INTRODUCTION TO ER MODEL

ER model stands for an Entity-Relationship model it is also known as ERD is a diagram that displays the relationship of entity sets stored in a database. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.

In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attributes, and relationships. ER Diagrams contain different symbols that use rectangles to represent entities, ovals to define attributes and diamond shapes to represent relationships.

1. ER Diagrams Symbols & Notations

ER Diagrams Symbols & Notations mainly contains three basic symbols which are rectangle, oval and diamond to represent relationships between elements, entities and attributes. There are some sub-elements which are based on main elements in ERD Diagram. ER Diagram is a visual representation of data that describes how data is related to each other using different ERD Symbols and Notations. Following are the main components and its symbols in ER Diagrams:

- Rectangles: This Entity Relationship Diagram symbol represents entity types
- Ellipses : Symbol represent attributes
- Diamonds: This symbol represents relationship types
- Lines: It links attributes to entity types and entity types with other relationship types
- Primary key: attributes are underlined.
- Double Ellipses: Represent multi-valued attributes.

2. Component of ER Diagram

Entity

The entity is denoted by a rectangle. This rectangle is divided into two parts: the entity name at the top and the entity attributes in the lower part.

Attributes

Attributes are represented by ovals. A key attribute is the unique, distinguishing characteristic of the entity.

Relationships

After creating entities, to specify the relationships between them. Relationships are denoted in diagrams by lines. Each relationship is one line and has cardinality and mandatory attributes. Relationships can be one-to-one, one-to-many, or many-to-many. The many side is denoted by the crow's foot symbol; one is denoted by a single line.

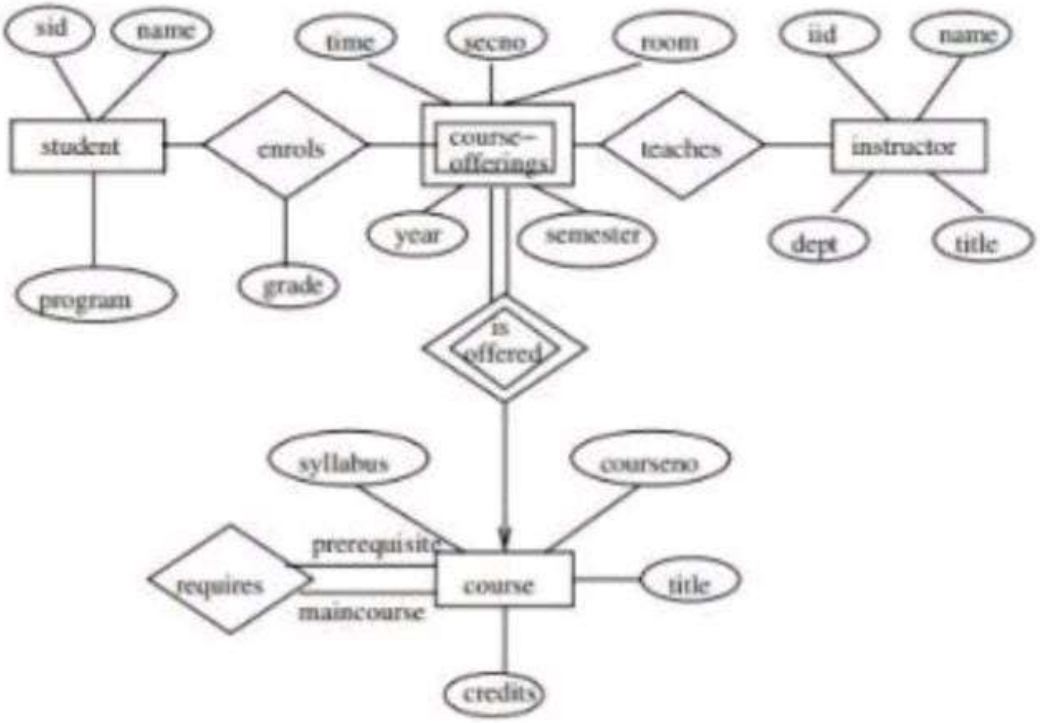

Figure 2.3 E-R diagram for a university.

Date:

Experiment No:1

UNIVERSITY MANAGEMENT SYSTEM ER DIAGRAM

AIM

The university management system database design was made based on managing university requirements. The system can encode students' information. University admin can have access to the students' status and information for important transaction. They can handle the data needed in managing student and instructors/teachers' files as well as the transactions made by their enrollee. To construct an ER diagram for university management system. Document all assumptions that make about the mapping constraints.

OUTPUT

University Management is the main feature of this system wherein ER diagram and it contains the basic details of university. These basic information was composed of the courses and programs offered. Student Management- This feature plays an important role for the system because this gathers the important information of the students. This information were used to track their transactions and other important matters regarding the system. Manage Courses and Subjects The university admin will have to set the selected courses of every students, then assign an appropriate instructors to the subjects covered by each course. Manage Instructors- This system handles the instructors' information as well as their complete schedule of classes and all that were needed to be done in managing instructors. Transaction and Scheduling Management- Its feature will store the transactions made by the students and instructors including their information and their schedules every subjects and timetables.

RESULT:

ER diagram for university management system has been drawn successfully.

INTRODUCTION TO SQL

SQL stands for Structured Query Language. SQL is used to create, remove, alter the database and database objects in a database management system and to store, retrieve, update the data in a database. SQL is a standard language for creating, accessing, manipulating database management system. SQL works for all modern relational database management systems, like SQL Server, Oracle, MySQL, etc.

1. History of SQL

Dr. E. F. Codd published the paper, "A Relational Model of Data for Large Shared Data Banks", in June 1970 in the Association of Computer Machinery (ACM) journal, Communications of the ACM. Codd's model is now accepted as the definitive model for relational database management systems (RDBMS). The language, Structured English Query Language (SEQUEL) was developed by IBM Corporation, Inc., to use Codd's model. SEQUEL later became SQL (still pronounced "sequel"). In 1979, Relational Software, Inc. (now Oracle) introduced the first commercially available implementation of SQL. Today, SQL is accepted as the standard RDBMS language.

2. Rules

SQL follows the following rules:

- Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, you can perform most of the actions in a database.
- SQL depends on tuple relational calculus and relational algebra.

3. SQL process:

- When an SQL command is executing for any RDBMS, then the system figure out the best way to carry out the request and the SQL engine determines that how to interpret the task.
- In the process, various components are included. These components can be optimization Engine, Query engine, Query dispatcher, classic, etc.
- All the non-SQL queries are handled by the classic query engine, but SQL query engine won't handle logical files.

4. SQL Commands

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.

1. Data Definition Language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Some commands that come under DDL are:

- CREATE
- ALTER
- DROP
- TRUNCATE

a. **CREATE:** It is used to create a new table in the database

Syntax:

CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);

Example:

CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DAT E);

b. **DROP**: It is used to delete both the structure and record stored in the table.

Syntax

DROP TABLE table_name;

Example

DROP TABLE EMPLOYEE;

c. **ALTER**: It is used to alter the structure of the database.This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

Syntax:

To add a new column in the table

ALTER TABLE table_name ADD column_name COLUMN-definition;

To modify existing column in the table:

ALTER TABLE table_name MODIFY(column_definitions....);

EXAMPLE

ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));

ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));

d. TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.

Syntax:

TRUNCATE TABLE table_name;

Example:

TRUNCATE TABLE EMPLOYEE;

2. Data Manipulation Language (DML)

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

a. INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax: INSERT INTO TABLE_NAME (col1, col2, col3,.... col N) VALUES (value1, value2, value3, ... valueN);

Or

INSERT INTO TABLE_NAME VALUES (value1, value2, value3, ... valueN);

For example:

INSERT INTO testtable (Author, Subject) VALUES ("Sonoo", "DBMS");

b. UPDATE: This command is used to update or modify the value of a column in the table.

Syntax:

UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]

For example:

UPDATE students SET User_Name = 'Sonoo' WHERE Student_Id = '3'

c. DELETE: It is used to remove one or more row from a table.

Syntax:

DELETE FROM table_name [WHERE condition];

For example:

DELETE FROM testtable WHERE Author="Sonoo";

3. Data Control Language (DCL)

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

- Grant
- Revoke

a. Grant: It is used to give user access privileges to a database.

Example

GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

b. Revoke: It is used to take back permissions from the user.

Example

REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

4. Transaction Control Language (TCL)

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

a. Commit: Commit command is used to save all the transactions to the database.

Syntax:

COMMIT;

Example:

DELETE FROM CUSTOMERS WHERE AGE = 25;

COMMIT;

b. Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

Syntax:

ROLLBACK;

Example:

DELETE FROM CUSTOMERS WHERE AGE = 25;

ROLLBACK;

c. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

SAVEPOINT SAVEPOINT_NAME;

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

SELECT expressions FROM TABLES WHERE conditions;

For example:

SELECT emp_name FROM employee WHERE age > 20;

Introduction to MySQL

MySQL is a very popular open-source relational database management system (RDBMS). MySQL is a relational database management system based on the Structured Query Language, which is the popular language for accessing and managing the records in the database. MySQL is open-source and free software under the GNU license. It is supported by Oracle Company.

History of MySQL

The project of MySQL was started in 1979 when MySQL's inventor Michael Widenius developed an in-house database tool called UNIREG for managing databases. After that, UNIREG has been rewritten in several different languages and extended to handle big databases. After some time, Michael Widenius contacted David Hughes, the author of mSQL, to see if Hughes would be interested in connecting mSQL to UNIREG's B+ ISAM handler to provide indexing to mSQL. That's the way MySQL came into existence.

MySQL is becoming so popular because of these following reasons:

- MySQL is an open-source database, so you don't have to pay a single penny to use it.
- MySQL is a very powerful program that can handle a large set of functionality of the most expensive and powerful database packages.
- MySQL is customizable because it is an open-source database, and the open-source GPL license facilitates programmers to modify the SQL software according to their own specific environment.
- MySQL is quicker than other databases, so it can work well even with the large data set.
- MySQL supports many operating systems with many languages like PHP, PERL, C, C++, JAVA, etc.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL is very friendly with PHP, the most popular language for web development.
- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).

MySQL Data Types

Data Type specifies a particular type of data, like integer, floating points, Boolean, etc. It also identifies the possible values for that type, the operations that can be performed on that type, and the way the values of that type are stored. In MySQL, each database table has many columns and contains specific data types for each column.

We can determine the data type in MySQL with the following characteristics:

- The type of values (fixed or variable) it represents.
- The storage space it takes is based on whether the values are a fixed-length or variable length.

- Its values can be indexed or not.
- How MySQL performs a comparison of values of a particular data type.

MySQL supports a lot number of SQL standard data types in various categories. It uses many different data types that can be broken into the following categories: numeric, date and time, string types, spatial types, and JSON data type .

1. Numeric Data Type

MySQL has all essential SQL numeric data types. These data types can include the exact numeric data types (For example, integer, decimal, numeric, etc.), as well as the approximate numeric data types (For example, float, real, and double precision). It also supports BIT datatype to store bit values. In MySQL, numeric data types are categories into two types, either signed or unsigned except for bit data type.

Data Type Syntax	Description
TINYINT	It is a very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. We can specify a width of up to 4 digits. It takes 1 byte for storage.
SMALLINT	It is a small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range is from 0 to 65535. We can specify a width of up to 5 digits. It requires 2 bytes for storage.
MEDIUMINT	It is a medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. We can specify a width of up to 9 digits. It requires 3 bytes for storage.
INT	It is a normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. We can specify a width of up to 11 digits. It requires 4 bytes for storage.
BIGINT	It is a large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615. We can specify a width of up to 20 digits. It requires 8 bytes for storage.

FLOAT(m,d)	It is a floating-point number that cannot be unsigned.You can define the display length (m) and number of decimals(d) .This is not required and will default to 10,2 where 2 is the numberof decimals,and 10 is the total number of digits(including decimals).Decimal precision can goto 24 places for a float type.It requires 2 bytes for storage.
DOUBLE(m,d)	It is a double-precision floating-point number that cannot be unsigned. You can define the display length (m) and the number of decimals (d). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a double. Real is a synonym for double. It requires 8 bytes for storage.
DECIMAL(m,d)	An unpacked floating-point number that cannot be unsigned. In unpacked decimals, each decimal corresponds to one byte. Defining the display length (m) and the number of decimals (d) is required. Numeric is a synonym for decimal.
BIT(m)	It is used for storing bit values into the table column. Here, M determines the number of bit per value that has a range of 1 to 64.
BOOL	It is used only for the true and false condition. It considered numeric value 1 as true and 0 as false.
BOOLEAN	It is Similar to the BOOL.

2. Date and Time Data Type:

This data type is used to represent temporal values such as date, time, datetime, timestamp, and year. Each temporal type contains values, including zero. When we insert the invalid value, MySQL cannot represent it, and then zero value is used.

Data Type Syntax	Maximum Size	Explanation
YEAR[(2|4)]	Year value as 2 digits or 4 digits.	The default is 4 digits. It takes 1 byte for storage.
DATE	Values range from '1000-01-01' to '9999-12-31'.	Displayed as 'yyyy-mm-dd'. It takes 3 bytes for storage.
TIME	Values range from '-838:59:59' to '838:59:59'.	Displayed as 'HH:MM:SS'. It takes 3 bytes plus fractional seconds for storage.
DATETIME	Values range from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.	Displayed as 'yyyy-mm-dd hh:mm:ss'. It takes 5 bytes plus fractional seconds for storage.
TIMESTAMP(m)	Values range from '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' TC.	Displayed as 'YYYY-MM-DD HH:MM:SS'. It takes 4 bytes plus fractional seconds for storage.

3. String Data Types:

The string data type is used to hold plain text and binary data, for example, files, images, etc. MySQL can perform searching and comparison of string value based on the pattern matching such as LIKE operator, Regular Expressions, etc.

Data Type Syntax	Maximum Size	Explanation
CHAR(size)	It can have a maximum size of 255 characters.	Here size is the number of characters to store. Fixed-length strings. Space padded on the right to equal size characters.
VARCHAR(size)	It can have a maximum size of 255 characters.	Here size is the number of characters to store. Variable-length string.
TINYTEXT(size)	It can have a maximum size of 255 characters.	Here size is the number of characters to store.
TEXT(size)	Maximum size of 65,535 characters.	Here size is the number of characters to store.
MEDIUMTEXT(size)	It can have a maximum size of 16,777,215 characters.	Here size is the number of characters to store.
LONGTEXT(size)	It can have a maximum size of 4GB or 4,294,967,295 characters.	Here size is the number of characters to store.

BINARY(size)	It can have a maximum size of 255 characters.	Here size is the number of binary characters to store. Fixed-length strings. Space padded on the right to equal size characters. (Introduced in MySQL 4.1.2)
VARBINARY(size)	It can have a maximum size of 255 characters.	Here size is the number of characters to store. Variable length string. (Introduced in MySQL 4.1.2)
ENUM	It takes 1 or 2 bytes that depend on the number of enumeration values. An ENUM can have a maximum of 65,535 values.	It is short for enumeration, which means that each column may have one of the specified possible values. It uses numeric indexes (1, 2, 3…) to represent string values.
SET	It takes 1, 2, 3, 4, or 8 bytes that depends on the number of set members. It can store a maximum of 64 members.	It can hold zero or more, or any number of string values. They must be chosen from a predefined list of values specified during table creation.

