INDEX PAGE

EXP NO. 2 :
Creation, modification, configuration, and deletion of databases UI anfSQL Commands

A **database** is a collection of related data. **Data** is the known facts that can be recorded and that have implicit meaning. A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.

A **database management system (DBMS)** is a collection of programs that enables users to create and maintain a database. The DBMS is a *general-purpose software system* that facilitates the processes of *defining, constructing, manipulating,* and *sharing* databases among various users and applications.

Defining a database involves specifying the data types, structures, and constraints of the data to be stored in the database. The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called **meta-data**.

Constructing the database is the process of storing the data on some storage medium that is controlled by the DBMS.

Manipulating a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the mini world, and generating reports from the data.

Sharing a database allows multiple users and programs to access the database simultaneously.

An **application program** accesses the database by sending queries or requests for data to the DBMS.

A **query** typically causes some data to be retrieved.

A **transaction** may cause some data to be read and some data to be written into the database.

A **data definition language (DDL)** is used to define the database conceptual schema.

The relational model represents the database as a collection of relations. Informally, each relation resembles **a table** of values or, to some extent, a flat file of records. When a relation is thought of as a **table** of values, each row in the table represents a collection of related data values. A row represents a fact that typically corresponds to a real-world entity or relationship. The table name and column names are used to help to interpret the meaning of the values in each row.

a) Create Database

CREATE DATABASE is the SQL command for creating a database.

```
1. CREATE DATABASE movies;
2. CREATE DATABASEIF NOT EXISTS movies;
```

List of existing databases can view by running the SQL command.

```
1. SHOW DATABASES;
```

To access the created database, use the following command.

```
1. USE database name;
```

b) Creating Tables MySQL

Tables can be created using CREATE TABLE statement.

```
1. CREATE  TABLE [IF NOT EXISTS] `TableName` (`fieldname` dataType
   [optional parameters]);
```

Example:-

```
CREATE TABLE IF NOT EXISTS Members(membership_number INT
AUTOINCREMENT,full_names VARCHAR(150) NOT NULL ,gender VARCHAR(6),
date_of_birth DATE ,physical_address VARCHAR(255) ,postal_address
VARCHAR(255) ,contact_number VARCHAR(75) ,email VARCHAR(255) , PRIMARY KEY
(`membership_number`) );
```

c) Deleting objects from Database

DROP command is used to delete objects from the database.

```
DROP TABLE „Table Name‟;
```

d) Modifying objects of a Database

ALTER TABLE is used to add, delete/drop or modify columns in the existing table. It is also used to add and drop various constraints on the existing table.

```
1. ALTER TABLE table_name ADD (Columnname_1  datatype);
2. ALTER TABLE table_name DROP COLUMN column_name;
3. ALTER TABLE table_name MODIFY column_name column_type;
4. ALTER TABLE table_name CHANGE column_old column_new type;
```

e) Finding the Description of Tables

DESC statement can be used to view the description about a Table.

DESC Table_Name;

Problem:

A. Create a DATABASE named as "UNIVERSITY" with the following TABLES.

1. STUDENT(Name, Student_Number, Semester, Branch)
2. COURSE(Name,Course_Number, Department, Credit)
3. SECTION(Section_Id,Course_Number,Semester,Year,Instructor)
4. GRADE_REPORT(Student_Number, Section_Id,Grade)

5. PREREQUISITE(Course_Number,Prerquisite_number)

Lab 1:

1. Identify different data types for defining the table attributes.
2. Create Database and Tables.
3. Change the Student_Number attribute as Roll_No of STUDENT table.
4. Change the type of Course_Number from INT to CHAR.
5. Add REF_BOOK attribute to COURSE table.
6. Delete all the tables.

B. Write a SQL statement to create a table job_history including columns employee_id, start_date, end_date, job_id and department_id and make sure that, the employee_id column does not contain any duplicate value at the time of insertion and the foreign key column job_id contain only those values which are exists in the jobs table.

Here is the structure of the table jobs;

```
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| JOB_ID     | varchar(10)  | NO   | PRI |         |       |
| JOB_TITLE  | varchar(35)  | NO   |     | NULL    |       |
| MIN_SALARY | decimal(6,0) | YES  |     | NULL    |       |
| MAX_SALARY | decimal(6,0) | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
```

EXP NO:3

Creation of database schema - DDL (create tables, set constraints, enforce relationships, create indices, delete and modify tables). Export ER diagram from the database and verify relationships

AIM: To export ER diagram from database

Entity-relationship diagrams are a useful tool for designing databases. They show the entities involved, their attributes, and the relationships between them. A method often used to create an entity-relationship diagram is crow's foot notation.

The symbols used in crow's foot notation

Entities

An entity is a real-world thing (person, place, object – whatever you can name in your database can be an entity) represented in crow's foot notation by a rectangle with a name in a box on top.

Attributes

Entities have attributes that describe them. An entity can have one or lots of attributes. One of those attributes will be its key attribute, or identifier, which uniquely identifies that entity. The identifier is shown with an asterisk next to its name.

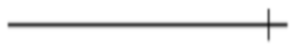
Relationships

The entities depicted in the diagram have relationships that describe how they interact. Relationships between entities are represented by a line with a verb written on the line.

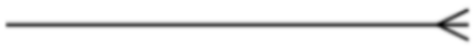
Cardinality and modality

Relationship lines in crow's foot notation have two indicators to describe the cardinality and the modality of the relationship. Cardinality tells you the maximum number of times that an instance of an entity can be associated with instances of the other entity. Modality tells you the minimum number of times that the instance can be associated with instances of the other entity.

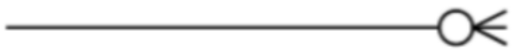
One



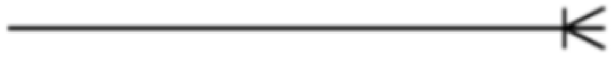
Many



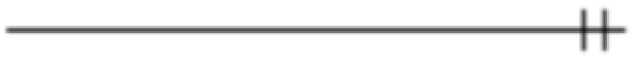
Zero or many



One or many

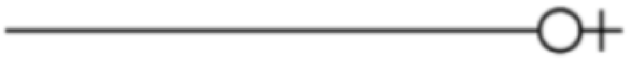


One and only one



No more than one relationship is possible between instances.

Zero or one



EXP NO:4

SQL commands for DML

AIM:To write queries to retrieve,insert ,update and delete data in a database.

a) The Insert Operation

(Relation means Table, Tuple means a Record)**

The **Insert** operation provides a list of attribute values for a new tuple *t* that is to be inserted into a relation *R*.

Insert can violate any of the four types of constraints.

Domain constraints can be violated if an attribute value is given that does not appear in the corresponding domain or is not of the appropriate data type.
Key constraints can be violated if a key value in the new tuple *t* already exists in another tuple in the relation *r*(*R*).
Entity integrity can be violated if any part of the primary key of the new tuple *t* is NULL.
Referential integrity can be violated if the value of any foreign key in *t* refers to a tuple that does not exist in the referenced relation.

INSERT statement is used to insert data into a table.

```
INSERT INTO <table name> VALUES (<value 1>, ... <value n>);
```

Example

```
INSERT INTO Student VALUES (‘Ram’,’CS01’,’S5’,’CSE’);
```

b) The Delete Operation

The **Delete** operation can violate only referential integrity. This occurs if the tuple being deleted is referenced by foreign keys from other tuples in the database. To specify deletion, a condition on the attributes of the relation selects the tuple (or tuples) to be deleted.

DELETE statement deletes row(s) from a table.

```
DELETE FROM <table name> WHERE <condition>;
```

Example

```
        DELETE FROM STUDENT WHERE Student_Number =”CS01”;
```

If the WHERE clause is omitted, then every row of the table is deleted.

c) *The Update Operation*

The **Update** (or **Modify**) operation is used to change the values of one or more attributes in a tuple (or tuples) of some relation *R*. It is necessary to specify a condition on the attributes of the relation to select the tuple (or tuples) to be modified.**SET** clause in the UPDATE command specifies the attributes to be modified and their new values.

UPDATE statement is used to change values that are already in a table.

```
 UPDATE <table name> SET <attribute> = <expression> WHERE <condition>;
```

Example: **UPDATE** STUDENT **SET** Name = „Amar‟ **WHERE** StudID=‟CS01‟;

Once the database schemas are compiled and the database is populated with data, users must have some means to manipulate the database. Typical manipulations include retrieval, insertion, deletion, and modification of the data. The DBMS provides a set of operations or a language called the **data manipulation language** (**DML**) for these purposes.

Retrieve data from Database

SELECT command is used to retrieve data from the a

database SELECT **SELECT** <attribute list> **FROM**

<table list>

Example

SELECT Name from Student;

Retrieve data from Database on a condition

```
        SELECT <attribute>, ….., <attribute n> FROM <table name> WHERE
<condition>;
```

Example

SELECT Name FROM Student WHERE Branch=”CSE”;

Problem:1

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Create the table as shown above

1. *Insert 5 rows of appropriate data into the tables using INSERT statement.*
2. *List the data of entire tables using SELECT command.*
3. *Find the student with Student Number CS01.*
4. *List all the courses under the Civil Department.*
5. *Find the prerequisite of course number BE301*
6. *Find the instructor who handles the course EE303*
7. *Find the grade of Student with Student_Number EE04.*
8. *Update the grade A to O of student with number CE09.*
9. *Update the Credit_hours of course CS301 to 4.*
10. *Delete the student with number EC09.*

2.Create a database orgz.
Worker(worker_id,first_name,last_name,salary,joining_date,deapertment)
Bonus(worker_ref_id, bonus_date, bonus_date)
Title(worker_ref_id,worker_title,affected_from)

1.Insert 5 rows of appropriate data into the tables using INSERT statement.
2.Write an SQL query to fetch “FIRST_NAME” from Worker table in upper case
3.Write an SQL query to fetch unique values of DEPARTMENT from Worker table.
4. Write an SQL query that fetches the unique values of DEPARTMENT from Worker table and prints its length.
5. Write an SQL query to print details of workers excluding first names, “Vipul” and “Satish” from Worker table.
6. Write an SQL query to print details of Workers with DEPARTMENT name as “Admin”
7. Write an SQL query to print details of the Workers whose FIRST_NAME ends with ‘h’ and contains six alphabets.
8. Write an SQL query to fetch the count of employees working in the department ‘Admin’.

