JSPM BSIOTR

Database Management Systems Lab Manual (310246): TE Computer Engineering

Savitribai Phule Pune University

Academic Year 2025-26
SEM-I

Department of Computer Engineering

TE (2019 Course)

Prepared By:

Prof. Poulami Das

(Assistant Professor, Department of Computer Engineering)

Savitribai Phule Pune University
Third Year of Computer Engineering (2019 Course)
Home
310246:Database Management Systems Laboratory

Teaching Scheme	Credit Scheme: 02	Examination Scheme and Marks
Practical: 04 Hours/Week		Term work: 25 Marks Practical: 25 Marks

Companion Course: Database Management Systems (310241)

Course Objectives:

- To develop Database programming skills
- To develop basic Database administration skills
- To develop skills to handle NoSQL database
- To learn, understand and execute process of software application development

Course Outcomes:

On completion of the course, learners will be able to

CO1: Design E-R Model for given requirements and convert the same into database tables

CO2: Design schema in appropriate normal form considering actual requirements

CO3: Implement SQL queries for given requirements , using different SQL concepts

CO4: Implement PL/SQL Code block for given requirements

CO5: Implement NoSQL queries using MongoDB

CO6: Design and develop application considering actual requirements and using database concepts

Guidelines for Instructor's Manual

The instructor‘s manual is to be developed as a reference and hands-on resource. It should include prologue (about University/program/ institute/ department/foreword/ preface), curriculum of the course, conduction and Assessment guidelines, topics under consideration, concept, objectives, outcomes, set of typical applications/assignments/ guidelines, and references.

Guidelines for Student's Laboratory Journal

The laboratory assignments are to be submitted by student in the form of journal. Journal consists of Certificate, table of contents, and handwritten write-up of each assignment (Title, Date of Completion, Objectives, Problem Statement, Software and Hardware requirements, Assessment grade/marks and assessor's sign, Theory- Concept in brief, algorithm, flowchart, test cases, Test Data Set(if applicable), mathematical model (if applicable), conclusion/analysis. Program codes with sample output of all performed assignments are to be submitted as softcopy. As a conscious effort and little contribution towards Green IT and environment awareness, attaching printed papers as part of write-ups and program listing to journal must be avoided. Use of DVD containing students programs maintained by Laboratory In-charge is highly encouraged. For reference one or two journals may be maintained with program prints in the Laboratory.

Guidelines for Laboratory /Term Work Assessment

Continuous assessment of laboratory work should be based on overall performance of Laboratory assignments by a student. Each Laboratory assignment assessment will assign grade/marks based on parameters, such as timely completion, performance, innovation, efficient codes, and punctuality.

Guidelines for Practical Examination

Problem statements must be decided jointly by the internal examiner and external examiner. During practical assessment, maximum weightage should be given to satisfactory implementation of the problem statement. Relevant questions may be asked at the time of evaluation to test the student’s understanding of the fundamentals, effective and efficient implementation. This will encourage, transparent evaluation and fair approach, and hence will not create any uncertainty or doubt in the minds of the students. So, adhering to these principles will consummate our team efforts to the promising start of student's academics.

Guidelines for Laboratory Conduction

The instructor is expected to frame the assignments by understanding the prerequisites, technological aspects, utility and recent trends related to the topic. The assignment framing policy need to address the average students and inclusive of an element to attract and promote the intelligent students. Use of open source software is encouraged. Based on the concepts learned. Instructor may also set one assignment or mini-project that is suitable to respective branch beyond the scope of syllabus.

Operating System recommended :- 64-bit Open source Linux or its derivative

Programming tools recommended: - MYSQL/Oracle, MongoDB, ERD plus, ER Win

Virtual Laboratory:

- http://vlabs.iitb.ac.in/vlabs-dev/labs/dblab/labs/index.php

Suggested List of Laboratory Experiments/Assignments

Assignments from all Groups (A, B, C) are compulsory

Sr. No.	Group A: SQL and PL/SQL
1.	ER Modeling and Normalization: Decide a case study related to real time application in group of 2-3 students and formulate a problem statement for application to be developed. Propose a Conceptual Design using ER features using tools like ERD plus, ER Win etc. (Identifying entities, relationships between entities, attributes, keys, cardinalities, generalization, specialization etc.) Convert the ER diagram into relational tables and normalize Relational data model. Note: Student groups are required to continue same problem statement throughout all the assignments in order to design and develop an application as a part Mini Project. Further assignments will be useful for students to develop a backend for system. To design front end interface students should use the different concepts learnt in the other subjects also.
2.	SQL Queries: a. Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym, different constraints etc. b. Write at least 10 SQL queries on the suitable database application using SQL DML statements. Note: Instructor will design the queries which demonstrate the use of concepts like Insert, Select, Update, Delete with operators, functions, and set operator etc.
3.	SQL Queries - all types of Join, Sub-Query and View: Write at least 10 SQL queries for suitable database application using SQL DML statements. Note: Instructor will design the queries which demonstrate the use of concepts like all types of Join, Sub-Query and View

4.	Unnamed PL/SQL code block: Use of Control structure and Exception handling is mandatory. Suggested Problem statement: Consider Tables: 1. Borrower(Roll_no, Name, DateofIssue, NameofBook, Status) 2. Fine(Roll_no,Date,Amt) • Accept Roll_no and NameofBook from user. • Check the number of days (from date of issue). • If days are between 15 to 30 then fine amount will be Rs 5per day. • If no. of days>30, per day fine will be Rs 50 per day and for days less than 30, Rs. 5 per day. • After submitting the book, status will change from I to R. • If condition of fine is true, then details will be stored into fine table. • Also handles the exception by named exception handler or user define exception handler.
5.	Write a PL/SQL code block to calculate the area of a circle for a value of radius varying from 5 to 9. Store the radius and the corresponding values of calculated area in an empty table named areas, consisting of two columns, radius and area. Note: Instructor will frame the problem statement for writing PL/SQL block in line with above statement.
6.	Named PL/SQL Block: PL/SQL Stored Procedure and Stored Function. Write a Stored Procedure namely proc_Grade for the categorization of student. If marks scoredby students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and 900 category is first class, if marks 899 and 825 category is Higher Second Class. Write a PL/SQL block to use procedure created with above requirement. Stud_Marks(name, total_marks) Result(Roll,Name, Class) Note: Instructor will frame the problem statement for writing stored procedure and Function in line with above statement.
7.	Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor) Write a PL/SQL block of code using parameterized Cursor that will merge the data availablein the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped. Note: Instructor will frame the problem statement for writing PL/SQL block using all types of Cursors in line with above statement.

8.	Database Trigger (All Types: Row level and Statement level triggers, Before and After Triggers). Write a database trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library_Audit table. Note: Instructor will Frame the problem statement for writing PL/SQL block for all types of Triggers in line with above statement.
9.	Database Connectivity: Write a program to implement MySQL/Oracle database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc.)
	Group B: NoSQL Databases
10.	MongoDB Queries: Design and Develop MongoDB Queries using CRUD operations. (Use CRUD operations, SAVE method, logical operators etc.).
11.	MongoDB - Aggregation and Indexing: Design and Develop MongoDB Queries using aggregation and indexing with suitable example using MongoDB.
12.	MongoDB - Map reduces operations: Implement Map reduces operation with suitable example using MongoDB.
13.	Database Connectivity: Write a program to implement MongoDB database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc.)
	Group C: Mini Project
1.	Using the database concepts covered in Group A and Group B, develop an application with following details: 1. Follow the same problem statement decided in Assignment -1 of Group A. 2. Follow the Software Development Life cycle and other concepts learnt in Software Engineering Course throughout the implementation. 3. Develop application considering: • Front End : Java/Perl/PHP/Python/Ruby/.net/any other language • Backend : MongoDB/MySQL/Oracle 4. Test and validate application using Manual/Automation testing. 5. Student should develop application in group of 2-3 students and submit the Project Report which will consist of documentation related to different phases of Software Development Life Cycle: • Title of the Project, Abstract, Introduction • Software Requirement Specification • Conceptual Design using ER features, Relational Model in appropriate Normalize form • Graphical User Interface, Source Code • Testing document • Conclusion. Note: • Instructor should maintain progress report of mini project throughout the semester from project group • Practical examination will be on assignments given above in Group A and Group B only • Mini Project in this course should facilitate the Project Based Learning among students

@The_CO-PO_Mapping_Matrix												
PO/CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	-	1	3	-	3	1	1	1	3	1	-	1
CO2	2	2	3	-	2	-	1	-	3	-	1	-
CO3	-	1	2	-	2	1	-	1	3	-	-	2
CO4	-	1	2	-	2	-	-	-	3	2	1	-
CO5	-	1	2	-	2	-	2	-	3	1	-	1
CO6	2	2	3	-	3	1	-	-	3	-	2	1

Contents

.

.

ASSIGNMENT NO: 1

Title : ER Modeling and Normalization:

Decide a case study related to real time application in group of 2-3 students and formulate a problem statement for application to be developed. Propose a Conceptual Design using ER features using tools like ERD plus, ER Win etc. (Identifying entities, relationships between entities, attributes, keys, cardinalities, generalization, specialization etc.) Convert the ER diagram into relational tables and normalize Relational data model.

Objectives: To learn and understand ER model with ER diagram and normalization.

Outcome : After completion of this Assignment students will be able to

CO1 : Design E-R Model for given requirements and convert the same into database tables

CO2 : Design schema in appropriate normal form considering actual requirements

Theory:

- **ER Model:**

• ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.

• It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.

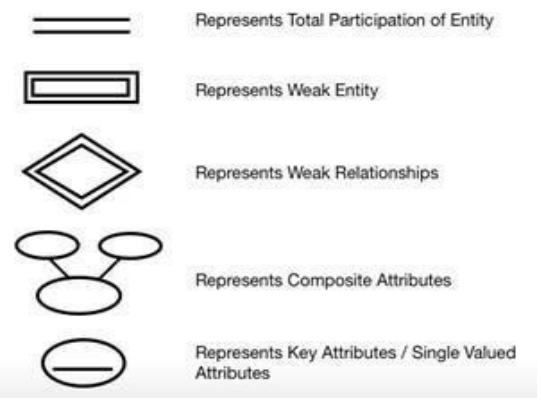
• In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

- **ER Diagram:**

• The pictorial representation of data using different conventions which state that how these data are related to each other is known as entity relationship diagram.

