BHARATI VIDYAPEETH'S COLLEGE OF ENGINEERING, LAVALE, PUNE

Department of Computer Engineering

THIRD YEAR ENGINEERING (2019 PATTERN)

Database management systems laboratory LAB MANUAL

prepared by

- Mrs. Prajakta Pawar -

List of Laboratory Assignments

Group A: SQL and PL/SQL	
1.	**ER Modeling and Normalization:** 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.
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. (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 least10 SQL queries for suitable database application using SQL DML statements. (Demonstrate the use of concepts like all types of Join ,Sub-Query and View)
4.	**Unnamed PL/SQLcode block: Use of Control structure:** 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. Write PL/SQL block in line with above statement.
5.	**Named PL/SQLcode block: Use of Control structure and Exception handling:** Consider Tables: 1. Borrower(Roll_no, Name, Date of Issue, Name of Book, Status) 2. Fine(Roll_no, Date, Amt) • Accept Roll_no and Name of Book 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.
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 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 marks899and 825 category is Higher Second Class. Write a PL/SQLblock to use procedure created with above requirement. Stud_Marks(name, total_marks), Result(Roll,Name, Class) Write stored procedure and Function inline with above statement.
7.	**Cursors:** Write a PL/SQL block of code using parameterized Cursor that will merge the data available in the newly created table N_Roll Call 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. Write PL/SQL block using all types of Cursors in line with above statement.
8.	**Database 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.Write PL/SQLblock 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 Mongo DB database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc.)
Group C: Mini Project	

.

ASSIGNMENT NO: 1

Title: ER Modeling and Normalization

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.

Problem Statement:

In an organization several projects are undertaken. Each project can employ one or more employees. Each employee can work on one or more projects. Each project is undertaken on the requirements of client. A client can request for several projects. Each project has only one client. A project can use a number of resources and a resource may be used by several projects. Draw an E-R diagram and convert it to a relational schema.

Theory:

Introduction:

DBMS stands for "Database Management System." It is software that allows users to interact with databases, store, retrieve, update, and manage data efficiently and securely. A database management system provides an interface for users and applications to access the data in the database while ensuring data integrity, consistency, and security. Examples of DBMSs include: Oracle Database, MySQL, Microsoft SQL Server, PostgreSQL, MongoDB , SQLite etc. Different types of DBMSs exist, including relational, object-oriented, graph databases, and more, catering to specific data storage and retrieval needs.

Drawing an Entity-Relationship (ER) diagram is a crucial step in the process of database design. It serves as a visual representation of the database schema and plays a significant role in helping developers, designers, and stakeholders understand, communicate, and refine the structure of the database. ER diagram accurately reflect the data requirements of the system and helps streamline the development process and contributes to building robust and well-structured databases.

Drawing an ER diagram is essential in database design as --

1. It is a blueprint for the database design that provides a clear and concise view of the database schema
2. It helps in identifying the data requirements of the system
3. It serves as a communication tool between developers, designers, and stakeholders
4. It may uncover inconsistencies or missing data requirements which can be addressed before implementing the database
5. It serves as an essential part of the database documentation.

ER Modeling Concepts

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. An Entity-Relationship (ER) diagram uses specific symbols to represent entities, attributes, and relationships in a visual and standardized way.

Example, suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc. and there will be a relationship between them.

The key components of the ER model are:

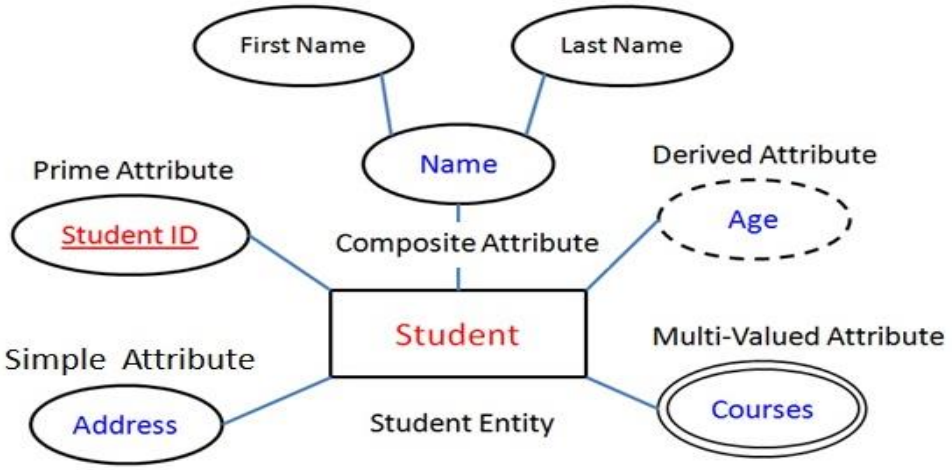
1. **Entity**:

- An entity is a real-world object or concept with its own set of attributes. Entities can be tangible objects like a person, place, or thing, or they can be intangible concepts like an event or an order.
- An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.

