DEPARTMENT OF COMPUTER ENGINEERING

LABORATORY MANUAL

DATABASE SYSTEM LABORATORY

Shri Vile Parle Kelavani Mandal's
INSTITUTE OF TECHNOLOGY DHULE (M.S.)

DEPARTMENT OF COMPUTER ENGINEERING

Vision of Institute

To prepare the students ready with socially sensitive Engineers of Excellence who will grow with the Institute and add value to the nation.

Mission of Institute

Strive to become one of the institute to be globally recognized as a Resource of Excellence focusing on nurturing and developing the Society

Vision of Department

To create quality computer professional through excellent academic environment.

Mission of Department

1. To empower students with fundamental of Computer Engineering to be successful professional.
2. To impart quality education to enable the students for higher studies, research and entrepreneurship.
3. To cater for the service to society.

Program Educational Objectives (PEOs) of Department

PEO I: Graduate shall have successful professional careers, lead and manage teams.
PEO II: Graduate shall exhibit functional and disciplinary skills to resolve real life problems.
PEO III: Graduate shall evolve as a professional or researcher and continue to learn and adapt emerging technology.

COMPUTER ENGINEERING DEPARTMENT

Program Outcomes

Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

COMPUTER ENGINEERING DEPARTMENT

Program Specific Outcomes (PSO) addressed by the Course:

A graduate of the Computer Engineering Program will demonstrate-

PSO1: Professional Skills-The ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, data science, and networking for efficient design of computer-based systems.

PSO2: Problem-Solving Skills- The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver advanced computing systems.

PSO3: Professional Career and Entrepreneurship- The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, and a zest for higher studies and research.

Dr. Babasaheb Ambedkar Technological University

Syllabus & Scheme

Course Code	Course Title	Weekly Practical Hrs.	Evaluation Scheme			Credit
			MSE	CA	ESE	
BTCOL506	**Database System Lab**	**2**	**20**	**20**	**60**	**2**

BTCOL506 Database System Lab

As a part of lab exercises for Database System Laboratory, it is suggested that the student should get hands-on experience on SQL/PLSQL.

Shri Vile Parle Kelavani Mandal's
Institute of Technology, Dhule.
www.svkm-iot.ac.in
Survey No. 499, Plot No.2, Behind Gurudwara, Mumbai-Agra Road, Dhule(M.S.)

DEPARMENT OF COMPUTER ENGG.

LABORATORY : (BTCOL506) Database System Laboratory	**SEMESTER : 5th Semester**

Master List of Practical

SR. NO.	PRACTICAL DESCRIPTION	Page No
1	Introduction to How to download and install MySQL database server	
2	Using SQL prompt create database and use of SQL commands (DDL, DML and DCL).	
3	Using SQL to Create Table and Rename table for suitable database schema also apply database constraint like primary key, foreign key and NOT Null. Then design at least ten SQL queries SQL DML statements: Insert, Select, Update, Delete using distinct and count clause.	
4	Consider the suitable database schema and design five SQL Nested Queries with/without Where clause.	
5	Write a stored procedure to insert and retrieve data from database table.	
6	Write a PLSQL procedure for cursor and trigger.	
7	Write SQL to apply Aggregating Data using Group functions	
8	Design a queries for suitable database application using SQL DML statements: all types of Join, Views.	
9	To study Large objects – CLOB, NCLOB, BLOB and BFILE.	
10	Design database schema using normalization concepts for office automation or hotel management or hospital management.	
11	Implement small database system for office automation or hotel management or hospital management.	

Subject In-charge:	**APPROVED BY :** Dr. Makarand Shahade HOD, Department of Computer Engineering

Rubrics for Assessment

Assignment/Experiment :

Date of Performance: **Date of submission :**

Marks Split Up to	Maximum Marks	Marks Obtained
Performance/ conduction	3	
Report Writing	3	
Attendance	2	
Viva/Oral	2	
Total Marks	10	
Signature of Subject Teacher		

COURSE OBJECTIVES:

• This lab enables the students to practice the concepts learnt in the subject DBMS by developing a database.

• The student is expected to practice the designing, developing and querying a database.

• Students are expected to use “Mysql/Oracle” database

COURSE OUTCOMES:

At the end of this course Students will be able to:

CO506L.1 To Discuss the installation procedure of DBMS software

CO506L.2 To Implement database language commands for database concepts

CO506L.3 To Examine the data using queries to retrieve data from database

CO506L.4 To Use PL/SQL concepts for processing a data.

CO506L.5 To Develop solutions using database concepts for requirements.

GUIDELINES FOR STUDENT'S LAB JOURNAL

1. The laboratory assignments are to be submitted by student in the form of journal.
2. The Journal consists of prologue, Certificate, table of contents, and handwritten write-up of each assignment (Title, Objectives, Problem Statement, Outcomes, software & Hardware requirements, Date of Completion, Assessment grade/marks and assessor's sign, Theory-Concept, circuit diagram, pin configuration, conclusion/analysis), printouts of the code written using coding standards, sample test cases etc.
3. Practical Examination will be based on the term work submitted by the student in the form of Journal
4. Candidate is expected to know the theory involved in the experiment
5. The practical examination should be conducted if the journal of the candidate is completed in all respects and certified by concerned faculty and head of the department
6. All the assignment mentioned in the syllabus must be conducted.

Experiment No. 1

Problem Statement: Download and install MySQL database server

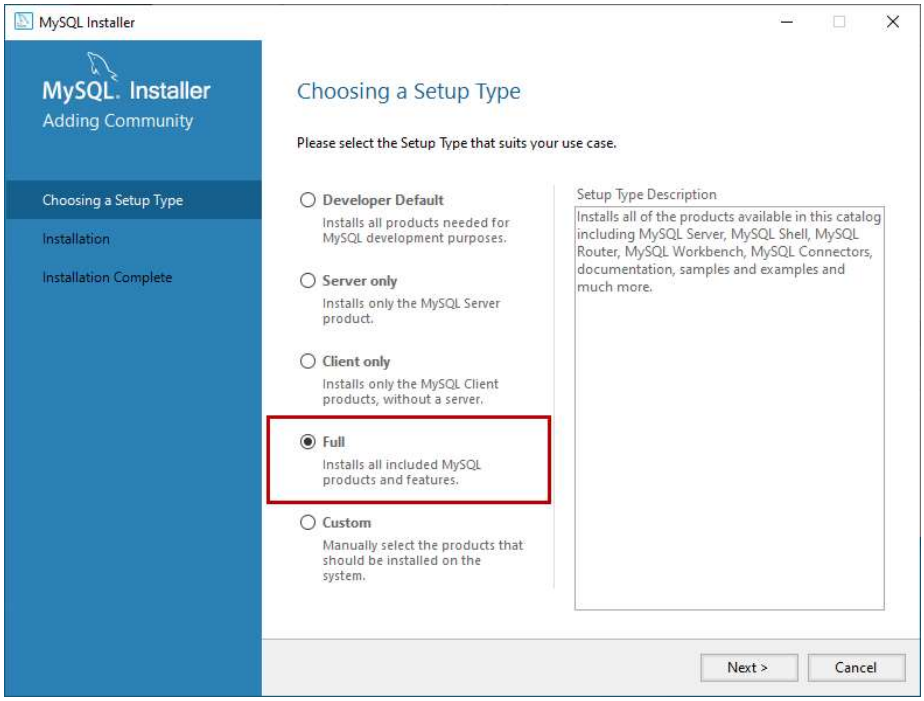
Software Required: MySQL, Ubuntu 14.04 or above

Theory:

Once the installer has been downloaded, double-click the setup file to start the installation process. On the **Choosing a Setup Type** page, you can see four installation options.

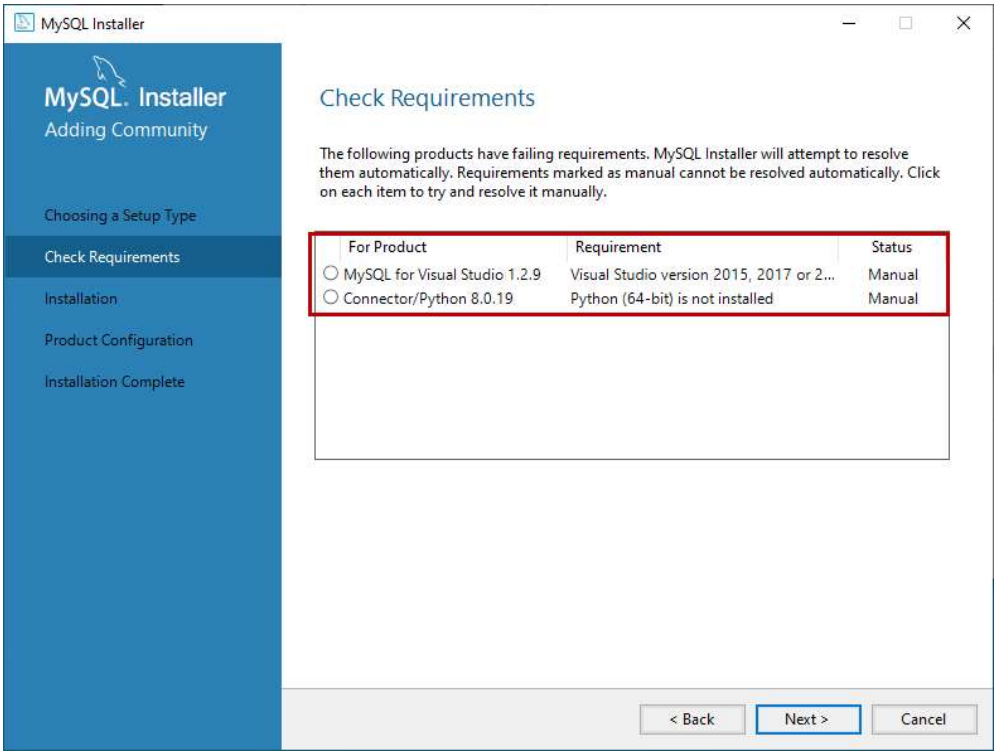
1. **Developer default:** If you want to create a development machine, you can use this option. It installs the components which are required for application development, e.g., MySQL Server, MySQL Shell, MySQL connectors, MySQL
2. **Server Only:** If you want to create a standalone database server with specific components, you can use this option
3. **Full:** If you want to install MySQL Server with its all components, then you can use this option
4. **Custom:** If your requirements are limited to the few components, you can use this option

We are going to install MySQL Server with all components; hence, choose "**Full**" and click on **Next**.

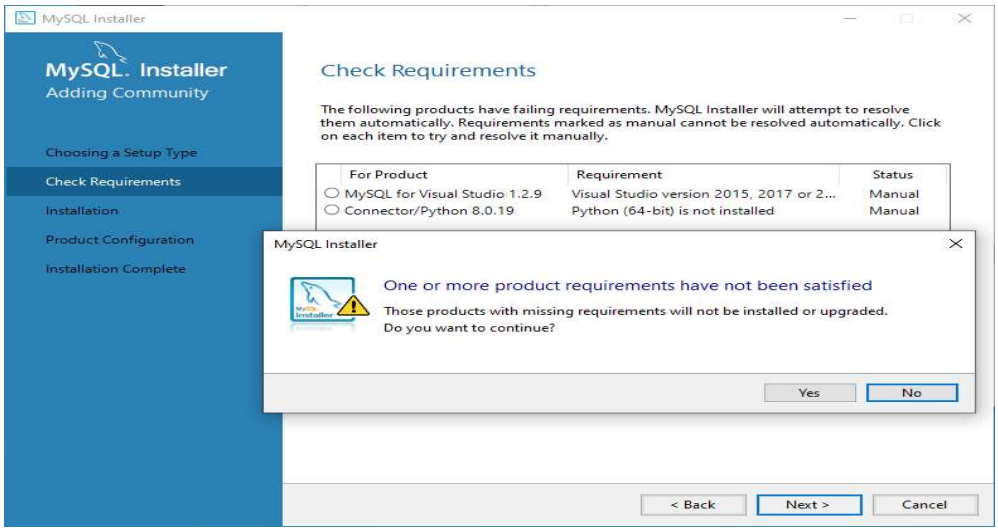


Before installation begins, the installer checks all the prerequisites that are required to install all the components of the MySQL database server. If any software prerequisites are missing, then you can

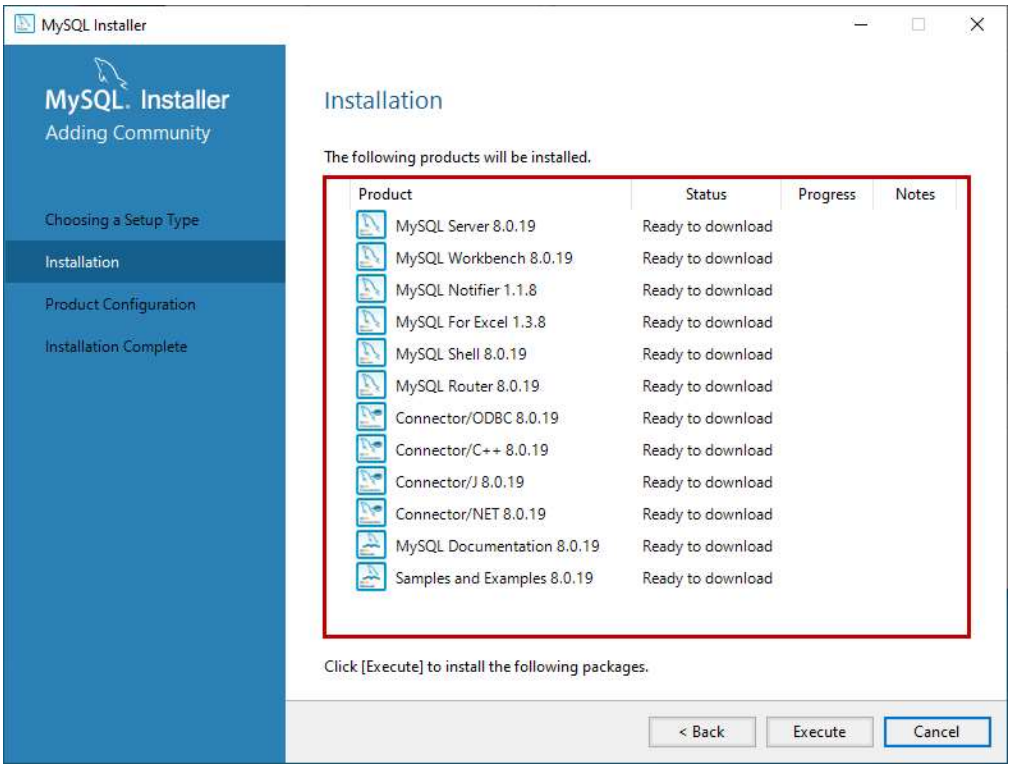
see the details of failing requirements on the **"Check Requirements"** screen. It shows the name of the product, required component/software, and its status. As you can see, to install the MySQL database server for visual studio, we must install visual studio 2015 or above. Similarly, to install Python connector, we must install python on the work station. Click on **Next**.



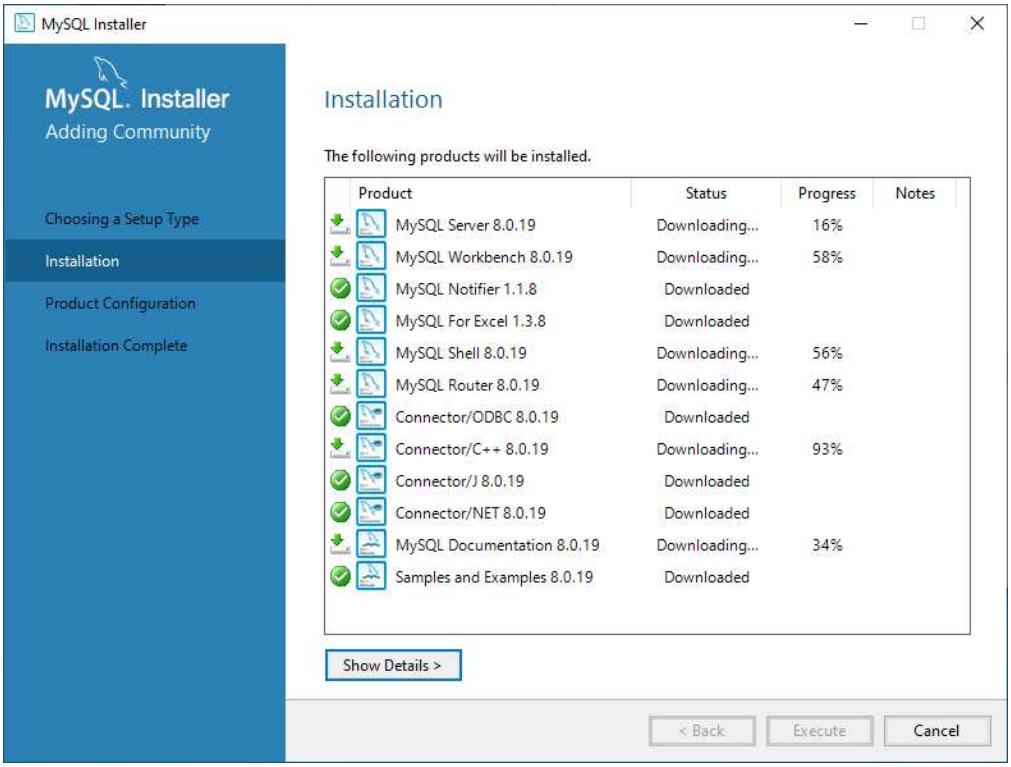
An installer gives us a warning. We can continue our installation without installing the visual studio and python. Click on **Yes**.