• E-R diagram represents the logical structure of the database in graphical manner.

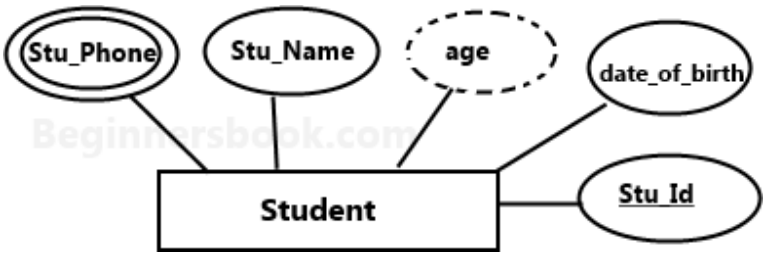
• Special symbols are used in E-R diagram which has it's own meaning.



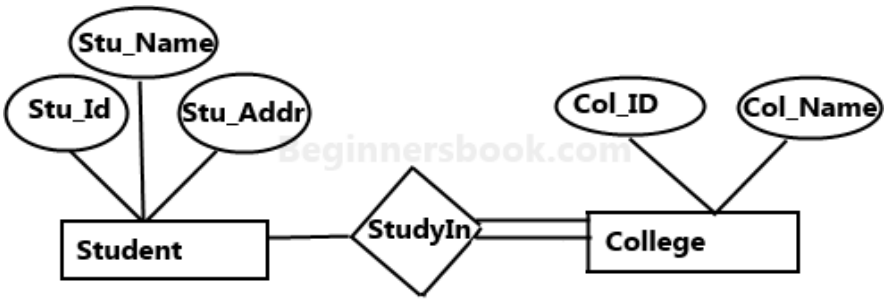
- **Points to remember while drawing E-R diagram.**
  - Initially identify all entities and their relationships with each other.
  - No entity should be repeated in a particular diagram.
  - Provide precise and appropriate name for each entity, attribute and relationship in the diagram. Give user friendly names to attributes, entities etc, which should be unique meaningful and easily understandable.
  - Do not set unclear, redundant or unnecessary relationships between entities.
  - Never connect a relationship to another relationship.

- **E-R diagram examples :**

- E-R diagram with multi-valued and derived attributes :



- E-R diagram for total participation for an entity :



E-R Digram with total participation of College entity set in StudyIn relationship Set - This indicates that each college must have atleast one associated Student.

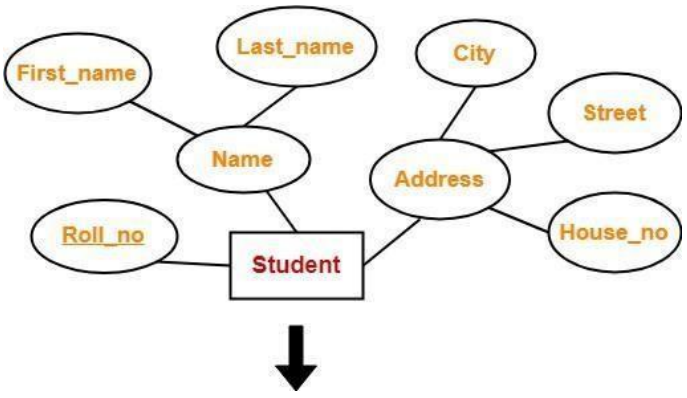
- **Converting E-R & EER diagram into tables**
  - ER diagram gives us good knowledge of entities and relations. We can understand various mapping cardinalities from ER-Diagram.
  - We can convert the ER diagram into tables by following steps:

Rule-01: For Strong Entity Set With Only Simple Attributes :

- A strong entity set with only simple attributes will require only one table in relational model.
- Attributes of the table will be the attributes of the entity set.
- The primary key of the table will be the key attribute of the entity set.
- For above entity student, table will be student and attributes will be Roll_no and Name.

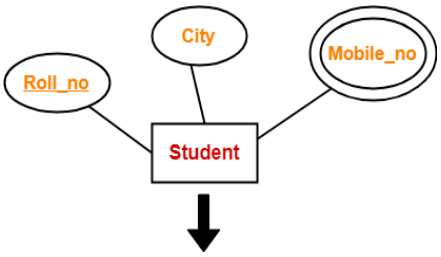
Rule-02: For Strong Entity Set With Composite Attributes-

- A strong entity set with any number of composite attributes will require only one table in relational model.
- While conversion, simple attributes of the composite attributes are taken into account and not the composite attribute itself.



Roll_no	First_name	Last_name	House_no	Street	City

- **Rule-03: For Strong Entity Set With Multi Valued Attributes:**
  - A strong entity set with any number of multi valued attributes will require two tables in relational model. One table will contain all the simple attributes with the primary key.
  - Other table will contain the primary key and all the multi valued attributes.



Roll_no	City

Roll_no	Mobile_no

- **Rule-04: Translating Relationship Set into a Table-**
  - A relationship set will require one table in the relational model. Attributes of the table are-
  - Primary key attributes of the participating entity sets
  - Its own descriptive attributes if any.
  - Set of non-descriptive attributes will be the primary key.
  - If we consider the overall ER diagram, three tables will be required in relational model-
  - One table for the entity set “Employee”
  - One table for the entity set “Department”
  - One table for the relationship set “Works in”

Normalization:

Normalization is the process of minimizing **redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updation anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

Conclusion: Hence we have successful implementation assignment on ER Modeling and Normalization, we have achieved CO1 and CO2

ASSIGNMENT NO: 2

Title: SQL Queries:

a. Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Index, Sequence, Synonym, different constraints etc.

b. Write at least 10 SQL queries on the suitable database application using SQL DML statements

Objectives:

1. To learn and SQL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym
2. To understand the concept of DML statement like Insert, Select, Update, operators and set operator.

Outcome : After completion of this Assignment students will be able to
CO3: Implement SQL queries for given requirements , using different SQL concepts

Theory:

Data Definition Language: DDL (Data Definition Language) statements are used to create, delete, or change the objects of a database. Typically a database administrator is responsible for using DDL statements or production databases in a large database system. The commands used are

- Create - It is used to create a table.
- Alter - This command is used to add a new column, modify the existing column definition and to include or drop integrity constraint.
- Drop - It will delete the table structure provided the table should be empty.
- Truncate - If there is no further use of records stored in a table and the structure has to be retained, and then the records alone can be deleted.
- Describe - This is used to view the structure of the table

1) Create table command :

Syntax

CREATE TABLE table_name
(column_name1 data_type(size), column_name2 data_type(size),.....)

Example 1

This example demonstrates how you can create a table named "Person", with four columns. The column names will be "LastName", "FirstName", "Address", and "Age":

```
CREATE TABLE Person
( LastName varchar,
  FirstName varchar,
  Address varchar,
```

```
Age int  )
```

This example demonstrates how you can specify a maximum length for some columns:

Example 2

```
CREATE TABLE Person
(
LastName varchar(30),
FirstName varchar,
Address varchar,
Age int(3)
)
```

Creating table from another (existing table) table:

Syntax

```
CREATE TABLE tablename
[(columnname,columnaname)]]
AS SELECT columnname,columnaname
FROM tablename;
```

Alter table command:

Once table is created within a database, we may wish to modify the definition of that table.The ALTER command allows to make changes to the structure of a table without deleting and recreating it.

Let's begin with creation of a table called **testalter_tbl.**

```
mysql> create table test alter_tbl
   -> (
   -> i INT,
   -> c CHAR(1)
   -> );
Query OK, 0 rows affected (0.05 sec)
```

Dropping, Adding or Repositioning a Column:

- Drop an existing column **i** from above MySQL table then you will use **DROP** clause along with **ALTER** command as follows:

```
mysql> ALTER TABLE testalter_tbl  DROP i;
```

- To add a column, use ADD and specify the column definition. The following statement restores the **i**column to testalter_tbl:

```
mysql> ALTER TABLE testalter_tbl ADD i INT;
```

- To add column at a specific position within the table, either use FIRST to make it the first column or AFTER col_name to indicate that the new column should be placed after col_name.

```
ALTER TABLE testalter_tbl DROP i;
ALTER TABLE testalter_tbl ADD i INT FIRST;
ALTER TABLE testalter_tbl DROP i;
```

```
ALTER TABLE testalter_tbl ADD i INT AFTER c;
```

The FIRST and AFTER specifiers work only with the ADD clause.

- **Changing a Column Definition or Name:**

To change a column's definition, use MODIFY or CHANGE clause along with ALTER command. For example, to change column c from CHAR(1) to CHAR(10), do this:

```
mysql> ALTER TABLE testalter_tbl MODIFY c CHAR(10);
```

- **Changing a Column's Default Value**:

You can change a default value for any column using ALTER command. Try out the following example.

```
mysql> ALTER TABLE testalter_tbl ALTER i SET DEFAULT 1000;
mysql> SHOW COLUMNS FROM testalter_tbl;
+-------+---------+------+-----+---------+ ------ +
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+ ------ +
| c     | char(1) | YES  |     | NULL    |        |
| i     | int(11) | YES  |     | 1000    |        |
+-------+---------+------+-----+---------+ ------ +
2 rows in set (0.00 sec)
```

- **Renaming a Table:**

To rename a table, use the **RENAME** option of the ALTER TABLE statement. Try out the following example to rename testalter_tbl to alter_tbl.

```
mysql> ALTER TABLE testalter_tbl RENAME TO alter_tbl;
```

3. Drop table command :

DROP command allows us to remove entire database objects from our DBMS. For example, if we want to permanently remove the personal_info table that we created, we'd use the following command:

Syntax

DROP TABLE table_name;

Example

```
DROP TABLE personal_info;
```

Example

4. MySQL CREATE Views

A view is a virtual table.View is a data object which does not contain any data. Contents of the view are the resultant of a base table. They are operated just like base table but they don't contain any data of their own.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

- MYSQL CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

View name : Each view is associated with a specific database therefore you can have database name prefix with the view name.

The CREATE VIEW statement creates a new view, or replaces an existing one if the OR REPLACE clause is given. This statement was added in MySQL 5.0.1. If the view does not exist, CREATE OR REPLACE VIEW is the same as CREATE VIEW. If the view does exist, CREATE OR REPLACE VIEW is the same as ALTER VIEW.
The select_statement is a SELECT statement that provides the definition of the view.select_statement can select from base tables or other views.