OUTPUT

```
mysql> CREATE DATABASE sanju;
Query OK, 1 row affected (0.00 sec)

mysql> USE sanju;
Database changed
```

1) mysql> CREATE TABLE student22(ID INT(10),STUDNAME VARCHAR(20),DEPT VARCHAR(20));

```
mysql> DESC student22;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| ID       | int(10)     | YES  |     | NULL    |       |
| STUDNAME | varchar(20) | YES  |     | NULL    |       |
| DEPT     | varchar(20) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.02 sec)
```

2) mysql> ALTER TABLE student22 ADD EMAIL_ID VARCHAR(20);

```
Query OK, 0 rows affected (0.51 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC student22;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| ID       | int(10)     | YES  |     | NULL    |       |
| STUDNAME | varchar(20) | YES  |     | NULL    |       |
| DEPT     | varchar(20) | YES  |     | NULL    |       |
| EMAIL_ID | varchar(20) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

3)mysql> DESC student22;

```
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| ID       | int(10)     | YES  |     | NULL    |       |
| STUDNAME | varchar(20) | YES  |     | NULL    |       |
| DEPT     | varchar(20) | YES  |     | NULL    |       |
| EMAIL_ID | varchar(20) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Date:

Experiment No:2

CREATION,MODIFICATION,CONFIGURATION AND DELTION OF DATABASE

AIM: To create ,modify, configure and delete database using DDL commands

QUESTION

1. Create a table STUDENT.
2. Add the column email ID to the table.
3. Show the structure of the table STUDENT.
4. Modify the table by changing student name field size with 40.
5. Modify the table by adding field credit to the table.
6. Drop the table STUDENT.

4) mysql> ALTER TABLE student22 MODIFY STUDNAME VARCHAR(40);
Query OK, 0 rows affected (0.06 sec)
Records: 0 Duplicates: 0 Warnings: 0

```
mysql> DESC student22;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| ID       | int(10)     | YES  |     | NULL    |       |
| STUDNAME | varchar(40) | YES  |     | NULL    |       |
| DEPT     | varchar(20) | YES  |     | NULL    |       |
| EMAIL_ID | varchar(20) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

5) mysql> ALTER TABLE student22 ADD CREDIT INT(10);
Query OK, 0 rows affected (0.48 sec)
Records: 0 Duplicates: 0 Warnings: 0

```
mysql> DESC student22;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| ID       | int(10)     | YES  |     | NULL    |       |
| STUDNAME | varchar(40) | YES  |     | NULL    |       |
| DEPT     | varchar(20) | YES  |     | NULL    |       |
| EMAIL_ID | varchar(20) | YES  |     | NULL    |       |
| GENDER   | varchar(1)  | YES  |     | NULL    |       |
| CREDIT   | int(10)     | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

6) mysql> DROP TABLE student22;
Query OK, 0 rows affected (0.14 sec)

```
mysql> DESC student22;
ERROR 1146 (42S02): Table 'sanju.student22' doesn't exist
mysql>
```

RESULT: Query to create, modify, configure and delete database using DDL commands has executed successfully and output obtained

OUTPUT

1) mysql> CREATE TABLE EMPLOYEE (EMPNUM INT(10),EMPNAME VARCHAR(25),JOB VARCHAR(15),SALARY INT(12),DEPT VARCHAR(5));
Query OK, 0 rows affected (0.45 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> DESC EMPLOYEE;

```
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| EMPNUM     | int(10)     | YES  |     | NULL    |       |
| EMPNAME    | varchar(25) | YES  |     | NULL    |       |
| JOB        | varchar(15) | YES  |     | NULL    |       |
| SALARY     | int(12)     | YES  |     | NULL    |       |
| DEPT       | varchar(5)  | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

6 rows in set (0.02 sec)

2) ALTER TABLE EMPLOYEE ADD EXPERIENCE INT(10);
Query OK, 0 rows affected (0.43 sec)
Records: 0 Duplicates: 0 Warnings: 0

3)mysql> DESC EMPLOYEE;

```
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| EMPNUM     | int(10)     | YES  |     | NULL    |       |
| EMPNAME    | varchar(25) | YES  |     | NULL    |       |
| JOB        | varchar(15) | YES  |     | NULL    |       |
| SALARY     | int(12)     | YES  |     | NULL    |       |
| DEPT       | varchar(5)  | YES  |     | NULL    |       |
| EXPERIENCE | int(10)     | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

6 rows in set (0.00 sec)

4) mysql> INSERT INTO EMPLOYEE VALUES (100,"SANJU","HECKER",40000,"IT",3),(101,"MANJU","PROGRAMMER",25000,"IT",2),(102,"KUNJU","MANAGER",60000,"HR",5),(103,"ANJU","INSTRUCTOR",55000,"HR",3),(104,"RENJU","SUPERVISOR",40000,"IT",3);

Query OK, 5 rows affected (0.13 sec)
Records: 5 Duplicates: 0 Warnings: 0

Date:

Experiment No:3

SQL COMMANDS FOR DML

AIM: To implement SQL commands for DML

QUESTION

1. Create a table EMPLOYEE.
2. Add the column experience to table EMPLOYEE.
3. Show the structure of the table EMPLOYEE.
4. Insert five different rows to table relevant data.
5. Display the table EMPLOYEE.
6. Update the EMPLOYEE table where set job = programmer where dept=it
7. Delete a employee from the table whose emp_no is 101
8. Display the details of emp_name, job, salary from EMPLOYEE.
9. Insert two different rows to the table EMPLOYEE.
10. Display emp_name, job, salary from EMPLOYEE table.
11. Update the salary =25000 where job=tester from the EMPLOYEE table.
12. Display the salary by order from the employee table.
13. Display the salary by descending order from the EMPLOYEE table.
14. Display the details of employee where salary>25000.
15. Display the distinct department from table EMPLOYEE.
16. Display the emp_no and emp_name from the EMPLOYEE table where salary is between 3000 and 45000.

5) mysql> SELECT * FROM EMPLOYEE;

```
+--------+---------+------------+--------+------+------------+
| EMPNUM | EMPNAME | JOB        | SALARY | DEPT | EXPERIENCE |
+--------+---------+------------+--------+------+------------+
|   100 | SANJU   | HECKER     |  40000 | IT   |          3 |
|   101 | MANJU   | PROGRAMMER |  25000 | IT   |          2 |
|   102 | KUNJU   | MANAGER    |  60000 | HR   |          5 |
|   103 | ANJU    | INSTRUCTOR |  55000 | HR   |          3 |
|   104 | RENJU   | SUPERVISOR |  40000 | IT   |          3 |
+--------+---------+------------+--------+------+-----------
```

6) mysql> UPDATE EMPLOYEE SET JOB="PROGRAMMER" WHERE DEPT="IT";
Query OK, 2 rows affected (0.05 sec)
Rows matched: 3 Changed: 2 Warnings: 0

mysql> SELECT * FROM EMPLOYEE;

```
+--------+---------+------------+--------+------+------------+
| EMPNUM | EMPNAME | JOB        | SALARY | DEPT | EXPERIENCE |
+--------+---------+------------+--------+------+------------+
|   100 | SANJU   | PROGRAMMER |  40000 | IT   |          3 |
|   101 | MANJU   | PROGRAMMER |  25000 | IT   |          2 |
|   102 | KUNJU   | MANAGER    |  60000 | HR   |          5 |
|   103 | ANJU    | INSTRUCTOR |  55000 | HR   |          3 |
|   104 | RENJU   | PROGRAMMER |  40000 | IT   |          3 |
+--------+---------+------------+--------+------+------------+
```

5 rows in set (0.00 sec)

7) mysql> DELETE FROM EMPLOYEE WHERE EMPNUM = 101;
Query OK, 1 row affected (0.03 sec)

mysql> SELECT * FROM EMPLOYEE;

```
+--------+---------+------------+--------+------+------------+
| EMPNUM | EMPNAME | JOB        | SALARY | DEPT | EXPERIENCE |
+--------+---------+------------+--------+------+------------+
|   100 | SANJU   | PROGRAMMER |  40000 | IT   |          3 |
|   102 | KUNJU   | MANAGER    |  60000 | HR   |          5 |
|   103 | ANJU    | INSTRUCTOR |  55000 | HR   |          3 |
|   104 | RENJU   | PROGRAMMER |  40000 | IT   |          3 |
+--------+---------+------------+--------+------+------------+
```

4 rows in set (0.00 sec)

17. Display the emp_no and emp_name from the EMPLOYEE table where emp_name should start with “S”.
18. Display the emp_no and emp_name from the table where department=IT and emp_name is greater than 110.
19. Display emp_no and emp_name from the table where department=IT or emp_name is greater than 110
20. Rename the table EMPLOYEE to EMPLY.
21. Display the distinct employee name from the table EMPLOYEE.

8) mysql> SELECT EMPNAME,JOB,SALARY FROM EMPLOYEE;

```
+---------+------------+--------+
| EMPNAME | JOB        | SALARY |
+---------+------------+--------+
| SANJU   | PROGRAMMER |  40000 |
| KUNJU   | MANAGER    |  60000 |
| ANJU    | INSTRUCTOR |  55000 |
| RENJU   | PROGRAMMER |  40000 |
+---------+------------+--------+
4 rows in set (0.00 sec)
```

9) mysql> INSERT INTO EMPLOYEE
VALUES(106,"CHINJU","SUPERVISOR",50000,"HR",4),(107,"RINJU","TESTER",10000
0,"DR",10);
Query OK, 2 rows affected (0.04 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM EMPLOYEE;

```
+--------+---------+------------+--------+------+------------+
| EMPNUM | EMPNAME | JOB        | SALARY | DEPT | EXPERIENCE |
+--------+---------+------------+--------+------+------------+
|    100 | SANJU   | PROGRAMMER |  40000 | IT   |          3 |
|    102 | KUNJU   | MANAGER    |  60000 | HR   |          5 |
|    103 | ANJU    | INSTRUCTOR |  55000 | HR   |          3 |
|    104 | RENJU   | PROGRAMMER |  40000 | IT   |          3 |
|    106 | CHINJU  | SUPERVISOR |  50000 | HR   |          4 |
|    107 | RINJU   | TESTER     | 100000 | DR   |         10 |
+--------+---------+------------+--------+------+------------+
6 rows in set (0.00 sec)
```

10) mysql> SELECT EMPNAME,JOB,SALARY FROM EMPLOYEE;

```
+---------+------------+--------+
| EMPNAME | JOB        | SALARY |
+---------+------------+--------+
| SANJU   | PROGRAMMER |  40000 |
| KUNJU   | MANAGER    |  60000 |
| ANJU    | INSTRUCTOR |  55000 |
| RENJU   | PROGRAMMER |  40000 |
| CHINJU  | SUPERVISOR |  50000 |
| RINJU   | TESTER     | 100000 |
+---------+------------+--------+
6 rows in set (0.00 sec)
```

11) mysql> UPDATE EMPLOYEE SET SALARY=25000 WHERE JOB="TESTER";
Query OK, 1 row affected (0.04 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM EMPLOYEE;

EMPNUM	EMPNAME	JOB	SALARY	DEPT	EXPERIENCE
100	SANJU	PROGRAMMER	40000	IT	3
102	KUNJU	MANAGER	60000	HR	5
103	ANJU	INSTRUCTOR	55000	HR	3
104	RENJU	PROGRAMMER	40000	IT	3
106	CHINJU	SUPERVISOR	50000	HR	4
107	RINJU	TESTER	25000	DR	10

6 rows in set (0.00 sec)

12) mysql> SELECT SALARY FROM EMPLOYEE ORDER BY SALARY;

SALARY
25000
40000
40000
50000
55000
60000

6 rows in set (0.04 sec)

13) mysql> SELECT SALARY FROM EMPLOYEE ORDER BY SALARY DESC;

SALARY
60000
55000
50000
40000
40000
25000

6 rows in set (0.00 sec)

14) mysql> SELECT *
-> FROM EMPLOYEE
-> WHERE SALARY>25000;

```
+--------+---------+------------+--------+------+------------+
| EMPNUM | EMPNAME | JOB        | SALARY | DEPT | EXPERIENCE |
+--------+---------+------------+--------+------+------------+
|   100 | SANJU   | PROGRAMMER |  40000 | IT   |          3 |
|   102 | KUNJU   | MANAGER    |  60000 | HR   |          5 |
|   103 | ANJU    | INSTRUCTOR |  55000 | HR   |          3 |
|   104 | RENJU   | PROGRAMMER |  40000 | IT   |          3 |
|   106 | CHINJU  | SUPERVISOR |  50000 | HR   |          4 |
+--------+---------+------------+--------+------+------------+
5 rows in set (0.02 sec)
```

15) mysql> SELECT DISTINCT DEPT
-> FROM EMPLOYEE;

```
+------+
| DEPT |
+------+
| IT   |
| HR   |
| DR   |
+------+
3 rows in set (0.02 sec)
```

16) mysql> SELECT EMPNUM,EMPNAME
-> FROM EMPLOYEE
-> WHERE SALARY BETWEEN 30000 AND 45000;

```
+--------+---------+
| EMPNUM | EMPNAME |
+--------+---------+
|   100 | SANJU   |
|   104 | RENJU   |
+--------+---------+
```

17) mysql> SELECT EMPNUM,EMPNAME FROM EMPLOYEE WHERE EMPNAME LIKE "S%";

```
+--------+---------+
| EMPNUM | EMPNAME |
+--------+---------+
|   100 | SANJU   |
+--------+---------+
1 row in set (0.02 sec)
```

18) mysql> SELECT EMPNUM,EMPNAME FROM EMPLOYEE WHERE DEPT = "IT" AND EMPNUM >103;

```
+--------+---------+
| EMPNUM | EMPNAME |
+--------+---------+
|   104 | RENJU   |
+--------+---------+
```

19)mysql> SELECT EMPNUM,EMPNAME FROM EMPLY WHERE DEPT = "IT" OR EMPNUM >103
-> ;

```
+--------+---------+
| EMPNUM | EMPNAME |
+--------+---------+
|   100 | SANJU   |
|   104 | RENJU   |
|   106 | CHINJU  |
|   107 | RINJU   |
+--------+---------+
```

4 rows in set (0.01 sec)

20) mysql> RENAME TABLE EMPLOYE TO EMPLY;
Query OK, 0 rows affected (0.15 sec)
mysql> SELECT * FROM EMPLY;

```
+--------+---------+------------+--------+------+------------+
| EMPNUM | EMPNAME | JOB        | SALARY | DEPT | EXPERIENCE |
+--------+---------+------------+--------+------+------------+
|   100 | SANJU   | PROGRAMMER |  40000 | IT   |          3 |
|   102 | KUNJU   | MANAGER    |  60000 | HR   |          5 |
|   103 | ANJU    | INSTRUCTOR |  55000 | HR   |          3 |
|   104 | RENJU   | PROGRAMMER |  40000 | IT   |          3 |
|   106 | CHINJU  | SUPERVISOR |  50000 | HR   |          4 |
|   107 | RINJU   | TESTER     |  25000 | DR   |         10 |
+--------+---------+------------+--------+------+------------+
```

6 rows in set (0.00 sec)

21)
mysql> SELECT DISTINCT EMPNAME FROM EMPLY;

```
+---------+
| EMPNAME |
+---------+
| SANJU   |
| KUNJU   |
| ANJU    |
| RENJU   |
| CHINJU  |
| RINJU   |
+---------+
```

6 rows in set (0.00 sec)

RESULT: Queries To implement SQL commands for DML has executed successfully and output obtained