OUTPUT

```
Microsoft Windows [Version 10.0.18363.449]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\DELL>cd C:\Program Files\MySQL\MySQL Server 8.0\bin

C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u root -p
Enter password: **********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 19
Server version: 8.0.28 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> drop database company;
Query OK, 1 row affected (0.03 sec)

mysql> create database company;
Query OK, 1 row affected (0.01 sec)

mysql> use company;
Database changed
mysql> create table worker(w
    -> ^Z
Bye

C:\Program Files\MySQL\MySQL Server 8.0\bin>
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u root -p
Enter password: **********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 20
Server version: 8.0.28 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create table worker(w_id int primary key not null,f_name varchar(100),l_name varchar(100),salary
int,doj datetime,dept varchar(100));
ERROR 1046 (3D000): No database selected
  mysql> use company;
Database changed
mysql> create table worker(w_id int primary key not null,f_name varchar(100),l_name varchar(100),salary
int,doj datetime,dept varchar(100));
Query OK, 0 rows affected (0.04 sec)

mysql> desc worker;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| w_id   | int          | NO   | PRI | NULL    |       |
| f_name | varchar(100) | YES  |     | NULL    |       |
| l_name | varchar(100) | YES  |     | NULL    |       |
| salary | int          | YES  |     | NULL    |       |
| doj    | datetime     | YES  |     | NULL    |       |
| dept   | varchar(100) | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
6 rows in set (0.01 sec)

mysql> create table bonus(wref_id int primary key not null,b_date datetime,b_amt int);
Query OK, 0 rows affected (0.03 sec)

mysql> create table title(wref_id int,w_title varchar(100),affected_from datetime);
Query OK, 0 rows affected (0.03 sec)

mysql> desc bonus;
+---------+----------+------+-----+---------+-------+
| Field   | Type     | Null | Key | Default | Extra |
+---------+----------+------+-----+---------+-------+
| wref_id | int      | NO   | PRI | NULL    |       |
| b_date  | datetime | YES  |     | NULL    |       |
| b_amt   | int      | YES  |     | NULL    |       |
+---------+----------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> drop table title;
Query OK, 0 rows affected (0.03 sec)

mysql> create table title(wref_id int,w_title varchar(100),affected_from datetime,foreign
key(wref_id)references bonus(wref_id)on delete cascade);
Query OK, 0 rows affected (0.06 sec)

mysql> desc title;
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
  | wref_id       | int          | YES  | MUL | NULL    |       |
| w_title       | varchar(100) | YES  |     | NULL    |       |
| affected_from | datetime     | YES  |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
3 rows in set (0.01 sec)

mysql> insert into
worker(w_id,f_name,l_name,salary,doj,dept)values(123,"VIBUL","S",30000,20101230121110,"ADMIN")
;
Query OK, 1 row affected (0.02 sec)

mysql> insert into worker values(345,"SURESH","W",25000,20111021110707,"ACCOUNT");
Query OK, 1 row affected (0.01 sec)

mysql> insert into worker values(567,"ANJANA","S",35000,20121021110707,"CS");
Query OK, 1 row affected (0.01 sec)

mysql> insert into worker values(567,"ANJANA","S",35000,20131111121212,"CS");
ERROR 1062 (23000): Duplicate entry '567' for key 'worker.PRIMARY'
mysql> insert into worker values(789,"ANJU","S",35000,20131111121212,"CS");
Query OK, 1 row affected (0.02 sec)
```

```
mysql> insert into worker values(789,"ANJU","S",35000,20141215010101,"CIVIL");
ERROR 1062 (23000): Duplicate entry '789' for key 'worker.PRIMARY'
mysql> insert into worker values(012,"RESHMI","P",35000,20141215010101,"CIVIL");
Query OK, 1 row affected (0.01 sec)

mysql> select * from student;
ERROR 1146 (42S02): Table 'company.student' doesn't exist
my sql> select * from worker;
+------+--------+--------+--------+---------------------+---------+
| w_id | f_name | l_name | salary | doj                 | dept    |
+------+--------+--------+--------+---------------------+---------+
|   12 | RESHMI | P      |  35000 | 2014-12-15 01:01:01 | CIVIL   |
|  123 | VIBUL  | S      |  30000 | 2010-12-30 12:11:10 | ADMIN   |
|  345 | SURESH | W      |  25000 | 2011-10-21 11:07:07 | ACCOUNT |
|  567 | ANJANA | S      |  35000 | 2012-10-21 11:07:07 | CS      |
|  789 | ANJU   | S      |  35000 | 2013-11-11 12:12:12 | CS      |
+------+--------+--------+--------+---------------------+---------+
5 rows in set (0.00 sec)

mysql> select * from worker;
+------+--------+--------+--------+---------------------+---------+
| w_id | f_name | l_name | salary | doj                 | dept    |
+------+--------+--------+--------+---------------------+---------+
|   12 | RESHMI | P      |  35000 | 2014-12-15 01:01:01 | CIVIL   |
|  123 | VIBUL  | S      |  30000 | 2010-12-30 12:11:10 | ADMIN   |
|  345 | SURESH | W      |  25000 | 2011-10-21 11:07:07 | ACCOUNT |
|  567 | ANJANA | S      |  35000 | 2012-10-21 11:07:07 | CS      |
|  789 | ANJU   | S      |  35000 | 2013-11-11 12:12:12 | CS      |
+------+--------+--------+--------+---------------------+---------+
5 rows in set (0.00 sec)

mysql> desc bonus;
+---------+----------+------+-----+---------+-------+
| Field   | Type     | Null | Key | Default | Extra |
+---------+----------+------+-----+---------+-------+
| wref_id | int      | NO   | PRI | NULL    |       |
| b_date  | datetime | YES  |     | NULL    |       |
| b_amt   | int      | YES  |     | NULL    |       |
+---------+----------+------+-----+---------+-------+
3 rows in set (0.01 sec)

mysql> insert into worker(wref_id,b_date,b_amt)values(1,20220125)121212,5000);
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your
MySQL server version for the right syntax to use near '121212,5000)' at line 1
mysql> insert into worker(wref_id,b_date,b_amt)values(1,20220125121212,5000);
ERROR 1054 (42S22): Unknown column 'wref_id' in 'field list'
mysql> insert into worker(wref_id,b_date,b_amt)values(134,20220125121212,5000);
ERROR 1054 (42S22): Unknown column 'wref_id' in 'field list'
mysql> insert into bonus(wref_id,b_date,b_amt)values(1,20220125)121212,5000);
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your
MySQL server version for the right syntax to use near '121212,5000)' at line 1
mysql> insert into bonus(wref_id,b_date,b_amt)values(1,20220125121212,5000);
Query OK, 1 row affected (0.01 sec)

mysql> insert into bonus values(2,20220126121212,4000);
Query OK, 1 row affected (0.01 sec)

mysql> insert into bonus values(3,20220127121212,3000);
```

```
Query OK, 1 row affected (0.01 sec)

mysql> insert into bonus values(4,20220123121212,5000);
Query OK, 1 row affected (0.02 sec)

mysql> insert into bonus values(5,20220122121212,6000);
Query OK, 1 row affected (0.01 sec)

mysql> select * from bonus;
+---------+---------------------+-------+
| wref_id | b_date              | b_amt |
+---------+---------------------+-------+
|       1 | 2022-01-25 12:12:12 |  5000 |
|       2 | 2022-01-26 12:12:12 |  4000 |
|       3 | 2022-01-27 12:12:12 |  3000 |
|       4 | 2022-01-23 12:12:12 |  5000 |
|       5 | 2022-01-22 12:12:12 |  6000 |
+---------+---------------------+-------+
5 rows in set (0.00 sec)

mysql> desc title;
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| wref_id       | int          | YES  | MUL | NULL    |       |
| w_title       | varchar(100) | YES  |     | NULL    |       |
| affected_from | datetime     | YES  |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> insert into worker(wref_id,w_date,affected_from)values(1,"HR",20201231101010);
ERROR 1054 (42S22): Unknown column 'wref_id' in 'field list'
mysql> insert into title(wref_id,w_date,affected_from)values(1,"HR",20201231101010);
ERROR 1054 (42S22): Unknown column 'w_date' in 'field list'
mysql> insert into title(wref_id,w_title,affected_from)values(1,"HR",20201231101010);
Query OK, 1 row affected (0.02 sec)

mysql> insert into title values(2,"MANAGER",20201231101010);
Query OK, 1 row affected (0.02 sec)

mysql> insert into title values(3,"CLERK",20201231101010);
Query OK, 1 row affected (0.01 sec)

mysql> insert into title values(4,"ASST.MANAGER",20201231101010);
Query OK, 1 row affected (0.02 sec)

mysql> insert into title values(5,"DESIGNMANAGER",20201231101010);
Query OK, 1 row affected (0.01 sec)

mysql> select * from title;
+---------+---------------+---------------------+
| wref_id | w_title       | affected_from       |
+---------+---------------+---------------------+
|       1 | HR            | 2020-12-31 10:10:10 |
|       2 | MANAGER       | 2020-12-31 10:10:10 |
|       3 | CLERK         | 2020-12-31 10:10:10 |
|       4 | ASST.MANAGER  | 2020-12-31 10:10:10 |
|       5 | DESIGNMANAGER | 2020-12-31 10:10:10 |
+---------+---------------+---------------------+
5 rows in set (0.00 sec)
 n
mysql> select * from table where f_name(UPPER);
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your
MySQL server version for the right syntax to use near 'table where f_name(UPPER)' at line 1
mysql> select * from employee UPPER(f_name);
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your
MySQL server version for the right syntax to use near '(f_name)' at line 1
mysql> select UPPER(f_name) from employee;
ERROR 1146 (42S02): Table 'company.employee' doesn't exist
mysql> select UPPER(f_name) from worker;
+---------------+
| UPPER(f_name) |
+---------------+
| RESHMI        |
| VIBUL         |
| SURESH        |
| ANJANA        |
| ANJU          |
+---------------+
5 rows in set (0.00 sec)

mysql> select distinct dept from worker;
+---------+
| dept    |
+---------+
| CIVIL   |
| ADMIN   |
| ACCOUNT |
| CS      |
+---------+
4 rows in set (0.00 sec)

mysql> select dept from worker group by dept;
+---------+
| dept    |
+---------+
| CIVIL   |
| ADMIN   |
| ACCOUNT |
| CS      |
+---------+
4 rows in set (0.00 sec)

mysql> select char_length(dept) from worker group by dept;
+-------------------+
| char_length(dept) |
+-------------------+
|                 5 |
|                 5 |
|                 7 |
|                 2 |
+-------------------+
4 rows in set (0.02 sec)

mysql> select * from worker where f_name not in("VIBUL","SATHEESH");
+------+--------+--------+--------+---------------------+---------+
| w_id | f_name | l_name | salary | doj                 | dept    |
+------+--------+--------+--------+---------------------+---------+
|   12 | RESHMI | P      |  35000 | 2014-12-15 01:01:01 | CIVIL   |
|  345 | SURESH | W      |  25000 | 2011-10-21 11:07:07 | ACCOUNT |
|  567 | ANJANA | S      |  35000 | 2012-10-21 11:07:07 | CS      |
|  789 | ANJU   | S      |  35000 | 2013-11-11 12:12:12 | CS      |
 +------+--------+--------+--------+---------------------+---------+
4 rows in set (0.00 sec)

mysql> select * from worker where dept+"ADMIN";
Empty set, 10 warnings (0.00 sec)

mysql> select * from worker where dept="ADMIN";
+------+--------+--------+--------+---------------------+-------+
| w_id | f_name | l_name | salary | doj                 | dept  |
+------+--------+--------+--------+---------------------+-------+
|  123 | VIBUL  | S      |  30000 | 2010-12-30 12:11:10 | ADMIN |
+------+--------+--------+--------+---------------------+-------+
1 row in set (0.02 sec)

mysql> select * from worker where f_name like "-----h";
Empty set (0.00 sec)

mysql> select * from worker where f_name like "_____h";
+------+--------+--------+--------+---------------------+---------+
| w_id | f_name | l_name | salary | doj                 | dept    |
+------+--------+--------+--------+---------------------+---------+
|  345 | SURESH | W      |  25000 | 2011-10-21 11:07:07 | ACCOUNT |
+------+--------+--------+--------+---------------------+---------+
1 row in set (0.00 sec)

mysql> select COUNT(*) from worker where dept="ADMIN"
    -> select COUNT(*) from worker where dept="ADMIN";
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your
MySQL server version for the right syntax to use near 'select COUNT(*) from worker where
dept="ADMIN"' at line 2
mysql> select COUNT(*) from worker where dept="ADMIN";\\
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.02 sec)

ERROR:
Unknown command '\\'.
    -> select COUNT(*) from worker where dept="ADMIN";
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your
MySQL server version for the right syntax to use near '\\
select COUNT(*) from worker where dept="ADMIN"' at line 1
mysql> select COUNT(*) from worker where dept="ADMIN";
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.00 sec)

mysql>
```

EXP NO: 5

Implementation of built-in functions in RDBMS

AIM: To implement built-in functions.