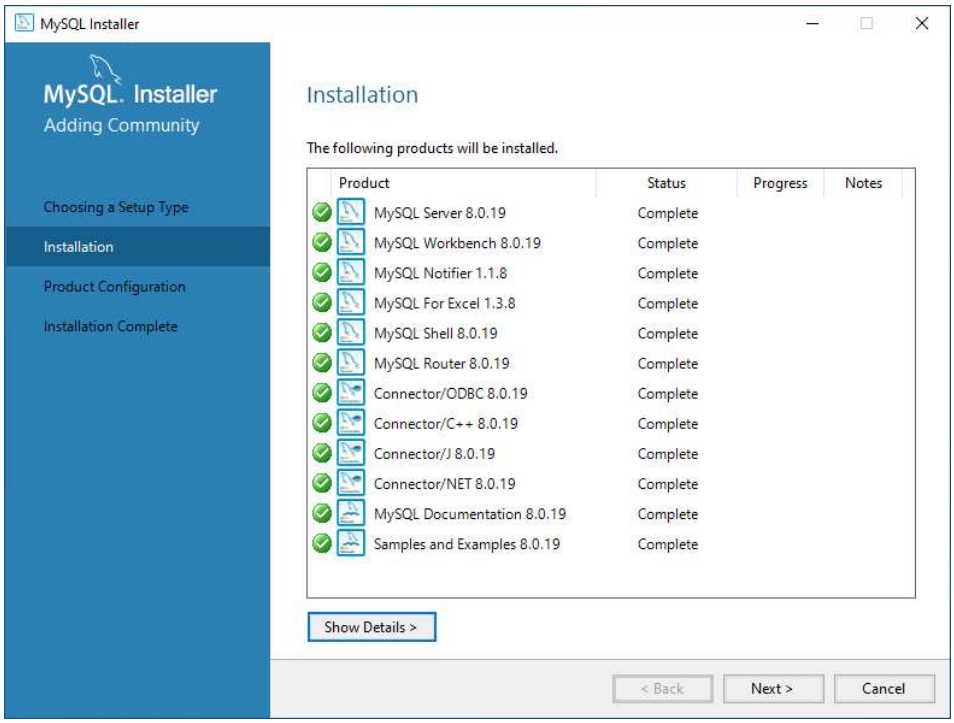
On the Installation screen, you can see the list of the MySQL products/software that are going to be installed on my workstation. Review the list and click on **Execute**.



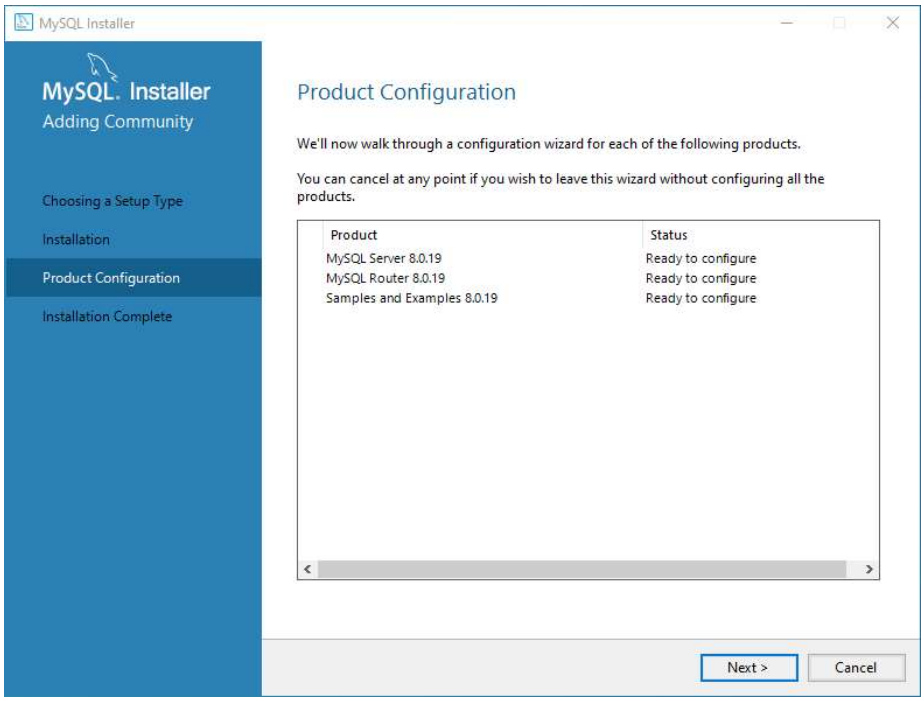
The installer downloads all the products/software. After that, it installs all the products.



Wait for a few mins. Once the installation process completes, we are ready to configure the MySQL database server and other components. Click on **Next**.

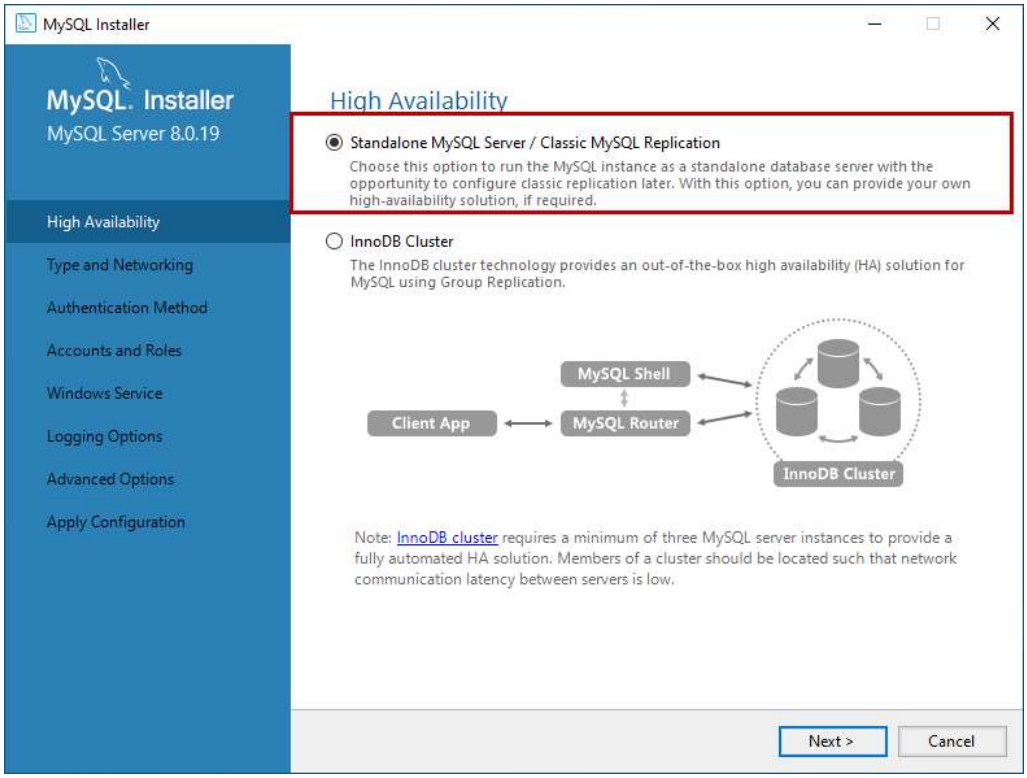


On the Product configuration screen, you can see the list of the products that need to be configured. First, let us configure the MySQL Server. Click on **Next**.



On the High availability screen, we can choose to install the **InnoDB cluster** or **Standalone MySQL Server**. **InnoDB cluster** is the High availability solution of MySQL. It uses group replication. I will

explain more about it in my future series of articles. We are going to perform a standalone installation of MySQL Server hence choose "**Standalone MySQL Server / Classic MySQL Replication**".



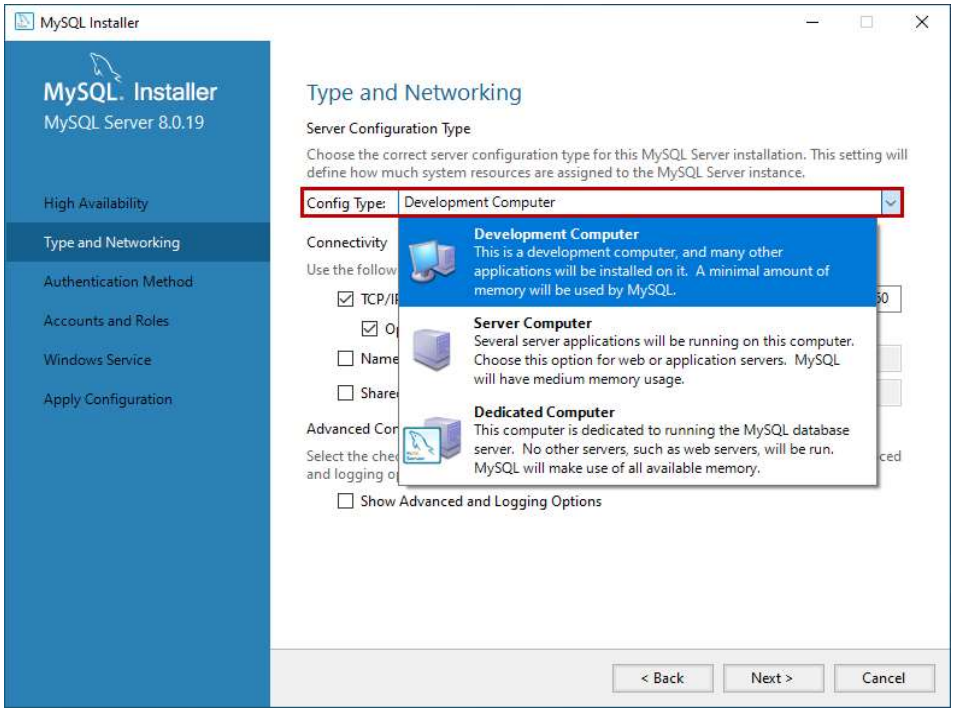
On **Type and Networking** screen, we can configure the following:

The type of MySQL configuration.

The type of MySQL configuration is a predefined set of configuration parameter that determines how much resources should be allocated to the MySQL Services. You have three configuration options:

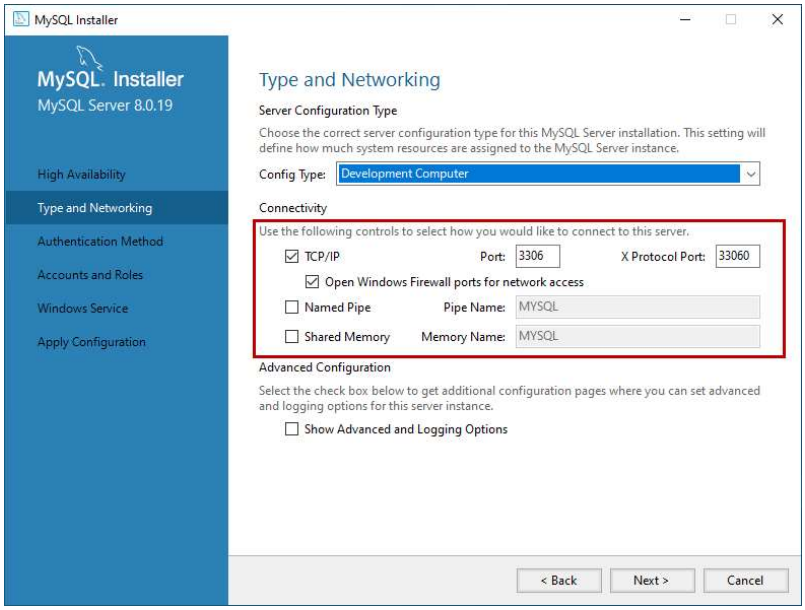
1. **Development Computer:** This configuration uses a minimal amount of the resources to MySQL Service
2. **Server Computer:** This configuration uses a minimal amount of resources. This option is suitable when we are installing database servers and web servers on the same machine. The configuration allocates an average amount of resources to MySQL Service
3. **Dedicated Computer:** This option is used when we have created a dedicated MySQL Server. The configuration allocates a high amount of resources to MySQL Service

We would configure the server with minimal resources hence select "Development computer" from the Config Type drop-down box.

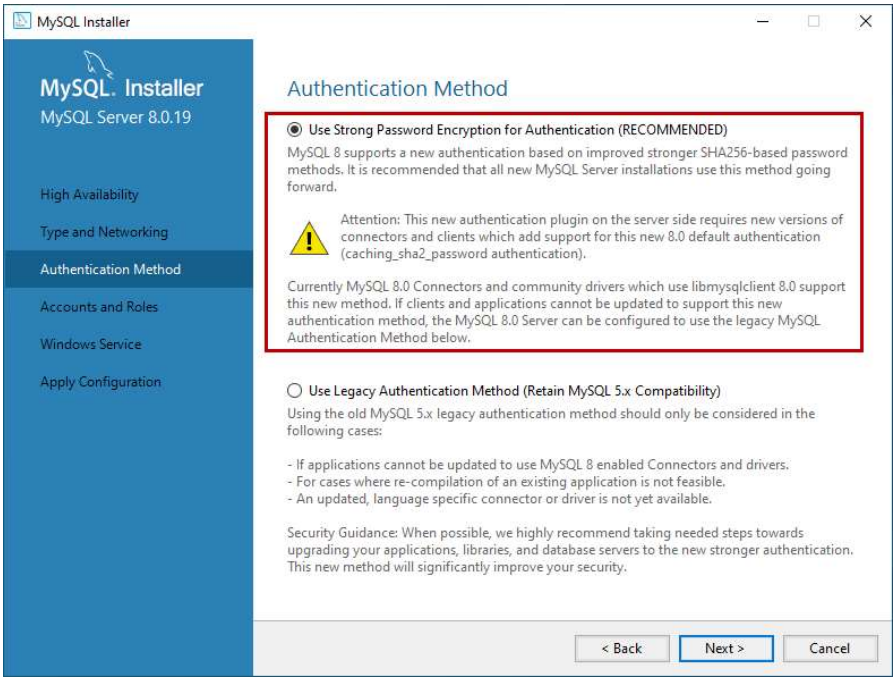


Network Connectivity

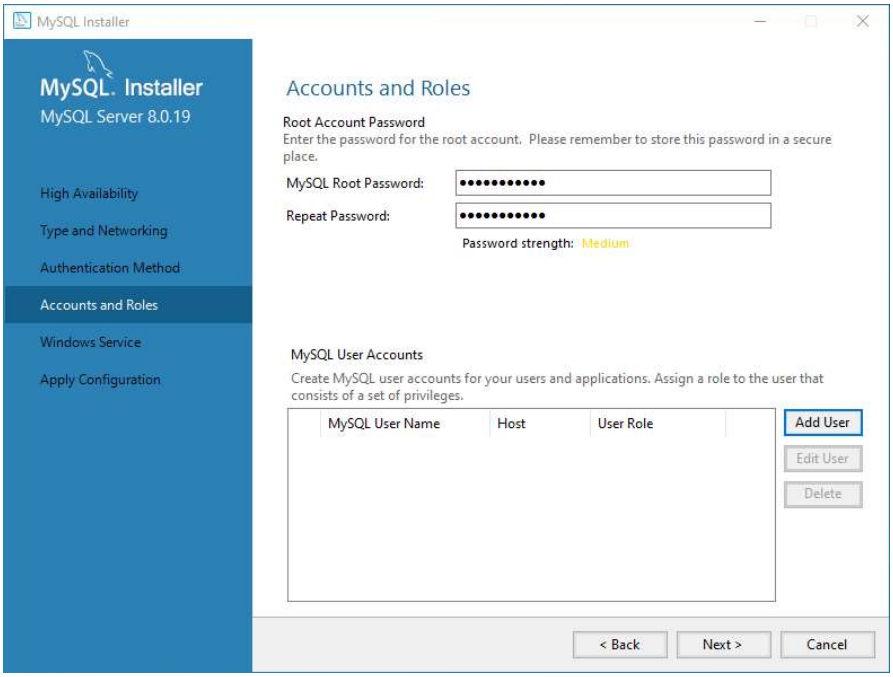
In this section, we can control how clients can connect to MySQL databases. We can use TCP/IP protocol or Named Pipe or Shared Memory. If you want to configure Named Pipe / Shared Memory, we must provide the Pipe Name and Memory Name. You can also specify the default port to connect to the database server. You can also choose to allow the port number specified in Port textbox in the firewall. See the following image:



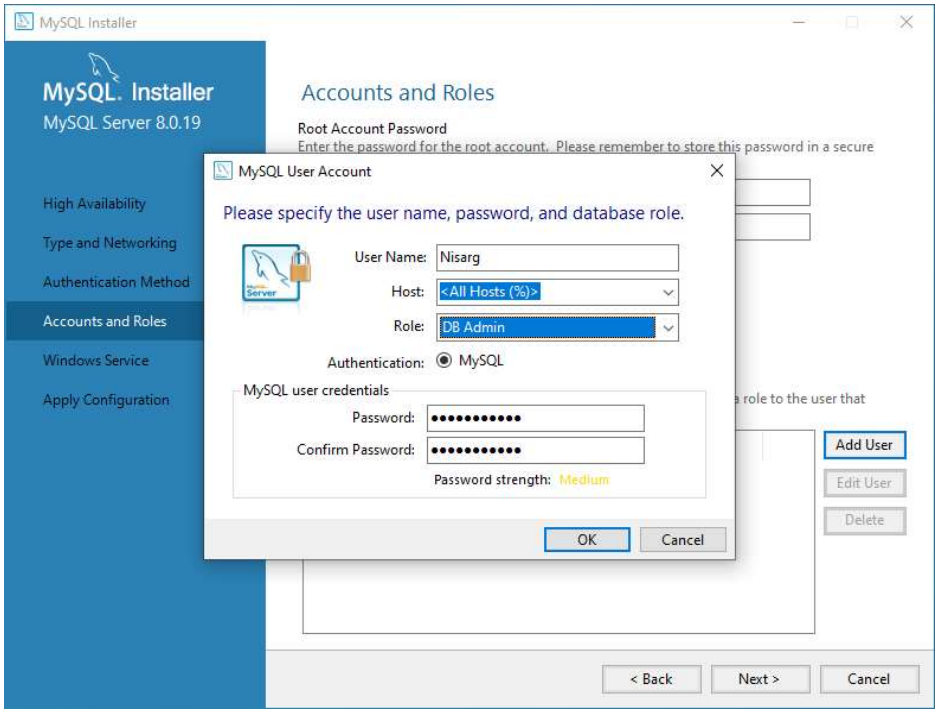
In MySQL 8.0 version, we can use SHA256 based strong passwords. On the **Authentication Method** screen, choose the option to use the Legacy authentication method or Strong password for authentication. Note: If you are using **Strong Password Encryption for Authentication,** then make sure that all the connectors must be updated to the latest version. We are going to use **Strong password Encryption for Authentication**.



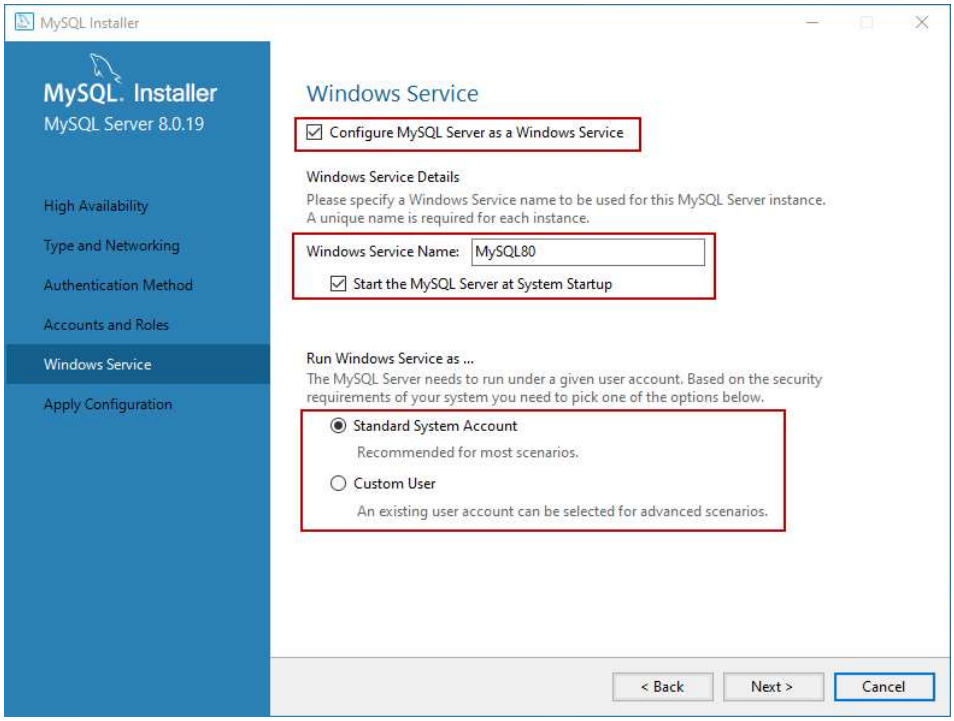
On **Accounts and Roles** screen, you can specify the MySQL root account password. MySQL Root account is a default sysadmin account, and it must be disabled.



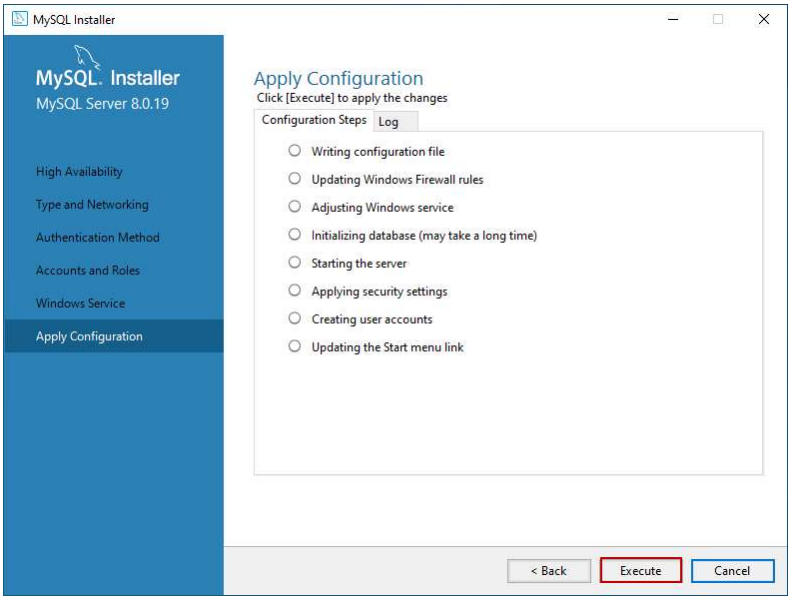
You can also create other users to do that click on Add user. On MySQL User account dialog box, provide a username, hostname, Role of the User, type of authentication, and password. Once the user is created, click on **Next**. See the following image:



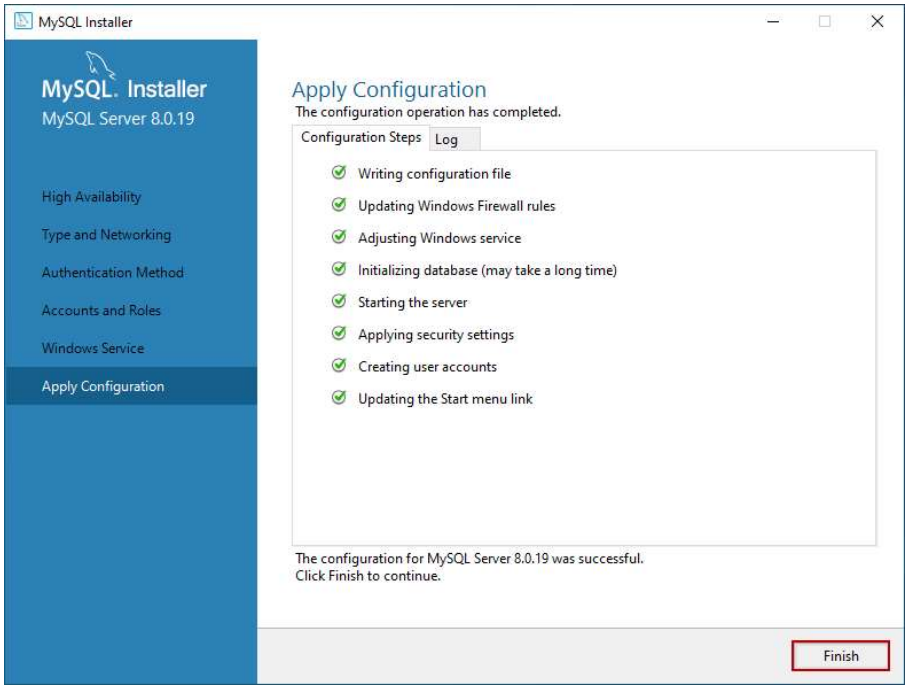
On the **Windows Service** screen, you can configure the MySQL server to run as a windows service. You can provide the desired name and configure it to auto-start the service when the system reboots. Moreover, you can provide the credentials under which the MySQL Service will run. You can choose the standard system account or provide a specific user. See the following image:



On the **Apply Configuration** screen, you can see the list of confirmation steps. Once all the configuration settings are verified, click on **Execute**.

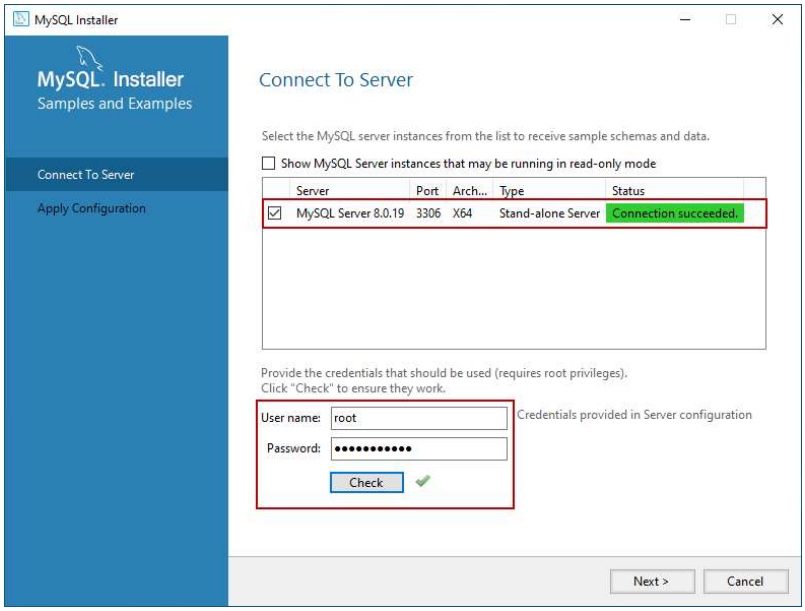


The MySQL installation process starts. You can view the installation process in the "Log" tab. Once installation completes successfully, click on "**Finish**" to close the installer.

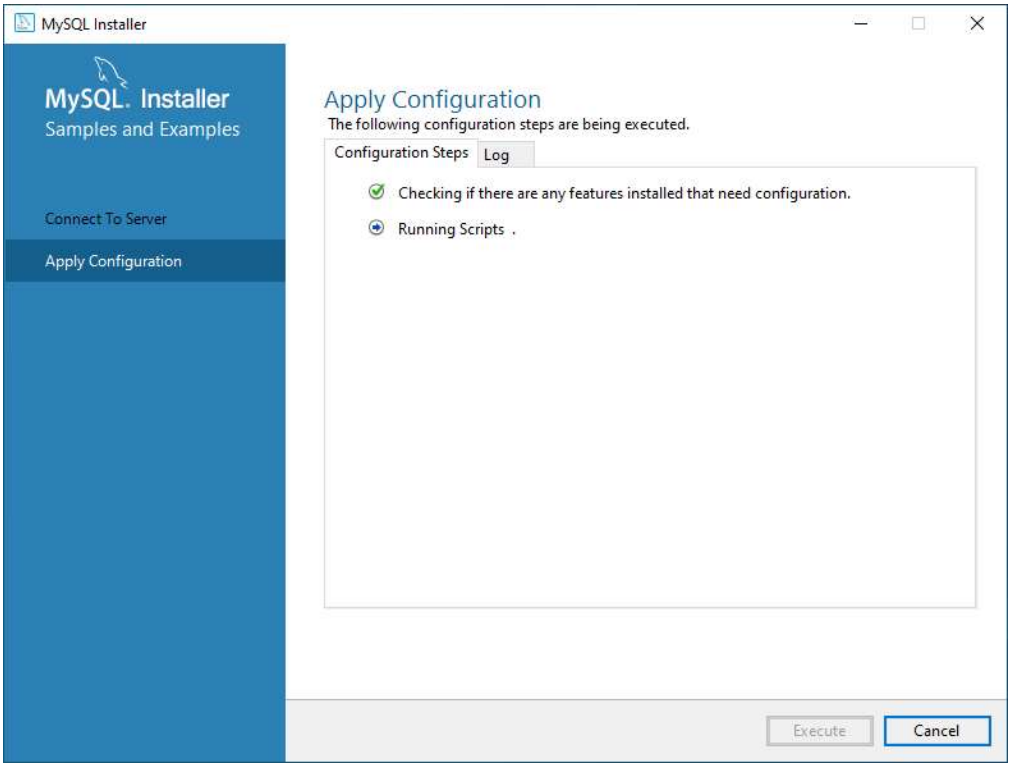


Install the sample database

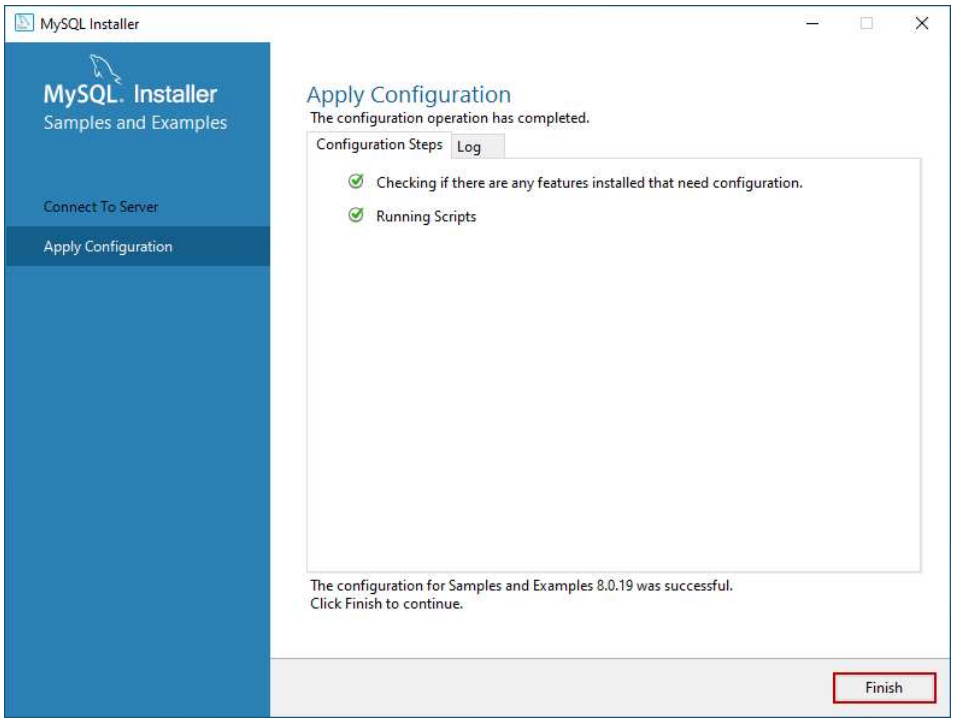
If you have chosen to install all the components of MySQL Server (Full Setup Type), MySQL installer moves to Sample and Example screen. On this screen, provide username and password of the user that has root/sysadmin privileges and click on Check. If the connection establishes successfully, click on next. See the following image:



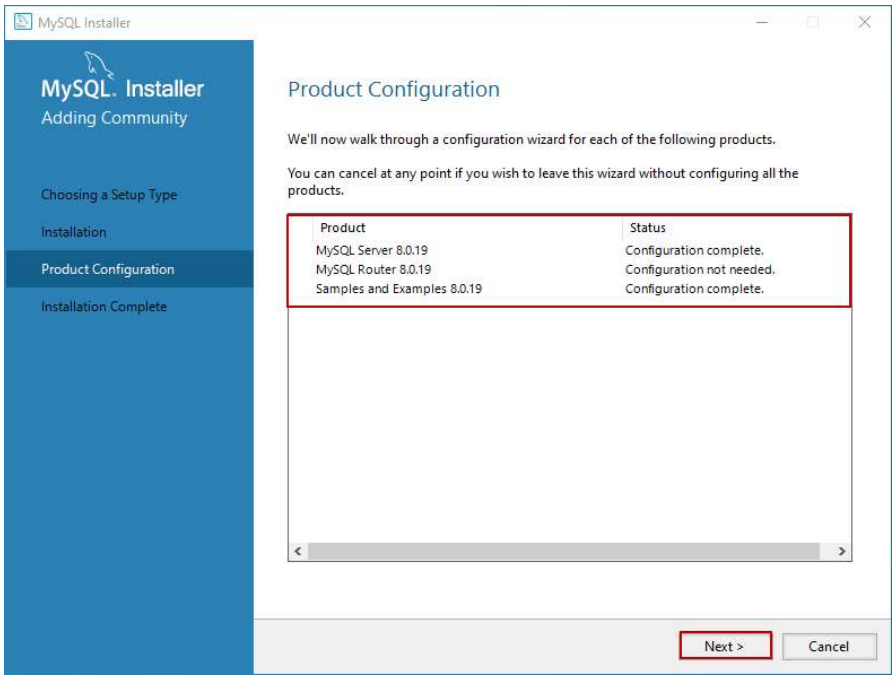
On the **Apply Configuration Screen**, click on **Execute** to start the installation of the Sample database. See the following:



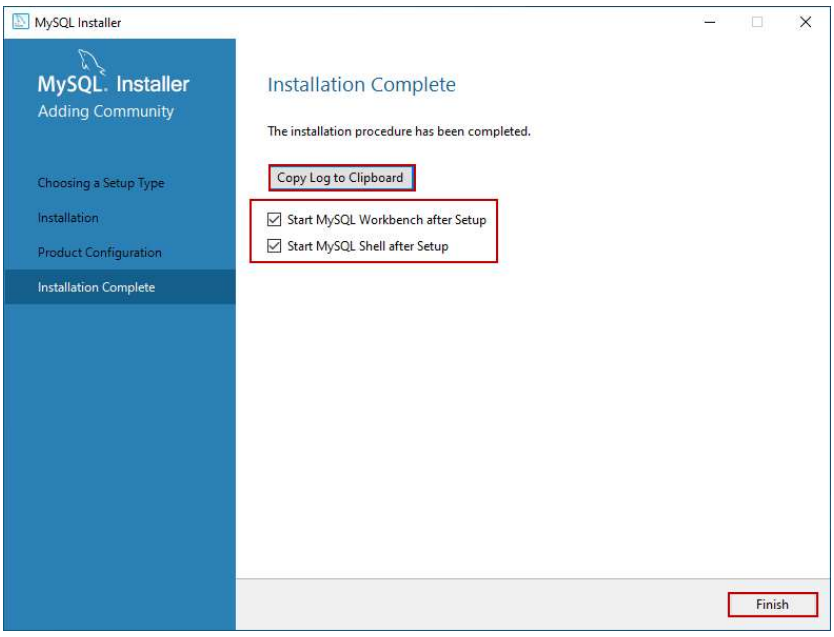
Once the sample database has been installed, click on the **Finish** button.



The installer continues to the **Product Configuration** screen. On this screen, you can see that the installation of the **MySQL Server 8.0.19** and **Sample and Example 8.0.19** has been completed successfully. See the following image:



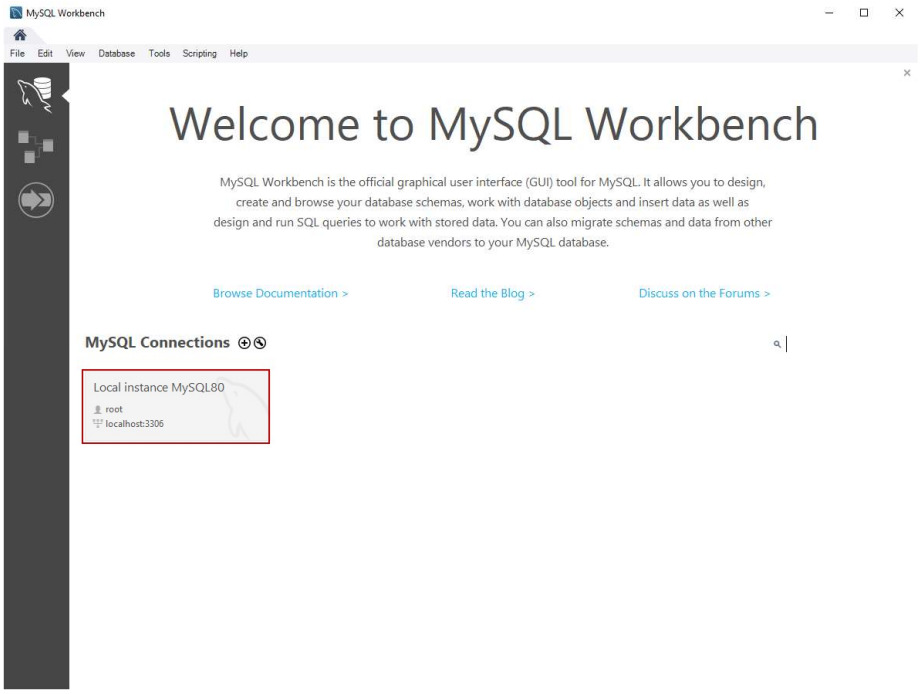
Once the installation completes, you can copy the installation logs on the clipboard to review it later. Moreover, if you want to start exploring MySQL straight away, then you can select "**Start MySQL workbench after Setup**" and "**Start MySQL Shell after Setup**" and click on **Finish**. See the following image:



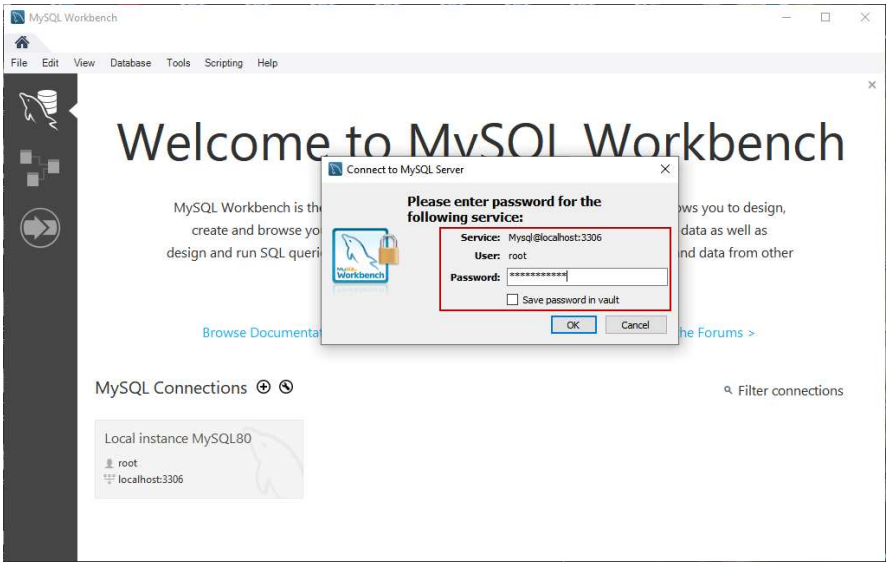
Connect to MySQL Server

Once the installation completes, let us connect to the server and execute the first MySQL Query. Open MySQL workbench. Just like SQL Server management studio, MySQL workbench is the development tool which is used to querying the database and create database objects.

On MySQL workbench welcome screen, you can see the list of MySQL connections. We have not configured multiple connections; hence you can see "**Local instance MySQL80.**" Click on it to open the new query editor window.

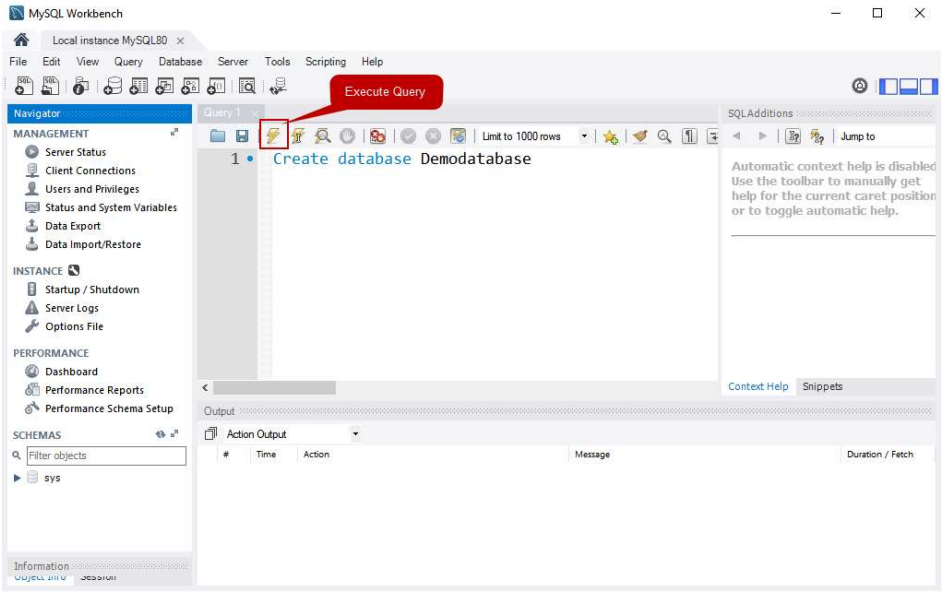


When you click on the connection, you must enter the credentials to connect the database server. Enter the password and click on **OK.**

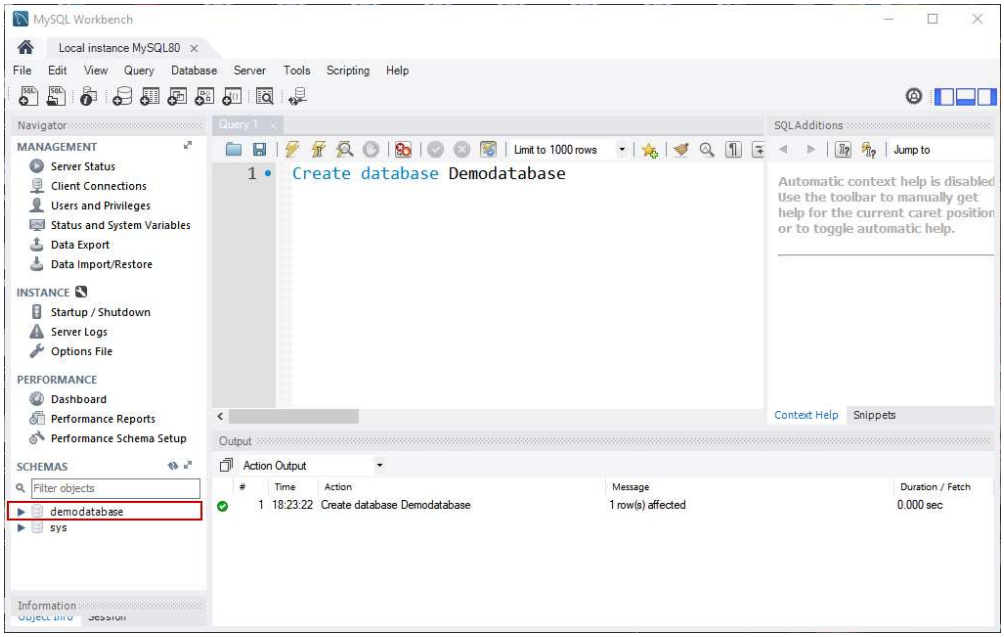


First, let's create a simple database on MySQL Server. Write the following query in the query editor window and click on execute. See the following image:

```
1 Create database Demodatabase
```



Once the query executes successfully, you can see the new database in the "SCHEMAS" pan. See the following image:

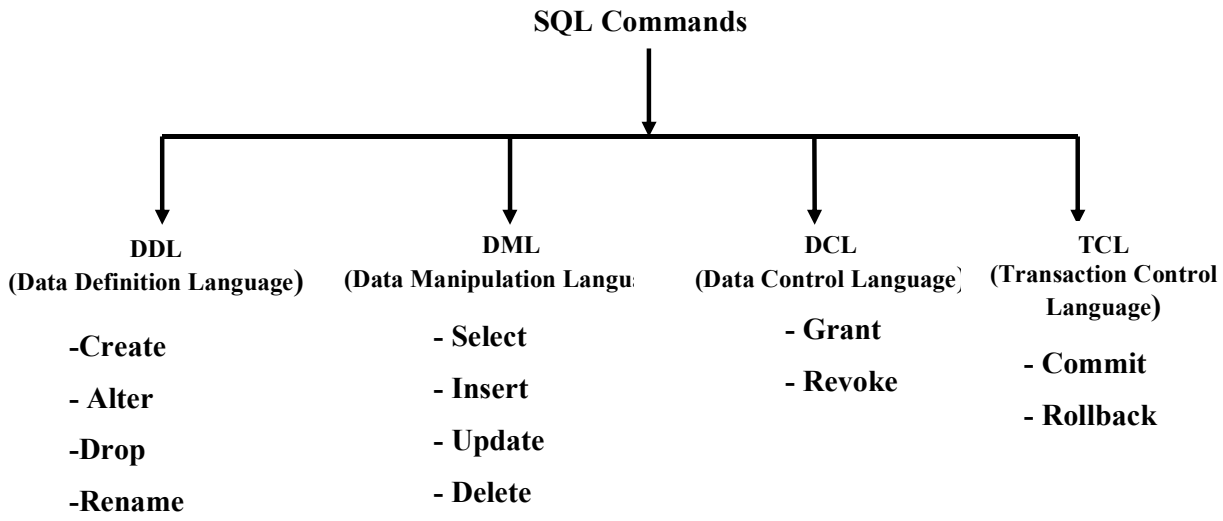


Experiment No. 2

Problem Statement: : Using SQL prompt create database and use of SQL commands (DDL, DML and DCL).

Software Required: MySQL, Ubuntu 14.04 or above

Theory:



Data Definition Language (DDL) is used different statements:

- CREATE - to create objects in the database
- ALTER - alters the structure of the database
- DROP - delete objects from the database
- RENAME - rename an object

1. **CREATE TABLE**

   Syntax: Create table table name(fieldname1 datatype(),fieldname2 datatype()...);
2. **ALTER**

A) ADD

B) MODIFY

ADD

Syntax: Alter table table name ADD (fieldname datatype()...);

MODIFY : To modify existing table structure

syntax: Alter table table name modify (fieldname datatype()...);

3. **DROP**

   Syntax: Drop table table_name;

4. **RENAME**

   Syntax: Rename Table Old_Table TO New_Table;

DML- Data Manipulation Language (DML) statements

Are used for managing data within schema objects DML deals with data manipulation, and therefore includes most common SQL statements such SELECT, INSERT, etc. DML allows to add / modify / delete data itself. DML is used to manipulate with the existing data in the database objects (insert, select, update, delete).

1. **INSERT:**

   Syntax: INSERT INTO Table name values();

2. **SELECT:**

   Syntax: Select*from <table name>

3. **UPDATE:**

   Syntax: Update<table name> set to(calculation);

4. **DELETE:**

   Syntax: Delete from<table name> OR Delete from<table name> Where Condition.

DCL- DCL is the abstract of Data Control Language.

Data Control Language includes commands such as GRANT, and concerns with rights, permissions and other controls of the database system. DCL is used to grant / revoke permissions on databases and their contents. DCL is simple, but MySQL permissions are a bit complex. DCL is about security. DCL is used to control the database transaction. DCL statement allows you to control who has access to specific object in your database.

1. GRANT
2. REVOKE

GRANT: It provides the user's access privileges to the database. In the MySQL database offers both the administrator and user a great extent of the control options. By the administration side of the process includes the possibility for the administrators to control certain user privileges over the MySQL server by restricting their access to an entire the database or use limiting permissions for a specific table. It Creates an entry in the security system that allows a user in the current database to work with data in the current database or execute specific statements.

Syntax :

Grant Privileges_Names ON Object To User

1. **privileges_name**: These are the access rights or privileges granted to the user.
2. **Object:** It is the name of the database object to which permissions are being granted. In the case of granting privileges on a table, this would be the table name.
3. **User:** It is the name of the user to whom the privileges would be granted.