The view definition is "frozen" at creation time, so changes to the underlying tables afterward do not affect the view definition. For example, if a view is defined as **SELECT *** on a table, new columns added to the table later do not become part of the view.

```
mysql> CREATE VIEW test.v AS SELECT * FROM t;
```

A view can be created from many kinds of **SELECT** statements. It can refer to base tables or other views. It can use joins, **UNION**, and sub queries. The **SELECT** need not even refer to any tables. The following example defines a view that selects two columns from another table, as well as an expression calculated from those columns

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    3 |    50 |   150 |
+------+-------+-------+
```

5. MySQL CREATE INDEX

MySQL, index can be created on a table when the table is created with CREATE TABLE command. Otherwise, CREATE INDEX enables to add indexes to existing tables. A multiple-column index can be created using multiple columns.

The indexes are formed by concatenating the values of the given columns.

CREATE INDEX cannot be used to create a PRIMARY KEY.

Automatically: - A unique index is created automatically when you define a primary key or unique key constraint in a table definition.

Manually: - users can create non unique indexes or columns to speed up access time to the rows.

Syntax:-

Syntax CREATE INDEX [index name] ON [table name] ([column name]);

Syntax:

Create index<index_name> On table (column[, column]…);

ic.table_name='emp';

Removing an Index

Syntax:-

Drop index <index_name>;

eg. Drop index emp_name_idx;

Note:

1) We cannot modify indexes.

2) To change an index, we must drop it and the re-create it.

6. My SQL Sequence

A sequence is a set of integers 1, 2, 3, ... that are generated in order on demand. Sequences are frequently used in databases because many applications require each row in a table to contain a unique value and sequences provide an easy way to generate them.

Using AUTO_INCREMENT column:

The simplest way in MySQL to use Sequences is to define a column as AUTO_INCREMENT and leave rest of the things to MySQL to take care.

Example:

Try out the following example. This will create table and after that it will insert few rows in this table where it is not required to give record ID because it's auto incremented by MySQL.

```
mysql> CREATE TABLE insect
   -> (
   -> id INT UNSIGNED NOT NULL AUTO_INCREMENT,
   -> PRIMARY KEY (id),
   -> name VARCHAR(30) NOT NULL, # type of insect
   -> date DATE NOT NULL, # date collected
   -> origin VARCHAR(30) NOT NULL # where collected
);
Query OK, 0 rows affected (0.02 sec)
mysql> INSERT INTO insect (id,name,date,origin) VALUES
   -> (NULL,'housefly','2001-09-10','kitchen'),
   -> (NULL,'millipede','2001-09-10','driveway'),
   -> (NULL,'grasshopper','2001-09-10','front yard');
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM insect ORDER BY id;
+----+-------------+------------+ ---------+
| id | name        | date       | origin    |
+----+-------------+------------+ ---------+
|  1 | housefly    | 2001-09-10 | kitchen    |
```

```
| 2 | millipede   | 2001-09-10 | driveway   |
| 3 | grasshopper | 2001-09-10 | front yard |
+----+-------------+------------+ - - - - - - - - - +
3 rows in set (0.00 sec)
```

7. My SQL Synonym

A *synonym* is merely another name for a table or a view. Synonyms are usually created so that a user can avoid having to qualify another user's table or view to access the table or view. Synonyms can be created as PUBLIC or PRIVATE. A PUBLIC synonym can be used by any user of the database; a PRIVATE synonym can be used only by the owner and any users that have been granted privileges.

Creating Synonyms

The general syntax to create a synonym is as follows:

CREATE [PUBLIC|PRIVATE] SYNONYM *SYNONYM_NAME* FOR TABLE|VIEW

You create a synonym called CUST, short for CUSTOMER_TBL, in the following example. This frees you from having to spell out the full table name.

CREATE SYNONYM CUST FOR CUSTOMER_TBL;
SELECT CUST_NAME FROM CUST;

Dropping Synonyms

Dropping synonyms is like dropping most any other database object. The general syntax to drop a synonym is as follows:

DROP [PUBLIC|PRIVATE] SYNONYM *SYNONYM_NAME*
DROP SYNONYM CUST;

8. Constraints

Both the Create table & Alter Table SQL can be used to write SQL sentences that attach constraints. Basically constraints are of three types(Domain, Entegrity and Referential Constraints)

SQL Constraint	Function
NOT NULL	It ensures that a column does not accept NULL values.
CHECK	It ensures that a column accepts values within the specified range of values.
UNIQUE	It ensures that a column does not accept duplicate values.
PRIMARY KEY	It uniquely identifies a row in the table. It is a combination of NOT NULL and UNIQUE constraints.
FOREIGN KEY	It is like a primary key constraint only. But it uniquely identifies a row in another table.
DEFAULT	It ensures that the column sets a default value for empty records.

- **Domain**
- **Not Null Constraint:.**

Syntax:

CREATE TABLE tableName
(ColumnName1 datatype(size) NOT NULL,

ColumnName2 datatype(size) NOT NULL);
After the table creation
ALTER TABLE tableName Modify ColumnName datatype(size) NOT NULL
Example

```
CREATE TABLE students (
student_ID int NOT NULL,
student_Name varchar(255) NOT NULL,
class_name varchar(255) NOT NULL,
Age int
);
```

- **Unique Constraint:**

The unique column constraint permits multiple entries of NULL into the column. Unique key not allowed duplicate values**.** Unique index is automatically created. Table can have more than one unique key.

UNIQUE constraint defined at column level

Syntax:
Create table tablename(<columnname> <datatype>(<Size> UNIQUE),<columnname> datatype(<size>)……);

UNIQUE constraint defined at table level

Syntax:
CREATE TABLE tablename (<columnname> <datatype>(<Size>), <columnname> <datatype>(<Size>), UNIQUE(<columnname>, <columnname>));

Example :

```
CREATE TABLE students (
student_ID int UNIQUE,
student_Name varchar(255) NOT NULL,
class_name varchar(255) NOT NULL,
Age int
);
```

- **Check Constraint**:

Syntax :

```
CREATE TABLE <tableName>
(<ColumnName> datatype(size),
<ColumnName> datatype(size),....,
CHECK (columnName condition));
```

Example:

Example :

```
CREATE TABLE students (
student_ID int NOT NULL,
student_Name varchar(255) NOT NULL,
class_name varchar(255) NOT NULL,
Age int
CHECK(Age >9)
);
```

- **Primary Key Constraint :**

Syntax:

```
CREATE TABLE <TableName>
(<ColumnName1 > <DataType>(<Size>)PRIMARY KEY,
```

<columnname2 <datatype(<size>),
…..);

Primary key constraint defined at Table level

Syntax:
CREATE TABLE <TableName>
(<ColumnName1> <DataType> (<Size>) ,
…,
PRIMARY KEY(<ColumnName1> <ColumnName2>));

Example :
CREATE TABLE Students(
Student_ID int NOT NULL,
Student_Name varchar(255) NOT NULL,
Class_Name varchar(255),
Age int,
PRIMARY KEY (Student_ID)
);

- **Referential Constraint**

Foreign key: Foreign key represents relationships between tables. A foreign key is a column (or group of columns) whose values are derived from primary key or unique key of some other table.
Foreign key constraint defined at column level
Syntax: create table <table name>(
<column Name> <Data type> (<size>),
<columnName> <DataType> (<size>) REFERENCES <TableName>[(<ColumnName>)] [ON DELETE CASCADE]

Example :
CREATE TABLE Students(
Student_ID int NOT NULL,
Student_Name varchar(255) NOT NULL,
Class_Name varchar(255),
Age int,
PRIMARY KEY (Student_ID)
FOREIGN KEY (Class_Name) References classes(Class_Name)
);

9. Default Constraint

Syntax :
CREATE TABLE table_name(
column_name_1 datatype NOT NULL,
column_name_2 datatype DEFAULT 'default_value',
.
.
);

Example :
CREATE TABLE Students(
Student_ID int NOT NULL,
Student_Name varchar(255) NOT NULL,

```
Class_Name varchar(255) DEFAULT 'IV'
);
```

Data Manipulation Language (DML):

After the database structure is defined with DDL, database administrators and users can utilize the Data Manipulation Language to insert, retrieve and modify the data contained within it.

- **INSERT COMMAND:**

The INSERT command in MYSQL is used to add records to an existing table.

Format 1:-Inserting a single row of data into a table

Syntax

INSERT INTO table_name
[(columnname,columnname)]
VALUES (expression,expression);

To add a new employee to the personal_info table

Example

```
INSERT  INTO  personal_info values('bart','simpson',12345,$45000)
```

Format 2 : Inserting data into a table from another table

Syntax

INSERT INTO tablename
SELECT columnname,columnname
FROM tablename

- **SELECT COMMAND:**

Syntax

SELECT * FROM tablename.
OR
SELECT columnname,columnname,…..
FROM tablename ;

- **UPDATE COMMAND:**

The UPDATE command can be used to modify information contained within a table.

Syntax UPDATE tablename
SET columnname=expression,columnname=expression,…..
WHERE columnname=expression;

Each year,company gives all employees a 3% cost-of-living increase in their salary. The following SQL command could be used to quickly apply this to all of the employees stored in the database:

Example

```
UPDATE personal_info SET salary=salary*1.03
```

- **DELETE COMMAND :**

The DELETE command can be used to delete information contained within a table.

Syntax

DELETE FROM tablename
WHERE search condition

The DELETE command with a WHERE clause can be used to remove his record from the personal_info table:

DELETE FROM personal_info WHERE employee_id=12345

The following command deletes all the rows from the table

Example DELETE FROM personal_info;

➢ **Selecting distinct rows:** To prevent the selection of duplicate rows, we include distinct clause in the select Command. For example,
Syntax: select distinct name from student

➢ **Aggregate Functions:**
Aggregate functions are statistical functions such as count, min, max etc. They are used to compute a single value from a set of attribute values of a column: The SUM and AVG must be a collection of numbers, but the other operators can operate on collections of nonnumeric data types such as string.

 AVG:Itis used to find out the average value.
Syntax: avg(column_name)
 SUM:It is used to find out the total or sum.
Syntax: sum(column_name)
 MIN:It is used to find out the minimum value.

Syntax: min(column_name)
 MAX:It is used to find out the average value.

COUNT:It is used to find out the total number of record or rows present in the relation or table.
Syntax: max(column_name)

 Count: It is used to eliminate the duplicate and null values in the specified column.
Syntax: count(Distinct column_name)

➢ **Ordering operation:** using ORDER BY: The order by cause is used to arrange the records in ascending or descending order
Syntax:Select<expr> from <table_name>[where condition(s)][order by{column,expr}[asc|desc]];

- Asc : orders the rows in ascending order.by default order are asc.
- Desc : orders the rows in descending order.