OUTPUT

1.NUMERIC FUNCTION

a) mysql> SELECT ABS(-15) AS ABS;

```
+-----+
| ABS |
+-----+
|  15 |
+-----+
1 row in set (0.00 sec)
```

b) mysql> SELECT CEIL(55.67) AS CEIL;

```
+------+
| CEIL |
+------+
|   56 |
+------+
1 row in set (0.00 sec)
```

c) mysql> SELECT EXP(4) AS EXP;

```
+--------------------+
| EXP                |
+--------------------+
| 54.598150033144236 |
+--------------------+
1 row in set (0.00 sec)
```

d)mysql> SELECT FLOOR(100.2) AS FLOOR;

```
+-------+
| FLOOR |
+-------+
|   100 |
+-------+
1 row in set (0.00 sec)
```

e)mysql> SELECT MOD(10,3);;

```
+-----------+
| MOD(10,3) |
+-----------+
|         1 |
+-----------+
1 row in set (0.00 sec)
```

Date:

Experiment No:4

IMPLEMENTATION OF BUILT-IN FUNCTIONS IN RDBMS

AIM: To implement built in functions in RDBMS

MySQL has many built-in functions. This reference contains string, numeric, date, and some advanced functions in MySQL

QUESTION

1. **NUMERIC FUNCTIONS**
   - **a.** Write a query to apply ABS function to -15
   - **b.** Write a query to apply CEIL function to 55.67
   - **c.** Write a query to apply EXP function to 4
   - **d.** Write a query to apply FLOOR function to 100.2
   - **e.** Write a query to apply MOD function to 10,3
   - **f.** Write a query to apply ROUND function to 100.256,2
   - **g.** Write a query to apply SQRT function to 16
   - **h.** Write a query to apply TRUNCATE function to 100.256,2
   - **i.** Write a query to apply GREATEST functions
   - **j.** Write a query to apply LEAST function
   - **k.** Write a query to apply POWER function
   - **l.** Write a query to apply SIGN function

f)mysql> SELECT ROUND(100.256,2);
```
+------------------+
| ROUND(100.256,2) |
+------------------+
|           100.26 |
+------------------+
1 row in set (0.00 sec)
```

g)mysql> SELECT SQRT(16);
```
+----------+
| SQRT(16) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)
```

h)mysql> SELECT TRUNCATE (100.256,2);
```
+----------------------+
| TRUNCATE (100.256,2) |
+----------------------+
|               100.25 |
+----------------------+
1 row in set (0.00 sec)
```

i)mysql> SELECT GREATEST(5,10,2,15) AS GREAT;
```
+-------+
| GREAT |
+-------+
|    15 |
+-------+
1 row in set (0.00 sec)
```

j)mysql> SELECT LEAST(5,10,2,15) AS LEAST;
```
+-------+
| LEAST |
+-------+
|     2 |
+-------+
1 row in set (0.00 sec)
```

k)mysql> SELECT POWER(3,3) AS POWER;
```
+-------+
| POWER |
+-------+
|    27 |
+-------+
```

2. STRING FUNCTIONS

a. Write a query to apply the ASCII function.

b. Write a query to apply the CHAR_LENGTH function.

c. Write a query to apply the CONCAT function.

d. Write a query to apply the LCASE function.

e. Write a query to apply the FORMAT function.

f. Write a query to apply the LEFT function.

g. Write a query to apply the LOCATE function.

h. Write a query to apply the REPEAT function.

i. Write a query to apply the REVERSE function.

j. Write a query to apply the STRCMP function.

k. Write a query to apply the SUBSTR function.

l. Write a query to apply the UCASE function.

m. Write a query to apply the LOWER function.

n. Write a query to apply the UPPER function.

o. Write a query to apply the LTRIM function.

p. Write a query to apply the RTRIM function.

l)mysql> SELECT SIGN(-23) AS SIGN;
```
+------+
| SIGN |
+------+
|   -1 |
+------+
1 row in set (0.00 sec)
```

2.STRING FUNCTION

a)mysql> SELECT ASCII('A') AS ASCII;
```
+-------+
| ASCII |
+-------+
|    65 |
+-------+
1 row in set (0.00 sec)
```

b)SELECT CHAR_LENGTH("SANJU") AS LENGTH;
```
+--------+
| LENGTH |
+--------+
|      5 |
+--------+
1 row in set (0.00 sec)
```

c)mysql> SELECT CONCAT("SANJU"," V PAULOSE") AS CONCAT;
```
+-----------------+
| CONCAT          |
+-----------------+
| SANJU V PAULOSE |
+-----------------+
1 row in set (0.01 sec)
```

d)mysql> SELECT LCASE("SANJU") AS CASEL;
```
+-------+
| CASEL |
+-------+
| sanju |
+-------+
    1   row in set (0.01 sec)
```

3. DATE FUNCTION

a. Write a query to apply the CURRENT_DATE function.

b. Write a query to apply the DATE function.

c. Write a query to apply the DATE_FORMAT function.

d. Write a query to apply the DAY function.

e. Write a query to apply the DAYNAME function.

f. Write a query to apply the DAYOFYEAR function.

g. Write a query to apply the DAYOFWEEK function.

h. Write a query to apply the MONTH function.

i. Write a query to apply the HOUR function.

j. Write a query to apply the MINUTE function.

k. Write a query to apply the SECOND function.

l. Write a query to apply the TIME function.

4. ADVANCED FUNCTIONS

a. Write a query to apply the USER function.

b. Write a query to apply the DATABASE function.

c. Write a query to apply the SYSTEM_USER function.

d. Write a query to apply the VERSION function.

e)mysql> SELECT FORMAT(1234.345,2) AS MAR;
```
+----------+
| MAR      |
+----------+
| 1,234.35 |
+----------+
1 row in set (0.00 sec)
```

f)SELECT LEFT("SANJU",3) AS LEF;
```
+------+
| LEF  |
+------+
| SAN  |
+------+
1 row in set (0.00 sec)
```

g)mysql> SELECT LOCATE("J","SANJUVPAULOSE");
```
+-----------------------------+
| LOCATE("J","SANJUVPAULOSE") |
+-----------------------------+
|                           4 |
+-----------------------------+
1 row in set (0.00 sec)
```

h)mysql> SELECT REPEAT("SANJU ",4) AS RE;
```
+--------------------------+
| RE                       |
+--------------------------+
| SANJU SANJU SANJU SANJU  |
+--------------------------+
1 row in set (0.00 sec)
```

i)mysql> SELECT REVERSE("SANJU") AS REV;
```
+-------+
| REV   |
+-------+
| UJNAS |
+-------+
1 row in set (0.00 sec)
```

j)mysql> SELECT STRCMP("STRINGCOMPARING","STRINGCOMPARING") AS COMP;
```
+------+
| COMP |
+------+
|    0 |
+------+
1 row in set (0.01 sec)
```

k)mysql> SELECT SUBSTR("THIS IS SUBSTRING",3,8) AS SUB;;
```
+----------+
| SUB      |
+----------+
| IS IS SU |
+----------+
1 row in set (0.00 sec)
```

l)mysql> SELECT UCASE("sanju") AS U;
```
+-------+
| U     |
+-------+
| SANJU |
+-------+
1 row in set (0.00 sec)
```

m)mysql> SELECT LOWER("ANJU") AS LOWE;
```
+------+
| LOWE |
+------+
| anju |
+------+
1 row in set (0.00 sec)
```

n)mysql> SELECT UPPER("anju") AS UP;
```
+------+
| UP   |
+------+
| ANJU |
+------+
1 row in set (0.00 sec)
```

o)mysql> SELECT LTRIM ("   SANJUUUUU") AS LT;
```
+-----------+
| LT        |
+-----------+
| SANJUUUUU |
+-----------+
1 row in set (0.00 sec)
```

p)mysql> SELECT RTRIM("SANJUUUUU   ") AS RT;
```
+-----------+
| RT        |
+-----------+
| SANJUUUUU |
+-----------+
```

3.STRING FUNCTION

a)mysql> SELECT CURRENT_DATE() AS DATE;
```
+------------+
| DATE       |
+------------+
| 2022-11-17 |
+------------+
1 row in set (0.00 sec)
```

b)mysql> SELECT DATE('2001/08/10') AS DA;
```
+------------+
| DA         |
+------------+
| 2001-08-10 |
+------------+
1 row in set (0.00 sec)
```

c)mysql> SELECT DATE_FORMAT('2022/10/12','%M') AS DAT;
```
+---------+
| DAT     |
+---------+
| October |
+---------+
1 row in set (0.00 sec)
```

d)mysql> SELECT DAY('2001/08/10') AS DAY;
```
+------+
| DAY  |
+------+
|   10 |
+------+
1 row in set (0.00 sec)
```

e)mysql> SELECT DAYNAME('2001/08/10') AS DAYN;
```
+--------+
| DAYN   |
+--------+
| Friday |
+--------+
1 row in set (0.00 sec)
```

f)mysql> SELECT DAYOFYEAR('2022/11/17') AS YER;
```
+------+
| YER  |
+------+
|  321 |
+------+
1 row in set (0.00 sec)
```

g)mysql> SELECT DAYOFWEEK('2022/11/17') AS WEK;
```
+------+
| WEK  |
+------+
|    5 |
+------+
1 row in set (0.00 sec)
```

h)mysql> SELECT MONTH('2022/11/17') AS MON;
```
+------+
| MON  |
+------+
|   11 |
+------+
1 row in set (0.00 sec)
```

i) SELECT HOUR('13:05:00') AS HOR;
```
+------+
| HOR  |
+------+
|   13 |
+------+
1 row in set (0.00 sec)
```

j)mysql> SELECT MINUTE('13:05:56') AS MIN;
```
+------+
| MIN  |
+------+
|    5 |
+------+
1 row in set (0.00 sec)
```

k)mysql> SELECT SECOND('13:05:56') AS SEC;
```
+------+
| SEC  |
+------+
|   56 |
+------+
1 row in set (0.00 sec)
```

l)mysql> SELECT TIME('13:05:56') AS TIM;
```
+----------+
| TIM      |
+----------+
| 13:05:56 |
+----------+
1 row in set (0.00 sec)
```

4.ADVANCED FUNCTIONS

a)mysql> SELECT USER() AS USER;

```
+----------------+
| USER           |
+----------------+
| root@localhost |
+----------------+
1 row in set (0.00 sec)
```

b)mysql> SELECT DATABASE() AS DB;

```
+-------+
| DB    |
+-------+
| sanju |
+-------+
1 row in set (0.00 sec)
```

c)mysql> SELECT SYSTEM_USER() AS SYSUSR;

```
+----------------+
| SYSUSR         |
+----------------+
| root@localhost |
+----------------+
1 row in set (0.00 sec)
```

d)mysql> SELECT VERSION() AS VER;

```
+-------------------------+
| VER                     |
+-------------------------+
| 5.7.32-0ubuntu0.16.04.1 |
+-------------------------+
1 row in set (0.00 sec)
```

RESULT : Queries to implement built-in functions in RDBMS has executed successfully and output obtained

OUTPUT

1)mysql> CREATE TABLE STUDENT(IDNUMBER INT(5),STUDENTNAME VARCHAR(20),DEPT VARCHAR(6),ADDRESS VARCHAR(10),MARKS INT(5),TOTAL INT(5),AVG INT(5));
Query OK, 0 rows affected (0.31 sec)

```
mysql> DESC STUDENT
    -> ;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| IDNUMBER    | int(5)      | YES  |     | NULL    |       |
| STUDENTNAME | varchar(20) | YES  |     | NULL    |       |
| DEPT        | varchar(6)  | YES  |     | NULL    |       |
| ADDRESS     | varchar(10) | YES  |     | NULL    |       |
| MARKS       | int(5)      | YES  |     | NULL    |       |
| TOTAL       | int(5)      | YES  |     | NULL    |       |
| AVG         | int(5)      | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
7 rows in set (0.01 sec)
```

2)mysql> INSERT INTO STUDENT VALUES(1,"MANJU","CSE","ABC",10,50,5),(2,"ANJU","CE","DEF",20,50,10),(3,"KUNJU","EC","GHI",30,50,15),(4,"CHINJU","EEE","JKL",40,50,20),(5,"SANJU","MECH","MNO",50,50,25);
Query OK, 5 rows affected (0.72 sec)
Records: 5 Duplicates: 0 Warnings: 0

```
mysql> SELECT * FROM STUDENT;
+----------+-------------+------+---------+-------+-------+------+
| IDNUMBER | STUDENTNAME | DEPT | ADDRESS | MARKS | TOTAL | AVG  |
+----------+-------------+------+---------+-------+-------+------+
|        1 | MANJU       | CSE  | ABC     |    10 |    50 |    5 |
|        2 | ANJU        | CE   | DEF     |    20 |    50 |   10 |
|        3 | KUNJU       | EC   | GHI     |    30 |    50 |   15 |
|        4 | CHINJU      | EEE  | JKL     |    40 |    50 |   20 |
|        5 | SANJU       | MECH | MNO     |    50 |    50 |   25 |
+----------+-------------+------+---------+-------+-------+------+
5 rows in set (0.03 sec)
```

3)mysql> SELECT MIN(MARKS) FROM STUDENT;

```
+------------+
| MIN(MARKS) |
+------------+
|         10 |
+------------+
1 row in set (0.01 sec)
```

Date:

Experiment No:5

IMPLEMENTATION OF VARIOUS AGGREGATE FUNCTION IN SQL

AIM: To implement various aggregate functions in SQL

QUESTION

1. Create a student table with attributes with id number, student name, department, address, marks, total and average.
2. Insert the values of student table.
3. Display the minimum marks from the student table.
4. Display the maximum marks from the student table.
5. Display the average marks from the student table.
6. Display the sum of the marks from the student table.
7. Insert two different values to the existing table.
8. Display the count of students from the table where department is equal to CSE.
9. Display the maximum marks minus minimum marks from the student table.
10. Display the minimum marks from the table where department is equal to CSE.
11. Display the maximum marks from the table where department is equal to CSE.
12. Display the average marks from the table where department is equal to CSE.
13. Display the minimum, maximum, average marks from student table.