2. **Attributes**:

   Attributes are properties or characteristics of entities that describe them. For example, a "Person" entity may have attributes like "name," "age," and "email."

   - Simple Attribute: A simple attribute is an attribute that cannot be divided further.
   - Composite Attribute: A composite attribute is an attribute that can be divided into sub-attributes, each representing a separate piece of information.
   - Single-Valued Attribute: A single-valued attribute is an attribute that can hold only one value for each entity. For example, the "DateOfBirth".
   - Multi-Valued Attribute: A multi-valued attribute is an attribute that can hold multiple values for each entity. For instance, the "PhoneNumbers" attribute of a "Contact" entity can store several phone numbers for a single contact.
   - Derived Attribute: A derived attribute is an attribute that can be derived or calculated from other attributes or entities. It does not store data directly but is obtained through some calculation or operation. An example of a derived attribute is "Age," which can be calculated from the "DateOfBirth" attribute of a "Person" entity.
   - Key Attribute: A key attribute is an attribute that uniquely identifies each entity instance in the database. It is used to establish relationships and ensure data integrity. For instance, the "PRN" or “StudentID” attribute of a "Student" entity serves as a key attribute.



3. **Relationships**:
    - Relationships represent the associations between entities. They define how entities are related to each other in the database.
    - For example, a "Customer" entity might be related to an "Order" entity through a "places" relationship.

4. **Cardinality**:
    - Cardinality describes the number of instances of one entity that can be associated with the number of instances of another entity. Common cardinality types are one-to-one, one-to-many, many-to-one, and many-to-many.

List of Entity-Relationship (ER) diagram symbols:

- Entity (Rectangle)
- Weak Entity (Double Rectangle)
- Primary Key Attribute (Underlined Attribute)
- Attribute (Oval/Ellipse)
- Composite Attribute (Double Oval)
- Derived Attribute (Dashed Oval)
- Relationship (Diamond)
- Cardinality Notations (Crow's Foot)
    - Zero : Represented by "0" or "o" near the end of the relationship line.
    - One : Represented by "1" near the end of the relationship line.
    - Many: Represented by "M" near the end of the relationship line.

Example for an online shopping system:

1. Entities:
    - Customer:
        - This entity represents the customers of the online shopping platform.

    - It may have attributes like "CustomerID" (unique identifier), "Name," "Email," and "Address."
- Product:
    - The "Product" entity represents the items available for purchase on the platform.
    - It may have attributes like "ProductID" (unique identifier), "Name," "Price," and "Description."
- Order:
    - The "Order" entity represents individual orders placed by customers.
    - It may have attributes like "OrderID" (unique identifier), "OrderDate," and "TotalAmount."
- Payment:
    - This entity represents the payments made for each order.
    - It may have attributes like "PaymentID" (unique identifier), "PaymentDate," and "Amount."

2. Relationships:
    - Customer - Places - Order:
        - This one-to-many relationship represents that each customer can place multiple orders, but each order is placed by only one customer.
    - Order - Contains - Product:
        - This is a many-to-many relationship representing that each order can contain multiple products, and each product can be a part of multiple orders.
    - Order - Has - Payment:
        - This one-to-one relationship represents that each order has one payment associated with it.

Important points while drawing an ER diagram:

1. Identify all the main entities in the domain of interest.
2. Determine the attributes for each entity
3. Select a primary key for each entity
4. Identify the relationships between entities.
5. Specify the cardinality
6. Use standard notations to represent entities, attributes, and relationships
7. Avoid Redundancy in the diagram
8. Ensure that the relationships and cardinality align with the business requirements.
9. Verify that the diagram accurately reflects the real-world scenario.
10. Include Foreign Keys in the diagram to depict the references between related tables.
11. Review the ER diagram for accuracy and completeness.
12. Use clear and consistent naming conventions for entities, attributes, and relationships.
13. Avoid overcrowding the diagram with too many entities and relationships.