```
SELECT * FROM members ORDER BY date_of_birth DESC;
SELECT * FROM `members` ORDER BY `gender`;
```

Conclusion: we have demonstrated the use of concepts like Insert, Select, Update, Delete with operators, functions, and set operator etc. mapped with CO3

ASSIGNMENT NO: 3

Title: SQL Queries - all types of Join, Sub-Query and View:

Write at least 10 SQL queries for suitable database application using SQL DML statements.

Objective: To understand the concept of DML statement SQL DML statements: all types of Join, Sub-Query and View.

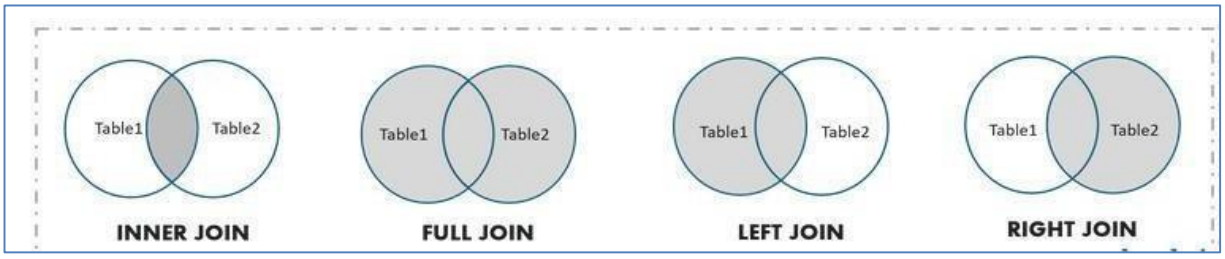
Outcome : After completion of this Assignment students will be able to
CO3 : Implement SQL queries for given requirements , using different SQL concepts

Theory:

➢ **Joins:** The purpose of a join concept is to combine data spread across tables. A join is actually performed by the "where" clause which combines specified rows of tables.

Types of JOIN:

1. **INNER JOIN** – Results return matching data from both tables.
2. **LEFT OUTER JOIN** – Results are from the left table and matching data from the right table.
3. **RIGHT OUTER JOIN** – Results are from the right table and matching data from the left table.
4. **FULL OUTER JOIN** – Results are from both tables when there is matching data.
5. **CROSS JOIN** – Results are a combination of every row from the joined tables



- **INNER JOIN**

Syntax:

SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1> INNER JOIN <tablename2>ON <tablename1>.<columnname> = <tablename2>.<columnname>

Example :

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders INNER JOIN Customers
ON Orders.CustomerID = Customers.CustomerID;
```

- **Left Outer Join**

Outer joins are similar to inner joins, but give a little bit more flexibility when selecting data from related tables. This type of joins can be used in situations where it is desired, to select all rows from the table on left (or right, or both) regardless of whether the other table has values in common & (usually) enter NULL where data is missing.

Syntax:

SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1> LEFT OUTER JOIN <tablename2>ON <tablename1>.<columnname> = <tablename2>.<columnname>

Example :

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers LEFT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

- **Right outer Join**

List the employee details with contact details (if any using right outer join. Since the RIGHT JOIN returns all the rows from the second table even if there are no matches in the first table.

Syntax:

SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1> RIGHT OUTER JOIN <tablename2>ON <tablename1>.<columnname> = <tablename2>.<columnname>

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers RIGHT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName
```

- **Cross Join**

A cross join returns what known as a Cartesian product. This means that the join combines every row from the left table with every row in the right table. As can be imagined, sometimes this join produces a mess, but under the right circumstances, it can be very useful. This type of join can be used in situation where it is desired, to select all possible combinations of rows & columns from both tables.

The kind of join is usually not preferred as it may run for a very long time & produce a huge result set that may not be useful.

Syntax:

SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1>, <tablename2>;

SELECT Customers.CustomerName, Orders.OrderID

FROM Customers , Orders

- **Self Join**

In some situation, it is necessary to join to itself, as though joining 2 separate tables.

This is referred to as self join

Syntax:

SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1> INNER JOIN <tablename1>ON <tablename1>.<columnname> = <tablename1>.<columnname> ;

SELECT Customers.CustomerName, Orders.OrderID

FROM Customers as C1 INNER JOIN Customers as C2

ON C1.CustomerID = C2.CustomerID

- **View :**

A. Creating Views

mysql> **CREATE** VIEW myView AS

-> **SELECT** id, first_name **FROM** employee **WHERE** id = 1;

Query OK, 0 rows affected (0.00 sec)

B. View with subquery

Create table penalties

(paymentno integer not null,

Employeeno integer not null,

Payment_date date not null,

Amount decimal(7,2) not null,

Primary key (paymentno));

Example:

CREATE VIEW COST_RAISERS AS SELECT * FROM Employee.

D. ALTER VIEW statement

mysql> ALTER VIEW myView (id,first_name,city)

-> AS SELECT id, upper(first_name), city FROM employee;

Query OK, 0 rows affected (0.00 sec)

E. Removing Views

mysql> drop view myView;

Query OK, 0 rows affected (0.00 sec)

F. Creating an Updatable View

mysql> CREATE OR REPLACE VIEW myView AS

-> SELECT id, first_name, city FROM employee

-> WHERE id = 3 WITH LOCAL CHECK OPTION;

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM myView;

+------+------------+ _ _ _ _ _ _ _ +

| id | first_name | city |

+------+------------+ _ _ _ _ _ _ _ +

| 3 | James | Vancouver |

+------+------------+ _ _ _ _ _ _ _ +

1 row in set (0.02 sec)

mysql>

mysql> UPDATE myView SET first_name = 'David';

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

- **Sub queries**

Subqueries in the WHERE Clause

mysql> SELECT Name FROM Course

-> WHERE CourseID =

-> (SELECT CourseID from EXAM

-> WHERE SustainedOn='10-MAR-03'

->);

- Subqueries That Return Multiple Results

mysql> SELECT Name FROM Course

-> WHERE CourseID IN

-> (SELECT CourseID from EXAM

-> WHERE SustainedOn='26-MAR-03'

->);

- Update command with sub query

```
mysql> UPDATE Articles
-> SET ArticleTitle='The Way of Zen', Copyright=1957
-> WHERE ArticleID=
-> ( SELECT ab.ArticleID
-> FROM Authors AS a, AuthorArticle AS ab
-> WHERE a.AuthID=ab.AuthID AND a.AuthorLastName='Yin'
-> );
```

- Delete command with sub query

```
DELETE ab, b
FROM AuthorArticle AS ab, Articles AS b
WHERE ab.ArticleID=b.ArticleID
AND ab.AuthID=(SELECT AuthID FROM Authors WHERE AuthorLastName
='Yin');
```

Conclusion : Demonstrate the use of concepts like all types of Join, Sub-Query and View mapped with CO3

ASSIGNMENT NO: 4

Title: Unnamed PL/SQL code block: Use of Control structure and Exception handling is mandatory

Suggested Problem statement:

Consider Tables:

1. Borrower(Roll_no, Name, DateofIssue, NameofBook, Status)

2. Fine(Roll_no,Date,Amt)

Accept Roll_no & NameofBook from user.

- Check the number of days (from date of issue),
- If days are between 15 to 30 then fine amount will be Rs 5per day.
- If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.
- After submitting the book, status will change from I to R.
- If condition of fine is true, then details will be stored into fine table.
- Also handles the exception by named exception handler or user define exception handler.

Objective: Understand the concept of Unnamed PL/SQL code, different Control Structure and exception handling

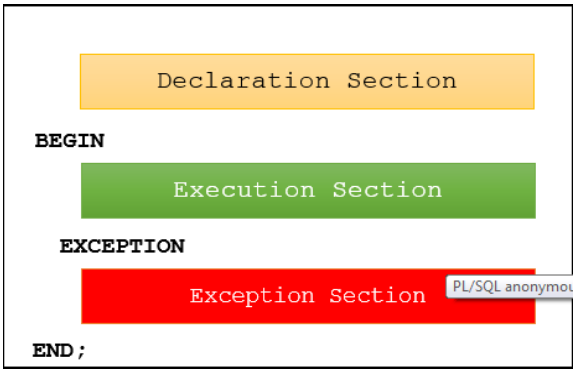
Outcome : After completion of this Assignment students will be able to
CO4: Implement PL/SQL Code block for given requirements

Theory:

- **PL/SQL :** In PL/SQL, the code is not executed in single line format, but it is always executed by grouping the code into a single element called Blocks. In this tutorial, you are going to learn about these blocks.

Blocks contain both PL/SQL as well as SQL instruction. All these instruction will be executed as a whole rather than executing a single instruction at a time.

The following picture illustrates the structure of a PL/SQL block




```
[DECLARE]
   Declaration statements;
BEGIN
   Execution statements;
  [EXCEPTION]
      Exception handling statements;
END;
```

➢ **Types of PL/SQL block**

PL/SQL blocks are of mainly two types.

1. **Anonymous blocks :** Anonymous blocks are PL/SQL blocks which do not have any names assigned to them. They need to be created and used in the same session because they will not be stored in the server as database objects.

2. **Named Blocks :**Named blocks have a specific and unique name for them. They are stored as the database objects in the server. Since they are available as database objects, they can be referred to or used as long as it is present on the server. The compilation process for named blocks happens separately while creating them as a database objects.

➢ **Unnamed PL/SQL hello world**

```
BEGIN
   DBMS_OUTPUT.put_line ('Hello World!');
END;
```

```
SQL> SET SERVEROUTPUT ON;
SQL> BEGIN
  2     DBMS_OUTPUT.PUT_LINE('Hello World');
  3  END;
  4  /
Hello World

PL/SQL procedure successfully completed.
```

SET SERVEROUTPUT ON command so that the DBMS_OUTPUT.PUT_LINE procedure will display text on the screen.