4)mysql> SELECT MAX(MARKS) FROM STUDENT;

```
+------------+
| MAX(MARKS) |
+------------+
|         50 |
+------------+
```

1 row in set (0.00 sec)

5)mysql> SELECT AVG(MARKS) FROM STUDENT;

```
+------------+
| AVG(MARKS) |
+------------+
|    30.0000 |
+------------+
```

1 row in set (0.03 sec)

6)mysql> SELECT SUM(MARKS) FROM STUDENT;

```
+------------+
| SUM(MARKS) |
+------------+
|        150 |
+------------+
```

1 row in set (0.00 sec)

7)mysql> INSERT INTO STUDENT
VALUES(6,"RENJU","CSE","PQR",25,50,18),(7,"RINJU","CSE","STU",35,50,20);
Query OK, 2 rows affected (0.02 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM STUDENT ;

```
+----------+-------------+------+---------+-------+-------+------+
| IDNUMBER | STUDENTNAME | DEPT | ADDRESS | MARKS | TOTAL | AVG  |
+----------+-------------+------+---------+-------+-------+------+
|        1 | MANJU       | CSE  | ABC     |    10 |    50 |    5 |
|        2 | ANJU        | CE   | DEF     |    20 |    50 |   10 |
|        3 | KUNJU       | EC   | GHI     |    30 |    50 |   15 |
|        4 | CHINJU      | EEE  | JKL     |    40 |    50 |   20 |
|        5 | SANJU       | MECH | MNO     |    50 |    50 |   25 |
|        6 | RENJU       | CSE  | PQR     |    25 |    50 |   18 |
|        7 | RINJU       | CSE  | STU     |    35 |    50 |   20 |
+----------+-------------+------+---------+-------+-------+------+
```

7 rows in set (0.00 sec)

8)mysql> SELECT COUNT(STUDENTNAME) FROM STUDENT WHERE DEPT="CSE";

```
+--------------------+
| COUNT(STUDENTNAME) |
+--------------------+
|                  3 |
+--------------------+
1 row in set (0.07 sec)
```

9)mysql> SELECT MAX(MARKS)- MIN(MARKS) FROM STUDENT;

```
+------------------------+
| MAX(MARKS)- MIN(MARKS) |
+------------------------+
|                     40 |
+------------------------+
1 row in set (0.00 sec)
```

10)mysql> SELECT MIN(MARKS) FROM STUDENT WHERE DEPT="CSE";

```
+------------+
| MIN(MARKS) |
+------------+
|         10 |
+------------+
1 row in set (0.00 sec)
```

11)mysql> SELECT MAX(MARKS) FROM STUDENT WHERE DEPT="CSE";

```
+------------+
| MAX(MARKS) |
+------------+
|         35 |
+------------+
1 row in set (0.00 sec)
```

12)mysql> SELECT AVG(MARKS) FROM STUDENT WHERE DEPT="CSE";

```
+------------+
| AVG(MARKS) |
+------------+
|    23.3333 |
+------------+
1 row in set (0.00 sec)
```

1)mysql> SELECT MAX(MARKS),MIN(MARKS),AVG(MARKS) FROM STUDENT;

```
+------------+------------+------------+
| MAX(MARKS) | MIN(MARKS) | AVG(MARKS) |
+------------+------------+------------+
|         50 |         10 |    30.0000 |
+------------+------------+------------+
1 row in set (0.00 sec)
```

RESULT : Queries to implement various aggregate functions in SQL has executed successfully and output obtained

OUTPUT

1)mysql>SELECT * FROM STUDENT ORDER BY DEPP="CSE" ASC;

```
+----------+-------------+------+---------+-------+-------+------+
| IDNUMBER | STUDENTNAME | DEPP | ADDRESS | MARKS | TOTAL | AVG  |
+----------+-------------+------+---------+-------+-------+------+
|        2 | ANJU        | CE   | DEF     |    20 |    50 |   10 |
|        3 | KUNJU       | EC   | GHI     |    30 |    50 |   15 |
|        4 | CHINJU      | EEE  | JKL     |    40 |    50 |   20 |
|        5 | SANJU       | MECH | MNO     |    50 |    50 |   25 |
|        1 | MANJU       | CSE  | ABC     |    10 |    50 |    5 |
|        6 | RENJU       | CSE  | PQR     |    25 |    50 |   18 |
|        7 | RINJU       | CSE  | STU     |    35 |    50 |   20 |
+----------+-------------+------+---------+-------+-------+------+
7 rows in set (0.01 sec)
```

2)mysql> SELECT * FROM STUDENT ORDER BY IDNUMBER DESC;

```
+----------+-------------+------+---------+-------+-------+------+
| IDNUMBER | STUDENTNAME | DEPP | ADDRESS | MARKS | TOTAL | AVG  |
+----------+-------------+------+---------+-------+-------+------+
|        7 | RINJU       | CSE  | STU     |    35 |    50 |   20 |
|        6 | RENJU       | CSE  | PQR     |    25 |    50 |   18 |
|        5 | SANJU       | MECH | MNO     |    50 |    50 |   25 |
|        4 | CHINJU      | EEE  | JKL     |    40 |    50 |   20 |
|        3 | KUNJU       | EC   | GHI     |    30 |    50 |   15 |
|        2 | ANJU        | CE   | DEF     |    20 |    50 |   10 |
|        1 | MANJU       | CSE  | ABC     |    10 |    50 |    5 |
+----------+-------------+------+---------+-------+-------+------+
7 rows in set (0.00 sec)
```

3)mysql> SELECT MIN(MARKS),MAX(MARKS),AVG(MARKS) FROM STUDENT GROUP BY MARKS;

```
+------------+------------+------------+
| MIN(MARKS) | MAX(MARKS) | AVG(MARKS) |
+------------+------------+------------+
|         10 |         10 |    10.0000 |
|         20 |         20 |    20.0000 |
|         25 |         25 |    25.0000 |
|         30 |         30 |    30.0000 |
|         35 |         35 |    35.0000 |
|         40 |         40 |    40.0000 |
|         50 |         50 |    50.0000 |
+------------+------------+------------+
7 rows in set (0.02 sec)
```

Date:

Experiment No:6

IMPLEMENTATION OF ORDER BY, GROUP BY & HAVING CLAUSE

AIM: To implement order by, group by, having clause

ID NUMBER	STUDENT NAME	DEPT	ADDRESS	MARKS	TOTAL	AVERAGE
1	MANJU	CSE	ABC	10	50	5
2	ANJU	CE	DEF	20	50	10
3	KUNJU	EC	GHI	30	50	15
4	CHINJU	EEE	JKL	40	50	20
5	SANJU	MECH	MNO	50	50	25
6	RENJU	CSE	PQR	25	50	18
7	RINJU	CSE	STU	35	50	20

QUESTIONS

1. Display the student table order by CSE in ascending order.
2. Display the student table order by id number in descending order.
3. Display the minimum, maximum, average marks from student table department using group by operation.
4. Display the distinct department in minimum, maximum marks from student table using group by operation.
5. Display the count idnumber, department from student table by using group by having function which is greater than the count idnumber by order by function in descending order.

4)mysql> SELECT DISTINCT DEPP,MIN(MARKS),MAX(MARKS) FROM STUDENT GROUP BY DEPP;

```
+------+------------+------------+
| DEPP | MIN(MARKS) | MAX(MARKS) |
+------+------------+------------+
| CE   |         20 |         20 |
| CSE  |         10 |         35 |
| EC   |         30 |         30 |
| EEE  |         40 |         40 |
| MECH |         50 |         50 |
+------+------------+------------+
5 rows in set (0.00 sec)
```

5)mysql> SELECT COUNT(IDNUMBER), DEPP FROM STUDENT GROUP BY DEPP HAVING COUNT(IDNUMBER) >1 ORDER BY COUNT(IDNUMBER) DESC;

```
+-----------------+------+
| COUNT(IDNUMBER) | DEPP |
+-----------------+------+
|               3 | CSE  |
+-----------------+------+
1 row in set (0.00 sec)
```

RESULT : Queries to implement order by, group by, having clause has executed successfully and output obtained

OUTPUT

1. Set operators.

```
mysql> CREATE TABLE EMPY1(ID VARCHAR(4), FNAME VARCHAR(10),LNAME
VARCHAR(10),DEP VARCHAR(10),DESTN VARCHAR(15),SALARY INT(10));
Query OK, 0 rows affected (0.29 sec)
mysql> DESC EMPY1;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| ID     | varchar(4)  | YES  |     | NULL    |       |
| FNAME  | varchar(10) | YES  |     | NULL    |       |
| LNAME  | varchar(10) | YES  |     | NULL    |       |
| DEP    | varchar(10) | YES  |     | NULL    |       |
| DESTN  | varchar(15) | YES  |     | NULL    |       |
| SALARY | int(10)     | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
mysql> CREATE TABLE EMPY2(ID VARCHAR(4), FNAME VARCHAR(10),LNAME
VARCHAR(10),DEP VARCHAR(10),DESTN VARCHAR(15),SALARY INT(10));
Query OK, 0 rows affected (0.28 sec)
mysql> DESC EMPY2;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| ID     | varchar(4)  | YES  |     | NULL    |       |
| FNAME  | varchar(10) | YES  |     | NULL    |       |
| LNAME  | varchar(10) | YES  |     | NULL    |       |
| DEP    | varchar(10) | YES  |     | NULL    |       |
| DESTN  | varchar(15) | YES  |     | NULL    |       |
| SALARY | int(10)     | YES  |     | NULL    |
```

Date:

Experiment No:7

IMPLEMENTATION OF SET OPERATOR, NESTED QUERIES AND JOIN QUERIES

AIM: To implement set operators, nested queries and join queries

1. Set Operators

Consider this schema.

1. Employee1(id, fname, lname, dept, designation, salary)
2. Employee2(id, fname, lname, dept, designation, salary)

QUESTIONS

a. Display all the employee details from employee 1 & 2 using union all.

b. Display all the employee details from employee 1 & 2 using union.

c. Display all the employee details from employee 1 & 2 using union & order by fname.

d. Display all the employee details from employee 1 where fname in & fname from employee 2.

e. Display all the employee details from employee 1 where fname not in & fname from employee 2.

```
mysql> INSERT INTO EMPY1
VALUES("E100","AMAL","RAVI","IT","MANAGER",50000),("E101","SNEHA","MANO
J","PURCHASE","ACCOUNTANT",400000),("E102","RAHUL","RAJ","IT","DESIGNER"
,30000);

Query OK, 3 rows affected (0.04 sec)

Records: 3  Duplicates: 0  Warnings: 0


mysql> SELECT * FROM EMPY1;

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

| ID   | FNAME | LNAME | DEP      | DESTN      | SALARY |

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

| E100 | AMAL  | RAVI  | IT       | MANAGER    |  50000 |

| E101 | SNEHA | MANOJ | PURCHASE | ACCOUNTANT | 400000 |

| E102 | RAHUL | RAJ   | IT       | DESIGNER   |  30000 |

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

3 rows in set (0.00 sec)


mysql> INSERT INTO EMPY2
VALUES("E100","AMAL","RAVI","IT","MANAGER",50000),("E104","KAVYA","MUK
UND","HR","MANAGER",50000),("E105","AMAL","SAJEEV","MARKETING","MANA
GER",50000);

Query OK, 3 rows affected (0.05 sec)

mysql>SELECT * FROM EMPLY2

Records: 3  Duplicates: 0  Warnings: 0

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

| ID   | FNAME | LNAME  | DEP       | DESTN   | SALARY |

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

| E100 | AMAL  | RAVI   | IT        | MANAGER |  50000 |

| E104 | KAVYA | MUKUND | HR        | MANAGER |  50000 |

| E105 | AMAL  | SAJEEV | MARKETING | MANAGER |  50000 |

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

3 rows in set (0.00 sec)
```

2. Nested Queries

Consider the schema.

Employee55(empno,ename,job,manager,sal,deptno,commn,hiredate)

QUESTIONS

a. Write a SQL query to display the lastname,hiredate of any employee in the same department as "Albin" , exclude Albin.

b. Write a query to display employee number, lastname and salary of all employees who earn more than the average salary.Sort the result in ascending order.

c. Write a query to display employee number, lastname of all employees who were in a department with any employee whose last name contain a 'U'.

3. Join Queries

Consider the schema.

1. Members(member_id,name)
2. Committees(committee_id,name)

QUESTIONS

a. Find members who are also committee members (using join).

b. Join the members with the committees table using left join.

c. Join the members with the committees table using right join.

d. Join the members with the committees table using cross join.

a) mysql> SELECT * FROM EMPY1 UNION ALL SELECT * FROM EMPY2;

```
+------+-------+--------+-----------+------------+--------+
| ID   | FNAME | LNAME  | DEP       | DESTN      | SALARY |
+------+-------+--------+-----------+------------+--------+
| E100 | AMAL  | RAVI   | IT        | MANAGER    |  50000 |
| E101 | SNEHA | MANOJ  | PURCHASE  | ACCOUNTANT | 400000 |
| E102 | RAHUL | RAJ    | IT        | DESIGNER   |  30000 |
| E100 | AMAL  | RAVI   | IT        | MANAGER    |  50000 |
| E104 | KAVYA | MUKUND | HR        | MANAGER    |  50000 |
| E105 | AMAL  | SAJEEV | MARKETING | MANAGER    |  50000 |
+------+-------+--------+-----------+------------+--------+
```

6 rows in set (0.02 sec)

b) mysql> SELECT * FROM EMPY1 UNION SELECT * FROM EMPY2;

```
+------+-------+--------+-----------+------------+--------+
| ID   | FNAME | LNAME  | DEP       | DESTN      | SALARY |
+------+-------+--------+-----------+------------+--------+
| E100 | AMAL  | RAVI   | IT        | MANAGER    |  50000 |
| E101 | SNEHA | MANOJ  | PURCHASE  | ACCOUNTANT | 400000 |
| E102 | RAHUL | RAJ    | IT        | DESIGNER   |  30000 |
| E104 | KAVYA | MUKUND | HR        | MANAGER    |  50000 |
| E105 | AMAL  | SAJEEV | MARKETING | MANAGER    |  50000 |
+------+-------+--------+-----------+------------+--------+
```

5 rows in set (0.00 sec)

c) mysql> SELECT * FROM EMPY1 UNION SELECT * FROM EMPY2 ORDER BY FNAME;

```
+------+-------+--------+-----------+------------+--------+
| ID   | FNAME | LNAME  | DEP       | DESTN      | SALARY |
+------+-------+--------+-----------+------------+--------+
| E105 | AMAL  | SAJEEV | MARKETING | MANAGER    |  50000 |
```

| E100 | AMAL | RAVI | IT | MANAGER | 50000 |

| E104 | KAVYA | MUKUND | HR | MANAGER | 50000 |

| E102 | RAHUL | RAJ | IT | DESIGNER | 30000 |

| E101 | SNEHA | MANOJ | PURCHASE | ACCOUNTANT | 400000 |

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

5 rows in set (0.00 sec)

d) mysql> SELECT FNAME FROM EMPY1 WHERE FNAME IN(SELECT FNAME FROM EMPY2);

+-------+

| FNAME |

+-------+

| AMAL |

+-------+

1 row in set (0.00 sec)

e) mysql> SELECT FNAME FROM EMPY1 WHERE FNAME NOT IN(SELECT FNAME FROM EMPY2);

+-------+

| FNAME |

+-------+

| SNEHA |

| RAHUL |

+-------+

2 rows in set (0.00 sec)