Normalization:

Normalization is a fundamental step to ensure data integrity, eliminate redundancies, reduce data anomalies and create a solid foundation for data management and retrieval in any database system. It is a systematic process used in database design to organize data into multiple related tables, eliminating data redundancy and potential anomalies while ensuring data integrity and efficient data management. The normalization process follows a set of rules and guidelines called normal forms, each addressing specific issues related to data dependencies and anomalies.

There are several normal forms, including First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), and higher forms like Boyce-Codd Normal Form (BCNF) and Fourth Normal Form (4NF). Let's discuss each normal form in detail:

1. First Normal Form (1NF):

1NF ensures that each attribute in a table contains only atomic (indivisible) values, and there are no repeating groups or arrays in a row. It eliminates data duplication within a single row. To achieve 1NF, we split multi-valued attributes into separate rows, create separate tables for related entities, and assign a primary key to each table.

2. Second Normal Form (2NF):

2NF builds upon 1NF and eliminates partial dependencies, where non-key attributes depend on only part of the primary key. To achieve 2NF, we identify partial dependencies and move them to separate tables, establishing relationships using foreign keys. Each non-key attribute should depend fully on the entire primary key.

3. Third Normal Form (3NF):

3NF builds upon 2NF and eliminates transitive dependencies, where non-key attributes depend on other non-key attributes. To achieve 3NF, we identify transitive dependencies and move them to separate tables. Each non-key attribute should depend only on the primary key, not on other non-key attributes.

4. Boyce-Codd Normal Form (BCNF):

BCNF is a more stringent form of normalization that addresses dependencies involving candidate keys, not just the primary key. It ensures that for every non-trivial functional dependency X → Y in a table, X must be a superkey (a candidate key). If a table is in BCNF, it is also in 3NF.

5. Fourth Normal Form (4NF):

4NF addresses multi-valued dependencies, where a non-key attribute has multiple values for each combination of the primary key's values. To achieve 4NF, we identify multi-valued dependencies and move them to separate tables.

6. Fifth Normal Form (5NF) or Project-Join Normal Form (PJNF):

5NF addresses join dependencies and ensures that a table is free from redundancy resulting from join operations. It involves decomposing tables into smaller tables to eliminate such redundancies.

Each higher normal form provides more strict rules for data organization and reduces the potential for data anomalies and inconsistencies. The choice of normalization level depends on the complexity of the data and the specific requirements of the application. Normalization helps maintain data integrity, reduce data redundancy, improve data consistency, and ensure efficient data management in a relational database. However, it is essential to strike a balance between normalization and performance, as higher normalization can lead to increased complexity and query complexity.

Process of normalization

Consider the following unnormalized table that contains information about products and their suppliers:

Product_Supplier (Unnormalized Table)

Product_ID	**Product_Name**	**Supplier_ID**	**Supplier_Name**	**Supplier_City**
101	Laptop	501	ABC Electronics	New York

101	Laptop	502	XYZ Gadgets	Los Angeles
102	Smartphone	501	ABC Electronics	New York
102	Smartphone	503	PQR Tech	Chicago
103	Printer	502	XYZ Gadgets	Los Angeles
104	Monitor	503	PQR Tech	Chicago

In this table, we can observe the following issues:

1. Data Redundancy:
The Product_Name and Supplier_Name are repeated for each product-supplier combination. This redundancy can lead to data inconsistency and increased storage requirements.

2. Update Anomalies:
If a supplier's name or city needs to be updated, we would have to update multiple rows, potentially leading to inconsistencies.

3. Insertion Anomalies:
If a new product is added or a new supplier starts supplying a product, we need to insert multiple rows with the same product information.

Now, let's normalize this table step by step:

Step 1: First Normal Form (1NF)

In 1NF, we ensure that each cell of the table contains atomic (indivisible) values. To achieve this, we split the table into two separate tables: one for products and another for suppliers.

Product Table (1NF)
Product_ID	Product_Name
101	Laptop
102	Smartphone
103	Printer
104	Monitor

Supplier Table (1NF)
Supplier_ID	Supplier_Name	Supplier_City
501	ABC Electronics	New York
502	XYZ Gadgets	Los Angeles
503	PQR Tech	Chicago

Step 2: Second Normal Form (2NF)

In 2NF, we remove partial dependencies by creating additional tables and establishing relationships using foreign keys.