```
SQL> /
Hello World

PL/SQL procedure successfully completed.
```

If you want to edit the code block, use the edit command. SQL*Plus will write the code block to a file and open it in a text editor as shown in the following picture:

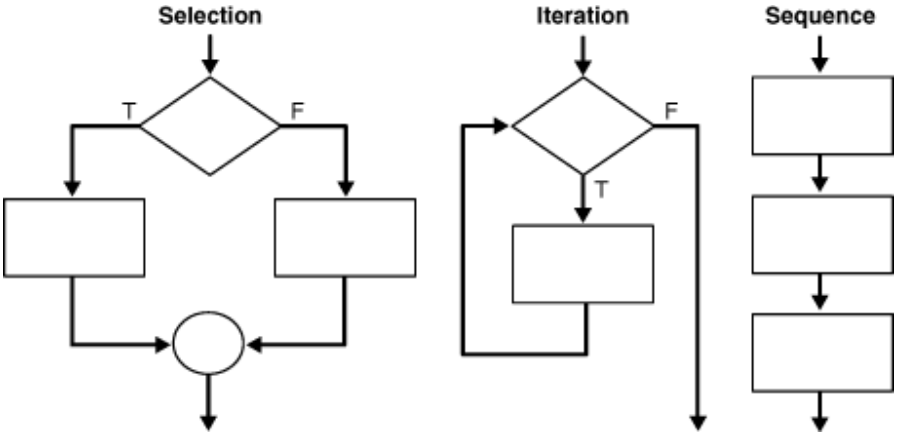
```
SQL> edit
Wrote file afiedt.buf
```

afiedt.buf - Notepad

File Edit Format View Help

```
begin
        dbms_output.put_line('Hello World');
end;
/
```

- **Control Structures**



The IF statement executes a sequence of statements depending on the value of a condition. There are three forms of IF statements: IF-THEN, IF-THEN-ELSE, and IF-THEN-ELSIF.

Syntax:

IF condition1 THEN statement1;

ELSEIF condition2 THEN statement2;

ELSEIF condition3 THEN statement3;

END IF;

The CASE statement is a compact way to evaluate a single condition and choose between many alternative actions. It makes sense to use CASE when there are three or more alternatives to choose from.

The simplest form of LOOP statement is the basic loop, which encloses a sequence of statements between the keywords LOOP and END LOOP, as follows:

LOOP

sequence_of_statements

END LOOP;

With each iteration of the loop, the sequence of statements is executed, then control resumes at the top of the loop. You use an EXIT statement to stop looping and prevent an infinite loop. You can place

one or more EXIT statements anywhere inside a loop, but not outside a loop.

- **EXCEPTIONS handling**: 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

- System-defined exceptions
- User-defined exceptions

Syntax for Exception Handling

MySQL provides an easy way to define handlers that handle from general conditions such as warnings or exceptions to specific conditions e.g., specific error codes.

Declaring a handler

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

DECLARE action HANDLER FOR condition_value statement;

If a condition whose value matches the condition_value , MySQL will execute the statement and continue or exit the current code block based on the action .

The action accepts one of the following values:

- CONTINUE : the execution of the enclosing code block (BEGIN ... END) continues.
- EXIT : the execution of the enclosing code block, where the handler is declared, terminates.

The condition_value specifies a particular condition or a class of conditions that activate the handler.

The condition_value accepts one of the following values:

- **A MySQL error code**. : An integer literal indicating a MySQL error code, such as 1051 to specify "unknown table":

```
DECLARE CONTINUE HANDLER FOR 1051
 BEGIN
  -- body of handler
 END;
```

- **SQLSTATE [VALUE]** ***sqlstate_value***: A 5-character string literal indicating an SQLSTATE value, such as '42S01' to specify "unknown table":

```
DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
 BEGIN
```

```
  -- body of handler
END;
```

***condition_name*:** A condition name previously specified with DECLARE ... CONDITION. A condition name can be associated with a MySQL error code or SQLSTATE value

- **SQL WARNING:** Shorthand for the class of SQLSTATE values that begin with '01'.

```
DECLARE CONTINUE HANDLER FOR SQLWARNING
 BEGIN
  -- body of handler
END;
```

- **NOT FOUND:** Shorthand for the class of SQLSTATE values that begin with '02', used to control what happens when a cursor reaches the end of a data set. If no more rows are available, a No Data condition occurs with SQLSTATE value '02000'.

To detect this condition, you can set up a handler for it or for a NOT FOUND condition.

```
DECLARE CONTINUE HANDLER FOR NOT FOUND
 BEGIN
  -- body of handler
 END;
```

- **SQLEXCEPTION:** Shorthand for the class of SQLSTATE values that do not begin with '00', '01', or '02'.

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
 BEGIN
  -- body of handler
 END;
```

The statement could be a simple statement or a compound statement enclosing by the BEGIN and END keywords.

Demonstration of Exit handler

```
DELIMITER #
create procedure demo_ex(in id int,in name varchar(10))
BEGIN
            // duplicate entry handler
   DECLARE EXIT HANDLER FOR 1062
   BEGIN
            SELECT 'Duplicate Key' as Message;
   END ;
insert into demo_exception values(id,'name');
select count(*) from demo_exception;
END #
DELIMITER ;
```

```
mysql> call demo_ex(9,'qq');
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.15 sec)

Query OK, 0 rows affected (0.15 sec)

mysql> call demo_ex(3,'tt');
+---------------+
| Message       |
+---------------+
| Duplicate Key |
+---------------+
1 row in set (0.00 sec)

+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

Demonstration of Continue handler

```
DELIMITER #
create procedure demo_ex(in id int,in name varchar(10))
BEGIN
   DECLARE CONTINUE HANDLER FOR 1062
   BEGIN
                        SELECT 'Duplicate Key' as Message;
   END ;
insert into demo_exception values(id,'name');
select count(*) from demo_exception;
END #
DELIMITER ;
```

```
mysql> call demo_ex(9,'qq');
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.15 sec)

Query OK, 0 rows affected (0.15 sec)

mysql> call demo_ex(3,'tt');
+---------------+
| Message       |
+---------------+
| Duplicate Key |
+---------------+
1 row in set (0.00 sec)

+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

Conclusion: Implemented some basics of PL/SQL with control structure and exception handling mapped with CO4

ASSIGNMENT NO: 5

Title: PL/SQL

Objectives: Implement PL/SQL Code block for given requirements.

Problem Statement:

Write a PL/SQL code block to calculate the area of a circle for a value of radius varying from 5 to 9.

Store the radius and the corresponding values of calculated area in an empty table named areas, consisting of two columns, radius and area.

Software and Hardware requirements:

* Software Tool: Oracle/SQL Plus

* Processor: A dual-core or quad-core processor (Intel Core i3, i5, i7 or equivalent AMD processor).

* Operating System: 64-bit Opensource Linux or its derivative.

Theory:

PL/SQL Looping

1. Basic Loop

A Basic Loop is an indefinite loop that executes its statements at least once before checking any condition. It runs until an EXIT statement is explicitly called.

Syntax:

```
LOOP
   -- Statements to be executed
   EXIT WHEN condition;  -- Exit condition
END LOOP;
```

2. WHILE Loop

A WHILE Loop is a conditional loop that executes as long as a specified condition is true.

The condition is evaluated at the beginning of each iteration.

Syntax:

WHILE condition LOOP

-- Statements to be executed

END LOOP;

3. FOR Loop

A FOR Loop is a definite loop that executes a specified number of times. The number of iterations is determined by a range of values.

Syntax:

FOR loop_variable IN [REVERSE] lower_bound .. upper_bound LOOP

-- Statements to be executed

END LOOP;

➢ Execution Of problem statement:

CONCLUSION:

This PL/SQL block calculates the area of a circle for radius values ranging from 5 to 9 and stores the results in the areas table. Each radius and its corresponding calculated area are inserted into the table. The solution ensures the data is saved, and optional output displays the results for verification. This approach efficiently handles calculations and database storage in a simple loop.

ASSIGNMENT NO: 6

Title: PL/SQL Stored Procedure and Stored Function.

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

Write a PL/SQL block for using procedure created with above requirement. Stud_Marks(name, total_marks) Result(Roll,Name, Class)

Frame the separate problem statement for writing PL/SQL Stored Procedure and function, inline with above statement. The problem statement should clearly state the requirements.

Objective: 1) To understand the differences between procedure and function

2) To understand commands related to procedure and function

Outcome : After completion of this Assignment students will be able to CO4 : Implement PL/SQL Code block for given requirements

Theory:

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

A subprogram can be created:

At schema level

Inside a package

Inside a MYSQL block

Parts of a MYSQL Subprog ram

Each MYSQL subprogram has a name, and may have a parameter list. Like anonymous PL/SQL blocks and, the named blocks a subprograms will also have following three parts:

Declarative Part

Executable part

Exception-handling

What is procedure? How to create it?

Procedures: these subprogram rams do not return a value directly, mainly used to perform an action.

- **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 [, ...])]
BEGIN
< procedure_body >
END ;
```

Where, *procedure-name* specifies the name of the procedure.

[OR REPLACE] option allows modifying an existing procedure.

The optional parameter list contains name, mode and types of the parameters. IN represents, that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.

Procedure-body contains the executable part.

- **Creating a standalone procedure.**

The following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed.

```
Delimiter //
CREATE OR REPLACE PROCEDURE greeting
Select concat('Hello World!');
```

- **Executing a Standalone Procedure**

Calling the name of the procedure from a PL/SQL block

```
Call greeting( )
Hello World
PL/SQL procedure successfully completed
```

- **Deleting a Standalone Procedure**

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

```
DROP PROCEDURE procedure-name;
```

So you can drop *greetings* procedure by using the following statement:

```
DROP PROCEDURE greetings;
```

- **PROCEDURES ON TABLES**

To run the procedures on table, lets create a sample table and insert some values in that.