CREATE USER 'NBhushan'@'127.0.0.1 ' IDENTIFIED BY 'Test@123';
Grant SELECT On EMP TO 'NBhushan@127.0.0.1';

OR

Grant SELECT, INSERT, UPDATE, DELETE On EMP TO NBhushan@127.0.0.1;

OR

Grant ALL On EMP TO NBhushan@127.0.0.1;

Granting a Privilege to all Users in a Table

Grant SELECT On EMP TO '*'@127.0.0.1;

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

REVOKE privileges ON Object FROM User;

1. **Object:** It is the name of the database object from which permissions are being revoked. In the case of revoking privileges from a table, this would be the table name.
2. **User:** It is the name of the user from whom the privileges are being revoked.

REVOKE SELECT On EMP TO NBhushan@127.0.0.1;

View Created User from Root login

Mysql> select host, user, password from mysql.user;

Login Trough New User-

bhushan@ubuntu:~$ mysql –u NBhushan –p –h 127.0.0.1

Enter Password:Test@123

Drop User from Root login:

Mysql> drop user NBhushan@127.0.0.1;

Experiment No. 3

Problem Statement: : Create tables for suitable database schema with constraint like primary key, foreign key and NOT Null. Then design at least ten SQL queries SQL DML statements: Insert, Select, Update, Delete using distinct and count clause.

Software Required: MySQL, Ubuntu 14.04 or above

Theory:

SQL Constraints:

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

1. **NOT NULL Constraint:** Ensures that a column cannot have NULL value. By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such constraint on this column specifying that NULL is now not allowed for that column.

   Example: Create Table Student(Roll_No Int Primary Key,
   F_Name Varchar(20) NOT NULL,
   L_Name Varchar(20) NOT NULL);
2. **Primary Key Constraint:** A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values. A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a **composite key**. Consider the above example.
3. **Foreign Key Constraint:** A foreign key is a key used to link two tables together. This is sometimes called a referencing key. Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.

   The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.

   Example: Create Table Country (Country_Code Primary Key, Name);

```
Create Table Info(Person_Name varchar(20), CountryCode Int,
                  Foreign key(CountryCode) References Country (Country_Code)
                 );
```

SQL SELECT Statement:

```
Syntax:  SELECT column1, column2....columnN
         FROM   table_name
```

SQL WHERE Clause:

```
Syntax : SELECT column1, column2....columnN
         FROM   table_name
         WHERE  CONDITION;
```

SQL AND/OR Clause:

```
Syntax: SELECT column1, column2....columnN
         FROM   table_name
         WHERE  CONDITION-1 {AND|OR} CONDITION-2;
```

SQL IN Clause:

```
Syntax: SELECT column1, column2....columnN
        FROM   table_name
        WHERE  column_name IN (val-1, val-2,...val-N);
```

SQL BETWEEN Clause:

```
Syntax: SELECT column1, column2....columnN
        FROM   table_name
        WHERE  column_name BETWEEN val-1 AND val-2;
```

SQL LIKE Clause:

```
Syntax: SELECT column1, column2....columnN
        FROM   table_name
       WHERE  column_name LIKE { PATTERN };
```

SQL ORDER BY Clause:

Syntax: SELECT column1, column2....columnN
FROM table_name
WHERE CONDITION
ORDER BY column_name {ASC|DESC};

SQL GROUP BY Clause:

Syntax: SELECT SUM(column_name)
FROM table_name
WHERE CONDITION
GROUP BY column_name;

SQL COUNT Clause:

Syntax: SELECT COUNT(column_name)
FROM table_name
WHERE CONDITION;

SQL HAVING Clause:

Syntax: SELECT SUM(column_name)
FROM table_name
WHERE CONDITION
GROUP BY column_name
HAVING (arithematic function condition);

SQL UPDATE Statement:

Syntax: UPDATE table_name
SET column1 = value1, column2 = value2....columnN=valueN
[WHERE CONDITION];

SQL DELETE Statement:

Syntax: DELETE FROM table_name
WHERE {CONDITION};

SQL CREATE DATABASE Statement:

CREATE DATABASE database_name;

SQL DROP DATABASE Statement:

DROP DATABASE database_name;

SQL USE Statement:

USE DATABASE database_name;

SQL COMMIT Statement:

COMMIT;

SQL ROLLBACK Statement:

ROLLBACK;

Consider the following database schema and creates tables with primary key, foreign key and not null constraints.

mysql> Create Database NBhushan;
Query OK, 1 row affected (0.11 sec)

mysql> Use NBhushan
Database changed

Create the following databases as per sequence

First Table	Second Table
Create table Dept (Dept_No int Primary Key, Dept_Name varchar(20) Not Null, Mgr_ID int Not Null, Mgr_Start_Date Date Not Null);	**Create table Employe** (Emp_Id int primary key, F_Name varchar(20)Not Null, L_Name varchar(20) Not Null, DOB date Not Null, Join_Date date Not Null, Address varchar(20) Not Null, Gender varchar(20) Not Null, Dept_Id int Not Null, Salary int Not Null, foreign key(Dept_Id) references Dept(Dept_No));
Third Table	**Forth Table**
Create table Dept_Location (Dept_No int primary key, Dept_Loc varchar(20) Not Null, foreign key(Dept_No) references Dept(Dept_No));	**Create table Project** (Pro_Name varchar(20) Not Null, Pro_No int Primary key, Pro_Location varchar(20)Not Null, Dept_Num int Not Null, foreign key(Dept_Num) references Dept(Dept_No));
Fifth Table	
Create table Works_On (W_Emp_ID int Not Null, W_Pro_Num int , Dept_No int, Relation Varchar(15), Hrs int Not Null, foreign key (W_Emp_ID) references Employe(Emp_ID), foreign key (W_Pro_Num) references Project(Pro_No), foreign key (Dept_No) references Dept(Dept_No));	

1) Insert Data into Dept Table

mysql> Insert into Dept Values(111,'Computer',101,'2010-06-23');

mysql> Insert into Dept Values(222,'Civil',102,'2011-08-17');

mysql> Insert into Dept Values(333,'Mechanical',103,'2013-09-13');

mysql> Insert into Dept Values(444,'Electrical',104,'2012-10-29');

```
mysql> select * from Dept;
+---------+------------+--------+----------------+
| Dept_No | Dept_Name  | Mgr_ID | Mgr_Start_Date |
+---------+------------+--------+----------------+
|     111 | Computer   |    101 | 2010-06-23     |
|     222 | Civil      |    102 | 2011-08-17     |
|     333 | Mechanical |    103 | 2013-09-13     |
|     444 | Electrical |    104 | 2012-10-29     |
+---------+------------+--------+----------------+
4 rows in set (0.00 sec)
```

2) Insert Data Into Employe Table

mysql> Insert into Employe values

(101,'Amit','Patil','1985-04-08','2008-05-05','Pune','M',111,95000);

mysql> Insert into Employe values

(102,'Mohan','Sharma','1983-07-23','2009-08-11','Delhi','M',222,60000);

mysql> Insert into Employe values

(103,'Rghuvir','Prasad','1982-07-23','2009-06-21','Delhi','M',333,65000);

mysql> Insert into Employe values

(104,'Ram','Kulkarni','1979-09-11','2008-04-23','Mubai','M',444,77000);

mysql> Insert into Employe values

(105,'Rohit','Jadhav','1987-03-03', '2009-03-03','Nashik','M',111,55000);

mysql> Insert into Employe values

(106,'Mrunali','Muley','1988-02-22', '2011-03-22','Indore','F',111,37000);

mysql> Insert into Employe values

(107,'Kunal','Ambekar','1989-06-23','2012-07-21','Pune','M',111,30000);

mysql> Insert into Employe values

(108,'Amol','Gupta','1987-01-17', '2012-04-21','Surat','M',222,55000);

mysql> Insert into Employe values

(109,'Manish','Morya','1985-09-21','2012-07-22','Noida','M',222,56000);

mysql> Insert into Employe values

(110,'Trupti','Pawar','1985-09-21','2013-02-17','Noida','F',222,56000);

mysql> Insert into Employe values

(111,'Lalit','Gaidhani','1984-07-07', '2012-03-17','Mumbai','M',333,76000);

mysql> Insert into Employe values

(112,'Mayur','Bhat','1986-07-03','2013-07-13','Jaipur','M',444,60000);

Emp_Id	F_Name	L_Name	DOB	Join_Date	Address	Gender	Dept_Id	Salary
101	Amit	Patil	1985-04-08	2008-05-05	Pune	M	111	95000
102	Mohan	Sharma	1983-07-23	2009-08-11	Delhi	M	222	60000
103	Rghuvir	Prasad	1982-07-23	2009-06-21	Delhi	M	333	65000
104	Ram	Kulkarni	1979-09-11	2008-04-23	Mubai	M	444	77000
105	Rohit	Jadhav	1987-03-03	2009-03-03	Nashik	M	111	55000
106	Mrunali	Muley	1988-02-22	2011-03-22	Indore	F	111	37000
107	Kunal	Ambekar	1989-06-23	2012-07-21	Pune	M	111	30000
108	Amol	Gupta	1987-01-17	2012-04-21	Surat	M	222	55000
109	Manish	Morya	1985-09-21	2012-07-22	Noida	M	222	56000
110	Trupti	Pawar	1985-09-21	2013-02-17	Noida	F	222	56000
111	Lalit	Gaidhani	1984-07-07	2012-03-17	Mumbai	M	333	76000
112	Mayur	Bhat	1986-07-03	2013-07-13	Jaipur	M	444	60000

3) Insert Data Into Dept_Locations Table

mysql> Insert into Dept_Location values (111,'Mumbai');

mysql> Insert into Dept_Location values (222,'Delhi');

mysql> Insert into Dept_Location values (333,'Pune');

mysql> Insert into Dept_Location values (444,'Kolkata');

```
mysql> select * from Dept_Location;
+---------+----------+
| Dept_No | Dept_Loc |
+---------+----------+
|     111 | Mumbai   |
|     222 | Delhi    |
|     333 | Pune     |
|     444 | Kolkata  |
+---------+----------+
4 rows in set (0.00 sec)
```

4) Insert Data Into Project Table

mysql> Insert into Project values('CodePro',4141,'Nagpur',111);

mysql> Insert into Project values('NH47',5151,'Jabalpur',222);

mysql> Insert into Project values('HAL',6161,'Nashik',333);

mysql> Insert into Project values('BHEL',7171,'Surat',444);

```
mysql> Select * from Project;
+----------+--------+--------------+----------+
| Pro_Name | Pro_No | Pro_Location | Dept_Num |
+----------+--------+--------------+----------+
| CodePro  |   4141 | Nagpur       |      111 |
| NH47     |   5151 | Jabalpur     |      222 |
| HAL      |   6161 | Nashik       |      333 |
| BHEL     |   7171 | Surat        |      444 |
+----------+--------+--------------+----------+
4 rows in set (0.00 sec)
```

5) Insert Data Into Works_On Table

mysql> Insert into Works_On Values(101,4141,111,'Manager',310);

mysql> Insert into Works_On Values(102,5151,222,'Manager',390);

mysql> Insert into Works_On Values(103,6161,333,'Manager',380);

mysql> Insert into Works_On Values(104,7171,444,'Manager',336);

mysql> Insert into Works_On Values(105,4141,111,'Jr.Eng',220);

mysql> Insert into Works_On Values(106,4141,111,'Sr.Eng',256);

mysql> Insert into Works_On Values(107,4141,111,'Clerk',120);

mysql> Insert into Works_On Values(108,5151,222,'Sr.Eng',247);

mysql> Insert into Works_On Values(109,5151,222,'Clrek',230);

mysql> Insert into Works_On Values(110,5151,222,'Jr.Eng',220);

mysql> Insert into Works_On Values(111,6161,333,'Clerk',280);

mysql> Insert into Works_On Values(112,7171,444,'Clerk',180);

```
mysql> Select * from Works_On;
+----------+-----------+---------+----------+-----+
| W_Emp_ID | W_Pro_Num | Dept_No | Relation | Hrs |
+----------+-----------+---------+----------+-----+
|      101 |      4141 |     111 | Manager  | 310 |
|      102 |      5151 |     222 | Manager  | 390 |
|      103 |      6161 |     333 | Manager  | 380 |
|      104 |      7171 |     444 | Manager  | 336 |
|      105 |      4141 |     111 | Jr.Eng   | 220 |
|      106 |      4141 |     111 | Sr.Eng   | 256 |
|      107 |      4141 |     111 | Clerk    | 120 |
|      108 |      5151 |     222 | Sr.Eng   | 247 |
|      109 |      5151 |     222 | Clrek    | 230 |
|      110 |      5151 |     222 | Jr.Eng   | 220 |
|      111 |      6161 |     333 | Clerk    | 280 |
|      112 |      7171 |     444 | Clerk    | 180 |
+----------+-----------+---------+----------+-----+
12 rows in set (0.00 sec)
```

Solve the following Queries

1. List the employee who join the before 2012.
2. List the emps in the asc order of their Salaries?
3. List the all emps in the asc order of Annsal.
4. Find the all details of employee who work for HAL project.
5. List the emps Who's Annual sal ranging from 1000000 and 1200000.
6. List the emps who joined in January.
7. List the emps whose Sal is five-digit number and not starting with digit 3.
8. Find the project location of 4141 and 5151.
9. List the department, details where at least two emps are working.
10. Update the salary of employee 104.
11. Find the maximum salary of each department.
12. List the First Name of employes F_names contains 'A'.
13. List the employe whose Emp_Id not starting with digit 3
14. Display unique Jobs from Works_on table.
15. Add new department 555 'Chemical' in department table and then delete entry from department table.
16. Check whether all the employe number are indeed unique.
17. Select the maximum average salary drawn for each dept.
18. List the unique jobs of dept 111 and 222 in desc order.
19. List the highest paid employee.

Experiment No. 4

Problem Statement: : Consider suitable database schema and design five SQL Nested Queries with/without where clause.

Software Required: MySQL, Ubuntu 14.04 or above

Theory:

What is Sub Quires?

In SQL a Subquery can be simply defined as a query within another query. In other words we can say that a Subquery is a query that is embedded in WHERE clause of another SQL query.

Important rules for Subqueries:

- You can place the Subquery in a number of SQL clauses: WHERE clause, HAVING clause, FROM clause. Subqueries can be used with SELECT, UPDATE, INSERT, DELETE statements along with expression operator. It could be equality operator or comparison operator such as =, >, =, <= and Like operator.
- A subquery is a query within another query. The outer query is called as **main query** and inner query is called as **subquery**.
- The subquery generally executes first, and its output is used to complete the query condition for the main or outer query.
- Subquery must be enclosed in parentheses.
- Subqueries are on the right side of the comparison operator.
- ORDER BY command **cannot** be used in a Subquery. GROUPBY command can be used to perform same function as ORDER BY command.
- Use single-row operators with singlerow Subqueries. Use multiple-row operators with multiple-row Subqueries.

1. Subqueries with the Select Statement:

- SQL subqueries are most frequently used with the Select statement.

Syntax: Select Column
from Table
where Column_name Expression Operator (Select Column
from Table
Where..);

Example:

```
SELECT *
FROM  EMPLOYE
WHERE EMP_ID   IN
                    (SELECT EMP_ID
                     FROM EMPLOYE
                     WHERE SALARY > 4500);
```

2. Subqueries with the INSERT Statement:

- SQL subquery can also be used with the Insert statement. In the insert statement, data returned from the subquery is used to insert into another table.
- In the subquery, the selected data can be modified with any of the character, date functions.

Syntax:

```
INSERT INTO table_name (column1, column2, column3....)
SELECT *   FROM table_name
WHERE VALUE OPERATOR
```

Example:

Consider a table EMPLOYEE_BKP with similar as EMPLOYEE. Now use the following syntax to copy the complete EMPLOYEE table into the EMPLOYEE_BKP table.

```
INSERT INTO EMPLOYE_BKP
SELECT * FROM EMPLOYEE
WHERE EMP_ID IN
                    (
                      SELECT EMP_ID
                      FROM EMPLOYE
```

```
);
```

3. Subqueries with the UPDATE Statement

The subquery of SQL can be used in conjunction with the Update statement. When a subquery is used with the Update statement, then either single or multiple columns in a table can be updated.

Syntax:

```
UPDATE table
SET column_name = new_value
WHERE VALUE OPERATOR
                    (SELECT COLUMN_NAME
                     FROM TABLE_NAME
                     WHERE condition);
```

Example:

```
UPDATE EMPLOYE
SET SALARY = SALARY * 0.25
WHERE Join_Date IN (SELECT Join_Date FROM
                    EMPLOYE
                    WHERE Join_Date > '2012-12-31');
```

4. Subqueries with the DELETE Statement

Syntax:

```
DELETE FROM TABLE_NAME
WHERE VALUE OPERATOR
                    (SELECT COLUMN_NAME
                     FROM TABLE_NAME
                     WHERE condition);
```

Example:

```
DELETE FROM EMPLOYE_BKP
WHERE Salary IN
                (SELECT Salary FROM EMPLOYEE
                 WHERE Salary>= 75000);
```

Note: Consider the database schema of Assignment No-03 and solve the following Queries.