Product_Supplier Table (2NF)

Product_ID	Supplier_ID
101	501
101	502
102	501
102	503
103	502
104	503

Step 3: Third Normal Form (3NF)

In 3NF, we remove transitive dependencies.

Product_Supplier Table (3NF)

Product_ID	Supplier_ID
101	501
101	502
102	501
102	503
103	502
104	503

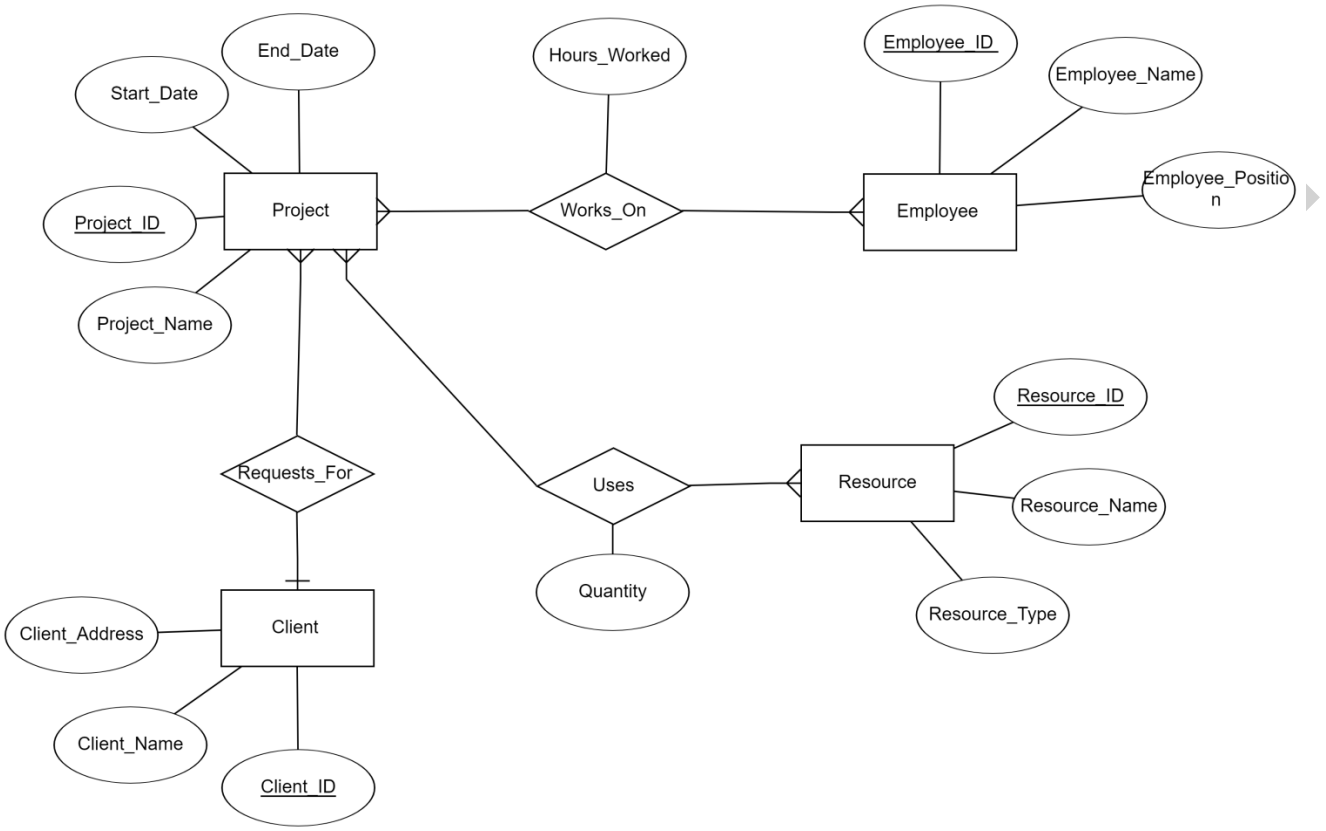
Supplier Table (3NF)

Supplier_ID	Supplier_Name	Supplier_City
501	ABC Electronics	New York
502	XYZ Gadgets	Los Angeles
503	PQR Tech	Chicago