2.NESTED QUERIES

```
mysql> INSERT INTO NEST
VALUES("E234","SUBIN","CLERK","DAYAL",8000,2,75,'1998/05/01');

Query OK, 1 row affected (0.03 sec)


mysql> INSERT INTO NEST
VALUES("E563","LINSON","ACCOUNTANT","SREEHARI",12000,3,3,'1999/05/04'),("E
101","SONNET","ANALYST","DHANYA",16000,1,280,'1996/05/25'),("E407","ANUPAM
A","PROGRAMMER","BINITH",17000,4,4,'2001/07/23');

Query OK, 3 rows affected (0.05 sec)

Records: 3  Duplicates: 0  Warnings: 0


mysql> INSERT INTO NEST
VALUES("E305","ALBIN","TECHNICIAN","SANOOT",15000,3,190,'2000/08/24'),("E497
","PINTO","SUPERVISOR","PRINCE",9000,5,0,'2006/07/04'),("E457","SAMJITH","MAN
AGER","VIMAL",18000,1,200,'2006/06/05');

Query OK, 3 rows affected (0.06 sec)

Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM NEST;

+-------+---------+------------+----------+-------+-------+-------+------------+
| EMPNO | EMPNME  | JOB        | MANAGER  | SAL   | DEPNO | COMMN | HIERDATE
|
+-------+---------+------------+----------+-------+-------+-------+------------+
| E234  | SUBIN   | CLERK      | DAYAL    |  8000 |     2 |    75 | 1998-05-01 |
| E563  | LINSON  | ACCOUNTANT | SREEHARI | 12000 |     3 |     3 | 1999-05-04 |
| E101  | SONNET  | ANALYST    | DHANYA   | 16000 |     1 |   280 | 1996-05-25 |
| E407  | ANUPAMA | PROGRAMMER | BINITH   | 17000 |     4 |     4 | 2001-07-23 |
| E305  | ALBIN   | TECHNICIAN | SANOOT   | 15000 |     3 |   190 | 2000-08-24 |
| E497  | PINTO   | SUPERVISOR | PRINCE   |  9000 |     5 |     0 | 2006-07-04 |
| E457  | SAMJITH | MANAGER    | VIMAL    | 18000 |     1 |   200 | 2006-06-05 |
+-------+---------+------------+----------+-------+-------+-------+------------+

7 rows in set (0.00 sec)
```

a) mysql> SELECT EMPNME,HIERDATE FROM NEST WHERE DEPNO=(SELECT DEPNO FROM NEST WHERE EMPNME="ALBIN") AND EMPNME!="ALBIN";

```
+--------+------------+
| EMPNME | HIERDATE   |
+--------+------------+
| LINSON | 1999-05-04 |
+--------+------------+
```

1 row in set (0.02 sec)

b) mysql> SELECT EMPNO,EMPNME,SAL FROM NEST WHERE SAL>(SELECT AVG(SAL) FROM NEST) ORDER BY EMPNO;

```
+-------+---------+-------+
| EMPNO | EMPNME  | SAL   |
+-------+---------+-------+
| E101  | SONNET  | 16000 |
| E305  | ALBIN   | 15000 |
| E407  | ANUPAMA | 17000 |
| E457  | SAMJITH | 18000 |
+-------+---------+-------+
```

4 rows in set (0.00 sec)

c) mysql> SELECT EMPNO,EMPNME FROM NEST WHERE DEPNO IN(SELECT DEPNO FROM NEST WHERE EMPNME LIKE '%U%');

```
+-------+---------+
| EMPNO | EMPNME  |
+-------+---------+
| E234  | SUBIN   |
| E407  | ANUPAMA |
+-------+---------+
```

2 rows in set (0.00 sec)

3.JOIN QUERIES

```
mysql> CREATE TABLE MEMB(MEMID INT(4),NAME VARCHAR(10));
Query OK, 0 rows affected (0.30 sec)

mysql> INSERT INTO MEMB
VALUES(100,"JOHN"),(200,"JANE"),(300,"MARY"),(400,"DAVID"),(500,"AMBELIA");
Query OK, 5 rows affected (0.04 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM MEMB;
+-------+---------+
| MEMID | NAME    |
+-------+---------+
|   100 | JOHN    |
|   200 | JANE    |
|   300 | MARY    |
|   400 | DAVID   |
|   500 | AMELIA |
+-------+---------+
5 rows in set (0.00 sec)

mysql> CREATE TABLE CMTS(CMTID INT(3),NAME VARCHAR(10));
Query OK, 0 rows affected (0.26 sec)

mysql> INSERT INTO CMTS
VALUES(100,"JOHN"),(200,"MARY"),(300,"AMBELLA"),(400,"JOE");
Query OK, 4 rows affected (0.04 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

```
mysql> SELECT * FROM CMTS;
+-------+---------+
| CMTID | NAME    |
+-------+---------+
|   100 | JOHN    |
|   200 | MARY    |
|   300 | AMELIA |
|   400 | JOE     |
+-------+---------+
4 rows in set (0.00 sec)
```

a) mysql> SELECT MEMB.NAME FROM MEMB INNER JOIN CMTS ON MEMB.NAME=CMTS.NAME;

```
+---------+
| NAME    |
+---------+
| JOHN    |
| MARY    |
| AMELIA |
```

b) mysql> SELECT * FROM MEMB LEFT JOIN CMTS ON MEMB.NAME=CMTS.NAME;

```
+-------+---------+-------+---------+
| MEMID | NAME    | CMTID | NAME    |
+-------+---------+-------+---------+
|   100 | JOHN    |   100 | JOHN    |
|   300 | MARY    |   200 | MARY    |
|   500 | AMBELIA |   300 | AMBELIA |
|   200 | JANE    |  NULL | NULL    |
|   400 | DAVID   |  NULL | NULL    |
+-------+---------+-------+---------+
```

c) mysql> SELECT * FROM MEMB RIGHT JOIN CMTS ON MEMB.NAME=CMTS.NAME;

```
+-------+---------+-------+---------+
| MEMID | NAME    | CMTID | NAME    |
+-------+---------+-------+---------+
|   100 | JOHN    |   100 | JOHN    |
|   300 | MARY    |   200 | MARY    |
|   500 | AMBELIA |   300 | AMBELIA |
|  NULL | NULL    |   400 | JOE     |
+-------+---------+-------+---------+
```

4 rows in set (0.00 sec)

d) mysql> SELECT * FROM MEMB CROSS JOIN CMTS ON MEMB.NAME=CMTS.NAME;

```
+-------+---------+-------+---------+
| MEMID | NAME    | CMTID | NAME    |
+-------+---------+-------+---------+
|   100 | JOHN    |   100 | JOHN    |
|   300 | MARY    |   200 | MARY    |
|   500 | AMBELIA |   300 | AMBELIA |
+-------+---------+-------+---------+
```

3 rows in set (0.00 sec)

RESULT: Queries to implement set operations, nested queries and join queries has implemented successfully and output obtained.

OUTPUT

1) mysql> CREATE TABLE STDT21(ID INT PRIMARY KEY AUTO_INCREMENT,NAME VARCHAR(15),DOB DATE,AGE INT(2));

Query OK, 0 rows affected (0.25 sec)

mysql> DESC STDT21;

```
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| ID    | int(11)     | NO   | PRI | NULL    | auto_increment |
| NAME  | varchar(15) | YES  |     | NULL    |                |
| DOB   | date        | YES  |     | NULL    |                |
| AGE   | int(2)      | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
```

4 rows in set (0.00 sec)

mysql> INSERT INTO STDT21 VALUES(1,"REKHA","2002/03/20",20),(2,"RAHUL","1998/06/01",24),(3,"ARYA","2001/08/10",21),(4,"JENITA","2001/08/23",21);

Query OK, 4 rows affected (0.03 sec)

Records: 4 Duplicates: 0 Warnings: 0

2) mysql> START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM STDT21;

```
+----+--------+------------+------+
| ID | NAME   | DOB        | AGE  |
+----+--------+------------+------+
|  1 | REKHA  | 2002-03-20 |   20 |
|  2 | RAHUL  | 1998-06-01 |   24 |
|  3 | ARYA   | 2001-08-10 |   21 |
|  4 | JENITA | 2001-08-23 |   21 |
+----+--------+------------+------+
```

4 rows in set (0.00 sec)

Date:

Experiment No:8

SQL, TCL COMMADNS LIKE ROLLBACK, COMMIT,SAVEPOINT

AIM: To implement SQL TCL commands like Rollback, Commit, Save point.

QUESTIONS

1. Create a student table
2. .Write a query to start transaction.
3. Write a query to set autocommit to 0.
4. Write a query to implement savepoints 1.
5. Update student table and set age 29 where id is 9.
6. Write a query to implement rollback to S1.
7. Write a query to implement commit.
8. Insert a value to the table.
9. Write a query rollback.

3) mysql> SET AUTOCOMMIT=0;

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM STDT21;

```
+----+--------+------------+------+
| ID | NAME   | DOB        | AGE  |
+----+--------+------------+------+
|  1 | REKHA  | 2002-03-20 |   20 |
|  2 | RAHUL  | 1998-06-01 |   24 |
|  3 | ARYA   | 2001-08-10 |   21 |
|  4 | JENITA | 2001-08-23 |   21 |
+----+--------+------------+------+
```

4 rows in set (0.00 sec)

4) mysql> SAVEPOINT S1;

Query OK, 0 rows affected (0.00 sec)

5) mysql> UPDATE STDT21 SET AGE=19 WHERE ID=3;

Query OK, 1 row affected (0.02 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM STDT21;

```
+----+--------+------------+------+
| ID | NAME   | DOB        | AGE  |
+----+--------+------------+------+
|  1 | REKHA  | 2002-03-20 |   20 |
|  2 | RAHUL  | 1998-06-01 |   24 |
|  3 | ARYA   | 2001-08-10 |   19 |
|  4 | JENITA | 2001-08-23 |   21 |
+----+--------+------------+------+
```

4 rows in set (0.00 sec)

6) mysql> ROLLBACK TO S1;

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM STDT21;

```
+----+--------+------------+------+
| ID | NAME   | DOB        | AGE  |
+----+--------+------------+------+
|  1 | REKHA  | 2002-03-20 |   20 |
|  2 | RAHUL  | 1998-06-01 |   24 |
|  3 | ARYA   | 2001-08-10 |   21 |
|  4 | JENITA | 2001-08-23 |   21 |
+----+--------+------------+------+
```

4 rows in set (0.00 sec)

7) mysql> COMMIT;

Query OK, 0 rows affected (0.05 sec)

8) mysql> INSERT INTO STDT21(NAME,DOB,AGE)
VALUES(5,"ABHI","2003/05/17",18);

Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM STDT21;

```
+----+--------+------------+------+
| ID | NAME   | DOB        | AGE  |
+----+--------+------------+------+
|  1 | REKHA  | 2002-03-20 |   20 |
|  2 | RAHUL  | 1998-06-01 |   24 |
|  3 | ARYA   | 2001-08-10 |   21 |
|  4 | JENITA | 2001-08-23 |   21 |
|  5 | ABHI   | 2003-05-17 |   18 |
+----+--------+------------+------+
```

5 rows in set (0.00 sec)

9) mysql> ROLLBACK;

mysql> SELECT * FROM STD21;

```
+----+--------+------------+------+
| ID | NAME   | DOB        | AGE  |
+----+--------+------------+------+
|  1 | REKHA  | 2002-03-20 |   20 |
|  2 | RAHUL  | 1998-06-01 |   24 |
|  3 | ARYA   | 2001-08-10 |   21 |
|  4 | JENITA | 2001-08-23 |   21 |
+----+--------+------------+------+
```

4 rows in set (0.00 sec)

RESULT: Queries to implement SQL, TCL commands has executed successfully and output obtained.

OUTPUT

1) mysql> CREATE USER 'PQR'@'LOCALHOST' IDENTIFIED BY '678';

2) mysql> GRANT CREATE ON *.* TO 'PQR'@'LOCALHOST';

Query OK, 0 rows affected (0.00 sec)

3) mysql> GRANT CREATE ON STUDENT21.* TO 'PQR'@'LOCALHOST';

Query OK, 0 rows affected (0.00 sec)

4) mysql> GRANT SELECT ON STUDENT21.* TO 'PQR'@'LOCALHOST';

Query OK, 0 rows affected (0.00 sec)

5) mysql> GRANT DROP ON *.* TO 'PQR'@'LOCALHOST';

Query OK, 0 rows affected (0.00 sec)

6) mysql> GRANT ALL PRIVILEGES ON *.* TO 'PQR'@'LOCALHOST';

Query OK, 0 rows affected (0.00 sec)

7) mysql> SHOW GRANTS FOR 'PQR'@'LOCALHOST';

```
+-----------------------------------------------------------+
| Grants for PQR@localhost                                  |
+-----------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'PQR'@'localhost'          |
| GRANT SELECT, CREATE ON `STUDENT21`.* TO 'PQR'@'localhost' |
+-----------------------------------------------------------+
```

2 rows in set (0.00 sec)

Date:

Experiment No:9

DCL COMMANDS FOR GRANTING AND REVOKING USER PRIVILEGES

AIM: To implement DCL commands for granting and revoking user privileges.

The privileges granted to a MySQL account determine which operations the account can perform.

DCL commands are used to enforce database security in a multiple user database environment. Two types of DCL commands are GRANT and REVOKE. Only Database Administrator's or owner's of the database object can provide/remove privileges on a database object.

GRANT

SQL GRANT is a command used to provide access or privileges on the database objects to the users.

SYNTAX:

GRANT privilege_name ON object_name TO {user_name |PUBLIC |role_name} [WITH GRANT OPTION];

REVOKE

The REVOKE command removes user access rights or privileges to the database objects.

8) mysql> REVOKE ALL,GRANT OPTION FROM 'PQR'@'LOCALHOST';

Query OK, 0 rows affected (0.00 sec)

mysql> SHOW GRANTS FOR 'PQR'@'LOCALHOST';

```
+----------------------------------------+
| Grants for PQR@localhost               |
+----------------------------------------+
| GRANT USAGE ON *.* TO 'PQR'@'localhost' |
+----------------------------------------+
```

1 row in set (0.00 sec)

SYNTAX:

REVOKE privilege_name ON object_name FROM {user_name |PUBLIC |role_name}

COMMANDS

QUESTIONS

1. Create a new MYSQL user account.
2. Give privileges to create a new database to a user.
3. Give privileges to a user to create new tables in a specific database.
4. Give privileges to a user to read a specific database table to a user.Give privileges to delete database to a user.
5. Give privileges to user.
6. Check the privileges for a specific user.

REVOKE

1. Revoke all privileges

RESULT: Queries to implement DCL commands has been executed successfully and output obtained.

OUTPUT

1) mysql> CREATE VIEW EMP55 AS SELECT * FROM NEST WHERE SAL>10000;

Query OK, 0 rows affected (0.07 sec)

mysql> SELECT * FROM EMP55;

```
+-------+---------+------------+----------+-------+-------+-------+------------+
| EMPNO | EMPNME  | JOB        | MANAGER  | SAL   | DEPNO | COMMN | HIERDATE
|
+-------+---------+------------+----------+-------+-------+-------+------------+
| E563  | LINSON  | ACCOUNTANT | SREEHARI | 12000 |     3 |     3 | 1999-05-04 |
| E101  | SONNET  | ANALYST    | DHANYA   | 16000 |     1 |   280 | 1996-05-25 |
| E407  | ANUPAMA | PROGRAMMER | BINITH   | 17000 |     4 |     4 | 2001-07-23 |
| E305  | ALBIN   | TECHNICIAN | SANOOT   | 15000 |     3 |   190 | 2000-08-24 |
| E457  | SAMJITH | MANAGER    | VIMAL    | 18000 |     1 |   200 | 2006-06-05 |
+-------+---------+------------+----------+-------+-------+-------+------------+
```

5 rows in set (0.00 sec)

2) mysql> INSERT INTO EMP55
VALUES("E546","AKASH","HR","RAHUL",25000,2,150,'2000-12-05'),("E654","SANJU","HECKER","KAVYA",15000,3,180,'2000-08-10');

Query OK, 2 rows affected (0.09 sec)

Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM EMP55;