1. Find Employe ID of who works for Codepro project.
2. Find the First Name and Last Name of Employe who worked more than 200 hrs.
3. Find the Dept Name of project Jabalpur.
4. Select All data of employe who works as 'Jr.Eng.' and 'Sr.Eng.' and working Hrs are greater than 120.
5. A. Create Backup Table **EMP_Backup** Contains Only Emp_ID, F_Name,L_Name,Join_Date, Salary Columns.
   B. Insert the values into EMP_Backup table with the help of sub query from Employe table.
6. Update the salary in EMP_Backup table by 30% who works for project ‘HAL’.

Experiment No. 5

Problem Statement: : Write a stored procedure to insert and retrieve data from database table.

Software Required: MySQL, Ubuntu 14.04 or above

Theory:

Introduction:

A procedure (often called a stored procedure) is a subroutine like a subprogram in a regular computing language, stored in database. A procedure has a name, a parameter list, and SQL statement(s). All most all relational database system supports stored procedure.

A stored procedure has , a (possibly empty) parameter list, and an SQL statement, which can contain many more SQL statements. There is new syntax for local variables, error handling, loop con trol, and IF conditions.

Advantages

- **Share logic** with other applications. Stored procedures encapsulate functionality; this ensures that data access and manipulation are coherent between different applications.
- **Isolate** users from data tables. This gives you the ability to grant access to the stored procedures that manipulate the data but not directly to the tables.
- Provide a **security** mechanism. Considering the prior item, if you can only access the data using the stored procedures defined, no one else can execute a DELETE SQL statement and erase your data.
- To **improve performance** because it reduces network traffic. With a stored procedure, multiple calls can be melded into one.

Delimiter (//):

- The delimiter is the character or string of characters that you'll use to tell the mySQL client that you've finished typing in an SQL statement. For ages, the delimiter has always been a semicolon. That, however, causes problems, because, in a stored procedure, one can have many statements, and each must end with a semicolon.

Understand Basic about Procedures:

Consider the following schema and creates tables.

1. Borrower (Roll_No, Name, Date_of_Issue, Book_Name, Status)

Create Table Borrower (Roll_NO int Primary Key, Name Varchar(15),
Date_of_Issue Date, Book_Name Varchar(15), Status Varchar(10));

mysql> Insert Into Borrower Values(101,'Raghavi','2019-08-10','DBMS','Issue');

Query OK, 1 row affected (0.07 sec)

mysql> Insert Into Borrower Values(102,'Lobhas','2019-07-13','SQL','Issue');

Query OK, 1 row affected (0.10 sec)

mysql> Insert Into Borrower Values(103,'Sumit','2019-07-14','DS','Issue');

Query OK, 1 row affected (0.08 sec)

mysql> Insert Into Borrower Values(104,'Sham','2019-07-20','TOC','Issue');

Query OK, 1 row affected (0.15 sec)

mysql> Insert Into Borrower Values(105,'Mohit','2019-06-15','MATH','Issue');

Query OK, 1 row affected (0.15 sec)

```
mysql> select * from Borrower;
+---------+---------+---------------+-----------+--------+
| Roll_NO | Name    | Date_of_Issue | Book_Name | Status |
+---------+---------+---------------+-----------+--------+
|     101 | Raghavi | 2019-08-10    | DBMS      | Issue  |
|     102 | Lobhas  | 2019-07-13    | SQL       | Issue  |
|     103 | Sumit   | 2019-07-14    | DS        | Issue  |
|     104 | Sham    | 2019-07-20    | TOC       | Issue  |
|     105 | Mohit   | 2019-06-15    | MATH      | Issue  |
+---------+---------+---------------+-----------+--------+
5 rows in set (0.00 sec)
```

2. Fine(Roll_no,Fine_Date,Amount)

Create Table Fine(Roll_no int References Borrower(Roll_NO), Fine_Date Date, Amount int);

Before writing the procedure change the Delimiter //

mysql>Delimiter //

1. Create Procedure:

```
Create Procedure BN_GetData( )
Begin
    Select * from Borrower;
End //
```

Run procedure:- mysql> call BN_GetData( )//

2. With IN Parameter :

```
Create Procedure BN_GateData1( IN ID Int)
Begin
    Select * from Borrower
    Where Roll_NO= ID;
End //
```

Run procedure:- mysql> call BN_GetData(105)//

3. With Out Parameter:

```
Create Procedure BN_Data_out(OUT R1 Int)
Begin
    Set R1=10;
End //
```

Run procedure:- mysql> call BN_Data_out(@ S1)//

mysql>select @s1//

4. Variables in Procedures:

Query- Insert new row into table Borrower

Solution: As per the Borrower table we required five variables to insert data into borrower table

```
Create Procedure BN_Variables( )
Begin
    Declare Var1 int;
    Declare Var2, Var3,Var4 varchar (20);
    Declare Var5 Date;
    SET Var1=106;
```

```
    SET Var2= 'Nyan';
    SET Var3= 'ML';
    SET Var4= 'Issue';
    SET Var5= '2019-08-22';
    Insert into Borrower values (Var1,Var2,Var5,Var3,Var4);
    Select * from Borrower;
End//
```

```
mysql> call BN_Variables()//
+---------+---------+---------------+-----------+--------+
| Roll_NO | Name    | Date_of_Issue | Book_Name | Status |
+---------+---------+---------------+-----------+--------+
|     101 | Raghavi | 2019-08-10    | DBMS      | Issue  |
|     102 | Lobhas  | 2019-07-13    | SQL       | Issue  |
|     103 | Sumit   | 2019-07-14    | DS        | Issue  |
|     104 | Sham    | 2019-07-20    | TOC       | Issue  |
|     105 | Mohit   | 2019-06-15    | MATH      | Issue  |
|     106 | Nyan    | 2019-08-22    | ML        | Issue  |
+---------+---------+---------------+-----------+--------+
6 rows in set (0.08 sec)
```

4. IF-Then-Else

```
Create Procedure BN_IF_Then_Else(IN T1 Int)
 Begin
      IF(T1>101&&T1<107)
      Then
          select * from Borrower
          where Roll_NO=T1;
      Else
          Select 'Not Found';
      End If;
End//
```

```
mysql> Call BN_IF_Then_Else(102)//
+---------+--------+---------------+-----------+--------+
| Roll_NO | Name   | Date_of_Issue | Book_Name | Status |
+---------+--------+---------------+-----------+--------+
|     102 | Lobhas | 2019-07-13    | SQL       | Issue  |
+---------+--------+---------------+-----------+--------+
1 row in set (0.07 sec)

Query OK, 0 rows affected (0.07 sec)

mysql> Call BN_IF_Then_Else(108)//
+-----------+
| Not Found |
+-----------+
| Not Found |
+-----------+
1 row in set (0.00 sec)
```

5. Loops:

WHILE ... END WHILE

Create table Temp (val int)//

```
Create Procedure BN_While_Loop()
Begin
    DECLARE v INT;
    SET v = 0;
    WHILE v < 5 DO
        Insert Into Temp Values (v);
        SET v = v + 1;
    END WHILE;
END//
```

```
mysql> call BN_While_Loop()//
Query OK, 1 row affected (0.02 sec)

mysql> Select * from Temp//
+------+
| var  |
+------+
|    0 |
|    1 |
|    2 |
|    3 |
|    4 |
+------+
5 rows in set (0.00 sec)
```

REPEAT ... END REPEAT

Create table Temp1(val int)

```
Create Procedure BN_Reapet_Loop  ()
Begin
    Declare v Int;
    SET v = 0;
    REPEAT
        Insert into Temp1 values (v);
        SET v = v + 1;
   UNTIL v >= 5
   END REPEAT;
End; //
```

```
mysql> call BN_Reapet_Loop()//
Query OK, 1 row affected (0.09 sec)

mysql> Select * from Temp1;
    -> //
+------+
| var  |
+------+
|    0 |
|    1 |
|    2 |
|    3 |
|    4 |
+------+
5 rows in set (0.00 sec)
```

Query: Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day. If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day. After submitting the book, status will change from Issue to Return. If condition of fine is true, then details will be stored into fine table.

```
Delimiter //
/**************************************************************************
Description: Accpet Roll Number and Book Name and calculate Fine amount as per
             Number of days. Handle Exception if Roll Number not found.
Create Date: 03-05-2022
**************************************************************************/
Create Procedure BRN( Roll_New int,Book_Name Varchar(15))
BEGIN
   Declare X1 integer;
   Declare Continue Handler for NOT FOUND
    BEGIN
          Select 'Not Found';
    END;
    Select DATEDIFF ( curdate(), Date_of_Issue) into X1 from Borrower where
Roll_NO=Roll_New;
      IF (X1>15&&X1<30)
      Then
          Insert Into Fine Values( Roll_New, curdate(),(X1*5));
         UPDATE Borrower SET Status='Return' WHERE Roll_NO=Roll_New;

       End IF;
            IF (X1>30)
            Then
                Insert into Fine values( Roll_New, curdate(),(X1*50));
               UPDATE Borrower SET Status='Return' WHERE Roll_NO=Roll_New;

             End IF ;
END;
  //
```

Experiment No. 6

Problem Statement: : Write a SQL procedure for cursor and trigger.

Software Required: MySQL, Ubuntu 14.04 or above

Theory:

In MySQL, a cursor allows row-by-row processing of the result sets. A cursor is used for the result set and returned from a query. By using a cursor, you can iterate, or by step through the results of a query and perform certain operations on each row. The cursor allows you to iterate through the result set and then perform the additional processing only on the rows that require it.

In a cursor contains the data in a loop. Cursors may be different from SQL commands that operate on all the rows in the returned by a query at one time.

There are some steps we have to follow, given below:

- Declare a cursor
- open a cursor statement
- Fetch the cursor
- close the cursor

1. Declaration of Cursor: To declare a cursor you must use the DECLARE statement. With the help of the variables, conditions and handlers we need to declare a cursor before we can use it. first of all we will give the cursor a name, this is how we will refer to it later in the procedure. We can have more than one cursor in a single procedure so its necessary to give it a name that will in some way tell us what its doing. We then need to specify the select statement we want to associate with the cursor

Syntax: DECLARE *cursor_name* CURSOR FOR *select_statement;*

2. Open a cursor statement: For open a cursor we must use the open statement. If we want to fetch rows from it you must open the cursor.

Syntax: OPEN cursor_name;

3. Cursor fetch statement: When we have to retrieve the next row from the cursor and move the cursor to next row then you need to fetch the cursor.

Synatx : FETCH cursor_name INTO var_name;

If any row exists, then the above statement fetches the next row and cursor pointer moves ahead to the next row.

4. Cursor close statement: By this statement closed the open cursor.

Syntax: CLOSE_name;

When an error occurs inside a stored procedure, it is important to handle it appropriately, such as continuing or exiting the current code block's execution, and issuing a meaningful error message. MySQL provides an easy way to define handlers that handle from general conditions such as warnings or exceptions to specific conditions e.g., specific error codes.

Declaring a handler

To declare a handler, you use the `DECLARE HANDLER` statement as follows:

```
DECLARE action HANDLER FOR condition_value statement;
```

The action accepts one of the following values:

CONTINUE: the execution of the enclosing code block (BEGIN … END) continues.

EXIT: the execution of the enclosing code block, where the handler is declared, terminates.

Query:

Write a Stored Procedure namely proc_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and900 category is first class, if marks 899 and 825 category is Higher Second Class. Write a PL/SQL block for using procedure created with above requirement.

- Stud_Marks(Roll_No,name, total_marks)
- Result(Roll,Name, Class)

1.Stud_Marks

Mysql> Create Table Stud_Marks(Roll_No Int,Name Varchar(15),Total_Marks Int);

2. Result

Mysql> Create Table result(Roll_No Int,Class Varchar (15));

Insert the suitable Data into table Stud_Marks

```
/*************************************************************************
**
Description: Read Data from Stud_Marks Table i.e roll number and marks and as per problem
statement categorization of marks and store the result into result table.
 Create Date: 10-09-2022
*************************************************************************
*/
Create Procedure Pro_Grade()
Begin
Declare v1 Int;
Declare v2 Int;
Declare noMoreRows int;
Declare  Grade_Check cursor for select Roll_No,Total_Marks from  Stud_Marks ;
Declare continue handler for not found set noMoreRows = 1;
  Set noMoreRows = 0;
Open Grade_Check;
 Loop_Name:Loop
    Fetch Grade_Check into v1,v2;
        If (v2>=990&&v2<=1500)
        Then
              Insert into result values(v1,'Distinction');
        End If;
        If(v2>=900&&v2<=989)
        Then
             Insert Into result values(v1,'First Class');
         End If;

         If(v2>=825&&v2<=899)
         Then
               Insert Into result values(v1,'Higher Sec.Class');
          End If;
       If noMoreRows Then
            Leave Loop_Name;
       End If;
```

End loop Loop_Name;

Close Grade_Check;

End;

Triggers:

In MySQL, a trigger is a stored program invoked automatically in response to an event such as insert, update, or delete that occurs in the associated table. For example, you can define a trigger that is invoked automatically before a new row is inserted into a table.

MySQL supports triggers that are invoked in response to the INSERT, UPDATE or DELETE event.

The SQL standard defines two types of triggers: row-level triggers and statement-level triggers.

- A row-level trigger is activated for each row that is inserted, updated, or deleted. For example, if a table has 100 rows inserted, updated, or deleted, the trigger is automatically invoked 100 times for the 100 rows affected.
- A statement-level trigger is executed once for each transaction regardless of how many rows are inserted, updated, or deleted.

MySQL supports only row-level triggers. It doesn't support statement-level triggers.

Managing MySQL triggers

Create triggers – describe steps of how to create a trigger in MySQL.

Drop triggers – show you how to drop a trigger.

Create a BEFORE INSERT trigger – show you how to create a `BEFORE INSERT` trigger to maintain a summary table from another table.

Create an AFTER INSERT trigger – describe how to create an `AFTER INSERT` trigger to insert data into a table after inserting data into another table.

Create a BEFORE UPDATE trigger – learn how to create a `BEFORE UPDATE` trigger that validates data before it is updated to the table.

Create an AFTER UPDATE trigger – show you how to create an `AFTER UPDATE` trigger to log the changes of data in a table.

Create a BEFORE DELETE trigger – show how to create a `BEFORE DELETE` trigger.

Create an AFTER DELETE trigger – describe how to create an `AFTER DELETE` trigger.

Show triggers – list triggers in a database, table by specific patterns.

Query:

Write a database trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library_Audit table.

Create Tables:

Mysql> Create table Library (Book_ID int, Title Varchar(20), Auther_Name Varchar(20),Dept Varchar(20));

Insert suitable data into Library table.

Mysql> Create table Lib_Audit(Book_ID Int,Old_Book_ID_Before_Update Int, Up_Del_Date Date, Staus Vacrchar(20));

Create Trigger

1. Trigger for Update data of library table:

```
CREATE TRIGGER Lib_Trig_Update AFTER update ON Library
FOR EACH ROW
BEGIN
INSERT INTO Lib_Audit (Book_ID,Old_Book_ID_Before_Update,Up_Del_Date,Status)
Values( NEW.Book_ID,Old.Book_ID,Now(),'Update');
END;
```

2. Trigger for Delete Data of library table:

```
CREATE TRIGGER Lib_Trig_Delete AFTER Delete ON Library
FOR EACH ROW
BEGIN
INSERT INTO Lib_Audit (Book_ID,Old_Book_ID_Before_Update,Up_Del_Date,Status)
Values( Old.Book_ID, 0 ,Now(),'Delete');
END;
```

Experiment No. 7

Problem Statement: : Write SQL to apply Aggregating Data using Group functions.

Software Required: MySQL, Ubuntu 14.04 or above

Theory:

In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.
Include **COUNT, SUM, MAX, MIN, and AVG.**

Example 1**:** Find the maximum salary, the minimum salary, and the average salary among all employees.

```
SELECT      MAX(SALARY),MIN(SALARY), AVG(SALARY)
FROM        EMPLOYEE;
```

Example 2: Retrieve the total number of employees in the company , and the number of employees in the 'Research' department.

```
a)  SELECT  COUNT (*)
    FROM    EMPLOYEE;
b)  SELECT  COUNT (*)
    FROM    EMPLOYEE, DEPARTMENT
    WHERE DNO=DNUMBER AND DNAME='Research';
```

Grouping:

In many cases, we want to apply the aggregate functions to *subgroups of tuples* in a relation.Each subgroup of tuples consists of the set of tuples that have the *same value* for the *grouping attribute(s).*The function is applied to each subgroup independently.