```
mysql> create table student
   -> ( sid int(5) not null,
   -> student_name varchar(9),
   -> DOB date,
   -> primary key(sid));
mysql> insert into student values(5,'Harry',20130412);
mysql> insert into student values(6,'Jhon',20100215);
mysql> insert into student values(7,'Mary',20140516);
mysql> insert into student values(8,'Kay',20131116);
mysql> select * from student;
+-----+--------------+              +
| sid | student_name | DOB       |
+-----+--------------+              +
|   5 | Harry      | 2013-04-12 |
|   6 | Jhon       | 2010-02-15 |
|   7 | Mary       | 2014-05-16 |
|   8 | Kay        | 2013-11-16 |
+-----+--------------+              +
4 rows in set (0.00 sec)
```

Q]Write a Procedure to display SID & Student.

A]

```
mysql> delimiter //
mysql> create procedure myprocedure()
   -> select sid,student_name from student
   -> //

mysql> call myprocedure()//
+-----+                +
| sid | student_name |
+-----+                +
|   5 | Harry      |
|   6 | Jhon       |
|   7 | Mary       |
|   8 | Kay        |
+-----+                +
```

Q] **Write a procedure which gets the name of the student when the student id is passed ?**

```
mysql> create procedure stud(IN id INT(5),OUT name varchar(9))
   -> begin
   -> select student_name into name
```

```
   -> from student
   -> where sid=id;
   -> end//
mysql> call stud(5,@x)//
mysql> select @x//
+-----+
| @x    |
+-----+
| Harry |
+-----+
mysql> call stud(7,@x)//
mysql> select @x//
+----- +
| @x   |
+----- +
| Mary |
+----- +
mysql> call stud(5,@x);
   -> select @x;
   -> //
+-----+
| @x    |
+-----+
| Harry |
+-----+
1 row in set (0.00 sec)
```

2.*FUNCTIONS*

Functions: these subprograms return a single value, mainly used to compute and return a value.

Creating a Function:

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

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

RETURN return_datatype
BEGIN
< function_body >
RETURN variable
END [function_name];
Where,
function-name specifies the name of the function.
[OR REPLACE] option allows modifying an existing function.
The optional parameter list contains name, mode and types of the parameters. IN represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.

T he function must contain a **return** statement.

RETURN clause specifies that data type you are going to return from the function.

function-body contains the executable part.

Example:

T he following example illustrates creating and calling a standalone function. T his function returns the total number of CUSTOMERS in the customers table. We will use the CUSTOMERS table, which we had created in PL/SQL Variables chapter:

```
delimiter &
mysql> create function hello(s char(20))
   -> returns char(50)
   -> return concat('hello,s,!');
   ->  &
```

When above code is executed using MYSQL prompt, it will produce the following result:

Calling a Function

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

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

T o call a function you simply need to pass the required parameters along with function name and if function returns a value then you can store returned value. Following program calls the function from an anonymous block:

```
->select hello('world');
```

When the above code is executed at SQL prompt, it produces the following result:

```
-> Hello  world
mysql> delimiter *
mysql> create function add1(a int, b int) returns int
   -> return (a+b);
   -> select add1(10,20);
   -> *
+---------+
| add1(10,20) |
+---------+
|        30 |
+---------+
1 row in set (0.02 sec)
```

Example:

The following is one more example which demonstrates Declaring , Defining , and Invoking a Simple MYSQL Function that computes and returns the maximum of two values.

```
mysql> delimiter //
mysql> CREATE FUNCTION grt(a INT,b INT,c INT) RETURNS INT
   -> BEGIN
   -> if a>b AND a>c then
   -> RETURN a;
   -> end if;
   -> if b>c AND b>a then
   -> RETURN b;
   -> end if;
   -> RETURN c;
   -> end;
   -> //
Query OK, 0 rows affected (0.12 sec)

mysql> select grt(23,78,98);
   -> //
+----------+
| grt(23,78,98) |
+----------+
|         98 |
+----------+
1 row in set (0.05 sec)
mysql> select grt(23,98,72);
   -> //
+----------+
| grt(23,98,72) |
+----------+
|         98 |
+---------------
mysql> select grt(45,2,3); //
+---------+
| grt(45,2,3) |
+---------+
|        45 |
+---------+


mysql> delimiter //
mysql> CREATE FUNCTION odd_even(a INT) RETURNS varchar(20)
   -> BEGIN
   -> if a%2=0 then
   -> RETURN 'even';
   -> end if;

   -> RETURN 'odd';
   -> end;
   -> //
```

Query OK, 0 rows affected (0.06 sec)

```
mysql> select odd_even(54);
   -> //
+-----------+
| odd_even(54) |
+-----------+
| even       |
+-----------+
1 row in set (0.03 se
mysql> select odd_even(51); //
+-----------+
| odd_even(51) |
+-----------+
| odd        |
```

Conclusion: Thus, performed implementation of procedures and functions in MYSQL successfully mapped with CO4

Assignment No. 7

Title:- **Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor)**

Write a PL/SQL block of code using parameterized Cursor that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.

Objective: :- To study cursor PL/SQL block of code.

Outcome : After completion of this Assignment students will be able to

CO4 : Implement PL/SQL Code block for given requirements

Theory:

PL/SQL Cursor

When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement. In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it.

A cursor is a special kind of loop for traversing through an SQL resultset one row at a time. That allows us to perform operations on every record on a one-by-one basis. Just like loops, cursors are only supported within stored procedures and functions.

A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors:

- Implicit Cursors
- Explicit Cursors

1) PL/SQL Implicit Cursors

The implicit cursors are automatically generated by Oracle while an SQL statement is executed, if you don?t use an explicit cursor for the statement.

These are created by default to process the statements when DML statements like INSERT, UPDATE, DELETE etc. are executed.

Orcale provides some attributes known as Implicit cursor?s attributes to check the status of DML operations. Some of them are: %FOUND, %NOTFOUND, %ROWCOUNT and %ISOPEN.

For example: When you execute the SQL statements like INSERT, UPDATE, DELETE then the cursor

attributes tell whether any rows are affected and how many have been affected. If you run a SELECT INTO statement in PL/SQL block, the implicit cursor attribute can be used to find out whether any row has been returned by the SELECT statement. It will return an error if there no data is selected.

The following table specifies the status of the cursor with each of its attribute.

Attribute	Description
%FOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect at least one row or more rows or a SELECT INTO statement returned one or more rows. Otherwise it returns FALSE.
%NOTFOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect no row, or a SELECT INTO statement return no rows. Otherwise it returns FALSE. It is a just opposite of %FOUND.
%ISOPEN	It always returns FALSE for implicit cursors, because the SQL cursor is automatically closed after executing its associated SQL statements.
%ROWCOUNT	It returns the number of rows affected by DML statements like INSERT, DELETE, and UPDATE or returned by a SELECT INTO statement.

2) PL/SQL Explicit Cursors

The Explicit cursors are defined by the programmers to gain more control over the context area. These cursors 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.

Following is the syntax to create an explicit cursor:

Syntax of explicit cursor

Following is the syntax to create an explicit cursor:

```
CURSOR cursor_name IS select_statement;;
```

Steps:

You must follow these steps while working with an explicit cursor.

Declare the cursor to initialize in the memory.

Open the cursor to allocate memory.

Fetch the cursor to retrieve data.

Close the cursor to release allocated memory.

1) Declare the cursor:

It defines the cursor with a name and the associated SELECT statement.

Syntax for explicit cursor declaration

CURSOR name IS SELECT statement;

2) Open the cursor:

It is used to allocate memory for the cursor and make it easy to fetch the rows returned by the SQL statements into it.

Syntax for cursor open:

OPEN cursor_name;

3) Fetch the cursor:

It is used to access one row at a time. You can fetch rows from the above-opened cursor as follows:

Syntax for cursor fetch:

FETCH cursor_name **INTO** variable_list;

4) Close the cursor:

It is used to release the allocated memory. The following syntax is used to close the above-opened cursors.

Syntax for cursor close:

Close cursor_name;

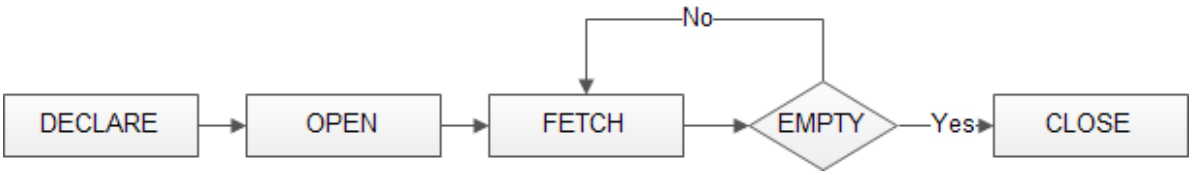
When working with MySQL cursor, you must also declare a NOT FOUND handler to handle the situation when the cursor could not find any row. Because each time you call the FETCH statement, the cursor attempts to read the next row in the result set. When the cursor reaches the end of the result set, it will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

To declare a NOT FOUND handler, you use the following syntax:

DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;

Where finished is a variable to indicate that the cursor has reached the end of the result set. Notice that the handler declaration must appear after variable and cursor declaration inside the stored procedures.

The following diagram illustrates how MySQL cursor works.



MySQL Cursor Example

We are going to develop a stored procedure that builds an email list of all employees in he employees table in the MySQL sample database.

First, we declare some variables, a cursor for looping over the emails of employees, and a NOT FOUND handler:

```
DELIMITER //
CREATE FUNCTION FindSiteID ( name_in VARCHAR(50) )
RETURNS INT
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE siteID INT DEFAULT 0;
  DECLARE c1 CURSOR FOR
   SELECT site_id
   FROM sites
   WHERE site_name = name_in;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
  OPEN c1;
  FETCH c1 INTO siteID;
  CLOSE c1;
  RETURN siteID;
END; //
DELIMITER ;
```

PROGRAM in MYSQL:

Create table stud_marks(roll_no int(2),name varchar(12),total_marks(12));

Insert record into that.

mysql> select * from stud_marks;

```
+---------+--------+ - - - - - - - - - +
| roll_no | name   | total_marks |
+---------+--------+ - - - - - - - - - +
|      1 | ravi   |       933 |
|      2 | sagar  |       450 |
|      3 | sarita |      1300 |
|      4 | avi    |       250 |
|      5 | raj    |       675 |
+---------+--------+ - - - - - - - - - +
```

5 rows in set (0.00 sec)

create table new_stud_marks(roll_no int, name char(10), grade char(10));

Query OK, 0 rows affected (0.12 sec)