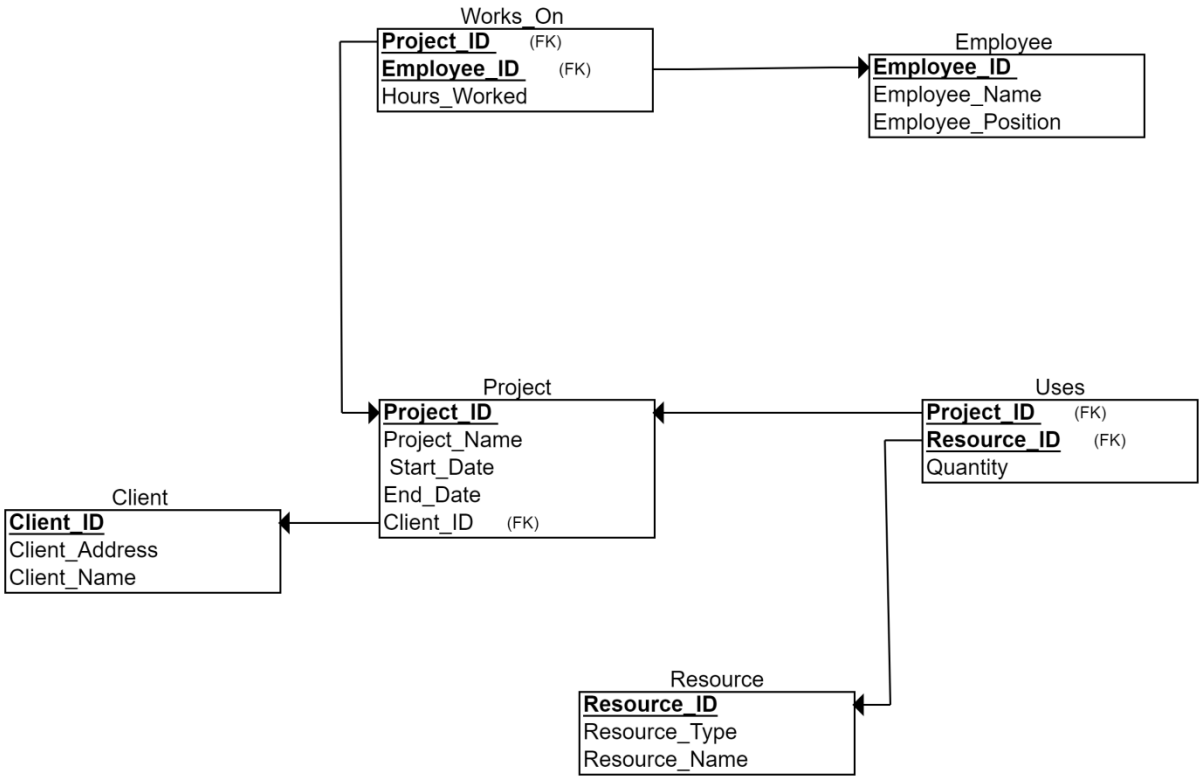
Now, the data is organized into multiple normalized tables, reducing redundancy and ensuring data integrity. Each table serves a specific purpose and can be independently updated and queried without duplication or data inconsistencies. The normalization process has improved data management and efficiency in the database. It eliminates data anomalies, minimizes redundancy, and ensures data consistency, making the database more maintainable and reliable.

Assignment No 1: Solution

ER Diagram



Relational Schema



Relational Schema

1. Project Table: **Project** (Project_ID [PK], Project_Name, Start_Date, End_Date, Client_ID [FK])
2. Employee Table: **Employee** (Employee_ID [PK], Employee_Name, Employee_Position)
3. Client Table:**Client** (Client_ID [PK], Client_Name, Client_Address)
4. Resource Table:**Resource** (Resource_ID [PK], Resource_Name, Resource_Type)
5. Works_On Table:**Works_On** (Project_ID [FK], Employee_ID [FK], Hours_Worked)
6. Requests_For Table:**Requests_For** (Project_ID [FK], Client_ID [FK])
7. Uses Table:**Uses** (Project_ID [FK], Resource_ID [FK], Quantity)

We need to analyze the dependencies for each table:

Project Table (Project_ID [PK], Project_Name, Start_Date, End_Date, Client_ID [FK]):
Project_ID, Project_Name, Start_Date, End_Date are directly dependent on the primary key (Project_ID). Client_ID depends on Project_ID, which is part of the primary key, so it is already in 3NF.