Built in functions are simply functions come already implemented in the RDBMS. These functions allow performing different types of manipulations on the data. The built in functions can be basically categorized into the following most used categories.

- **Strings functions** - operate on string data types

1. UCASE - Converts all the letters to upper case.
2. LCASE - Converts all the letters to lower case.
3. REVERSE - Reverse a string.
4. CHAR_LENGTH - Return the length of the string.
5. ASCII - Return the ASCII value of the first character

- *Numeric functions - operate on numeric data types*

1. ABS - Returns the absolute (positive) value of a number.
2. RAND - Return a random decimal number

- *Date functions - operate on date data types*

1. CURDATE - Return the current date.
2. CURTIME - Return current time.
3. DAY,MONTH,YEAR - Return the day, month and year of a date.
4. DAYNAME - Return the weekday name for a date.
5. DATEDIFF - Return the number of days between two date values.

Problem:

1. Student(Name, Sid, Address)
2. Record(Sid, DOB,Doadmission,Dept)

Questions:

1. Insert relevant data into tables (Student name in CAPS).
2. Enter current date to Doadmission field.
3. List all the students born on “Saturday”.
4. List the name and age of all students.
5. List all the students born on 1st of any month.
6. List all the student names in lower case.
7. List all students and his/her Department name (CAPS).
8. List the student names in reverse.
9. List the student names and its length.
10. Practice ASCII, ABS and RAND function.

OUTPUT

```
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.7.37-log MySQL Community Server (GPL)

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> desc student;
ERROR 1046 (3D000): No database selected
mysql> use stu;
Database changed
mysql> desc student;
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| name    | varchar(100) | YES  |     | NULL    |       |
| sid     | int(11)      | NO   | PRI | NULL    |       |
| address | varchar(100) | YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+
3 rows in set (0.45 sec)

mysql> desc recorcd;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| sid    | int(11)      | YES  | MUL | NULL    |       |
| dob    | date         | YES  |     | NULL    |       |
| doadmi | date         | YES  |     | NULL    |       |
| dept   | varchar(100) | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.02 sec)

mysql> select * from student;
+-------+-----+---------+
| name  | sid | address |
+-------+-----+---------+
| SAI   | 120 | UVW     |
| ARUN  | 121 | XYZ     |
| RIYA  | 122 | RST     |
| RAM   | 123 | ABC     |
| HENNA | 124 | DEF     |
+-------+-----+---------+
5 rows in set (0.05 sec)

mysql> select * from recorcd;
+------+------------+------------+------+
| sid  | dob        | doadmi     | dept |
+------+------------+------------+------+
|  120 | 1999-01-08 | 2022-02-20 | Cse  |
|  121 | 1999-02-01 | 2022-02-20 | ee   |
|  122 | 1998-08-06 | 2022-02-20 | ce   |
|  123 | 1999-11-02 | 2022-02-20 | mech |
|  123 | 1999-11-02 | 2022-02-20 | mech |
|  124 | 1999-12-07 | 2022-02-20 | che  |
+------+------------+------------+------+
6 rows in set (0.01 sec)

mysql> select * from student where sid
    -> select * from student where sid=(select sid from record where dayname
(dob)='sunday');
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
that corresponds to your MySQL server vers
ion for the right syntax to use near 'select * from student where sid=(select sid from
record where dayname (dob)='su
n' at line 2
mysql> select * from student where sid=(select sid from record where dayname
(dob)='sunday');
ERROR 1146 (42S02): Table 'stu.record' doesn't exist
mysql> select * from student where sid=(select sid from recorcd where dayname
(dob)='sunday');
Empty set (0.04 sec)

mysql> select * from student where sid=(select sid from recorcd where dayname
(dob)='monday');
+------+-----+---------+
| name | sid | address |
+------+-----+---------+
| ARUN | 121 | XYZ     |
+------+-----+---------+
1 row in set (0.02 sec)

mysql> select lower(name) from student;
```

```
+-------------+
| lower(name) |
+-------------+
| sai         |
| arun        |
| riya        |
| ram         |
| henna       |
+-------------+
5 rows in set (0.02 sec)

mysql> select lower(name),lower(dept) from student,record where
student.sid=recorcd.sid;
ERROR 1146 (42S02): Table 'stu.record' doesn't exist
mysql> select lower(name),lower(dept) from student,recorcd where
student.sid=recorcd.sid;
+-------------+-------------+
| lower(name) | lower(dept) |
+-------------+-------------+
| sai         | cse         |
| arun        | ee          |
| riya        | ce          |
| ram         | mech        |
| ram         | mech        |
| henna       | che         |
+-------------+-------------+
6 rows in set (0.03 sec)

mysql> select reverse(name) as rname from student;
+-------+
| rname |
+-------+
| IAS   |
| NURA  |
| AYIR  |
| MAR   |
| ANNEH |
+-------+
5 rows in set (0.02 sec)

mysql> select name,length(name) as lname from student;
+-------+-------+
| name  | lname |
+-------+-------+
| SAI   |     3 |
| ARUN  |     4 |
| RIYA  |     4 |
| RAM   |     3 |
| HENNA |     5 |
+-------+-------+
5 rows in set (0.03 sec)

mysql>
```

EXP NO:6

Implementation of various aggregate functions in SQL

AIM:To implement aggregate function in SQL

Aggregate functions are used to summarize information from multiple tuples into a single-tuple summary.

Grouping is used to create subgroups of tuples before summarization. Grouping and aggregation are required in many database applications.

A number of built-in aggregate functions exist:

COUNT, **SUM**, **MAX**, **MIN**, and **AVG**.

The COUNT function returns the number of tuples or values as specified in a query.

The functions SUM, MAX, MIN, and AVG can be applied to a set or multiset of numeric values and return, respectively, the sum, maximum value, minimum value, and average (mean) of those values.
These functions can be used in the SELECT clause or in a HAVING clause.

The functions MAX and MIN can also be used with attributes that have nonnumeric domains if the domain values have a *total ordering* among one another.

Example;

- SELECT SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary) FROM EMPLOYEE;
- SELECT COUNT (*) FROM EMPLOYEE;
- SELECT COUNT (*) FROM EMPLOYEE, DEPARTMENT WHERE DNO=DNUMBER AND DNAME='Research';
- SELECT COUNT (DISTINCT Salary) FROM EMPLOYEE; (duplicate values will be eliminated)

Problem:

1. Products(<u>Product_Id</u>, name,brand, category,year, price)

Questions:

1. Find the average price of all products in the products table.
2. List the number of products whose price is greater than 500.
3. Find the highest price of all products.
4. Find the lowest price of all products.

OUTPUT

```
mysql> create database shop;

Query OK, 1 row affected (0.05 sec)
```

```
mysql> use shop;

Database changed

mysql> create table products(p_id int primary key not null,name varchar(100),brand varchar(100),category
varchar(100),year int,price int);

Query OK, 0 rows affected (0.09 sec)



mysql> insert into
products(p_id,name,brand,category,year,price)values(123,"MOBILE","HUAWEI","ELECTRONIC
DEVICES",2019,20000);

Query OK, 1 row affected (0.03 sec)



mysql> insert into products
values(345,"REFRIGERATOR","WHIRLPOOL","HOMEAPPLIANCES",2020,10000);

Query OK, 1 row affected (0.01 sec)



mysql> insert into products values(567,"NOTEBOOK","APSARA","STATIONERY",2008,100);

Query OK, 1 row affected (0.02 sec)



mysql> insert into products values(789,"PEN","CLASSMATE","STATIONERY",2009,50);

Query OK, 1 row affected (0.01 sec)



mysql> insert into products values(901,"LAPTOP","DELL","ELECTRONIC DEVICES",2022,30000);

Query OK, 1 row affected (0.02 sec)



mysql> select * from  products;

+------+--------------+-----------+--------------------+------+-------+

| p_id | name         | brand     | category           | year | price |

+------+--------------+-----------+--------------------+------+-------+

|  123 | MOBILE       | HUAWEI    | ELECTRONIC DEVICES | 2019 | 20000 |

|  345 | REFRIGERATOR | WHIRLPOOL | HOMEAPPLIANCES     | 2020 | 10000 |

|  567 | NOTEBOOK     | APSARA    | STATIONERY         | 2008 |   100 |
|  789 | PEN             | CLASSMATE | STATIONERY         | 2009 |    50 |
|  901 | LAPTOP          | DELL      | ELECTRONIC DEVICES | 2022 | 30000 |
+------+-----------------+-----------+--------------------+------+-------+
5 rows in set (0.02 sec)

mysql> select COUNT(*),AVG(price) from products;
+----------+------------+
| COUNT(*) | AVG(price) |
+----------+------------+
|        5 | 12030.0000 |
+----------+------------+
1 row in set (0.01 sec)

mysql>  select AVG(price) from products;
+------------+
| AVG(price) |
+------------+
| 12030.0000 |
+------------+
1 row in set (0.00 sec)

mysql> select * from products where price>=500;
+------+-----------------+-----------+--------------------+------+-------+
| p_id | name            | brand     | category           | year | price |
+------+-----------------+-----------+--------------------+------+-------+
|  123 | MOBILE          | HUAWEI    | ELECTRONIC DEVICES | 2019 | 20000 |
|  345 |REFRIGERATOR     | WHIRLPOOL | HOMEAPPLIANCES     | 2020 | 10000 |
|  901 | LAPTOP          | DELL      | ELECTRONIC DEVICES | 2022 | 30000 |
+------+-----------------+-----------+--------------------+------+-------+
3 rows in set (0.00 sec)
```

```
mysql> select max(price) from  products;
+------------+
| max(price) |
+------------+
|      30000 |
+------------+
1 row in set (0.00 sec)

mysql> select  *,max(price) from  products;
+------+--------+---------+--------------------+------+-------+------------+
| p_id | name   | brand   | category           | year | price | max(price) |
+------+--------+---------+--------------------+------+-------+------------+
|  123 | MOBILE | HUAWEI  | ELECTRONIC DEVICES | 2019 | 20000 |      30000 |
+------+--------+---------+--------------------+------+-------+------------+
1 row in set (0.00 sec)

mysql> select COUNT(*),max(price) from  products;
+----------+------------+
| COUNT(*) | max(price) |
+----------+------------+
|        5 |      30000 |
+----------+------------+
1 row in set (0.00 sec)

mysql> select min(price) from products;
+------------+
| min(price) |
+------------+
|         50 |
```

EXP NO. 7

Implementation of Order By, Group By & Having clause.