```
create procedure set_cursor()
begin
declare rollno int;
declare marks int;
declare flag int;
declare c1 cursor for select roll_no, total_marks from stud_marks;
open c1;
l1:loop
fetch c1 into rollno, marks;
set flag=0;
select roll_no into flag from new_stud_marks where new_stud_marks.roll_no = rollno;
if flag = 0 then
if marks<=1500 and marks>=990 then insert into new_stud_marks values(rollno,name,'DIST'); end
if;
if marks<990 and marks>=900 then insert into new_stud_marks values(rolln o,name,'FC'); end if;
```

```
if marks<900 and marks>=825 then insert into new_stud_marks values(rollno,name,'HSC'); end if;
if marks<825 and marks>=750 then insert into new_stud_marks values(rollno,name,'SC'); end if;
if marks<750 and marks>=600 then  insert into new_stud_marks values(rollno,name,'PC'); end if;
if marks<600 then insert into new_stud_marks values(rollno,name,'FAIL'); end if;
end if;
end loop l1;
close c1;
end//

mysql> call set_cursor ();
   -> //
mysql> select * from new_stud_marks;//
+----------+-------+ _ _ _ _ _+
| roll_no | name | grade |
+----------+-------+ _ _ _ _ _+
|      2 | NULL | FAIL  |
|      3 | NULL | DIST  |
|      4 | NULL | FAIL  |
|      5 | NULL | PC    |
+----------+-------+ _ _ _ _ _+
5 rows in set (0.02 sec)
```

Conclusion: Hence we have successfully demonstrated cursor PL/SQL block using all types of Cursors in line with above statement mapped with CO4

ASSIGNMENT NO. 8

Title: Database Trigger (All Types: Row level and Statement level triggers, Before and After Triggers).

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

Objectives: To understand the concept of database MYSQL Trigger

Outcome : After completion of this Assignment students will be able to

CO4 : Implement PL/SQL Code block for given requirements

Theory:

Introduction to MYSQL Trigger

What is a Trigger?

A trigger is a MYSQL block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed.

Types of Triggers

There are two types of triggers based on the which level it is triggered.

1) Row level trigger - An event is triggered for each row upated, inserted or deleted.

2) Statement level trigger - An event is triggered for each sql statement executed.

Trigger Execution Hierarchy

The following hierarchy is followed when a trigger is fired.

1) BEFORE statement trigger fires first.

2) Next BEFORE row level trigger fires, once for each row affected.

3) Then AFTER row level trigger fires once for each affected row. This events will alternates between BEFORE and AFTER row level triggers.

4) Finally the AFTER statement level trigger fires.

Syntax of Triggers

The Syntax for creating a trigger is:

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER | }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
BEGIN
 --- sql statements
END;
```

- *CREATE TRIGGER trigger_name* - This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
  - *{BEFORE | AFTER | INSTEAD OF }* - This clause indicates at what time should the trigger get fired. i.e for example: before or after updating a table. INSTEAD OF is used to create a trigger on a view. before and after cannot be used to create a trigger on a view.
  - *{INSERT [OR] | UPDATE [OR] | DELETE}* - This clause determines the triggering event. More than one triggering events can be used together separated by OR keyword. The trigger gets fired at all the specified triggering event.
  - *[OF col_name]* - This clause is used with update triggers. This clause is used when you want to trigger an event only when a specific column is updated.
  - *CREATE [OR REPLACE] TRIGGER trigger_name* - This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
  - *[ON table_name]* - This clause identifies the name of the table or view to which the trigger is associated.
  - *[REFERENCING OLD AS o NEW AS n]* - This clause is used to reference the old and new values of the data being changed. By default, you reference the values as :old.column_name or :new.column_name. The reference names can also be changed from old (or new) to any other user-defined name. You cannot reference old values when inserting a record, or new values when deleting a record, because they do not exist.

- *[FOR EACH ROW]* - This clause is used to determine whether a trigger must fire when each row gets affected (i.e. a Row Level Trigger) or just once when the entire sql statement is executed(i.e.statement level Trigger).
- *WHEN (condition)* - This clause is valid only for row level triggers. The trigger is fired only for rows that satisfy the condition specified.

Trigger Examples

Example 1:

This example is based on the following two tables:

```
CREATE  TABLE  T4 ( a  INTEGER ,  b      CHAR(10));
CREATE  TABLE  T5 ( c  CHAR(10) ,  d      INTEGER);
```

-- create a trigger that may insert a tuple into T5 when a tuple is inserted into T4. inserts the reverse tuple into T5:

1) Create trigger as follows:

```
CREATE TRIGGER trig1 AFTER INSERT ON T4
 FOR EACH ROW BEGIN
INSERT INTO t5 SET c = NEW.b,d = NEW.a;
    END;
```

2) Insert values in T4.

3) Check the values in T5.

Example2:

1) The price of a product changes constantly. It is important to maintain the history of the prices of the products. Create a trigger to update the 'product_price_history' table when the price of the product is updated in the 'product' table.

Create the 'product' table and 'product_price_history' table

```
CREATE TABLE product_price_history

(product_id number(5),
product_name varchar2(32),
supplier_name varchar2(32),
unit_price number(7,2) );
```

```
CREATE TABLE product
(product_id number(5),
product_name varchar2(32),
supplier_name varchar2(32),
unit_price number(7,2) );
```

The price of a product changes constantly.

```
drop trigger if exists price_history_trigger;
CREATE TRIGGER price_history_trigger
BEFORE UPDATE on product1
FOR EACH ROW BEGIN
INSERT INTO product_price_history
set product_id=old.product_id,
product_name=old.product_name,
supplier_name=old.supplier_name,
unit_price=old.unit_price;
END
```

3) Lets update the price of a product.

```
UPDATE PRODUCT SET unit_price = 800 WHERE product_id = 100
```

Once the above update query is executed, the trigger fires and updates the 'product_price_history' table.

Example 3 :

```
create table account(accno int,amount int)
```

Create a trigger on account table before update in new inserted amount is less than “0” then set amount “0” else if amount is greater than 100 then set amount 100

```
CREATE TRIGGER upd_check BEFORE UPDATE ON account
FOR EACH ROW
BEGIN
 IF NEW.amount < 0 THEN
SET NEW.amount = 0;
 ELSEIF NEW.amount > 100 THEN
```

```
  SET NEW.amount = 100;
   END IF;
END
 update account set amount= -12 where accno=101
```

Deleting a trigger

DROP TRIGGER -- Removes a trigger definition from a database.

DROP TRIGGER *name* ON *table*

Conclusion: Studied and implemented MYSQL Trigger and achieved CO-4

ASSIGNMENT NO. 9

Title : Database Connectivity:

Write a program to implement MYSQL/Oracle database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc)

Objectives: To understand the database connectivity (MySQL and Java).

Outcome : After completion of this Assignment students will be able to

CO6 : Design and development applications considering actual requirements and using database concepts

Therory:

What is Database connectivity?

A **Database connection** is a facility in computer science that allows client software to talk to database server software, whether on the same machine or not. A **connection** is required to send commands and receive answers, usually in the form of a result set.

Connections are built by supplying an underlying driver or provider with a connection string, which is a way of addressing a specific database or server and instance as well as user authentication credentials (for example, ***Server=****sql_box;****Database=****Common;****User ID=****uid;****Pwd=****password;*). Once a connection has been built it can be opened and closed at will, and properties (such as the command time-out length, or transaction, if one exists) can be set. The Connection String is composed of a set of key/value pairs as dictated by the data access interface and data provider being used.

What is JDBC?

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver

A list of popular *interfaces* of JDBC API are given below:

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface

- CallableStatement interface
- ResultSet interface

ResultSetMetaData interface

- DatabaseMetaData interface
- RowSet interface

A list of popular *classes* of JDBC API are given below:

- DriverManager class
- Blob class
- Clob class
- Types class

We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

Java Database Connectivity with MySQL:

To connect Java application with the MySQL database, we need to follow 5 following steps.

In this example we are using MySql as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/abc** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and abc is the database name. We may use any database, in such case, we need to replace the abc with our database name.
3. **Username:** The default username for the mysql database is **root**.
4. **Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

To connect java application with the mysql database, **mysqlconnector.jar** file is required to be loaded.

Conclusion : Implemented the database connectivity (Java with MySQL)mapped with CO6

ASSIGNMENT NO. 10

Title: MongoDB Queries:

Design and Develop MongoDB Queries using CRUD operations. (Use CRUD operations, SAVE method, logical operators etc)

Objective: :- To study and learn MongoDB all basic operations as well as SAVE method and logical operators.

Outcome : After completion of this Assignment students will be able to,

CO5 : Implement NoSQL queries using MongoDB

Theory:-

What is MongoDB?

MongoDB is a document database with the scalability and flexibility that you want with the querying and indexing that you need.

MongoDB **stores data in flexible, JSON-like documents**, meaning fields can vary from document to document and data structure can be changed over time.

- The document model **maps to the objects in your application code**, making data easy to work with.
- **Ad hoc queries, indexing, and real time aggregation** provide powerful ways to access and analyze your data.
- MongoDB is a **distributed database at its core**, so high availability, horizontal scaling, and geographic distribution are built in and easy to use.
- MongoDB is **free and open-source**, published under the GNU Affero General Public License.

MongoDB Crud operations(*create*, *read*, *update*, and *delete.*)

Create operation: Create or insert operations add new documents to collection. If the collection does not currently exist, insert operations will create the collection. MongoDB provides the following methods to insert documents into a collection:

Syntax: db.collection.insert()

For example: db.info(collection).insert("name":"ABC","marks":50);

Read operation: Read operations retrieve documents from a collection; i.e. queries a collection for documents. MongoDB provides the following methods to read documents from a collection:

Syntax: db.collection.find()

For example: db.info(collection).find()

Update operation: Update operations modify existing documents in a collection.

Syntax: db.collection.update()

For example: db.info(collection).update({ type: "book", item : "journal"

}, { $set : { qty: 10 } }, { upsert : true }))

Delete operation: Delete operations remove documents from a collection.

Syntax: db.collection.remove()

MongoDB SAVE Method:

The **save()**method replaces the existing document with the new document passed in the save() method.

Syntax: db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})

For example: db.mycol.save({ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point New Topic", "by":"Tutorials Point" })

Logical Operators(AND,OR,AND-OR,$lt(less than),$gt(greater than)):

AND:- In the **find()**method, if you pass multiple keys by separating them by ',' then MongoDB treats it as **AND** condition.

