alter, Rename, Drop And Truncate) statement is issued.

Outcome:

Students will be able to execute various transactions commands..

Oral Questions:

1. What is transaction?
2. What is schedule?
3. What do you mean by serial schedule? View seriazibility means what?
4. ACID properties.