SQL allows the user to order the tuples in the result of a query by the values of one or more of the attributes that appear in the query result, by using the **ORDER BY** clause.

Example:
SELECT Fname, Pname **FROM** DEPARTMENT **WHERE** Dnumber=101 **ORDER BY** Fname [ASC|DESC]

To apply the aggregate functions *to subgroups of tuples in a relation*, where the subgroups are based on some attribute values. For example, to find the average salary of employees *in each department* or the number of employees who work *on each project*, **partition** the relation into nonoverlapping subsets (or **groups**) of tuples.

Each group (partition) will consist of the tuples that have the same value of some attribute(s), called the **grouping attribute(s)**. Apply the function to each such group independently to produce summary information about each group.

SQL has a **GROUP BY** clause for this purpose. The GROUP BY clause specifies the grouping attributes, which should *also appear in the SELECT clause*, so that the value resulting from applying each aggregate function to a group of tuples appears along with the value of the grouping attribute(s).

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;

To retrieve the values of these functions for *groups that satisfy certain conditions* **HAVING** clause can be used.

HAVING clause, appear in conjunction with a GROUP BY clause. HAVING provides a condition on the summary information regarding the group of tuples associated with each value of the grouping attributes. Only the groups that satisfy the condition are retrieved in the result of the query.

Syntax:

```
SELECT column-names  FROM table-name  WHERE condition  GROUP BY column-
names HAVING condition
```

Example:

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

SELECT Pnumber, Pname, **COUNT** (*) **FROM** PROJECT, WORKS_ON **WHERE**

Pnumber=Pno **GROUP BY** Pnumber, Pname **HAVING COUNT** (*) > 2;

Problem:

EMPLOYEE(Emp_id,Name,DoJ,Designation,salary,Dept**)**
ORDER(Ordr_Id**,** Prod_Id, Price, Quantity, Discount**)**
PRODUCTS(Prod_Id, Prod_Name, Supplier_Id, Category_Id, Unit_Price**)**

Lab 7:

1. Find the sum of Salary paid to each department from EMPLOYEE Table.
2. List the salary of employee in the alphabetic order of Name from EMPLOYEE Table.
3. List the Employee name based on the salary (Highest to Lowest) from EMPLOYEE Table..
4. Find the number employees in each deprtment from EMPLOYEE Table.
5. Find the number employees in each department & sum of Salary paid to each department from EMPLOYEE Table.
6. Find the number employees in each department & sum of Salary paid to each department by Designation from EMPLOYEE Table.
7. Find the sum of Salary paid to CS Department from EMPLOYEE Table.
8. Find the number of employees from each department where at least two employee present in department from EMPLOYEE Table.
9. Find the sale orders whose total sale is greater than Rs. 12000 from ORDER Table.
10. Find all the orders that have at least 5 items from ORDER Table.
11. Find the most expensive product in each category from PRODUCTS Table.
12. Find the most expensive product that has the price greater than Rs 10000 from PRODUCTS Table.
13. Find the least expensive product in each category from PRODUCTS Table.

OUTPUT

```
Microsoft Windows [Version 10.0.19044.1706]
(c) Microsoft Corporation. All rights reserved.

C:\Users\pc>cd C:\Program Files\MySQL\MySQL Server 8.0\bin

C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u root -p
Enter password: ************
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.28 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
```

```
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database employment;
Query OK, 1 row affected (0.08 sec)

mysql> use employment;
Database changed
mysql> create table employee(emp_id int(11) primary key,name varchar(11),doj date,designation
varchar(11),salary int(11),department varchar(11));
Query OK, 0 rows affected, 2 warnings (0.09 sec)

mysql> show table;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to
your MySQL server version for the right syntax to use near '' at line 1
mysql> desc employee;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| emp_id      | int         | NO   | PRI | NULL    |       |
| name        | varchar(11) | YES  |     | NULL    |       |
| doj         | date        | YES  |     | NULL    |       |
| designation | varchar(11) | YES  |     | NULL    |       |
| salary      | int         | YES  |     | NULL    |       |
| department  | varchar(11) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
6 rows in set (0.04 sec)

mysql> create table products(prod_id varchar(11) primar key,pname varchar(11),supplier_id
int(11),category_id varchar(11),unit_price int(11));
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to
your MySQL server version for the right syntax to use near 'primar key,pname
varchar(11),supplier_id int(11),category_id varchar(11),unit_pr' at line 1
mysql> create table products(prod_id varchar(11) primar key,pname varchar(11),supplier_id
int(11),category_id varchar(11),uprice int(11));
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to
your MySQL server version for the right syntax to use near 'primar key,pname
varchar(11),supplier_id int(11),category_id varchar(11),uprice ' at line 1
mysql> \c
mysql> \c
mysql> create table products(prod_id varchar(11) primary key,pname varchar(11),supplier_id
int(11),category_id varchar(11),uprice int(11));
Query OK, 0 rows affected, 2 warnings (0.03 sec)

mysql> desc products;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| prod_id     | varchar(11) | NO   | PRI | NULL    |       |
| pname       | varchar(11) | YES  |     | NULL    |       |
| supplier_id | int         | YES  |     | NULL    |       |
| category_id | varchar(11) | YES  |     | NULL    |       |
| uprice      | int         | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

```
mysql> create table orders(id int(11) primary key,prod_id varchar(11),cost int(11),quantity
varchar(11),discount varchar(11),foreign key(prod_id) references products(prod_id));
Query OK, 0 rows affected, 2 warnings (0.07 sec)

mysql> desc orders;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| id       | int         | NO   | PRI | NULL    |       |
| prod_id  | varchar(11) | YES  | MUL | NULL    |       |
| cost     | int         | YES  |     | NULL    |       |
| quantity | varchar(11) | YES  |     | NULL    |       |
| discount | varchar(11) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)

mysql> insert into employee values(100,"anu",20200202,"d1",2000,"CS");
Query OK, 1 row affected (0.01 sec)

mysql> insert into employee values(120,"ajith",20230402,"d2",20000,"CS");
Query OK, 1 row affected (0.01 sec)

mysql> insert into employee values(158,"laya",20190406,"d4",19000,"ME");
Query OK, 1 row affected (0.01 sec)

mysql> insert into employee values(160,"Rahul",20190904,"d3",15000,"EC");
Query OK, 1 row affected (0.01 sec)

mysql> insert into employee values(203,"liam",20201012,"d5",20000,"EC");
Query OK, 1 row affected (0.01 sec)

mysql> select * from employee;
+--------+-------+------------+-------------+--------+------------+
| emp_id | name  | doj        | designation | salary | department |
+--------+-------+------------+-------------+--------+------------+
|    100 | anu   | 2020-02-02 | d1          |   2000 | CS         |
|    120 | ajith | 2023-04-02 | d2          |  20000 | CS         |
|    158 | laya  | 2019-04-06 | d4          |  19000 | ME         |
|    160 | Rahul | 2019-09-04 | d3          |  15000 | EC         |
|    203 | liam  | 2020-10-12 | d5          |  20000 | EC         |
+--------+-------+------------+-------------+--------+------------+
5 rows in set (0.00 sec)

mysql> insert into products
values("P","n1","s1","c1",100),("P2","n2","s2","c2",199),("P3","n3","s3","c3",250),("P4","n4",
"s4","c4",210),("P5","n5","s5","c5",180);
ERROR 1366 (HY000): Incorrect integer value: 's1' for column 'supplier_id' at row 1
mysql> insert into products
values("P","n1","1","c1",100),("P2","n2","2","c2",199),("P3","n3","3","c3",250),("P4","n4","4
","c4",210),("P5","n5","5","c5",180);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from products;
+---------+-------+-------------+-------------+--------+
| prod_id | pname | supplier_id | category_id | uprice |
+---------+-------+-------------+-------------+--------+
| P       | n1    |           1 | c1          |    100 |
| P2      | n2    |           2 | c2          |    199 |
| P3      | n3    |           3 | c3          |    250 |
| P4      | n4    |           4 | c4          |    210 |
| P5      | n5    |           5 | c5          |    180 |
+---------+-------+-------------+-------------+--------+
5 rows in set (0.00 sec)

mysql> insert into orders values("1","p",1000,10,"10%");
Query OK, 1 row affected (0.01 sec)

mysql> insert into orders values("2","p2",995,5,"10%");
Query OK, 1 row affected (0.01 sec)

mysql> insert into orders values("3","p3",2500,10,"50%");
Query OK, 1 row affected (0.01 sec)

mysql> insert into orders values("4","p4",2520,12,"10%");
Query OK, 1 row affected (0.01 sec)

mysql> insert into orders values("5","p5",2700,15,"10%");
Query OK, 1 row affected (0.01 sec)

mysql> select * from orders;
+----+---------+------+----------+----------+
| id | prod_id | cost | quantity | discount |
+----+---------+------+----------+----------+
|  1 | p       | 1000 | 10       | 10%      |
|  2 | p2      |  995 | 5        | 10%      |
|  3 | p3      | 2500 | 10       | 50%      |
|  4 | p4      | 2520 | 12       | 10%      |
|  5 | p5      | 2700 | 15       | 10%      |
+----+---------+------+----------+----------+
5 rows in set (0.00 sec)

mysql> select salary from employee order by name ASC;
+--------+
| salary |
+--------+
|  20000 |
|   2000 |
|  19000 |
|  20000 |
|  15000 |
+--------+
5 rows in set (0.00 sec)

mysql> select name from employee order by salary desc;
+-------+
| name  |
+-------+
| ajith |
| liam  |
| laya  |
| Rahul |
| anu   |
+-------+
5 rows in set (0.00 sec)

mysql> select count(*) dept from employee group by dept;
ERROR 1056 (42000): Can't group on 'dept'
mysql> select count(*) from employee group by department;
+----------+
| count(*) |
+----------+
|        2 |
|        1 |
|        2 |
+----------+
3 rows in set (0.00 sec)

mysql> select count(*)department from employee group by department;
+------------+
| department |
+------------+
|          2 |
|          1 |
|          2 |
+------------+
3 rows in set, 1 warning (0.00 sec)

mysql> select count(*)department,sum(salary) from employee group by department;
+------------+-------------+
| department | sum(salary) |
+------------+-------------+
|          2 |       22000 |
|          1 |       19000 |
|          2 |       35000 |
+------------+-------------+
3 rows in set, 1 warning (0.00 sec)

mysql> select count(*)department,sum(salary) from employee group by designation;
+------------+-------------+
| department | sum(salary) |
+------------+-------------+
|          1 |        2000 |
|          1 |       20000 |
|          1 |       19000 |
|          1 |       15000 |
|          1 |       20000 |
+------------+-------------+
5 rows in set (0.00 sec)

mysql> select sum(salary) from employee where department="CS";
+-------------+
| sum(salary) |
+-------------+
|       22000 |
+-------------+
1 row in set (0.00 sec)
```