Syntax of AND: db.mycol.find({ $and: [{key1: value1}, {key2:value2}] }).pretty()

OR:- To query documents based on the OR condition, you need to use **$or**keyword.

Syntax of OR: db.mycol.find({ $or: [{key1: value1}, {key2:value2}] }).pretty()

AND-OR:-The query documents based on the both AND-OR condition, you need to use both.

Syntax of AND-OR: db.mycol.find({"likes": {$gt:10}, $or: [{"by": "tutorials"}, {"title": "MongoDB Overview"}]}).pretty()

$lt(less than):- For Specify AND Conditions,An equality match on the field food **and** a less than ($lt) comparison match on the field price.

Example: db.inventory.find({ type: 'food', price: { $lt: 9.95 } })

$gt(greater than):-** ForSpecify OR Conditions,The field qty has a value greater than ($gt) 100 **or the value of the price field is less than ($lt)9.95

Example: db.inventory.find({ type: 'food', $or: [{ qty: { $gt: 100 } }, { price: { $lt: 9.95 } }] })

Conclusion: Hence we Study of Open Source NOSQL Database: MongoDB mapped with CO5

ASSIGNMENT NO. 11

Title: MongoDB - Aggregation and Indexing:

Design and Develop MongoDB Queries using aggregation and indexing with suitable example using MongoDB.

Objective: :- To study and learn aggregation and indexing with suitable example using MongoDB

Outcome : After completion of this Assignment students will be able to,

CO5 : Implement NoSQL queries using MongoDB

Theory:-

Introduction to Indexing

Typically, Indexes are data structures that can store collection's data set in a form that is easy to traverse. Queries are efficiently executed with the help of indexes in MongoDB. Indexes help MongoDB find documents that match the query criteria without performing a collection scan. If a query has an appropriate index, MongoDB uses the index and limits the number of documents it examines.

Single Field Index

MongoDB supports indexes on any document field in a collection. By default, the _id field in all collections have indexes.

Compound Indexes

MongoDB supports compound indexes to query multiple fields. A compound index contains multiple single field indexes separated by a comma.

The createIndex() Method

To create an index, you need to use createIndex() method of MongoDB.

Syntax

The basic syntax of createIndex() method is as follows().

>db.COLLECTION_NAME.createIndex({KEY:1})

Here key is the name of the field on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.

The dropIndex() method

>db.COLLECTION_NAME.dropIndex({KEY:1})

The aggregate() Method

For the aggregation in MongoDB, you should use aggregate() method.

Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])
$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}])
$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

Example:

```
db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
{
   "result" : [
      {
         "_id" : "tutorials point",
         "num_tutorial" : 2
      },
      {
         "_id" : "Neo4j",
         "num_tutorial" : 1
      }
   ],
   "ok" : 1
}
```

Pipeline Concept

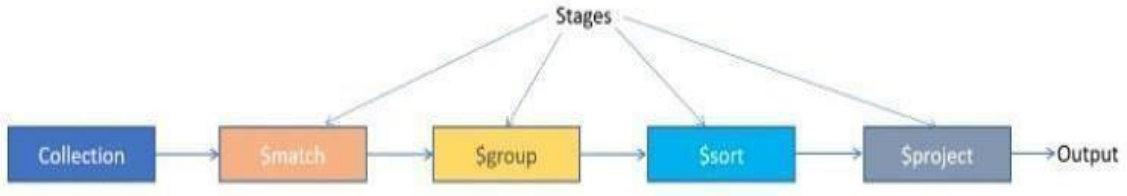
In UNIX command, shell pipeline means the possibility to execute an operation on some input and use the output as the input for the next command and so on. MongoDB also supports same concept in aggregation framework. There is a set of possible stages and each of those is taken as a set of documents as an input and produces a resulting set of documents (or the final resulting JSON document at the end of the pipeline). This can then in turn be used for the next stage and so on.

Following are the possible stages in aggregation framework –

- **$project** – Used to select some specific fields from a collection.
- **$match** – This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.
- **$group** – This does the actual aggregation as discussed above.
- **$sort** – Sorts the documents.
- **$skip** – With this, it is possible to skip forward in the list of documents for a given amount of documents.
- **$limit** – This limits the amount of documents to look at, by the given number starting from the current positions.
- **$unwind** – This is used to unwind document that are using arrays. When using an array, the data is kind of pre-joined and this operation will be undone with this to have individual documents again. Thus with this stage we will increase the amount of documents for the next stage.

Aggregation framework processes the pipeline of stages on the collection data and gives you output in the form you needed.

Every stage receives the output of the previous stage, processes the data further, and sends it to the next stage as input data. Aggregation pipeline executes on the server can take advantage of indexes. See the list of stages



Conclusion: Hence we study the MongoDB aggregation and indexing commands which is mapped with CO5.

ASSIGNMENT NO : 12

Title: MongoDB - Map reduces operations:

Implement Map reduces operation with suitable example using MongoDB.

Objective: :- To learn the Map reduces operation with MongoDB.

Outcome : After completion of this Assignment students will be able to,

CO5 : Implement NoSQL queries using MongoDB

Theory:-

What is MapReduce?

As per the MongoDB documentation, Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results. MongoDB uses mapReduce command for map-reduce operations. MapReduce is generally used for processing large data sets.

MapReduce Command

```
db.collection.mapReduce(
  function() {emit(key,value);},  //map  function
  function(key,values) {return reduceFunction}, {  //reduce function
    out: collection,
    query: document,
    sort: document,
    limit: number
  }
)
```

In the above syntax –

- **map** is a javascript function that maps a value with a key and emits a key-value pair
- **reduce** is a javascript function that reduces or groups all the documents having the same key
- **out** specifies the location of the map-reduce query result
- **query** specifies the optional selection criteria for selecting documents
- **sort** specifies the optional sort criteria
- **limit** specifies the optional maximum number of documents to be returned.

```
db.posts.mapReduce(
   function() { emit(this.user_id,1); },
   function(key, values) {return Array.sum(values)}, {
      query:{status:"active"},
      out:"post_total"
   }
)
```

The above mapReduce query outputs the following result −

```
{
   "result" : "post_total",
   "timeMillis" : 9,
   "counts" : {
      "input" : 4,
      "emit" : 4,
      "reduce" : 2,
      "output" : 2
   },
   "ok" : 1,
}
```

The result shows that a total of 4 documents matched the query (status:"active"), the map function emitted 4 documents with key-value pairs and finally the reduce function grouped mapped documents having the same keys into 2.

Conclusion: Implemented the Map reduces operation with MongoDB mapped with CO5

ASSIGNMENT NO. : 13

Title: MongoDB - Database Connectivity:

Write a program to implement MongoDB database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc)

Objective : To understand the database connectivity (MongoDB and Java).

Outcome : After completion of this Assignment students will be able to,

CO6 : Design and development applications considering actual requirements and using database concepts

Theory :

NoSQL database system which has become very popular for recent years due to its dynamic schema nature and advantages over big data like high performance, horizontal scalability, replication, etc. Unlike traditional relational database systems which provide JDBC-compliant drivers, MongoDB comes with its own non-JDBC driver called Mongo Java Driver. That means we cannot use JDBC API to interact with MongoDB from Java. Instead, we have to use its own Mongo Java Driver API.

1. Downloading Mongo Java Driver

   download latest version of Mongo Java Driver (version 2.11.1 as of this writing). The JAR file name is **mongo-java-driver-VERSION.jar** (around 400KB). Copy the downloaded JAR file into your classpath.

2. **Connecting to MongoDB using MongoClient**

   The MongoClientclass is used to make a connection with a MongoDB server and perform database-related operations. Here are some examples:

   Creating a MongoClient instance that connects to a default MongoDB server running on localhost and default port:

   ```
   MongoClient mongoClient = new MongoClient();
   ```

   Connecting to a named MongoDB server listening on the default port (27017):

   ```
   MongoClient mongoClient = new MongoClient("localhost");
   ```

   Or:

   ```
   MongoClient mongoClient = new MongoClient("db1.server.com");
   ```

Connecting to a named MongoDB server listening on a specific port:

```
MongoClient mongoClient = new MongoClient("localhost", 27017);
```

Or:

```
MongoClient mongoClient = new MongoClient("db1.server.com", 27018);
```

Connection to MongoDB Database

Once we get the connection to MongoDB server, next step is to create the connection to the database, as shown below. Note that if database is not present, MongoDB will create it for you.

```
MongoClient mongo = new MongoClient("localhost", 27017);
DB db = mongo.getDB("journaldev");
```

MongoClient provide a useful method to get all the database names, as shown below.

```
MongoClient mongo = new MongoClient("localhost", 27017);

List<String> dbs = mongo.getDatabaseNames();
System.out.println(dbs); // [journaldev, local, admin]
```

MongoDB and Collections

Every database can have zero or multiple collections, they are like tables in relational database

servers except that you don't have specific format of data. Think of it like a generic list vs list of Strings in terms of java programming language. We can get all the collections names using below code.

```
MongoClient mongo = new MongoClient("localhost", 27017);
DB db = mongo.getDB("journaldev");

Set<String> collections = db.getCollectionNames();
System.out.println(collections); // [datas, names, system.indexes, users]
```

We can get a specific collection by providing it's name, as shown below.

```
DB db = mongo.getDB("journaldev");
DBCollection col = db.getCollection("users");
```

Again if the collection doesn't exist, MongoDB will create it for you. All the data in MongoDB goes into some collection, so at this point we are ready to perform insert/update/delete operations. We can use DBCollection drop() method to drop a collection from the database.

Following is the code snippet to connect to the database

```
import com.mongodb.client.MongoDatabase;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;
public class ConnectToDB {

  public static void main( String args[] ) {

    // Creating a Mongo client
    MongoClient mongo = new MongoClient( "localhost" , 27017 );

    // Creating Credentials
    MongoCredential credential;
    credential = MongoCredential.createCredential("sampleUser", "myDb",
      "password".toCharArray());
    System.out.println("Connected to the database successfully");

    // Accessing the database
    MongoDatabase database = mongo.getDatabase("myDb");
    System.out.println("Credentials ::"+ credential);
  }
}
```

Conclusion: Implemented the connectivity of MongoDB with Java hence mapped with CO6.