```
+-------+---------+------------+----------+-------+-------+-------+------------+
| EMPNO | EMPNME  | JOB        | MANAGER  | SAL   | DEPNO | COMMN | HIERDATE
|
+-------+---------+------------+----------+-------+-------+-------+------------+
| E563  | LINSON  | ACCOUNTANT | SREEHARI | 12000 |     3 |     3 | 1999-05-04 |
| E101  | SONNET  | ANALYST    | DHANYA   | 16000 |     1 |   280 | 1996-05-25 |
| E407  | ANUPAMA | PROGRAMMER | BINITH   | 17000 |     4 |     4 | 2001-07-23 |
| E305  | ALBIN   | TECHNICIAN | SANOOT   | 15000 |     3 |   190 | 2000-08-24 |
```

Date:

Experiment No:10

SQL COMMANDS FOR CREATION OF VIEWS AND ASSERTIONS

AIM: To implement SQL commands to creation of views and assertions.

EMPLOYEE55

EMP NO	EMPNME	JOB	MANAGER	SAL	DEP NO	COM MN	HIERDATE
E234	SUBIN	CLERK	DAYAL	8000	2	75	1998-05-01
E563	LINSON	ACCOUNTANT	SREEHARI	12000	3	120	1999-05-04
E101	SONNET	ANALYST	DHANYA	16000	1	280	1996-05-25
E407	ANUPAMA	PROGRAMMER	BINITH	17000	4	180	2001-07-23
E305	ALBIN	TECHNICIAN	SANOOT	15000	3	190	2000-08-24
E497	PINTO	SUPERVISOR	PRINCE	9000	5	0	2006-07-04
E457	SAMJITH	MANAGER	VIMAL	18000	1	200	2006-06-05

QUESTIONS

1. Write a query to display the emp55 from employee55 where salary is greater than 10000.
2. Insert values into the emp55 table.
3. Write a query to update the emp55 table by setting the manager as Anoop where job as hr.
4. Write a query to display the employee name & department number from the table.
5. Write a query to display employee number, employee name who were in a department with any employee where name contains a 'I' .

| E457 | SAMJITH | MANAGER | VIMAL | 18000 | 1 | 200 | 2006-06-05 |

| E546 | AKASH | HR | RAHUL | 25000 | 2 | 150 | 2000-12-05 |

| E654 | SANJU | HECKER | KAVYA | 15000 | 3 | 180 | 2000-08-10 |

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

7 rows in set (0.00 sec)

3) mysql> UPDATE EMP55 SET MANAGER="ANOOP" WHERE JOB="HR";

Query OK, 1 row affected (0.06 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM EMP55;

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

| EMPNO | EMPNME | JOB | MANAGER | SAL | DEPNO | COMMN | HIERDATE |

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

| E563 | LINSON | ACCOUNTANT | SREEHARI | 12000 | 3 | 3 | 1999-05-04 |

| E101 | SONNET | ANALYST | DHANYA | 16000 | 1 | 280 | 1996-05-25 |

| E407 | ANUPAMA | PROGRAMMER | BINITH | 17000 | 4 | 4 | 2001-07-23 |

| E305 | ALBIN | TECHNICIAN | SANOOT | 15000 | 3 | 190 | 2000-08-24 |

| E457 | SAMJITH | MANAGER | VIMAL | 18000 | 1 | 200 | 2006-06-05 |

| E546 | AKASH | HR | ANOOP | 25000 | 2 | 150 | 2000-12-05 |

| E654 | SANJU | HECKER | KAVYA | 15000 | 3 | 180 | 2000-08-10 |

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

7 rows in set (0.00 sec)

4) mysql> SELECT EMPNME,DEPNO FROM EMP55;

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

| EMPNME | DEPNO |

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

| LINSON | 3 |

| SONNET | 1 |

| ANUPAMA | 4 |

6. Write a query to delete a row where the department number is 3.
7. Write a query to drop the table.
8. Write a query to view emp55 from employee55 where commn is less than 150 with check option.

| ALBIN | 3 |

| SAMJITH | 1 |

| AKASH | 2 |

| SANJU | 3 |

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

7 rows in set (0.00 sec)

5) mysql> SELECT EMPNME,DEPNO FROM EMP55 WHERE DEPNO IN(SELECT DEPNO FROM EMP55 WHERE EMPNME LIKE '%I%');

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

| EMPNME | DEPNO |

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

| LINSON | 3 |

| ALBIN | 3 |

| SANJU | 3 |

| SONNET | 1 |

| SAMJITH | 1 |

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

5 rows in set (0.01 sec)

6) mysql> DELETE FROM EMP55 WHERE DEPNO=3;

Query OK, 3 rows affected (0.05 sec)

mysql> SELECT * FROM EMP55;

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

| EMPNO | EMPNME | JOB | MANAGER | SAL | DEPNO | COMMN | HIERDATE |

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

| E101 | SONNET | ANALYST | DHANYA | 16000 | 1 | 280 | 1996-05-25 |

| E407 | ANUPAMA | PROGRAMMER | BINITH | 17000 | 4 | 4 | 2001-07-23 |

| E457 | SAMJITH | MANAGER | VIMAL | 18000 | 1 | 200 | 2006-06-05 |

| E546 | AKASH | HR | ANOOP | 25000 | 2 | 150 | 2000-12-05 |

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

7) mysql> DROP VIEW EMP55;

Query OK, 0 rows affected (0.00 sec)

8) mysql> CREATE VIEW EMP55 AS SELECT * FROM NEST WHERE COMMN<150 WITH CHECK OPTION;

Query OK, 0 rows affected (0.04 sec)

mysql> SELECT * FROM EMP55;

```
+-------+---------+------------+---------+-------+-------+-------+------------+
| EMPNO | EMPNME  | JOB        | MANAGER | SAL   | DEPNO | COMMN | HIERDATE   |
+-------+---------+------------+---------+-------+-------+-------+------------+
| E234  | SUBIN   | CLERK      | DAYAL   |  8000 |     2 |    75 | 1998-05-01 |
| E407  | ANUPAMA | PROGRAMMER | BINITH  | 17000 |     4 |     4 | 2001-07-23 |
| E497  | PINTO   | SUPERVISOR | PRINCE  |  9000 |     5 |     0 | 2006-07-04 |
+-------+---------+------------+---------+-------+-------+-------+------------+
```

3 rows in set (0.00 sec)

RESULT: Queries to implement SQL commands to creation of views and assertions has been executed successfully and output obtained.

INTRODUCTION TO PL/SQL

PL/SQL is a block structured language that can have multiple blocks in it. The programs of PL/SQL are logical blocks that can contain any number of nested sub-blocks. PL/SQL stands for "Procedural Language extension of SQL" that is used in Oracle. PL/SQL is integrated with Oracle database (since version 7). The functionalities of PL/SQL usually extended after each release of Oracle database. Although PL/SQL is closely integrated with SQL language, yet it adds some programming constraints that are not available in SQL.

1. Features of PL/SQL

PL/SQL has the following features –

- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It offers a variety of programming structures.
- It supports structured programming through functions and procedures.
- It supports object-oriented programming.
- It supports the development of web applications and server pages.

2. Advantages of PL/SQL

PL/SQL has the following advantages –

- SQL is the standard database language and PL/SQL is strongly integrated with SQL. PL/SQL supports both static and dynamic SQL. Static SQL supports DML operations and transaction control from PL/SQL block. In Dynamic SQL, SQL allows embedding DDL statements in PL/SQL blocks.
- PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.

- PL/SQL gives high productivity to programmers as it can query, transform, and update data in a database.
- PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.
- Applications written in PL/SQL are fully portable.
- PL/SQL provides high security level.
- PL/SQL provides access to predefined SQL packages.
- PL/SQL provides support for Object-Oriented Programming.
- PL/SQL provides support for developing Web Applications and Server Pages.

3. Basic Syntax

Basic Syntax of PL/SQL which is a block-structured language; this means that the PL/SQL programs are divided and written in logical blocks of code. Each block consists of three subparts –

S.NO	SECTION AND DESCRIPTION
1	**Declarations** This section starts with the keyword **DECLARE**. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.
2	**Executable Commands** This section is enclosed between the keywords **BEGIN** and **END** and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a **NULL** command to indicate that nothing should be executed.
3	**Exception Handling** This section starts with the keyword **EXCEPTION**. This optional section contains **exception(s)** that handle errors in the program.

Every PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks using **BEGIN** and **END**. Following is the basic structure of a PL/SQL block –

```
DECLARE
<declaration section>
BEGIN
<executable command(s)
EXCEPTION
<exception handling>
END;
```

4. PL/SQL Comments

Program comments are explanatory statements that can be included in the PL/SQL code that you write and helps anyone reading its source code. All programming languages allow some form of comments. The PL/SQL supports single-line and multi-line comments. All characters available inside any comment are ignored by the PL/SQL compiler. The PL/SQL single-line comments start with the delimiter -- (double hyphen) and multi-line comments are enclosed by /* and */.

Example:

```
DECLARE
    -- variable declaration
    message varchar2(20):= 'Hello, World!';
BEGIN
/*
```

```
* PL/SQL executable statement(s)
*/
 dbms_output.put_line(message);
END;
```

5. PL/SQL Identifiers

PL/SQL identifiers are constants, variables, exceptions, procedures, cursors, and reserved words. The identifiers consist of a letter optionally followed by more letters, numerals, dollar signs, underscores, and number signs and should not exceed 30 characters. By default**, identifiers are not case-sensitive**. So you can use **integer** or **INTEGER** to represent a numeric value. You cannot use a reserved keyword as an identifier.

Date:

Experiment No:11

IMPLEMENTATION OF VARIOUS CONTROL STRUCTURES USING PL/SQL

AIM: To Implement various control structures like IF-THEN, IF-THEN-ELSE, IF-THEN ELSIF, CASE, WHILE using PL/SQL

1. Decision-Making Statements

Decision-making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false

S.NO	STATEMENTS AND DESCRIPTION
1	**IF - THEN statement** The IF statement associates a condition with a sequence of statements enclosed by the keywords THEN and END IF. If the condition is true, the statements get executed and if the condition is false or NULL then the IF statement does nothing.
2	**IF-THEN-ELSE statement** IF statement adds the keyword ELSE followed by an alternative sequence of statement. If the condition is false or NULL, then only the alternative sequence of statements get executed. It ensures that either of the sequence of statements is executed.
3	**IF-THEN-ELSIF statement** It allows you to choose between several alternatives

4	**Case statement** Like the IF statement, the CASE statement selects one sequence of statements to execute. However, to select the sequence, the CASE statement uses a selector rather than multiple Boolean expressions. A selector is an expression whose value is used to select one of several alternatives.
5	**Searched CASE statement** The searched CASE statement has no selector, and it's WHEN clauses contain search conditions that yield Boolean values.
6	**Nested IF-THEN-ELSE** Use one IF-THEN or IF-THEN-ELSIF statement inside another IF-THEN or IF-THENELSIF statement(s)

2. Looping Statements

A loop statement allows us to execute a statement or group of statements multiple times.

S.NO	Loop Type & Description
1	**PL/SQL Basic LOOP** In this loop structure, sequence of statements is enclosed between the LOOP and the END LOOP statements. At each iteration, the sequence of statements is executed and then control resumes at the top of the loop.
2	**PL/SQL WHILE LOOP** Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
3	**PL/SQL FOR LOOP** Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable
4	**Nested loops in PL/SQL** Use one or more loop inside any another basic loop, while, or for loop.

PROGRAM

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  A NUMBER(4):=&NUM1;
  3  B NUMBER(4):=&NUM2;
  4  SUM1 NUMBER(8):=0;
  5  BEGIN
  6  SUM1:=A+B;
  7  DBMS_OUTPUT.PUT_LINE('THE SUM OF TWO NUMBER IS:'||SUM1);
  8  END;
  9  /
```

OUTPUT

```
Enter value for num1: 28
old   2: A NUMBER(4):=&NUM1;
new   2: A NUMBER(4):=28;
Enter value for num2: 22
old   3: B NUMBER(4):=&NUM2;
new   3: B NUMBER(4):=22;
THE SUM OF TWO NUMBER IS:50
PL/SQL procedure successfully completed.
```

QUESTIONS

1. Write a PL/SQL program to find the sum of two numbers

ALGORITHM

Step1: Start

Step2: Enter number A

Step3: Enter number B

Step4: Compute SUM1 = A+B

Step5: Display SUM1

Step6: End

PROGRAM (IF-THEN)

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  NUM1 NUMBER:=0;
  3  BEGIN
  4  IF NUM1 = 0 THEN
  5  DBMS_OUTPUT.PUT_LINE('THE NUMBER IS ZERO');
  6  END IF;
  7  DBMS_OUTPUT.PUT_LINE('END OF PROGRAM');
  8  END;
  9  /
```

OUTPUT

```
THE NUMBER IS ZERO
END OF PROGRAM

PL/SQL procedure successfully completed.
```

1. IF-THEN QUESTION

1. Write a PL/SQL program to display given number is zero

ALGORITHM

Step1: Start

Step2: Set NUM1=0

Step3: Check whether NUM1 is equal to 0

Step4: Display the message the number is zero

Step5: Display end of the program

Step6: End

PROGRAM (IF-THEN-ELSE)

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  A NUMBER(4):=&NUM;
  3  BEGIN
  4  IF MOD(A,2)=0 THEN
  5  DBMS_OUTPUT.PUT_LINE('THE NUMBER IS EVEN');
  6  ELSE
  7  DBMS_OUTPUT.PUT_LINE('THE NUMBER IS ODD');
  8  END IF;
  9  END;
 10  /
```

OUTPUT

```
Enter value for num: 2
old   2: A NUMBER(4):=&NUM;
new   2: A NUMBER(4):=2;
THE NUMBER IS EVEN
PL/SQL procedure successfully completed.
SQL> /
Enter value for num: 7
old   2: A NUMBER(4):=&NUM;
new   2: A NUMBER(4):=7;
THE NUMBER IS ODD
PL/SQL procedure successfully completed.
```

2. IF-THEN-ELSE QUESTIONS

1. Write a PL/SQL program to find whether a given number is odd or even

ALGORITHM

Step 1: Start

Step 2: Enter the number to check

Step 3: If number is divisible by 2 then the number is Even

Step 4: Else number is odd

Step 5: Print the output

Step 6: Stop

PROGRAM

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  A NUMBER(4):=&NUM1;
  3  B NUMBER(4):=&NUM2;
  4  C NUMBER(4):=&NUM3;
  5  LARGE NUMBER(4):=0;
  6  BEGIN
  7  IF A>B THEN
  8  LARGE:=A;
  9  ELSE
 10  LARGE:=B;
 11  END IF;
 12  IF LARGE>C THEN
 13  DBMS_OUTPUT.PUT_LINE('THE LARGEST OF THREE NUMBERS:'||LARGE);
 14  ELSE
 15  DBMS_OUTPUT.PUT_LINE('THE LARGEST OF THREE NUMBERS:'||C);
 16  END IF;
 17  END;
 18  /
```

2. Write a PL/SQL program to find the largest of three numbers

ALGORITHM

Step1: Start

Step2: Enter three numbers A, B and C

Step 3: Check if A is greater than B

3:1 If true, set A as large

3:2 If false, set B as large

Step 4: Check if large is greater than C

4:1 If true, display largest number is large

4:2 If false, display largest number is C

Step 5: Stop

OUTPUT

```
Enter value for num1: 23
old   2: A NUMBER(4):=&NUM1;
new   2: A NUMBER(4):=23;
Enter value for num2: 56
old   3: B NUMBER(4):=&NUM2;
new   3: B NUMBER(4):=56;
Enter value for num3: 87
old   4: C NUMBER(4):=&NUM3;
new   4: C NUMBER(4):=87;
THE LARGEST OF THREE NUMBERS:87


PL/SQL procedure successfully completed.
```

PROGRAM (IF-THEN-ELSEIF)