```
mysql> select department,count(*) from employee group by department having count(*)>=2;
+------------+----------+
| department | count(*) |
+------------+----------+
| CS         |        2 |
| EC         |        2 |
+------------+----------+
2 rows in set (0.00 sec)

mysql> select * from orders having cost>2000;
+----+---------+------+----------+----------+
| id | prod_id | cost | quantity | discount |
+----+---------+------+----------+----------+
|  3 | p3      | 2500 | 10       | 50%      |
|  4 | p4      | 2520 | 12       | 10%      |
|  5 | p5      | 2700 | 15       | 10%      |
+----+---------+------+----------+----------+
3 rows in set (0.00 sec)

mysql> select * from orders having quantity>5;
+----+---------+------+----------+----------+
| id | prod_id | cost | quantity | discount |
+----+---------+------+----------+----------+
|  1 | p       | 1000 | 10       | 10%      |
|  3 | p3      | 2500 | 10       | 50%      |
|  4 | p4      | 2520 | 12       | 10%      |
|  5 | p5      | 2700 | 15       | 10%      |
+----+---------+------+----------+----------+
4 rows in set (0.00 sec)

mysql> select prod_id,max(uprice) from products group by category_id;
+---------+-------------+
| prod_id | max(uprice) |
+---------+-------------+
| P       |         100 |
| P2      |         199 |
| P3      |         250 |
| P4      |         210 |
| P5      |         180 |
+---------+-------------+
5 rows in set (0.00 sec)

mysql> select max(uprice) from products,orders where orders.cost>2000;
+-------------+
| max(uprice) |
+-------------+
|         250 |
+-------------+
1 row in set (0.00 sec)

mysql> select min(uprice) from products group by category_id;
+-------------+
| min(uprice) |
+-------------+
|         100 |
|        199 |
|        250 |
|        210 |
|        180 |
+-------------+
5 rows in set (0.00 sec)

mysql>
```

EXP NO: 8

Implementation of set operators, nested queries and Join queries

Aim : To implement set operators, nested queries and Join queries in sql.

Set operators

SQL has directly incorporated some of the set operations from mathematical *set theory*, which are also part of relational algebra. There are set union (UNION), set difference (**EXCEPT**), and set intersection (**INTERSECT**) operations.

The relations resulting from these set operations are sets of tuples; that is, *duplicate tuples are eliminated from the result.* These set operations apply only to *union-compatible relations*, so the two relations on which applying the operation have the same attributes and that the attributes appear in the same order in both relations.

1. UNION: - Used to combine the results of two or more SELECT statements. It will eliminate duplicate rows from its resultset.

```
SELECT * FROM First
UNION
SELECT * FROM Second;
```

2. UNION ALL: - This operation is similar to Union. But it also shows the duplicate rows.

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

3. INTERSECT:- Used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements. In case of **Intersect** the number of columns and datatype must be same.

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

4. MINUS:- The Minus operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

Nested queries

Some queries require that existing values in the database be fetched and then used in a comparison condition. Such queries can be conveniently formulated by using **nested queries**, which are complete select-from-where blocks within the WHERE clause of another query. That other query is called the **outer query**

Join queries

The concept of a **joined table** (or **joined relation**) was incorporated into SQL to permit users to specify a table resulting from a join operation *in the* FROM *clause* of a query. This construct may be easier to comprehend than mixing together all the select and join conditions in the WHERE clause.

The concept of a joined table also allows the user to specify different types of join, such as NATURAL JOIN and various types of OUTER JOIN.

In a NATURAL JOIN on two relations *R* and *S*, no join condition is specified; an implicit *EQUIJOIN condition* for *each pair of attributes with the same name* from *R* and *S* is created. Each such pair of attributes is included *only once* in the resulting relation.

The default type of join in a joined table is called an **inner join**, where a tuple is included in the result only if a matching tuple exists in the other relation.

```
SELECT table1.column1, table1.column2, table2.column1, FROM table1

INNER JOIN table2

ON table1.matching_column = table2.matching_column;
```

In SQL, the options available for specifying joined tables include INNER JOIN (only pairs of tuples that match the join condition are retrieved, same as JOIN), LEFT OUTER JOIN (every tuple in the left table must appear in the result; if it does not have a matching tuple, it is padded with NULL values for the attributes of the right table), RIGHT OUTER JOIN (every tuple in the right table must appear in the result; if it does not have a matching tuple, it is padded with NULL values for the attributes of the left table), and FULL OUTER JOIN.

```
SELECT table1.column1, table1.column2, table2.column1, FROM table1

LEFT JOIN table2

ON table1.matching_column = table2.matching_column;
```

SELECT table1.column1, table1.column2, table2.column1, FROM table1

RIGHT JOIN table2

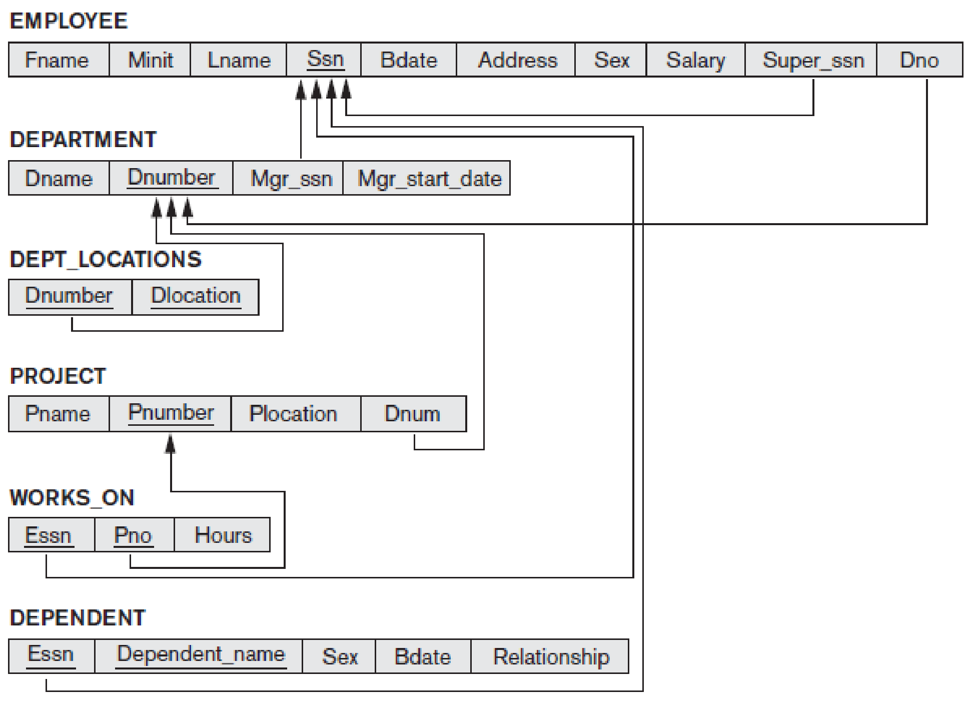
ON table1.matching_column = table2.matching_column;

In the latter three options, the keyword OUTER may be omitted. If the join attributes have the same name, one can also specify the natural join variation of outer joins by using the keyword NATURAL before the operation (for example, NATURAL LEFT OUTER JOIN). The keyword CROSS JOIN is used to specify the CARTESIAN PRODUCT operation

Questions

1. **Borrower (customer-name, loan-number)**
   Depositor (customer-name, account-number)
   Customer (customer-name, street-number, customer-city)

2.



Problem:

1. List all details of the customers who have either an account or a loan or both.
2. Find the names of all customers who have an account but not a loan.

3. List all the customers who have both a loan and an account.
4. Make a list of all project numbers for projects that involve an employee whose last name is „Smith‟, either as a worker or as a manager of the department that controls the project.
5. Select the Essns of all employees who work the same (project, hours) combination on some project that employee „John Smith‟ (whose Ssn = „123456789‟) works on.
6. Select the names of employees whose salary is greater than the salary of all the employees in department 5.
7. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.
8. Retrieves the name and address of every employee who works for the Research‟ department

EXP NO : 9
Practice of SQL TCL commands like Rollback, Commit, Savepoint

Aim: To practice TCL commands like Rollback, Commit, Savepoint

The following commands are used to control transactions.

- **COMMIT** − to save the changes.
- **ROLLBACK** − to roll back the changes.
- **SAVEPOINT** − creates points within the groups of transactions in which to ROLLBACK.

Transactional Control Commands

Transactional control commands are only used with the **DML Commands** such as - INSERT, UPDATE and DELETE only. They cannot be used while creating tables or dropping them because these operations are automatically committed in the database.

The COMMIT Command

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

The syntax for the COMMIT command is as follows.

```
COMMIT;
```

The ROLLBACK Command

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for a ROLLBACK command is as follows −

```
ROLLBACK;
```

The SAVEPOINT Command

A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

The syntax for a SAVEPOINT command is as shown below.

```
SAVEPOINT SAVEPOINT_NAME;
```

This command serves only in the creation of a SAVEPOINT among all the transactional statements. The ROLLBACK command is used to undo a group of transactions.

The syntax for rolling back to a SAVEPOINT is as shown below.

```
ROLLBACK TO SAVEPOINT_NAME;
```

EXP 10
Practice of SQL DCL commands for granting and revoking user privileges

Aim: To practice of SQL DCL commands for granting and revoking user privileges

You can GRANT and REVOKE privileges on various database objects in MySQL. You can then view the privileges assigned to a user using the SHOW GRANTS command.
You can grant users various privileges to tables. These permissions can be any combination of SELECT, INSERT, UPDATE, DELETE, INDEX, CREATE, ALTER, DROP, GRANT OPTION or ALL.
Syntax
The syntax for granting privileges on a table in MySQL is:
GRANT privileges ON object TO user;

privileges

It can be any of the following values:

Privilege	Description
SELECT	Ability to perform SELECT statements on the table.
INSERT	Ability to perform INSERT statements on the table.
UPDATE	Ability to perform UPDATE statements on the table.
DELETE	Ability to perform DELETE statements on the table.
INDEX	Ability to create an index on an existing table.
CREATE	Ability to perform CREATE TABLE statements.
ALTER	Ability to perform ALTER TABLE statements to change the table definition.
DROP	Ability to perform DROP TABLE statements.
GRANT OPTION	Allows you to grant the privileges that you possess to other users.
ALL	Grants all permissions except GRANT OPTION.

object

The name of the database object that you are granting permissions for. In the case of granting privileges on a table, this would be the table name.

user

The name of the user that will be granted these privileges.

if you wanted to grant SELECT, INSERT, UPDATE, and DELETE privileges on a table called contacts to a user name abc, you would run the following GRANT statement:
GRANT SELECT, INSERT, UPDATE, DELETE ON contacts TO 'abc'@'localhost';
You can also use the ALL keyword to indicate that you wish to grant all permissions except

GRANT OPTION to a user named abc. For example:
GRANT ALL ON contacts TO 'abc'@'localhost';
If you wanted to grant only SELECT access on the contacts table to all users, you could grant the privileges to *. For example:
GRANT SELECT ON contacts TO '*'@'localhost';

Revoke Privileges on Table

Once you have granted privileges, you may need to revoke some or all of these privileges. To do this, you can run a revoke command. You can revoke any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, or ALL.

Syntax

The syntax for revoking privileges on a table in MySQL is:

```
REVOKE privileges ON object FROM user;
```

privileges
It can be any of the following values:

Privilege	Description
SELECT	Ability to perform SELECT statements on the table.
INSERT	Ability to perform INSERT statements on the table.
UPDATE	Ability to perform UPDATE statements on the table.
DELETE	Ability to perform DELETE statements on the table.
INDEX	Ability to create an index on an existing table.
CREATE	Ability to perform CREATE TABLE statements.
ALTER	Ability to perform ALTER TABLE statements to change the table definition.
DROP	Ability to perform DROP TABLE statements.
GRANT OPTION	Allows you to grant the privileges that you possess to other users.
ALL	Grants all permissions except GRANT OPTION.

object

The name of the database object that you are revoking privileges for. In the case of revoking privileges on a table, this would be the table name.

user

The name of the user that will have these privileges revoked.

Example

if you wanted to revoke DELETE and UPDATE privileges on a table called contacts from a user named abc, you would run the following REVOKE statement:

```
REVOKE DELETE, UPDATE ON contacts FROM 'abc'@'localhost';
```

If you wanted to revoke all permissions (except GRANT OPTION) on a table for a user named abc, you could use the ALL keyword as follows:

```
REVOKE ALL ON contacts FROM 'abc'@'localhost';
```

If you had granted SELECT privileges to * (ie: all users) on the contacts table and you wanted to revoke these privileges, you could run the following REVOKE statement:

```
REVOKE SELECT ON contacts FROM '*'@'localhost';
```

EXP NO:11
Practice of SQL commands for creation of views and assertions

Aim: To practice of SQL commands for creation of views and assertions

Views (Virtual Tables) in SQL

A **view** in SQL terminology is a single table that is derived from other tables. These other tables can be *base tables* or previously defined views. A view does not necessarily exist in physical form; it is considered to be a **virtual table**, in contrast to **base tables**, whose tuples are always physically stored in the database. This limits the possible update operations that can be applied to views, but it does not provide any limitations on querying a view.

In SQL, the command to specify a view is **CREATE VIEW**. The view is given a (virtual) table name (or view name), a list of attribute names, and a query to specify the contents of the view.

CREATE VIEW `view_name`

AS

SELECT statement;

The DROP command can be used to delete a view

```
DROP VIEW `View _name`;
```

Assertions

An assertion is a named constraint that may relate to the content of individual rows of a table, to the entire contents of a table, or to a state required to exist among a number of tables.

Assertions are similar to check constraints, but unlike check constraints they are not defined on table or column level but are defined on schema level. (i.e., assertions are database objects of their own right and are not defined within a create table or alter table statement.)

The SQL syntax for create assertion is:

```
CREATE ASSERTION <constraint name> CHECK (<search condition>)
```

Questions

Product (BarCode, PName, Price, QuantityInStock)
Sale (SaleID, DeliveryAddress, CreditCard)
SaleItem (SaleID, BarCode, Quantity)

1.List all products, with barcode and name, and the sale id and quantity for the sales containing that product, if any (products that were never sold should still be listed in your result) Hint: use a left join

2.Write a SQL statement to create a view called AllProductsSales based on the query in the previous exercise.

4. A virtual table or view can be used as if it is a table in the database. Now write a SQL query against the AllProductsSales view as if it was a table: select everything from the AllProductsSales, ordered by BarCode and SaleID.

OUTPUT

```
MariaDB [(none)]> create database exp11;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> use exp11
Database changed
MariaDB [exp11]> create table product(barcode int(11) primary key not
null,pname varchar(100),price varchar(11),quantitystock int(10));
Query OK, 0 rows affected (0.641 sec)

MariaDB [exp11]> desc product;
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| barcode       | int(11)      | NO   | PRI | NULL    |       |
| pname         | varchar(100) | YES  |     | NULL    |       |
| price         | varchar(11)  | YES  |     | NULL    |       |
| quantitystock | int(10)      | YES  |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
4 rows in set (0.005 sec)

MariaDB [exp11]> create table sale(saleid int(11) primary key not
null,deliveryaddress varchar(100),creditcard varchar(20));
Query OK, 0 rows affected (0.599 sec)

MariaDB [exp11]> desc sale;
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| saleid          | int(11)      | NO   | PRI | NULL    |       |
| deliveryaddress | varchar(100) | YES  |     | NULL    |       |
| creditcard      | varchar(20)  | YES  |     | NULL    |       |
+-----------------+--------------+------+-----+---------+-------+
3 rows in set (0.004 sec)

MariaDB [exp11]> create table saleitem(saleid int(11),barcode
int(11),quantity int(10),foreign key(saleid)references sale(saleid),foreign
key(barcode)references product(barcode));
Query OK, 0 rows affected (0.890 sec)

MariaDB [exp11]> desc saleitem;
+----------+---------+------+-----+---------+-------+
| Field    | Type    | Null | Key | Default | Extra |
+----------+---------+------+-----+---------+-------+
| saleid   | int(11) | YES  | MUL | NULL    |       |
| barcode  | int(11) | YES  | MUL | NULL    |       |
| quantity | int(10) | YES  |     | NULL    |       |
+----------+---------+------+-----+---------+-------+
3 rows in set (0.005 sec)

MariaDB [exp11]> insert into product
values(123,"Phone",25000,15),(124,"Football",1500,25),(125,"Bat",950,10),(
126,"Book",25,250),(127,"Bag",250,50);
Query OK, 5 rows affected (0.145 sec)
Records: 5  Duplicates: 0  Warnings: 0

MariaDB [exp11]> select * from product;
+---------+----------+-------+---------------+
| barcode | pname    | price | quantitystock |
+---------+----------+-------+---------------+
|     123 | Phone    | 25000 |            15 |
|     124 | Football | 1500  |            25 |
|     125 | Bat      | 950   |            10 |
|     126 | Book     | 25    |           250 |
|     127 | Bag      | 250   |            50 |
+---------+----------+-------+---------------+
5 rows in set (0.001 sec)

MariaDB [exp11]> insert into sale
values(451,"Alappuzha",789),(452,"Kannur",809),(453,"Aaluva",810),(454,"
Ambhalappuzha",829),(455,"Kayamkulam",839);
Query OK, 5 rows affected (0.097 sec)
Records: 5  Duplicates: 0  Warnings: 0

MariaDB [exp11]> select * from sale;
+--------+-----------------+------------+
| saleid | deliveryaddress | creditcard |
+--------+-----------------+------------+
|    451 | Alappuzha       | 789        |
|    452 | Kannur          | 809        |
|    453 | Aaluva          | 810        |
|    454 | Ambhalappuzha   | 829        |
|    455 | Kayamkulam      | 839        |
+--------+-----------------+------------+
5 rows in set (0.001 sec)

MariaDB [exp11]> insert into saleitem
values(451,123,20),(452,124,50),(453,125,20),(454,126,500),(455,127,100);
Query OK, 5 rows affected (0.108 sec)
Records: 5  Duplicates: 0  Warnings: 0

MariaDB [exp11]> select * from saleitem;
+--------+---------+----------+
| saleid | barcode | quantity |
+--------+---------+----------+
|    451 |     123 |       20 |
|    452 |     124 |       50 |
|    453 |     125 |       20 |
|    454 |     126 |      500 |
|    455 |     127 |      100 |
+--------+---------+----------+
5 rows in set (0.001 sec)
```

⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿Creation And Insertion ☝ ⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿⦿

⦿First Question

```
select product.barcode,pname,saleitem.saleid,saleitem.quantity from product
left join saleitem on product.barcode = saleitem.barcode;
+---------+----------+--------+----------+
| barcode | pname    | saleid | quantity |
+---------+----------+--------+----------+
|     123 | Phone    |    451 |       20 |
|     124 | Football |    452 |       50 |
|     125 | Bat      |    453 |       20 |
|     126 | Book     |    454 |      500 |
|     127 | Bag      |    455 |      100 |
+---------+----------+--------+----------+
5 rows in set (0.013 sec)
```

⦿Second Question

```
MariaDB [exp11]> create view AllProductsSales as select
product.barcode,pname,saleitem.saleid,saleitem.quantity from product left
join saleitem on product.barcode = saleitem.barcode;
Query OK, 0 rows affected (0.156 sec)

MariaDB [exp11]> select * from AllProductsSales;
+---------+----------+--------+----------+
| barcode | pname    | saleid | quantity |
+---------+----------+--------+----------+
|     123 | Phone    |    451 |       20 |
|     124 | Football |    452 |       50 |
|     125 | Bat      |    453 |       20 |
|     126 | Book     |    454 |      500 |
|     127 | Bag      |    455 |      100 |
+---------+----------+--------+----------+
5 rows in set (0.010 sec)
```

● Third Question

```
select * from AllProductsSales order by barcode,saleid;
+---------+----------+--------+----------+
| barcode | pname    | saleid | quantity |
+---------+----------+--------+----------+
|     123 | Phone    |    451 |       20 |
|     124 | Football |    452 |       50 |
|     125 | Bat      |    453 |       20 |
|     126 | Book     |    454 |      500 |
|     127 | Bag      |    455 |      100 |
+---------+----------+--------+----------+
5 rows in set (0.024 sec)
```

EXP 12

Implementation of various control structures using PL/SQL

Aim: To implement various control structures using PL/SQL

PL/SQL is a procedural language designed specifically to embrace SQL statements within its syntax. PL/SQL program units are compiled by the Oracle Database server and stored inside the database. And at run-time, both PL/SQL and SQL run within the same server process, bringing optimal efficiency. PL/SQL automatically inherits the robustness, security, and portability of the Oracle Database.

PL/SQL is a powerful, yet straightforward database programming language. It is easy to both write and read, and comes packed with lots of out-of-the-box optimizations and security features.

PL/SQL stands for *Procedural Language/Structured Query Language*. PL/SQL offers a set of procedural commands (IF statements, loops, assignments), organized within blocks (explained below), that complement and extend the reach of SQL.

It is certainly possible to build applications on top of SQL and Oracle Database *without* using PL/SQL. Utilizing PL/SQL to perform database-specific operations, most notably SQL statement execution, offers several advantages, though, including tight integration with SQL, improved performance through reduced network traffic, and portability (PL/SQL programs can run on any Oracle Database instance). Thus, the front-end code of many applications executes both SQL statements and PL/SQL blocks, to maximize performance while improving the maintainability of those applications.

Building Blocks of PL/SQL Programs PL/SQL is a block-structured language. A PL/SQL block is defined by the keywords DECLARE, BEGIN, EXCEPTION, and END, which break up the block into three sections:

1. Declarative: statements that declare variables, constants, and other code elements, which can then be used within that block
2. Executable: statements that are run when the block is executed
3. Exception handling: a specially structured section you can use to “catch,” or trap, any exceptions that are raised when the executable section runs

Only the executable section is required for a simple PL/SQL program. Others are optional.

- The classic “Hello World!” block contains an executable section that calls the DBMS_OUTPUT.PUT_LINE procedure to display text on the screen:

```
BEGIN

DBMS_OUTPUT.put_line ('Hello World!');

END;
```

Another Example

```
DECLARE

l_message VARCHAR2 (100) := 'Hello World!'; BEGIN

DBMS_OUTPUT.put_line (l_message);
```

```
END;
```

MySQL Stored Procedures

MySQL has no such interface (PL/SQL client console) or structure as seen in Oracle‟s PL/SQL, but some another way it provides procedural language programming features to use with SQL.

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, MySQL 5 introduced stored procedure.

Why Stored Procedures?

- Stored procedures are fast. MySQL server takes some advantage of caching, just as prepared statements do. The main speed gain comes from reduction of network traffic. If you have a repetitive task that requires checking, looping, multiple statements, and no user interaction, do it with a single call to a procedure that's stored on the server.
- Stored procedures are portable. When you write your stored procedure in SQL, you know that it will run on every platform that MySQL runs on, without obliging you to install an additional runtime-environment package, or set permissions for program execution in the operating system, or deploy different packages if you have different computer types. That's the advantage of writing in SQL rather than in an external language like Java or C or PHP.
- Stored procedures are always available as 'source code' in the database itself. And it makes sense to link the data with the processes that operate on the data.

Create Procedure

The following CREATE PROCEDURE statement creates a new stored procedure.