SQL has a GROUP BY-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*

Example: For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
SELECT      DNO, COUNT (*), AVG (SALARY)
FROM        EMPLOYEE
GROUP BY    DNO;
```

In above query the EMPLOYEE tuples are divided into groups. Each group having the same value for the grouping attribute DNO. The COUNT and AVG functions are applied to each such group of tuples separately. The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples. A join condition can be used in conjunction with grouping.

Sometimes we want to retrieve the values of these functions for only those *groups that satisfy certain conditions.*

The **HAVING**-clause is used for specifying a selection condition on groups (rather than on individual tuples)

Example: For each project *on which more than two employees work*, retrieve the project number, project name, and the number of employees who work on that project.

```
Q22:    SELECT      PNUMBER, PNAME, COUNT(*)
        FROM        PROJECT, WORKS_ON
        WHERE       PNUMBER=PNO
        GROUP BY    PNUMBER, PNAME
        HAVING      COUNT (*) > 2
```

Experiment No. 8

Problem Statement: : Design a queries for suitable database application using SQL DML statements: all types of Join,Views.

Software Required: MySQL, Ubuntu 14.04 or above

Theory:

MySQL **JOINS** are used to retrieve data from multiple tables. A MySQL JOIN is performed whenever two or more tables are joined in a SQL statement.

INNER JOIN (or sometimes called simple join)

LEFT OUTER JOIN (or sometimes called LEFT JOIN)

RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)

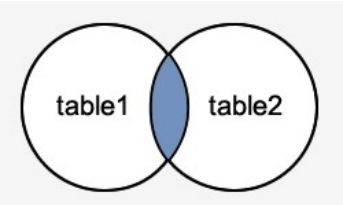
INNER JOIN (or sometimes called simple join):

MySQL INNER JOINS return all rows from multiple tables where the join condition is met.

Syntax

The syntax for the INNER JOIN in MySQL is:

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```



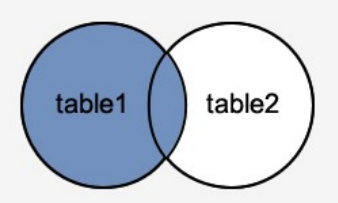
The MySQL INNER JOIN would return the records where *table1* and *table2* intersect.

LEFT OUTER JOIN (or sometimes called LEFT JOIN)

This type of join returns all rows from the LEFT-hand table specified in the ON condition and **only** those rows from the other table where the joined fields are equal (join condition is met).

The syntax for the LEFT OUTER JOIN in MySQL is:

```
SELECT columns
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```



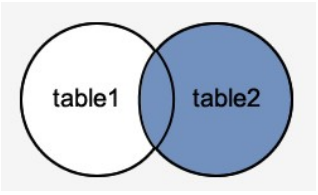
The MySQL LEFT OUTER JOIN would return the all records from *table1* and only those records from *table2* that intersect with *table1*.

RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)

This type of join returns all rows from the RIGHT-hand table specified in the ON condition and **only** those rows from the other table where the joined fields are equal (join condition is met).

The syntax for the RIGHT OUTER JOIN in MySQL is:

```
SELECT columns
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```



The MySQL RIGHT OUTER JOIN would return the all records from *table2* and only those records from *table1* that intersect with *table2*.

Create Table

Create table for following Schema

Schema:

Supplier(Sup_ID- Primary Key,Sup_Name ,Address)

Parts(Part_ID- Primary Key,Part_Name,Part_Type)

Orders(Order_No,Part_ID,Sup_ID,Order_Date)

Catalog(P_ID,Part_Name,Cost, Available_Stock)

Load Data into Supplier Table:

mysql> Insert into Supplier values(101,'Akash Auto','Chakan');

Query OK, 1 row affected (0.08 sec)

mysql> Insert into Supplier Values (102,'HITACH','Pune');

Query OK, 1 row affected (0.08 sec)

mysql> Insert into Supplier Values (103,'OM Breaks', 'Noida');

Query OK, 1 row affected (0.08 sec)

mysql> Insert into Supplier Values (104,'BMC Steel','Rachi');

Query OK, 1 row affected (0.08 sec)

mysql> Insert into Supplier Values (105,'Telco Glass','Mumbai');

Query OK, 1 row affected (0.08 sec)

mysql> Insert into Supplier Values (106,'Ashok Leyland','Chenni');

Query OK, 1 row affected (0.08 sec)

mysql> Insert into Supplier Values (107,'Bharat BENZ','Surat');

Query OK, 1 row affected (0.08 sec)

```
mysql> select * from Supplier;
+------+---------------+---------+
| S_ID | S_Name        | Address |
+------+---------------+---------+
|  101 | Akash Auto    | Chakan  |
|  102 | HITACH        | Pune    |
|  103 | OM Breaks     | Noida   |
|  104 | BMC Steel     | Rachi   |
|  105 | Telco Glass   | Mumbai  |
|  106 | Ashok Leyland | Chenni  |
|  107 | Bharat BENZ   | Surat   |
+------+---------------+---------+
7 rows in set (0.00 sec)
```

Load Data into Parts Table:

mysql>Insert into Parts Values(301,'Fuel Pump','Engine');

Query OK, 1 row affected (0.08 sec)

mysql> Insert into Parts Values(302,'Break Liners','Break System');

Query OK, 1 row affected (0.08 sec)

mysql> Insert into Parts Values(303,'Head Lamps','Car Body');

Query OK, 1 row affected (0.01 sec)

mysql> Insert into Parts Values(304,'Gear Box','Engine');

Query OK, 1 row affected (0.01 sec)

mysql> Insert into Parts Values(305,'AC Button','Dash Bord');

Query OK, 1 row affected (0.01 sec)

mysql> Insert into Parts Values(306,'Music System','Dash Bord');

Query OK, 1 row affected (0.08 sec)

mysql> Insert into Parts Values(307,'Battery','Electric');

Query OK, 1 row affected (0.07 sec)

```
mysql> Select * from Parts;
+---------+--------------+--------------+
| Part_ID | Part_Name    | Part_Type    |
+---------+--------------+--------------+
|     301 | Fuel Pump    | Engine       |
|     302 | Break Liners | Break System |
|     303 | Head Lamps   | Car Body     |
|     304 | Gear Box     | Engine       |
|     305 | AC Button    | Dash Bord    |
|     306 | Music System | Dash Bord    |
|     307 | Battery      | Electric     |
+---------+--------------+--------------+
7 rows in set (0.00 sec)
```

Load Data into Orders Table:

mysql> Insert into Orders Values(1001,301,102,'2017-02-03');

Query OK, 1 row affected (0.09 sec)

mysql> Insert into Orders Values(1002,302,103,'2017-04-10');

Query OK, 1 row affected (0.01 sec)

mysql> Insert into Orders Values(1003,303,104,'2017-05-11');

Query OK, 1 row affected (0.04 sec)

mysql> Insert into Orders Values(1004,304,102,'2017-05-11');

Query OK, 1 row affected (0.01 sec)

mysql> Insert into Orders Values(1005,305,101,'2017-06-11');

Query OK, 1 row affected (0.09 sec)

mysql> Insert into Orders Values(1006,306,103,'2017-07-11');

Query OK, 1 row affected (0.01 sec)

```
mysql> Select * from Orders;
+----------+---------+------+------------+
| Order_No | Part_ID | S_ID | Order_Date |
+----------+---------+------+------------+
|     1001 |     301 |  102 | 2017-02-03 |
|     1002 |     302 |  103 | 2017-04-10 |
|     1003 |     303 |  104 | 2017-05-11 |
|     1004 |     304 |  102 | 2017-05-11 |
|     1005 |     305 |  101 | 2017-06-11 |
|     1006 |     306 |  103 | 2017-07-11 |
+----------+---------+------+------------+
6 rows in set (0.00 sec)
```

Load Data into Catalog Table:

mysql> Insert into Catalog Values(301,'Fuel Pump',3000,12);

Query OK, 1 row affected (0.09 sec)

mysql> Insert into Catalog Values(302,'Break Liners',1600,10);

Query OK, 1 row affected (0.03 sec)

mysql> Insert into Catalog Values(303,'Head Lamps',7000,06);

Query OK, 1 row affected (0.08 sec)

mysql> Insert into Catalog Values(304,'Gear Box',12000,02);

Query OK, 1 row affected (0.01 sec)

mysql> Insert into Catalog Values(305,'AC Button',700,25);

Query OK, 1 row affected (0.02 sec)

mysql> Insert into Catalog Values(306,'Music System',22000,02);

Query OK, 1 row affected (0.01 sec)

```
mysql> Select * from Catalog;
+---------+--------------+-------+-----------------+
| Part_ID | Part_Name    | Cost  | Available_Stock |
+---------+--------------+-------+-----------------+
|     301 | Fuel Pump    |  3000 |              12 |
|     302 | Break Liners |  1600 |              10 |
|     303 | Head Lamps   |  7000 |               6 |
|     304 | Gear Box     | 12000 |               2 |
|     305 | AC Button    |   700 |              25 |
|     306 | Music System | 22000 |               2 |
+---------+--------------+-------+-----------------+
6 rows in set (0.00 sec)
```

1. Find the Supplier ID's and date of supply of supplier from the list of suppliers (Solve using Join).

```
mysql> select Supplier.S_ID,Orders.Order_Date
    -> from Supplier
    -> INNER JOIN Orders
    -> ON Supplier.S_ID=Orders.S_ID;
+------+------------+
| S_ID | Order_Date |
+------+------------+
|  102 | 2017-02-03 |
|  103 | 2017-04-10 |
|  104 | 2017-05-11 |
|  102 | 2017-05-11 |
|  101 | 2017-06-11 |
|  103 | 2017-07-11 |
+------+------------+
6 rows in set (0.05 sec)
```

2. Find All Suppliers who supply parts as well as not.

```
mysql> Select Supplier.S_ID,Orders.Order_Date
    -> from Supplier
    -> LEFT OUTER JOIN
    -> Orders
    -> ON
    -> Supplier.S_ID=Orders.S_ID;
+------+------------+
| S_ID | Order_Date |
+------+------------+
|  101 | 2017-06-11 |
|  102 | 2017-02-03 |
|  102 | 2017-05-11 |
|  103 | 2017-04-10 |
|  103 | 2017-07-11 |
|  104 | 2017-05-11 |
|  105 | NULL       |
|  106 | NULL       |
+------+------------+
8 rows in set (0.00 sec)
```

3. Find all suppliers who supply parts.(Using Join)

```
mysql> Select Supplier.S_ID,Orders.Order_Date
    -> from Supplier
    -> RIGHT OUTER JOIN
    -> Orders
    -> ON
    -> Supplier.S_ID=Orders.S_ID;
+------+------------+
| S_ID | Order_Date |
+------+------------+
|  102 | 2017-02-03 |
|  103 | 2017-04-10 |
|  104 | 2017-05-11 |
|  102 | 2017-05-11 |
|  101 | 2017-06-11 |
|  103 | 2017-07-11 |
+------+------------+
6 rows in set (0.00 sec)
```

4. Create View for total cost of every part which is ordered

```
mysql> Create View Total_Part_Cost As
    -> Select Part_Name,Cost * Available_Stock
    -> From Catalog;
Query OK, 0 rows affected (0.30 sec)

mysql> select * from Total_Part_Cost;
+--------------+------------------------+
| Part_Name    | Cost * Available_Stock |
+--------------+------------------------+
| Fuel Pump    |                  36000 |
| Break Liners |                  23000 |
| Head Lamps   |                  42000 |
| Gear Box     |                  24000 |
| AC Button    |                  17500 |
| Music System |                  44000 |
+--------------+------------------------+
6 rows in set (0.08 sec)
```

5. Find the Details of supplier who supply a 'Fuel Pumps'.

```
mysql> Select * From Supplier
    -> Where S_ID = (Select S_ID From Orders
    ->               Where Part_ID = ( Select Part_ID From Catalog
    ->                                 Where Part_Name='Fuel Pump'));
+------+--------+---------+
| S_ID | S_Name | Address |
+------+--------+---------+
|  102 | HITACH | Pune    |
+------+--------+---------+
1 row in set (0.00 sec)
```

Experiment No. 9

Problem Statement: : Normalization in DBMS

Normalization is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly

Anomalies in DBMS

There are three types of **anomalies that occur when the database is not normalized**. These are: Insertion, update and deletion anomaly. Let's take an example to understand this.

Example: A manufacturing company stores the employee details in a table `Employee` that has four attributes: `Emp_Id` for storing employee's id, `Emp_Name` for storing employee's name, `Emp_Address` for storing employee's address and `Emp_Dept` for storing the department details in which the employee works. At some point of time the table looks like this:

Emp_Id	Emp_Name	Emp_Address	Emp_Dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

This table is not normalized. We will see the problems that we face when a table in database is not normalized.

Update anomaly: In the above table we have two rows for employee `Rick` as he belongs to two departments of the company. If we want to update the address of `Rick` then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, `Rick` would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if `Emp_Dept` field doesn't allow `null`.

Delete anomaly: Let's say in future, company closes the department `D890` then deleting the rows that are having `Emp_Dept` as D890 would also delete the information of employee `Maggie` since she is assigned only to this department.

To overcome these anomalies we need to normalize the data. In the next section we will discuss about normalization.

Normalization

Here are the most commonly used normal forms:

- First normal form(1NF)
- Second normal form(2NF)
- Third normal form(3NF)
- Boyce & Codd normal form (BCNF)

First normal form (1NF)

A relation is said to be in **1NF (first normal form)**, if it doesn't contain any multi-valued attribute. In other words you can say that a relation is in 1NF if each attribute contains only atomic(single) value only.

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Example: Let's say a company wants to store the names and contact details of its employees. It creates a table in the database that looks like this:

Emp_Id	Emp_Name	Emp_Address	Emp_Mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212 , 9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123, 8123450987

Two employees (`Jon` & `Lester`) have two mobile numbers that caused the `Emp_Mobile` field to have multiple values for these two employees.

This table is **not in 1NF** as the rule says “each attribute of a table must have atomic (single) values”, the Emp_Mobile values for employees `Jon` & `Lester` violates that rule.

To make the table complies with 1NF we need to create separate rows for the each mobile number in such a way so that none of the attributes contains multiple values.

Emp_Id	Emp_Name	Emp_Address	Emp_Mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
102	Jon	Kanpur	9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123
104	Lester	Bangalore	8123450987

Second normal form (2NF)

A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- No non-prime attribute is dependent on the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

Example: Let's say a school wants to store the data of teachers and the subjects they teach. They create a table `Teacher` that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

Teacher_Id	Subject	Teacher_Age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {`Teacher_Id`, `Subject`}

Non prime attribute: `Teacher_Age`

This table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute `Teacher_Age` is dependent on `Teacher_Id` alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says **"no non-prime attribute is dependent on the proper subset of any candidate key of the table"**.

To make the table complies with 2NF we can disintegrate it in two tables like this:

Teacher_Details table:

Teacher_Id	Teacher_Age
111	38
222	38
333	40

Teacher_Subject table:

Teacher_Id	Subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.

An attribute that is not part of any candidate key is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a super key of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Let's say a company wants to store the complete address of each employee, they create a table named `Employee_Details` that looks like this:

Emp_Id	Emp_Name	Emp_Zip	Emp_State	Emp_City	Emp_District
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {`Emp_Id`}, {`Emp_Id`, `Emp_Name`}, {`Emp_Id`, `Emp_Name`, `Emp_Zip`}…so on

Candidate Keys: {`Emp_Id`}

Non-prime attributes: all attributes except `Emp_Id` are non-prime as they are not part of any candidate keys.

Here, `Emp_State`, `Emp_City` & `Emp_District` dependent on `Emp_Zip`. Further `Emp_zip` is dependent on `Emp_Id` that makes non-prime attributes (`Emp_State`, `Emp_City` & `Emp_District`) transitively dependent on super key (`Emp_Id`). This violates the rule of 3NF.

To make this table complies with 3NF we have to disintegrate the table into two tables to remove the transitive dependency:

Employee Table:

Emp_Id	Emp_Name	Emp_Zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

Employee_Zip table:

Emp_Zip	Emp_State	Emp_City	Emp_District
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

Boyce Codd normal form (BCNF)

It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.

Example: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

Emp_Id	Emp_Nationality	Emp_Dept	Dept_Type	Dept_No_Of_Emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Functional dependencies in the table above:

`Emp_Id` -> `Emp_Nationality`

`Emp_Dept` -> {`Dept_Type`, `Dept_No_Of_Emp`}

Candidate key: {`Emp_Id`, `Emp_Dept`}

The table is not in BCNF as neither `Emp_Id` nor `Emp_Dept` alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:

Emp_Nationality table:

Emp_Id	Emp_Nationality
1001	Austrian
1002	American

Emp_Dept table:

Emp_Dept	Dept_Type	Dept_No_Of_Emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

Emp_Dept_Mapping table:

Emp_Id	Emp_Dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

Functional dependencies:

`Emp_Id` -> `Emp_Nationality`

`Emp_Dept` -> {`Dept_Type`, `Dept_No_Of_Emp`}

Candidate keys:

For first table: `Emp_Id`

For second table: `Emp_Dept`

For third table: {`Emp_Id`, `Emp_Dept`}

This table is now in BCNF as in both the functional dependencies left side part is a key.

50 Queries for Practice