```
SQL>  SET SERVEROUTPUT ON
SQL> DECLARE
  2  SUB1 NUMBER(4):=&NUM1;
  3  SUB2 NUMBER(4):=&NUM2;
  4  SUB3 NUMBER(4):=&NUM3;
  5  TOTAL NUMBER(4):=0;
  6  PER DECIMAL(18,2):=0;
  7  BEGIN
  8  TOTAL:=SUB1+SUB2+SUB3;
  9  DBMS_OUTPUT.PUT_LINE('TOTAL='||TOTAL);
 10  PER:=(TOTAL/300)*100;
 11  DBMS_OUTPUT.PUT_LINE('PERCENTAGE='||PER);
 12  IF PER>=90 THEN
 13  DBMS_OUTPUT.PUT_LINE('GRADE:A');
 14  ELSIF PER>=80 AND PER<90 THEN
 15  DBMS_OUTPUT.PUT_LINE('GRADE:B');
 16  ELSE
 17  DBMS_OUTPUT.PUT_LINE('GRADE:C');
 19  END IF;
 19  END;
 20  /
```

3. IF-THEN ELSIF

QUESTION

1. Write a PL/SQL program to display grade list of a student

ALGORITHM

Step 1: Start

Step 2: Enter three marks

Step 3: Calculate total mark and percentage

Step 4: Check if percentage is greater than 90 then display grade as A grade

Step 5: Check if percentage is greater than 80 and less than 90 then display grade as B

Step 6: Otherwise display grade as C grade

Step 7: Stop

OUTPUT

```
Enter value for num1: 89
old   2: SUB1 NUMBER(4):=&NUM1;
new   2: SUB1 NUMBER(4):=89;
Enter value for num2: 95
old   3: SUB2 NUMBER(4):=&NUM2;
new   3: SUB2 NUMBER(4):=95;
Enter value for num3: 90
old   4: SUB3 NUMBER(4):=&NUM3;
new   4: SUB3 NUMBER(4):=90;
TOTAL=274
PERCENTAGE=91.33
GRADE:A


PL/SQL procedure successfully completed.
```

PROGRAM (CASE)

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
 2  CHOICE NUMBER;
 3  RADIUS NUMBER;
 4  AREA NUMBER;
 5  BASE NUMBER;
 6  HEIGHT NUMBER;
 7  BEGIN
 8  DBMS_OUTPUT.PUT_LINE('1.AREA OF CIRCLE');
 9  DBMS_OUTPUT.PUT_LINE('2.AREA OF TRIANGLE');
10  CHOICE:=&CHOICE;
11  CASE CHOICE
12  WHEN 1 THEN
13  RADIUS:=&RADIUS;
14  AREA:=RADIUS*RADIUS*3.14;
15  DBMS_OUTPUT.PUT_LINE('AREA OF CIRCLE IS:'||AREA);
16  WHEN 2 THEN
17  BASE:=&BASE;
18  HEIGHT:=&HEIGHT;
19  AREA:=0.5*BASE*HEIGHT;
20  DBMS_OUTPUT.PUT_LINE('AREA OF TRIANGLE IS :'||AREA);
21  ELSE
22  DBMS_OUTPUT.PUT_LINE('INVALID CHOICE');
```

4. CASE

QUESTION

1. Write a PL/SQL program to find the area of geometric shapes using CASE

ALGORITHM

Step 1: Start

Step 2: Take input for choice and then for area variables from the user

Step 3: Case 1: Circle:3.14*radius*radius Case 2: Triangle:0.5*base*height;

Step 4: Display output according to the case

Step 5: Stop

```
 23  END CASE;
 24  END;
 25  /
```

OUTPUT

```
Enter value for choice: 1
old  10: CHOICE:=&CHOICE;
new  10: CHOICE:=1;
Enter value for radius: 2
old  13: RADIUS:=&RADIUS;
new  13: RADIUS:=2;
Enter value for base: 3
old  17: BASE:=&BASE;
new  17: BASE:=3;
Enter value for height: 4
old  18: HEIGHT:=&HEIGHT;
new  18: HEIGHT:=4;
1.AREA OF CIRCLE
2.AREA OF TRIANGLE
AREA OF CIRCLE IS:12.56


PL/SQL procedure successfully completed.
```

PROGRAM (WHILE)

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  A NUMBER(5):=&NUM1;
  3  D NUMBER(5):=0;
  4  X NUMBER(5);
  5  SUM1 NUMBER(10):=0;
  6  BEGIN
  7  X:=A;
  8  WHILE A>0
  9  LOOP
 10  D:=MOD(A,10);
 11  SUM1:=SUM1+(D*D*D);
 12  A:=FLOOR(A/10);
 13  END LOOP;
 14  IF SUM1=X THEN
 15  DBMS_OUTPUT.PUT_LINE('THE NUMBER IS ARMSTRONG');
 16  ELSE
 17  DBMS_OUTPUT.PUT_LINE('THE NUMBER IS NOT ARMSRONG');
 18  END IF;
 19  END;
 20  /
```

5. WHILE

QUESTIONS

1. Write a PL/SQL program to check whether the number is Armstrong number or not

ALGORITHM

Step 1: Start

Step 2: Read a number from user

Step 3: Initialize variable sum1 and D to zero

Step 4: Set X=A

Step 5: Repeat until A>0

5:1 Calculate D=MOD(A,10)

5:2 Calculate the sum, sum1+(D*D*D)

5:3 Set A as A/10

Step 6: Check whether sum1 is equal to X

6:1 If true display number is Armstrong

6:2 If false display number is not an Armstrong

Step 7: Stop

OUTPUT

```
Enter value for num1: 153
old   2: A NUMBER(5):=&NUM1;
new   2: A NUMBER(5):=153;
THE NUMBER IS ARMSTRONG


PL/SQL procedure successfully completed.


SQL> /
Enter value for num1: 231
old   2: A NUMBER(5):=&NUM1;
new   2: A NUMBER(5):=231;
THE NUMBER IS NOT ARMSRONG


PL/SQL procedure successfully completed.
```

PROGRAM (FOR LOOP)

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  A NUMBER(4):=&LIMIT;
  3  B NUMBER(4):=1;
  4  C NUMBER:=0;
  5  BEGIN
  6  DBMS_OUTPUT.PUT_LINE('THE PRIME NUMBERS ARE:');
  7  FOR B IN 1 ..A
  8  LOOP
  9  C:=0;
 10  FOR I IN 1 ..A
 11  LOOP
 12  IF MOD(B,I)=0 THEN
 13  C:=C+1;
 14  END IF;
 15  END LOOP;
 16  IF C=2 THEN
 17  DBMS_OUTPUT.PUT_LINE(B);
 18  END IF;
 19  END LOOP;
 20  END;
 21  /
```

6. FOR LOOP

QUESTION

1. Write a PL/SQL program to display the prime numbers up to ‘n’

ALGORITHM

Step 1: Start

Step 2: Read the limit

Step 3 : Initilize variables

Step 4: Repeat loop for B=1 to limit

4:1 Repeat loop for I=1 to limit

a. Check if mod(B,I) is equal to 0 then set C as C+1

4:2 Check if C is eqaul to 2 then display prime numbers

Step 5: Stop

OUTPUT

```
Enter value for limit: 10
old   2: A NUMBER(4):=&LIMIT;
new   2: A NUMBER(4):=10;
THE PRIME NUMBERS ARE:
2
3
5
7

PL/SQL procedure successfully completed.
```

RESULT: Queries to Implement various control structures like IF-THEN, IF-THEN-ELSE, IF-THEN ELSIF, CASE, WHILE ,FOR using PL/SQL has been executed successfully and output obtained.

Date:

Experiment No:12

CREATION OF PROCEDURES, TRIGGERS AND FUNCTIONS

AIM: To create procedures, triggers and functions

1. Procedures

A **subprogram** is a program unit/module is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is that performs a particular task. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called the **calling program**.

At the schema level, subprogram is a **standalone subprogram**. It is created with the CREATE PROCEDURE or the CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement. A subprogram created inside a package is a **packaged subprogram**. It is stored in the database and can be deleted only when the package is deleted with the DROP PACKAGE statement.

PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters.

PL/SQL provides two kinds of subprograms –

- **Functions** − These subprograms return a single value; mainly used to compute and return a value.
- **Procedures** − These subprograms do not return a value directly; mainly used to perform an action.

1.1 Creating a Procedure

A procedure is created with the **CREATE OR REPLACE PROCEDURE** statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows –

CREATE [OR REPLACE] PROCEDURE procedure_name[(parameter_name [IN | OUT | IN OUT] type [, ...])]

{IS | AS}

BEGIN

< procedure_body >

END procedure_name;

Where,

- *procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows the modification of an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- *procedure-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

1.2 Deleting a Standalone Procedure

A standalone procedure is deleted with the **DROP PROCEDURE** statement. Syntax for deleting a procedure is –

DROP PROCEDURE procedure-name;

1.3 Parameter Modes in PL/SQL Subprograms

The following table lists out the parameter modes in PL/SQL subprograms –

S.NO	PARAMETER MODE & DESCRIPTION
1	**IN** An IN parameter lets you pass a value to the subprogram. It is a **read-only parameter**. Inside the subprogram, an IN parameter acts like a constant. It cannot be assigned a value. You can pass a constant, literal, initialized variable, or expression as an IN parameter. You can also initialize it to a default value; however, in that case, it is omitted from the subprogram call. **It is the default mode of parameter passing. Parameters are passed by reference.**
2	**OUT** An OUT parameter returns a value to the calling program. Inside the subprogram, an OUT parameter acts like a variable. You can change its value and reference the value after assigning it. **The actual parameter must be variable, and it is passed by value** .
3	**IN OUT** An **IN OUT** parameter passes an initial value to a subprogram and returns an updated value to the caller. It can be assigned a value and the value can be read. The actual parameter corresponding to an IN OUT formal parameter must be a variable, not a constant or an expression. Formal parameter must be assigned a value. **Actual parameter is passed by value**.

2. Function

2.1 Creating a Function

A standalone function is created using the CREATE FUNCTION statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows

CREATE [OR REPLACE] FUNCTION function_name [(parameter_name [IN | OUT | IN OUT] type [, ...])]

RETURN return_datatype

{IS | AS}

BEGIN

< function_body >

END [function_name];

Where,

- *function-name* specifies the name of the function.
- [OR REPLACE] option allows the modification of an existing function.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- The function must contain a return statement.
- The RETURN clause specifies the data type you are going to return from the function.
- *function-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

2.2 Calling a Function

While creating a function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. When a program calls a function, the program control is transferred to the called function.

A called function performs the defined task and when its return statement is executed or when the **last end statement** is reached, it returns the program control back to the main program

3. Trigger

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers can be defined on the table, view, schema, or database with which the event is associated.

3.1 Benefits of Triggers

Triggers can be written for the following purposes –

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

3.2Creating Triggers

The syntax:

```
CREATE [OR REPLACE ] TRIGGER trigger_name {BEFORE | AFTER |
INSTEAD OF } {INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name [
REFERENCING OLD AS.NEW AS n][FOR EACH ROW]
WHEN (condition)
DECLARE
        Declaration-statements
```

BEGIN

Executable-statements

EXCEPTION

Exception-handling-statements

END;

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation. •
- [OF col_name] − This specifies the column name that will be updated.
- [ON table_name] − This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers

PROGRAM

```
SQL> CREATE OR REPLACE PROCEDURE FIB(N IN NUMBER)
 2  IS
 3  A NUMBER(5):=0;
 4  B NUMBER(3):=1;
 5  C NUMBER(3);
 6  T NUMBER(5);
 7  VAL NUMBER(5):=0;
 8  BEGIN
 9  DBMS_OUTPUT.PUT_LINE('FIBONACCI SERIES IS:');
10  DBMS_OUTPUT.PUT_LINE(A);
11  DBMS_OUTPUT.PUT_LINE(B);
12  WHILE VAL!=N-2
13  LOOP
14  C:=A+B;
15  DBMS_OUTPUT.PUT_LINE(C);
16  T:=B;
17  B:=C;
18  A:=T;
19  VAL:=VAL+1;
20  END LOOP;
21  END;
22  /
```

Procedure created.

1. PROCEDURE

QUESTION

1. Write a PL/SQL program to generate Fibonacci series using procedure

ALGORITHM

Step 1: Start

Step 2 : read the limit

Step 3: Call procedure, FIB(number)

Step 4: Stop

FIB(number)

Step 1: Start

Step 2: Initialize A and VAL as zero

Step 3: Display the first two terms of series

Step 4: Repeat until VAL!= N-2

4:1 Calculate C=A+B

4:2 Display C

4:3 Perform swapping

4:4 Set Val as Val+1

Step 5: Stop

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  F NUMBER(5):=&LIMIT;
  3  BEGIN
 4  FIB(F);
  5  END;
  6  /
```

OUTPUT

```
Enter value for limit: 6
old   2: F NUMBER(5):=&LIMIT;
new   2: F NUMBER(5):=6;
FIBONACCI SERIES IS:
0
1
1
2
3
5
PL/SQL procedure successfully completed.
```

2. PROGRAM

```
SQL> CREATE OR REPLACE FUNCTION PMN(N IN NUMBER) RETURN NUMBER
 2  IS
 3  C NUMBER(5);
 4  P NUMBER(5):=0;
 5  I NUMBER(5);
 6  J NUMBER(5);
 7  BEGIN
 8  FOR I IN 2..100
 9  LOOP
10  C:=0;
11  FOR J IN 1..I
12  LOOP
13  IF MOD(I,J)=0 THEN
14  C:=C+1;
15  END IF;
16  END LOOP;
17  IF C=2 THEN
18  P:=P+1;
19  END IF;
20  IF P=N THEN
21  RETURN(I);
22  END IF;
22  END LOOP;
```

2. FUNCTION

QUESTION

1. Write a PL/SQL program to display Nth prime number using function

ALGORITHM

Step 1: Start

Step 2: Read a number, NO

Step 3: Call function PMN(NO)

Step 4: Display Nth prime number

Step 5: Stop

PMN(N)

Step 1 : Start

Step 2: Initialize P and C as Zero

Step 3: Repeat loop until I=100

3:1 Check if I%J=0 then set C=C+1

3:2 If C=2 then set P as P+1

3:3 Check if P=N then return i

Step 4: Stop

```
 24  END;
 25  /
```

Function created.

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  NO NUMBER(5):=&LIMIT;
  3  BEGIN
  4  DBMS_OUTPUT.PUT_LINE(NO||'TH PRIME NUMBER IS:'||PMN(NO));
  5  END;
  6  /
```

OUTPUT

```
Enter value for limit: 10
old   2: NO NUMBER(5):=&LIMIT;
new   2: NO NUMBER(5):=10;
10TH PRIME NUMBER IS:29
```

PL/SQL procedure successfully completed.

3. PROGRAM

1) SQL> CREATE TABLE CUSTOMER21(ID NUMBER,NAME VARCHAR(50),AGE NUMBER,CITY VARCHAR(50),DEP NUMBER,DESG VARCHAR(50),SAL NUMBER);

Table created.

SQL> INSERT INTO CUSTOMER21
VALUES(700,'REKHA',33,'ALAPPY',3,'DEVELOPER',35000);

1 row created.

SQL> INSERT INTO CUSTOMER21
VALUES(900,'ARYA',34,'PANDALAM',3,'DESIGNER',30000);

1 row created.

SQL> INSERT INTO CUSTOMER21
VALUES(1000,'JENITA',37,'KOLLAM',1,'CLERK',25000);

1 row created.