```
DELIMITER $$

CREATE PROCEDURE GetCustomers()
BEGIN
  SELECT customerName, city, state, postalCode, country FROM customers
  ORDER BY customerName;
END$$
DELIMITER ;
```

The stored procedure can be invoked by using the CALL statement.

```
CALL GetCustomers();
```

A stored procedure may contain control flow statements such as IF, CASE, and LOOP that allow you to implement the code in the procedural way.

Syntax:

```
CREATE PROCEDURE procedure_name(parameter_list)
BEGIN
  statement;
END //
```

The DROP PROCEDURE deletes a stored procedure from the database.

```
DROP PROCEDURE [IF EXISTS] stored_procedure_name;
```

MySQL stored procedure parameters

In MySQL, a parameter has one of three modes: IN,OUT, or INOUT.

IN parameters

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

OUT parameters

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

INOUT parameters

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

Example

```
DELIMITER //

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

DELIMITER ;
```

Invoke this procedure as

```
 CALL GetOfficeByCountry('USA');
```

The OUT parameter example

The following stored procedure returns the number of orders by order status.

```
 DELIMITER $$
```

```
CREATE PROCEDURE
   GetOrderCountByStatus ( IN
   orderStatus VARCHAR(25),
   OUT total INT
)
BEGI
N
   SELECT
   COUNT(orderNumber)
   INTO total
   FROM orders
   WHERE status =
orderStatus; END$$

DELIMITER ;
```

Invoke the procedure using **CALL GetOrderCountByStatus('Shipped',@total);** The result from total can be viewed by **SELECT @total;**

Declaring Variables

```
DELIMITER $$

CREATE PROCEDURE GetTotalOrder()
BEGIN
   DECLARE totalOrder INT DEFAULT 0;

   SELECT COUNT(*)
   INTO totalOrder
   FROM orders;

   SELECT totalOrder;
END$$

DELIMITER ;
```

Listing Procedures

```
SHOW PROCEDURE STATUS;
```

MySQL simple `IF-THEN` statement

The `IF-THEN` statement allows you to execute a set of SQL statements based on a specified condition.

The following illustrates the syntax of the `IF-THEN` statement

```
IF condition THEN
   statements;
END IF;
```

Example

```
    DELIMITER $$

CREATE PROCEDURE GetCustomerLevel( IN pCustomerNumber INT,
        OUT pCustomerLevel VARCHAR(20))
    BEGIN
        DECLARE credit DECIMAL(10,2) DEFAULT 0;

        SELECT creditLimit
        INTO credit
        FROM customers
        WHERE customerNumber = pCustomerNumber;

        IF credit > 50000 THEN
            SET pCustomerLevel = 'PLATINUM';
        END IF;
END$$
 DELIMITER ;
```

IF..THEN..ELSE

```
    IF condition THEN
       statements;
    ELSE
       else-statements;
    END IF;
```

IF-THEN-ELSEIF-ELSE

```
     IF condition THEN
        statements;
     ELSEIF elseif-condition THEN
        elseif-statements;
     ...
     ELSE
        else-statements;
     END IF;
```

Simple CASE statement

The following is the basic syntax of the simple `CASE` statement:

```
 CASE case_value
    WHEN when_value1 THEN statements
    WHEN when_value2 THEN statements
    ...
    [ELSE else-statements]
 END CASE;
```

MySQL LOOP statement

The LOOP statement allows you to execute one or more statements repeatedly. Here is the basic syntax of the LOOP statement:

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

Example

```
    DELIMITER $$
CREATE PROCEDURE loopDemo()
BEGIN
        DECLARE x INT;
        DECLARE str  VARCHAR(255);

           SET x = 1;
            SET str =  '';

        loop_label: LOOP
            IF x > 10 THEN
                LEAVE loop_label;
            END IF;

            SET x = x + 1;
            IF (x mod 2) THEN
                ITERATE loop_label;
            ELSE
                SET str = CONCAT(str,x,',');
            END IF;
        END LOOP;
        SELECT str;
END$$

 DELIMITER ;
```

WHILE Loop

```
WHILE counter <= day
            DO CALL InsertCalendar(dt);
            SET counter = counter + 1;
            SET dt = DATE_ADD(dt,INTERVAL 1 day);
END WHILE;
```

Repeat Loop

```
    REPEAT
           SET result= CONCAT(result,counter,',');
            SET counter = counter + 1;
        UNTIL counter >= 10
        END REPEAT;
```

Questions

1. Write a stored procedure to find the sum of 2 numbers.
2. Write a stored procedure to find the largest of three numbers.
3. Write a stored procedure to print the Fibonacci series.

4. Write a stored procedure to print the Odd numbers up to n.
5. Write a stored procedure to save 5 circles radius and its area to a table.

<u>Create the following tables.</u>

Student(Name, Sid, address)
Record(Sid, DOB,Dept)
Marks(M1,M2,M3)

6. Add records to the table using stored procedure.
7. List students‟ details using stored procedure.

EXP No. 13
Creation of Procedures, Triggers and Functions

AIM: To implement procedures,triggers and functions in mysql.

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

Different from a stored procedure, a stored function can be used in SQL statements as an expression. This helps improve the readability and maintainability of the procedural code.

To create a stored function, use the `CREATE FUNCTION` statement.

```
DELIMITER $$

CREATE FUNCTION
   function_name( param1,
   param2,…
)
RETURNS datatype
[NOT]
DETERMINISTIC
BEGIN
--
statements
END $$

DELIMITER ;
```

Example

```
DELIMITER $$

CREATE FUNCTION CustomerLevel(credit DECIMAL(10,2))
RETURNS VARCHAR(20)
BEGIN
   DECLARE customerLevel VARCHAR(20);

   IF credit > 50000 THEN
      SET customerLevel = 'PLATINUM';
   ..
   ..
;
   END IF;
   RETURN (customerLevel);
END$$
DELIMITER ;
```

Calling Stored Function

```
CALL GetCustomerLevel(-131,@customerLevel);
SELECT @customerLevel;
```

Trigger 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.

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.

Advantages of triggers

- Triggers provide another way to check the integrity of data.
- Triggers handle errors from the database layer.
- Triggers give an alternative way to run scheduled tasks. By using triggers, no need to wait for the scheduled events to run because the triggers are invoked automatically *before* or *after* a change is made to the data in a table.
- Triggers can be useful for auditing the data changes in tables.

Disadvantages of triggers

- Triggers can only provide extended validations, not all validations (PK,FK, NULL).
- Triggers can be difficult to troubleshoot because they execute automatically in the database, which may not invisible to the client applications.
- Triggers may increase the overhead of the MySQL Server.

Managing MySQL triggers

Create triggers – Describe steps of how to create a trigger in MySQL.
Drop triggers – Show you how to drop a trigger.
BEFORE INSERT trigger, AFTER INSERT trigger, BEFORE UPDATE trigger, AFTER UPDATE trigger, BEFORE DELETE trigger, AFTER DELETE trigger. Depending on the operations.

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

The `CREATE TRIGGER` statement creates a new trigger.

Syntax:

```
CREATE TRIGGER trigger_name {BEFORE | AFTER} {INSERT | UPDATE| DELETE }
ON table_name FOR EACH
ROW trigger_body;
```

Example :

```
CREATE TRIGGER before_employee_update BEFORE UPDATE ON employees
 FOR EACH ROW INSERT INTO employees_audit SET action = 'update',
      employeeNumber = OLD.employeeNumber, lastname =
      OLD.lastname, changedate = NOW();
```

Problem:

Student (No, Name, Address,

Percentage)

1.Create a stored function to insert values to the table.
2.Read a student's mark in function, display the students name and grade (If %>90 print O Grade, if %>80 print A Grade, if %>70 print B Grade, if %>60 print C Grade, if %>50 print D Grade, else print F)

Problem:

1. **Stock(Item_No, Name, Quantity, Unit_Prices)**
2. **Purchase(Item_No, Name, Quantity, Unit_Price)**
3. **Bill(Item_No, Name, Quantity)**

1. Create Trigger to update Stock table depending on the Purchase/Bill table entries.

EXP NO:14
Creation of cursors

AIM: To implement cursor.

Cursors are to handle a result set inside a stored procedure. A cursor allows you to iterate a set of rows returned by a query and process each row individually.

MySQL cursor is read-only, non-scrollable and asensitive.

- **Read-only**: Cannot update data in the underlying table through the cursor.
- **Non-scrollable**: Can only fetch rows in the order determined by the SELECT statement. Cannot fetch rows in the reversed order.Cannot skip rows or jump to a specific row in the result set.
- **Asensitive**: There are two kinds of cursors: asensitive cursor and insensitive cursor. An asensitive cursor points to the actual data, whereas an insensitive cursor uses a temporary copy of the data. An asensitive cursor performs faster than an insensitive cursor because it does not have to make a temporary copy of data. However, any change that made to the data from other connections will affect the data that is being used by an asensitive cursor, therefore, it is safer if you do not update the data that is being used by an asensitive cursor. MySQL cursor is asensitive.

MySQL cursors can be used in stored procedures, stored functions, and triggers.

Working with MySQL cursor

Declare a cursor by using the **DECLARE** statement:

DECLARE cursor_name CURSOR FOR SELECT_statement ;

The cursor declaration must be after any variable declaration. A cursor must always associate with a SELECT statement.

Open the cursor by using the OPEN statement. The OPEN statement initializes the result set for the cursor, therefore, must call the OPEN statement before fetching rows from the result set.

OPEN cursor_name;

Use the FETCH statement to retrieve the next row pointed by the cursor and move the cursor to the next row in the result set.

FETCH cursor_name INTO variables list;

Check if there is any row available before fetching it.

Finally, deactivate the cursor and release the memory associated with it using the **CLOSE** statement:

CLOSE cursor_name;

It is a good practice to always close a cursor when it is no longer used.

When working with MySQL cursor, declare a NOT FOUND handler to handle the situation when the cursor could not find any row, the cursor attempts to read the next row in the result set at each FETCH statement. When the cursor reaches the end of the result set, it will not be able to get the data, and a condition is raised. The handler is used to handle this condition.

To declare a NOT FOUND handler, use the following syntax:
DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;
The createEmailList stored procedure is as follows:

```
DELIMITER $$
CREATE PROCEDURE createEmailList ( INOUT emailList varchar(4000))
 BEGIN
DECLARE finished INTEGER DEFAULT 0;
DECLARE emailAddress varchar(100) DEFAULT "";

-- declare cursor for employee email DECLARE curEmail
CURSOR FOR
SELECT email FROM employees;
-- declare NOT FOUND handler
DECLARE CONTINUE HANDLER
FOR NOT FOUND SET finished = 1;
OPEN curEmail;

getEmail: LOOP
        FETCH curEmail INTO emailAddress;
         IF finished = 1 THEN
        LEAVE getEmail;
        END IF;
-- build email list
        SET emailList = CONCAT(emailAddress,";",emailList);
        END LOOP getEmail;
        CLOSE curEmail;

END$$
DELIMITER ;
```

Test the createEmailList stored procedure using the following script:

```
SET @emailList = "";
CALLcreateEmailList(@email
List); SELECT @emailList;
```

Question

Create table

Employee(eid,name,email)

Use cursors and create a procedure to print email from employee table

OUTPUT

```
DELIMITER $$
CREATE PROCEDURE createEmailList ( INOUT emailList varchar(4000))
 BEGIN
DECLARE finished INTEGER DEFAULT 0;
DECLARE emailAddress varchar(100) DEFAULT "";

-- declare cursor for employee email DECLARE curEmail
CURSOR FOR
SELECT email FROM employees;
-- declare NOT FOUND handler
DECLARE CONTINUE HANDLER
FOR NOT FOUND SET finished = 1;
OPEN curEmail;

getEmail: LOOP
        FETCH curEmail INTO emailAddress;
         IF finished = 1 THEN
        LEAVE getEmail;
        END IF;
-- build email list
        SET emailList = CONCAT(emailAddress,";",emailList);
        END LOOP getEmail;
        CLOSE curEmail;

END$$
DELIMITER ;
```

Exp No 15
Familiarization of NoSQL Databases and CRUD operations

NoSQL is a broad term that describes various database technologies that don't conform to the traditional relational database model. CRUD stands for create, read, update, and delete and refers to the basic functions you'll need to perform on your data. **CRUD operations** are usually performed using a language like SQL, but NoSQL databases often use different query languages or APIs. This can make it challenging to switch from a traditional relational database to a NoSQL database.

There are many different NoSQL databases, but some of the most popular ones include MongoDB, Cassandra, and Couchbase. And each of these databases have different query languages.

What are CRUD Operations?

Organizations have to keep track of customer data, accounts, payment information, and other records that require persistent storage. The basic operations performed on such data stored in databases are known as **CRUD operations. CRUD** stands for **Create, Read, Update, and Delete.**

A relational database has tables with columns and rows. Each row describes a single data point, such as an employee's name, address, phone number, salary, and so on. Each column represents a category of information, such as the employee's age, gender, marital status, and so forth. These data can be manipulated using the **CRUD (Create, Read, Update, and Delete) operations** in a database. These operations allow users to create records, read existing records, update existing records, and delete existing records.

The operations are as follows:

- Create
- Read
- Update
- Delete

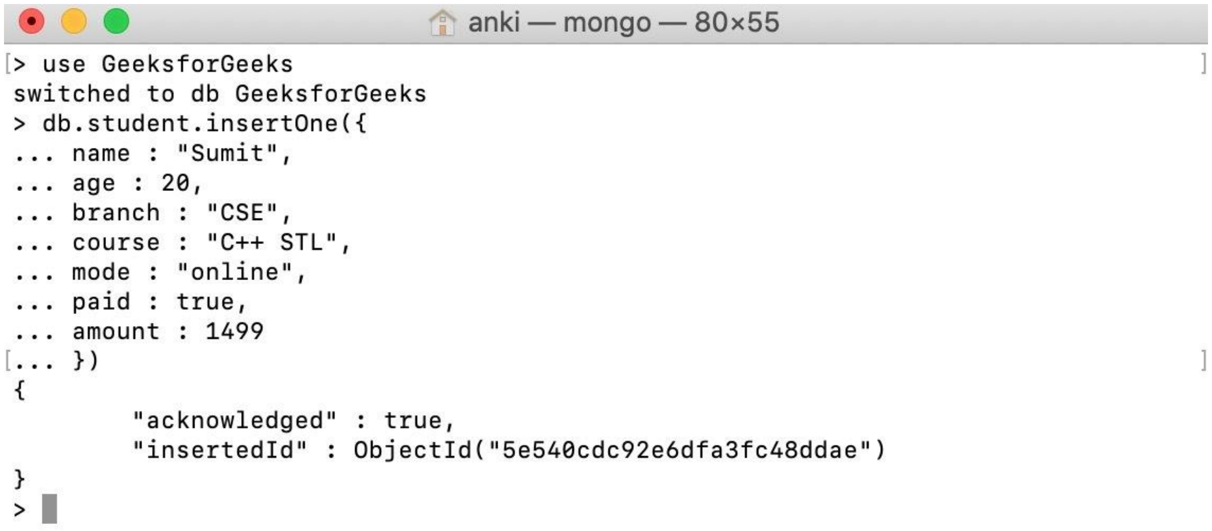
1) Create

The **Create** operations enables users to insert new records into the database. It is same as the functioning of the INSERT function in SQL relational database application. Only an admin can add new attributes to the table. Whereas, a user can create rows and populate them with data corresponding to each attribute.

You can perform, create operations using the following methods provided by the MongoDB:

Method	Description
db.collection.insertOne()	It is used to insert a single document in the collection.
db.collection.insertMany()	It is used to insert multiple documents in the collection.
db.createCollection()	It is used to create an empty collection.

Example 1: In this example, we are inserting details of a single student in the form of document in the student collection using db.collection.insertOne() method.



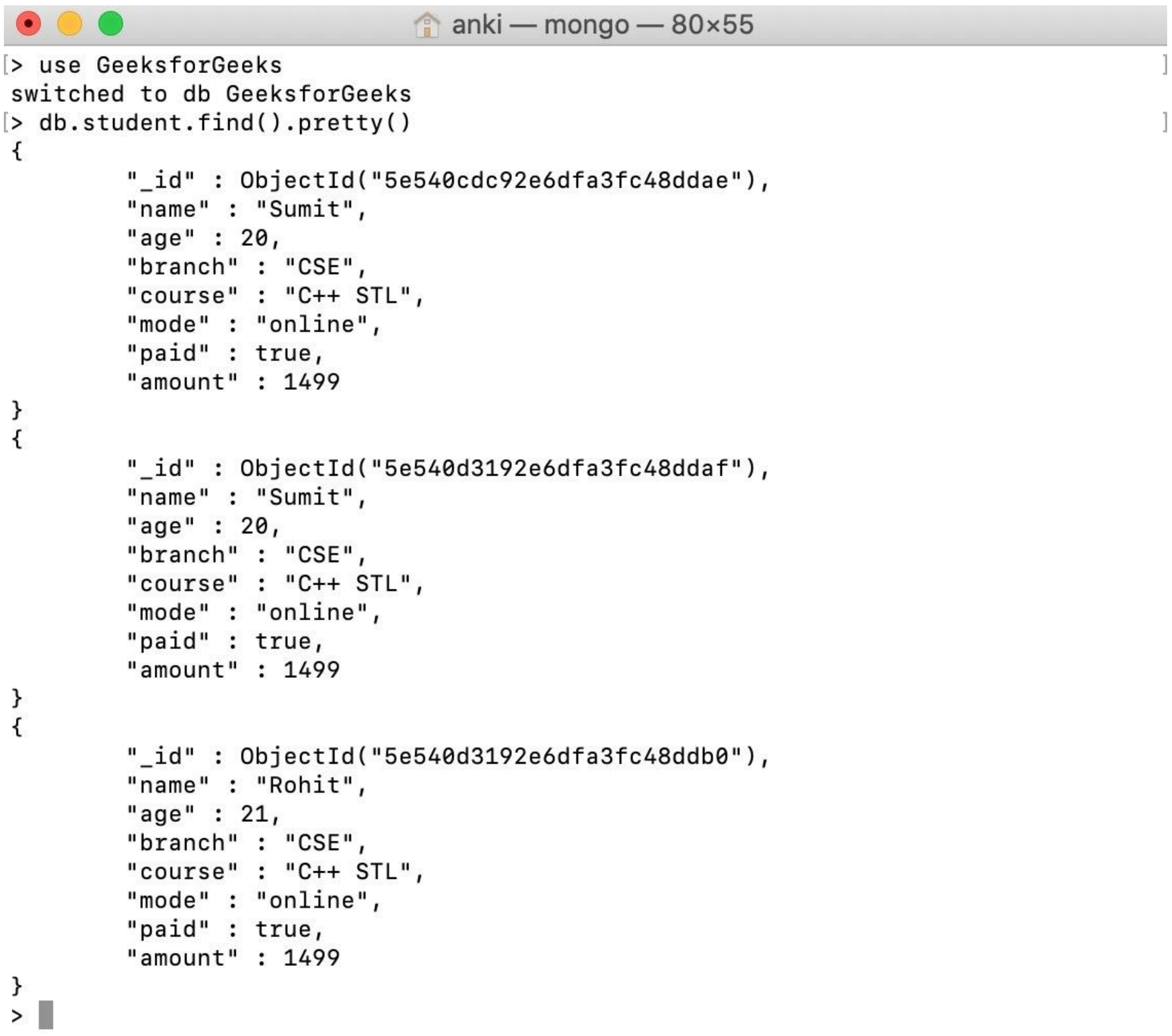
```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.insertOne({
... name : "Sumit",
... age : 20,
... branch : "CSE",
... course : "C++ STL",
... mode : "online",
... paid : true,
... amount : 1499
... })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5e540cdc92e6dfa3fc48ddae")
}
> 
```

2) Read

Similar to the search function, **Read** enables users to look up particular records in the table and access their values. Users can look for keywords or filter the data based on customized criteria to find records.

Method	Description
db.collection.find()	It is used to retrieve documents from the collection.

Example : In this example, we are retrieving the details of students from the student collection using db.collection.find()
method.



```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddb0"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
>
```

Update Operations –

The update operations are used to update or modify the existing document in the collection. You can perform update operations using the following methods provided by the MongoDB:

Method	Description
db.collection.updateOne()	It is used to update a single document in the collection that satisfy the given criteria.
db.collection.updateMany()	It is used to update multiple documents in the collection that satisfy the given criteria.
db.collection.replaceOne()	It is used to replace single document in the collection that satisfy the given criteria.

Delete Operations –

The delete operation are used to delete or remove the documents from a collection. You can perform delete operations using the following methods provided by the

MongoDB:

Method	Description
db.collection.deleteOne()	It is used to delete a single document from the collection that satisfy the given criteria.
db.collection.deleteMany()	It is used to delete multiple documents from the collection that satisfy the given criteria.