```
CREATE DATABASE Bhushan;
SHOW DATABASES;
USE Bhushan;

CREATE TABLE Worker (
        WORKER_ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
        FIRST_NAME CHAR(25),
        LAST_NAME CHAR(25),
        SALARY INT(15),
        JOINING_DATE DATETIME,
        DEPARTMENT CHAR(25)
);

INSERT INTO Worker (WORKER_ID, FIRST_NAME, LAST_NAME, SALARY,
JOINING_DATE, DEPARTMENT) VALUES
                (001, 'Monika', 'Arora', 100000, '14-02-20 09.00.00', 'HR'),
                (002, 'Niharika', 'Verma', 80000, '14-06-11 09.00.00', 'Admin'),
                (003, 'Vishal', 'Singhal', 300000, '14-02-20 09.00.00', 'HR'),
                (004, 'Amitabh', 'Singh', 500000, '14-02-20 09.00.00', 'Admin'),
                (005, 'Vivek', 'Bhati', 500000, '14-06-11 09.00.00', 'Admin'),
                (006, 'Vipul', 'Diwan', 200000, '14-06-11 09.00.00', 'Account'),
                (007, 'Satish', 'Kumar', 75000, '14-01-20 09.00.00', 'Account'),
                (008, 'Geetika', 'Chauhan', 90000, '14-04-11 09.00.00', 'Admin');
CREATE TABLE Bonus (
        WORKER_REF_ID INT,
        BONUS_AMOUNT INT(10),
        BONUS_DATE DATETIME,
        FOREIGN KEY (WORKER_REF_ID)
                REFERENCES Worker(WORKER_ID)
    ON DELETE CASCADE
);
```

```
INSERT INTO Bonus (WORKER_REF_ID, BONUS_AMOUNT, BONUS_DATE) VALUES
                (001, 5000, '16-02-20'),
                (002, 3000, '16-06-11'),
                (003, 4000, '16-02-20'),
                (001, 4500, '16-02-20'),
                (002, 3500, '16-06-11');

CREATE TABLE Title (
        WORKER_REF_ID INT,
        WORKER_TITLE CHAR(25),
        AFFECTED_FROM DATETIME,
        FOREIGN KEY (WORKER_REF_ID)
                REFERENCES Worker(WORKER_ID)
    ON DELETE CASCADE
);
INSERT INTO Title (WORKER_REF_ID, WORKER_TITLE, AFFECTED_FROM) VALUES
 (001, 'Manager', '2016-02-20 00:00:00'),
 (002, 'Executive', '2016-06-11 00:00:00'),
 (008, 'Executive', '2016-06-11 00:00:00'),
 (005, 'Manager', '2016-06-11 00:00:00'),
 (004, 'Asst. Manager', '2016-06-11 00:00:00'),
 (007, 'Executive', '2016-06-11 00:00:00'),
 (006, 'Lead', '2016-06-11 00:00:00'),
 (003, 'Lead', '2016-06-11 00:00:00');
```

Q-1. Write an SQL query to fetch "FIRST_NAME" from Worker table using the alias name as <WORKER_NAME>.

Ans.

```
Select FIRST_NAME AS WORKER_NAME from Worker;
```

Q-2. Write an SQL query to fetch "FIRST_NAME" from Worker table in upper case.

Ans.

The required query is:

```
Select upper(FIRST_NAME) from Worker;
```

Q-3. Write an SQL query to fetch unique values of DEPARTMENT from Worker table.

Ans.

The required query is:

```
Select distinct DEPARTMENT from Worker;
```

Q-4. Write an SQL query to print the first three characters of FIRST_NAME from Worker table.

Ans.

```
Select substring(FIRST_NAME,1,3) from Worker;
```

Q-5. Write an SQL query to find the position of the alphabet ('a') in the first name column 'Amitabh' from Worker table.

Ans.

The required query is:

```
Select INSTR(FIRST_NAME, BINARY'a') from Worker where FIRST_NAME = 'Amitabh';
```

Notes.

- The INSTR method is in case-sensitive by default.
- Using Binary operator will make INSTR work as the case-sensitive function.

Q-6. Write an SQL query to print the FIRST_NAME from Worker table after removing white spaces from the right side.

Ans.

The required query is:

```
Select RTRIM(FIRST_NAME) from Worker;
```

Q-7. Write an SQL query to print the DEPARTMENT from Worker table after removing white spaces from the left side.

Ans.

The required query is:

```
Select LTRIM(DEPARTMENT) from Worker;
```

Q-8. Write an SQL query that fetches the unique values of DEPARTMENT from Worker table and prints its length.

Ans.

The required query is:

```
Select distinct length(DEPARTMENT) from Worker;
```

Q-9. Write an SQL query to print the FIRST_NAME from Worker table after replacing 'a' with 'A'.

Ans.

The required query is:

```
Select REPLACE(FIRST_NAME,'a','A') from Worker;
```

Q-10. Write an SQL query to print the FIRST_NAME and LAST_NAME from Worker table into a single column COMPLETE_NAME. A space char should separate them.

Ans.

The required query is:

```
Select CONCAT(FIRST_NAME, ' ', LAST_NAME) AS 'COMPLETE_NAME' from Worker;
```

Q-11. Write an SQL query to print all Worker details from the Worker table order by FIRST_NAME Ascending.

Ans.

The required query is:

```
Select * from Worker order by FIRST_NAME asc;
```

Q-12. Write an SQL query to print all Worker details from the Worker table order by FIRST_NAME Ascending and DEPARTMENT Descending.

Ans.

The required query is:

```
Select * from Worker order by FIRST_NAME asc,DEPARTMENT desc;
```

Q-13. Write an SQL query to print details for Workers with the first name as "Vipul" and "Satish" from Worker table.

Ans.

The required query is:

```
Select * from Worker where FIRST_NAME in ('Vipul','Satish');
```

Q-14. Write an SQL query to print details of workers excluding first names, "Vipul" and "Satish" from Worker table.

Ans.

The required query is:

```
Select * from Worker where FIRST_NAME not in ('Vipul','Satish');
```

Q-15. Write an SQL query to print details of Workers with DEPARTMENT name as "Admin".

Ans.

The required query is:

```
Select * from Worker where DEPARTMENT like 'Admin%';
```

Q-16. Write an SQL query to print details of the Workers whose FIRST_NAME contains 'a'.

Ans.

The required query is:

```
Select * from Worker where FIRST_NAME like '%a%';
```

Q-17. Write an SQL query to print details of the Workers whose FIRST_NAME ends with 'a'.

Ans.

The required query is:

Select * from Worker where FIRST_NAME like '%a';

Q-18. Write an SQL query to print details of the Workers whose FIRST_NAME ends with ‘h’ and contains six alphabets.

Ans.

The required query is:

Select * from Worker where FIRST_NAME like '_____h';

Q-19. Write an SQL query to print details of the Workers whose SALARY lies between 100000 and 500000.

Ans.

The required query is:

Select * from Worker where SALARY between 100000 and 500000;

Q-20. Write an SQL query to print details of the Workers who have joined in Feb’2014.

Ans.

The required query is:

Select * from Worker where year(JOINING_DATE) = 2014 and month(JOINING_DATE) = 2;

Q-21. Write an SQL query to fetch the count of employees working in the department ‘Admin’.

Ans.

The required query is:

SELECT COUNT(*) FROM worker WHERE DEPARTMENT = 'Admin';

Q-22. Write an SQL query to fetch worker names with salaries >= 50000 and <= 100000.

Ans.

The required query is:

SELECT CONCAT(FIRST_NAME, ' ', LAST_NAME) As Worker_Name, Salary
FROM worker

```
WHERE WORKER_ID IN
(SELECT WORKER_ID FROM worker
WHERE Salary BETWEEN 50000 AND 100000);
```

Q-23. Write an SQL query to fetch the no. of workers for each department in the descending order.

Ans.

The required query is:

```
SELECT DEPARTMENT, count(WORKER_ID) No_Of_Workers
FROM worker
GROUP BY DEPARTMENT
ORDER BY No_Of_Workers DESC;
```

Q-24. Write an SQL query to print details of the Workers who are also Managers.

Ans.

The required query is:

```
SELECT DISTINCT W.FIRST_NAME, T.WORKER_TITLE
FROM Worker W
INNER JOIN Title T
ON W.WORKER_ID = T.WORKER_REF_ID
AND T.WORKER_TITLE in ('Manager');
```

Q-25. Write an SQL query to fetch duplicate records having matching data in some fields of a table.

Ans.

The required query is:

```
SELECT WORKER_TITLE, AFFECTED_FROM, COUNT(*)
FROM Title
GROUP BY WORKER_TITLE, AFFECTED_FROM
HAVING COUNT(*) > 1;
```

Q-26. Write an SQL query to show only odd rows from a table.

Ans.

The required query is:

SELECT * FROM Worker WHERE MOD (WORKER_ID, 2) <> 0;

Q-27. Write an SQL query to show only even rows from a table.

Ans.

The required query is:

SELECT * FROM Worker WHERE MOD (WORKER_ID, 2) = 0;

Q-28. Write an SQL query to clone a new table from another table.

Ans.

The general query to clone a table with data is:

SELECT * INTO WorkerClone FROM Worker;

The general way to clone a table without information is:

SELECT * INTO WorkerClone FROM Worker WHERE 1 = 0;

An alternate way to clone a table (for MySQL) without is:

CREATE TABLE WorkerClone LIKE Worker;

Q-29. Write an SQL query to fetch intersecting records of two tables.

Ans.

The required query is:

(SELECT * FROM Worker)

INTERSECT

(SELECT * FROM WorkerClone);

Q-30. Write an SQL query to show records from one table that another table does not have.

Ans.

The required query is:

SELECT * FROM Worker

MINUS

SELECT * FROM Title;

Q-31. Write an SQL query to show the current date and time.

Ans.

Following MySQL query returns the current date:

SELECT CURDATE();

Following MySQL query returns the current date and time:

SELECT NOW();

Following SQL Server query returns the current date and time:

SELECT getdate();

Following Oracle query returns the current date and time:

SELECT SYSDATE FROM DUAL;

Q-32. Write an SQL query to show the top n (say 10) records of a table.

Ans.

Following MySQL query will return the top n records using the LIMIT method:

SELECT * FROM Worker ORDER BY Salary DESC LIMIT 10;

Following SQL Server query will return the top n records using the TOP command:

SELECT TOP 10 * FROM Worker ORDER BY Salary DESC;

Following Oracle query will return the top n records with the help of ROWNUM:

SELECT * FROM (SELECT * FROM Worker ORDER BY Salary DESC)

WHERE ROWNUM <= 10;

Q-33. Write an SQL query to determine the nth (say n=5) highest salary from a table.

Ans.

The following MySQL query returns the nth highest salary:

SELECT Salary FROM Worker ORDER BY Salary DESC LIMIT n-1,1;

The following SQL Server query returns the nth highest salary:

```
SELECT TOP 1 Salary
FROM (
 SELECT DISTINCT TOP n Salary
 FROM Worker
 ORDER BY Salary DESC
 )
ORDER BY Salary ASC;
```

Q-34. Write an SQL query to determine the 5th highest salary without using TOP or limit method.

Ans.

The following query is using the correlated subquery to return the 5th highest salary:

```
SELECT Salary
FROM Worker W1
WHERE 4 = (
 SELECT COUNT( DISTINCT ( W2.Salary ) )
 FROM Worker W2
 WHERE W2.Salary >= W1.Salary
 );
```

Use the following generic method to find nth highest salary without using TOP or limit.

```
SELECT Salary
FROM Worker W1
WHERE n-1 = (
 SELECT COUNT( DISTINCT ( W2.Salary ) )
 FROM Worker W2
 WHERE W2.Salary >= W1.Salary
 );
```

Q-35. Write an SQL query to fetch the list of employees with the same salary.

Ans.

The required query is:

```
Select distinct W.WORKER_ID, W.FIRST_NAME, W.Salary
from Worker W, Worker W1
where W.Salary = W1.Salary
and W.WORKER_ID != W1.WORKER_ID;
```

Q-36. Write an SQL query to show the second highest salary from a table.

Ans.

The required query is:

```
Select max(Salary) from Worker
where Salary not in (Select max(Salary) from Worker);
```

Q-37. Write an SQL query to show one row twice in results from a table.

Ans.

The required query is:

```
select FIRST_NAME, DEPARTMENT from worker W where W.DEPARTMENT='HR'
union all
select FIRST_NAME, DEPARTMENT from Worker W1 where W1.DEPARTMENT='HR';
```

Q-38. Write an SQL query to fetch intersecting records of two tables.

Ans.

The required query is:

```
(SELECT * FROM Worker)
INTERSECT
(SELECT * FROM WorkerClone);
```

Q-39. Write an SQL query to fetch the first 50% records from a table.

Ans.

The required query is:

```
SELECT *
FROM WORKER
WHERE WORKER_ID <= (SELECT count(WORKER_ID)/2 from Worker);
```

Q-40. Write an SQL query to fetch the departments that have less than five people in it.

Ans.

The required query is:

```
SELECT DEPARTMENT, COUNT(WORKER_ID) as 'Number of Workers' FROM Worker
GROUP BY DEPARTMENT HAVING COUNT(WORKER_ID) < 5;
```

Q-41. Write an SQL query to show all departments along with the number of people in there.

Ans.

The following query returns the expected result:

SELECT DEPARTMENT, COUNT(DEPARTMENT) as 'Number of Workers' FROM Worker GROUP BY DEPARTMENT;

Q-42. Write an SQL query to show the last record from a table.

Ans.

The following query will return the last record from the Worker table:

```
Select * from Worker where WORKER_ID = (SELECT max(WORKER_ID) from Worker);
```

Q-43. Write an SQL query to fetch the first row of a table.

Ans.

The required query is:

```
Select * from Worker where WORKER_ID = (SELECT min(WORKER_ID) from Worker);
```

Q-44. Write an SQL query to fetch the last five records from a table.

Ans.

The required query is:

```
SELECT * FROM Worker WHERE WORKER_ID <=5
UNION
SELECT * FROM (SELECT * FROM Worker W order by W.WORKER_ID DESC) AS W1
WHERE W1.WORKER_ID <=5;
```

Q-45. Write an SQL query to print the name of employees having the highest salary in each department.

Ans.

The required query is:

```
SELECT t.DEPARTMENT,t.FIRST_NAME,t.Salary from(SELECT max(Salary) as
TotalSalary,DEPARTMENT from Worker group by DEPARTMENT) as TempNew
Inner Join Worker t on TempNew.DEPARTMENT=t.DEPARTMENT
 and TempNew.TotalSalary=t.Salary;
```

Q-46. Write an SQL query to fetch three max salaries from a table.

Ans.

The required query is:

SELECT distinct Salary from worker a WHERE 3 >= (SELECT count(distinct Salary) from worker b WHERE a.Salary <= b.Salary) order by a.Salary desc;

Q-47. Write an SQL query to fetch three min salaries from a table.

Ans.

The required query is:

SELECT distinct Salary from worker a WHERE 3 >= (SELECT count(distinct Salary) from worker b WHERE a.Salary >= b.Salary) order by a.Salary desc;

Q-48. Write an SQL query to fetch nth max salaries from a table.

Ans.

The required query is:

SELECT distinct Salary from worker a WHERE n >= (SELECT count(distinct Salary) from worker b WHERE a.Salary <= b.Salary) order by a.Salary desc;

Q-49. Write an SQL query to fetch departments along with the total salaries paid for each of them.

Ans.

The required query is:

SELECT DEPARTMENT, sum(Salary) from worker group by DEPARTMENT;

Q-50. Write an SQL query to fetch the names of workers who earn the highest salary.

Ans.

The required query is:

SELECT FIRST_NAME, SALARY from Worker WHERE SALARY=(SELECT max(SALARY) from Worker);