SQL> INSERT INTO CUSTOMER21
VALUES(1001,'AKASH',28,'PALA',1,'DESIGNER',27000);

1 row created.

SQL> INSERT INTO CUSTOMER21
VALUES(1002,'SANJU',39,'KOTYAM',3,'TESTER',20000);

1 row created.

SQL> INSERT INTO CUSTOMER21
VALUES(1003,'HARI',40,'TVM',1,'MANAGER',67000);

1 row created.

2)SQL> SET SERVEROUTPUT ON

SQL> CREATE OR REPLACE TRIGGER DISPLAY_SALARY_CHANGES BEFORE DELETE OR INSERT OR UPDATE ON CUSTOMER21 FOR EACH ROW

2 WHEN (NEW.ID >0)

3. TRIGGER

QUESTION

1. Create a table customer21 with the following fields.

```
----+----------+-----+-----------+----------+ -------+-----
| ID | NAME  | AGE | CITY   | DEP | DESG | SAL |
+----+----------+-----+-----------+----------+------+-----
|  700| Sethu |  33 | Wayanad| 3| Developer |  35000  |
| 900| Jaya  |  34 | Pandalam| 3| Designer| 30000 |
|1000 | Heera|  37 |  Kollam| 1|  Clerk| 25000|
|1001| Rima| 28| Pala| 1 | Designer | 27000 |
1002| Jani| 39| Kottayam| 3| Tester | 20000|
|1003 | Hari | 40| TVM| 1| Manager | 67000 |
+----+----------+-----+-----------+----------+ ------+
```

2. Write a PL/SQL program to display the salary diference of an employee in customer table using trigger

ALGORITHM

Step 1: Start

Step 2: Create a trigger for the customer21 table that fire for insert or update or delete operation to display the salary diference

Step 3: Inserting a new employee

Step 4: Modify the salary

Step 5: Calculate the salary difference

Step 6: Display the difference

Step 7 : Stop

```
 3  DECLARE
 4  SAL_DIFF NUMBER;
 5  BEGIN
 6  SAL_DIFF := :NEW.SAL - :OLD.SAL;
 7  DBMS_OUTPUT.PUT_LINE('OLD SALARY: ' || :OLD.SAL);
 8  DBMS_OUTPUT.PUT_LINE('NEW SALARY: ' || :NEW.SAL);
 9  DBMS_OUTPUT.PUT_LINE('SALARY DIFFERENCE: ' || SAL_DIFF);
10  END;
11  /
```

Trigger created.

```
SQL> INSERT INTO CUSTOMER21 VALUES('780','NEERAV',21,'TVM',2,'GM',80000);
OLD SALARY:
NEW SALARY: 80000
SALARY DIFFERENCE:
1 row created.

SQL> UPDATE CUSTOMER21 SET SAL=90000 WHERE ID='780';
OLD SALARY: 80000
NEW SALARY: 90000
SALARY DIFFERENCE: 10000
1 row updated.
```

RESULT: Procedures, triggers and functions are executed successfully and output is verified

Date:

Experiment No:13

CREATION OF PACKAGES

AIM: To create a package in PL/SQL

Packages are schema objects that groups logically related PL/SQL types, variables, and subprograms.

A package will have two mandatory parts –

- Package specification
- Package body or definition

1.Package Specification

The specification is the interface to the package. It just **DECLARES** the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package. In other words, it contains all information about the content of the package, but excludes the code for the subprograms. All objects placed in the specification are called **public** objects. Any subprogram not in the package specification but coded in the package body is called a **private** object.

2.Package Body

The package body has the codes for various methods declared in the package specification and other private declarations, which are hidden from the code outside the package. The CREATE PACKAGE BODY Statement is used for creating the package body

PROGRAM

```
SQL> CREATE PACKAGE CUSTOMER_SAL AS PROCEDURE FIND_SALARY(C_ID
CUSTOMER21.ID%TYPE);
  2  END CUSTOMER_SAL;
  3  /

Package created.

SQL> CREATE OR REPLACE PACKAGE BODY CUSTOMER_SAL AS PROCEDURE
FIND_SALARY(C_ID CUSTOMER21.ID%TYPE) IS C_SAL
CUSTOMER21.SAL%TYPE;
  2  BEGIN
  3  SELECT SAL INTO C_SAL FROM CUSTOMER21 WHERE ID=C_ID;
  4  DBMS_OUTPUT.PUT_LINE('SALARY:'||C_SAL);
  5  END FIND_SALARY;
  6  END CUSTOMER_SAL;
  7  /

Package body created.

SQL> DECLARE
  2  CODE CUSTOMER21.ID%TYPE:=&CC_ID;
  3  BEGIN
  4  CUSTOMER_SAL.FIND_SALARY(CODE);
  5  END;
  6  /
```

QUESTION

1. Create a package for the table customers

ALGORITHM

Package Specification

Step 1: Start

Step 2: Create a package and create a method for finding the salary of an employee whose id is given by the user

Step 3: Stop

Package Body

Step 1: Start

Step 2: Declaration of package body for the created package

Step 3: Check whether the entered id is equal to the id from table customer21 then display salary of the given employee

Step 4: Stop

Using the package elements

Step 1: Start

Step 2: Enter the employee id by the user

Step 3: Call the method for finding the salary using package

Step 4: Stop

OUTPUT

Enter value for cc_id: 1000

old 2: CODE CUSTOMER21.ID%TYPE:=&CC_ID;

new 2: CODE CUSTOMER21.ID%TYPE:=1000;

SALARY:25000

PL/SQL procedure successfully completed.

RESULT: PL/SQL program to create a package for a table has executed successfully and output is verified

Date:

Experiment No:14

CREATION OF CURSORS

AIM: To create a cursor in PL/SQL

A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.

There are two types of cursors –

- Implicit cursors
- Explicit cursors

1.Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

S.NO	ATTRIBUTES & DESCRIPTION
1	**%FOUND** Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.

2	**%NOTFOUND** The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE
3	**%ISOPEN** Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4	**%ROWCOUNT** Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement

2.Explicit Cursors

Explicit cursors are programmer-defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is –

CURSOR cursor_name IS select_statement;

Working with an explicit cursor includes the following steps –

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

2.1Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement.
CURSOR c_customers IS SELECT id, name, address FROM customers;

2.2Opening the Cursor

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it.

OPEN c_customers;

2.3Fetching the Cursor

Fetching the cursor involves accessing one row at a time.

FETCH c_customers INTO c_id, c_name, c_addr;

2.4Closing the Cursor

Closing the cursor means releasing the allocated memory.

CLOSE c_customers;

PROGRAM

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  CURSOR CUR IS SELECT ID,NAME,DESG,SAL FROM CUSTM25 WHERE
SAL>28000;
  3   C_ID CUSTM25.ID%TYPE;
  4  C_NAME CUSTM25.NAME%TYPE;
  5  C_DESG CUSTM25.DESG%TYPE;
  6  C_SAL CUSTM25.SAL%TYPE;
  7  BEGIN
  8  OPEN CUR;
  9  FETCH CUR INTO C_ID,C_NAME,C_DESG,C_SAL;
 10  WHILE CUR%FOUND
 11  LOOP
 12  IF C_SAL>28000 THEN
 13  DBMS_OUTPUT.PUT_LINE(C_ID||' '||C_NAME||' '||C_DESG||' '||C_SAL);
 14   END IF;
 15  FETCH CUR INTO C_ID,C_NAME,C_DESG,C_SAL;
 16  END LOOP;
 17  CLOSE CUR;
 18  END;
 19  /
```

OUTPUT

```
700 SANJU HECKER 35000
900 MANJU DESIGNER 30000
1002 RENJU MANAGER 67000
780 RINJU GM 90000

PL/SQL procedure successfully completed.
```

QUESTIONS

1. To write a PL/SQL program to display customer id, name, designation and salary of employees whose salary greater than 28000 from customer table using cursor

ALGORITHM

Step 1: Start

Step 2: Create cursor to select employee id, name, designation and salary from customer21

Step 3: Fetching the cursor into id, name, designation and salary

Step 4: Repeat the loop while data found

4:1 Check salary is greater than 28000 then display id, name, designation and salary of employee

Step 5: Stop

PROGRAM

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  CURSOR CURR2 IS SELECT ID,NAME,DEP,DESG,SAL FROM CUSTM25;
  3  CID CUSTM25.ID%TYPE;
  4  CNAME CUSTM25.NAME%TYPE;
  5  CDEP CUSTM25.DEP%TYPE;
  6  CDESG CUSTM25.DESG%TYPE;
  7  CSAL CUSTM25.SAL%TYPE;
  8  DP NUMBER:=&DID;
  9  BEGIN
 10  OPEN CURR2;
 11  FETCH CURR2 INTO CID,CNAME,CDEP,CDESG,CSAL;
 12  WHILE CURR2%FOUND
 13  LOOP
 14  IF CDEP=DP THEN
 15  DBMS_OUTPUT.PUT_LINE(CID||' '||CNAME||' '||CDESG||' '||CSAL);
 16  END IF;
 17  FETCH CURR2 INTO CID,CNAME,CDEP,CDESG,CSAL;
 18  END LOOP;
 19  CLOSE CURR2;
 20  END;
 21  /
```

2. To write a PL/SQL program to display customer id, name, designation and salary of employees whose salary greater than 28000 from customer table using cursor

ALGORITHM

Step 1: Start

Step 2: Create a cursor and select employee id, name, department, designation and salary from table customer21

Step 3: Read department number

Step 4: Fetching the cursor into employee number, name, department, designation and salary

Step 5: Repeat while data found

5:1 If department number from table is equal to department number entered by the user then display employee details

Step 6: Close the cursor

Step 7: Stop

OUTPUT

```
Enter value for did: 1
old   8: DP NUMBER:=&DID;
new   8: DP NUMBER:=1;
1000 KUNJU CLERK 25000
1001 ANJU DESIGNER 27000
1002 RENJU MANAGER 67000
780 RINJU GM 90000
PL/SQL procedure successfully completed
```

RESULT: PL/SQL program to implement cursor for a table has executed successfully and output is verified

Date:

Experiment No:15

CREATION OF PL/SQL BLOCK FOR EXCEPTION HANDLING

AIM: To create a PL/SQL block for exception handling

An exception is an error condition during a program execution. PL/SQL supports programmers to catch such conditions using EXCEPTION block in the program and an appropriate action is taken against the error condition.

There are two types of exceptions –

1. System-defined exceptions
2. User-defined exceptions

Syntax for Exception Handling

The general syntax for exception handling is as follows. The default exception will be handled using WHEN others THEN

DECLARE

<declaration section>

BEGIN

<executable command(s)>

EXCEPTION

<exception handling goes here >

WHEN exception1 THEN

exception1-handling-statements

WHEN exception2 THEN

exception2-handling-statements

WHEN exception3 THEN

exception3-handling-statements

.....

```
WHEN others THEN
        exception3-handling-statements
END;
```

2. Raising Exceptions

Exceptions are raised by the database server automatically whenever there is any internal database error, but exceptions can be raised explicitly by the programmer by using the command **RAISE**. Following is the simple syntax for raising an exception.

```
 DECLARE
exception_name EXCEPTION;
BEGIN
        IF condition THEN
                RAISE exception_name;
        END IF;
EXCEPTION
        WHEN exception_name THEN
        statement;
END;
```

PROGRAM

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  C_ID CUSTM25.ID%TYPE:=&ID;
  3  ENMAE CUSTM25.NAME%TYPE;
  4  ECITY CUSTM25.CITY%TYPE;
  5  EX_INVALID_ID EXCEPTION;
  6  BEGIN
  7  IF C_ID<=0 THEN
  8  RAISE EX_INVALID_ID;
  9  ELSE
 10  SELECT NAME,CITY INTO ENMAE,ECITY FROM CUSTM25 WHERE ID = C_ID;
 11   DBMS_OUTPUT.PUT_LINE('NAME:'||ENMAE);
 12  DBMS_OUTPUT.PUT_LINE('ADDRESS:'||ECITY);
 13  END IF;
 14  EXCEPTION
 15  WHEN EX_INVALID_ID THEN
 16  DBMS_OUTPUT.PUT_LINE('ID MUST BE GREATER THAN ZERO!');
 17  WHEN NO_DATA_FOUND THEN
 18  DBMS_OUTPUT.PUT_LINE('NO SUCH CUSTOMER!');
 19  WHEN OTHERS THEN
 20  DBMS_OUTPUT.PUT_LINE('ERROR');
 21  END;
 22  /
```

OUTPUT

```
Enter value for id: 1002
old   2: C_ID CUSTM25.ID%TYPE:=&ID;
new   2: C_ID CUSTM25.ID%TYPE:=1002;
NAME:RENJU
ADDRESS:TVM
PL/SQL procedure successfully completed.
```

QUESTIONS

1. Write a PL/SQL program to implement exception handling

ALGORITHM

Step 1: Start

Step 2: Read employee id from user

Step 3: Check employee id

3:1 If it is in table customer21 then display employee name and address

3:2 If employee id is not in table customer21 then raise no found data exception

3:3 If employee id is less than zero raise invalid exception

Step 4: Stop

```
SQL> /
Enter value for id: -9
old   2: C_ID CUSTM25.ID%TYPE:=&ID;
new   2: C_ID CUSTM25.ID%TYPE:=-9;
ID MUST BE GREATER THAN ZERO!

PL/SQL procedure successfully completed.

SQL> /
Enter value for id: 5000
old   2: C_ID CUSTM25.ID%TYPE:=&ID;
new   2: C_ID CUSTM25.ID%TYPE:=5000;
NO SUCH CUSTOMER!

PL/SQL procedure successfully completed.
```

PROGRAM

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  A INT:=&NUM1;
  3  B INT:=&NUM2;
  4  ANSWER INT;
  5  BEGIN
  6  ANSWER:=A/B;
  7  DBMS_OUTPUT.PUT_LINE('THE RESULT AFTER DIVISION IS:'||ANSWER);
  8  EXCEPTION
  9  WHEN ZERO_DIVIDE THEN
 10  DBMS_OUTPUT.PUT_LINE('DIVIDING BY ZERO PLESE CHECK THE VALUE
AGAIN');
 11  DBMS_OUTPUT.PUT_LINE('THE VALUE OF A IS '||A);
 12  DBMS_OUTPUT.PUT_LINE('THE VALUE OF B IS '||B);
 13  END;
 14  /
```

OUTPUT

```
Enter value for num1: 25
old   2: A INT:=&NUM1;
new   2: A INT:=25;
Enter value for num2: 5
old   3: B INT:=&NUM2;
new   3: B INT:=5;
THE RESULT AFTER DIVISION IS:5

PL/SQL procedure successfully completed.
```

2. Write a PL/SQL program to raise an exception when dividing with zero

ALGORITHM

Step 1: Start

Step 2: Enter two numbers

Step 3: Perform division operation

Step 4: Display the result

Step 5: Raise exception when dividing with zero

Step 6: Stop

```
SQL> /

Enter value for num1: 25

old   2: A INT:=&NUM1;

new   2: A INT:=25;

Enter value for num2: 0

old   3: B INT:=&NUM2;

new   3: B INT:=0;

DIVIDING BY ZERO PLEASE CHECK THE VALUES AGAIN

THE VALUE OF A IS25

THE VALUE OF B IS0


PL/SQL procedure successfully completed.
```

RESULT: PL/SQL program for exception handling has executed successfully and output is verified