Employee Table (Employee_ID [PK], Employee_Name, Employee_Position):
Employee_ID, Employee_Name, Employee_Position are directly dependent on the primary key (Employee_ID), so the table is already in 3NF.

Client Table (Client_ID [PK], Client_Name, Client_Address):

Client_ID, Client_Name, and Client_Address are directly dependent on the primary key (Client_ID), so the table is already in 3NF.

Resource Table (Resource_ID [PK], Resource_Name, Resource_Type):
Resource_ID, Resource_Name, and Resource_Type are directly dependent on the primary key (Resource_ID), so the table is already in 3NF.

Works_On Table (Project_ID [FK], Employee_ID [FK], Hours_Worked):
Project_ID and Employee_ID are part of the composite primary key and are directly dependent on it.
Hours_Worked depends only on the primary key (Project_ID, Employee_ID), so the table is already in 3NF.

Requests_For Table (Project_ID [FK], Client_ID [FK]):
Both Project_ID and Client_ID are part of the composite primary key and are directly dependent on it, so the table is already in 3NF.

Uses Table (Project_ID [FK], Resource_ID [FK], Quantity):
Project_ID and Resource_ID are part of the composite primary key and are directly dependent on it.
Quantity depends only on the primary key (Project_ID, Resource_ID), so the table is already in 3NF.

Since all the tables in the initial data model are already in 3NF, the relational data model is considered normalized. The normalization process ensures that the data is well-structured, eliminates data redundancy, and maintains data integrity in the database.

Conclusion:

In this assignment, we have studied the ER Model and its components in detail. We have designed the ER diagram for a given problem statements using ERD Plus and converted the ER diagram into relational tables. We have understood the process to normalize a relational data model.

ASSIGNMENT NO: 2

Title: 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.

(Demonstrate the use of concepts like Insert, Select, Update, Delete with operators, functions, and set operator etc.).

Problem Statement:

Consider following Database Schemas

1. **Branch (branch_name, branch_city, assets)**
2. **Account (acc_no, *branch_name*, balance)**
3. **Customer (cust_name, cust_street, cust_city)**
4. **Depositor (*cust_name*, *acc_no*)**
5. **Loan (loan_no, *branch_name*, amount)**
6. **Borrower (*cust_name*, *loan_no*)**

Part A:

Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym, different constraints like primary key, foreign key, check constrains, not null, default etc.

Part B:

Demonstrate the use of concepts like Insert, Select, Update, Delete with operators, functions, and set operator etc. for above mentioned Bank Database Schema and Solve following SQL queries on the Bank database using SQL DML statements.

1. Find the names of all branches in loan relation.
2. Find all loan numbers for loans made at Swargate Branch with loan amount > 25000.
3. Find all customers who have an account or loan or both at bank.
4. Find all customers who have both account and loan at bank.
5. Calculate total loan amount given by bank.
6. Calculate number of accounts at a given bank.
7. Change the value of loan amount to 18000 whose loan_no is 202206.
8. List all customers in alphabetical order who have loan from Swargate branch.
9. Find the average account balance at each branch.
10. Delete all loans with loan amount between 1200 and 1500.

Theory:

A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.

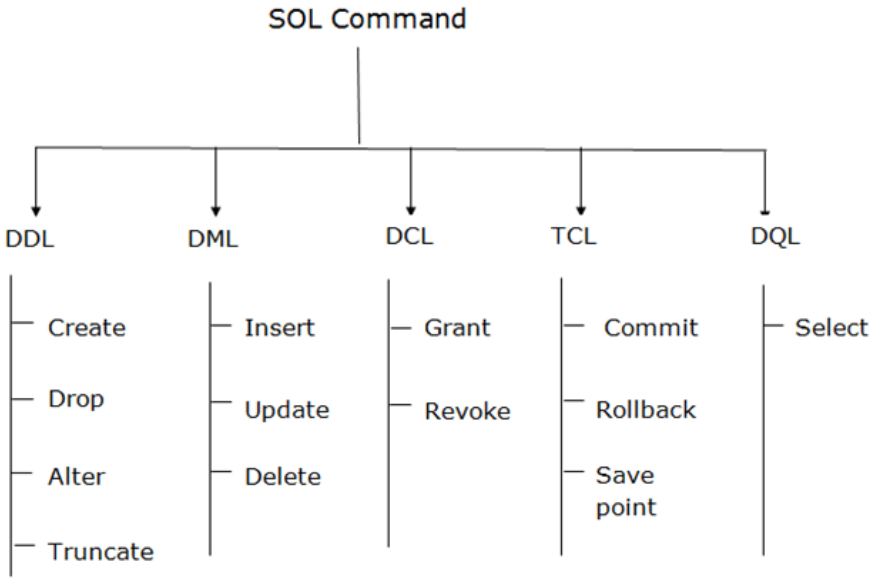
Database Applications:

- Banking: transactions
- Airlines: reservations, schedules
- Universities: registration, grades
- Sales: customers, products, purchases
- Online retailers: order tracking, customized recommendations
- Manufacturing: production, inventory, orders, supply chain
- Human resources: employee records, salaries, tax deductions

A Database Management System(DBMS) handles the way data is stored, maintained, and retrieved. In the case of a relational database, a Relational Database Management System (RDBMS) performs these tasks.

A relational database management system (RDBMS) is a program that lets you create, update, and administer a relational database. Most commercial RDBMS's use the Structured Query Language (SQL) to access the database, although SQL was invented after the development of the relational model and is not necessary for its use.

MySQL open source RDBMS overview: MySQL is a popular open source relational database management system (RDBMS) choice for web-based applications.



DATA DEFINITION LANGUAGE (DDL)

DDL-Data Definition Language (DDL) statements are used to define the database structure or schema. Data Definition Language understanding with database schemas and describes how the data should consist in the database, therefore language statements like CREATE TABLE or ALTER TABLE belongs to the DDL. DDL is about "metadata".
DDL includes commands such as CREATE, ALTER and DROP statements. DDL is used to CREATE, ALTER OR DROP the database objects (Table, Views, Users).
Data Definition Language (DDL) are used different statements :

- CREATE –to create objects in the database
- ALTER –alters the structure of the database
- DROP –delete objects from the database
- TRUNCATE –remove all records from a table, including all spaces allocated for the records are removed
- COMMENT –add comments to the data dictionary
- RENAME –rename an object

CREATE Command

CREATE is a DDL command used to create databases, tables, triggers and other database objects.

1.Syntax to Create a Database or schema:

CREATE Database Database_Name;
OR
CREATE Schema Schema _Name;

Suppose, you want to create a Books database in the SQL database. To do this, you have to write the following DDL Command:

Example:

Create Database Books;

2.Syntax to create a new table:

CREATE TABLE table_name
(
column_Name1 data_type (size of the column) ,
column_Name2 data_type (size of the column) ,
column_Name3 data_type (size of the column) ,
…
column_NameN data_type (size of the column)
) ;

Suppose, you want to create a Student table with five columns in the SQL database. To do this, you have to write the following DDL command:

Example:

```
CREATE TABLE Student
(
Roll_No. Int ,
First_Name Varchar (20) ,
Last_Name Varchar (20) ,
Age Int ,
Marks Int ,
) ;
```

Constraints

Constraints are the set of rules defined on tables to ensure data integrity.

1. Unique
2. Not null
3. Primary key
4. Default
5. Check
6. Foreign key/reference key

1.Unique constraint

The **UNIQUE** constraint in MySQL does not allow to insert a duplicate value in a column.

Syntax

CREATE TABLEtable_name (col_name data_type (size) Unique);

Example

CREATE TABLE Stud (Rno int Unique);

2.NOT NULL constraint

A **NOT NULL** constraint means that a data row must have a value for the column specified as NOT NULL.

Syntax

CREATE TABLE table_name (Col_name Data_type(size)not null, ...);

Example

Create table stud (rollno int ,name varchar2(20)not null);

3.Primary key constraint

Each table must normally contain a column or set of columns that uniquely identifies rows of data that are stored in the table. This column or set of columns is referred to as the primary key.
A table can have only one primary key.
Multiple columns can be clubbed under a composite primary key.
Primary key columns is combination of NOT NULL and UNIQUE.
Syntax

CREATE TABLE table_name (Col_name Data_type(size) CONSTRAINT constraint_name PRIMARY KEY, ...);

Example

Create table stud (rollno int primary key, name ….)
Create table stud1 (rollno int, name varchar(25), CONSTRAINT pk1 primary key(rollno));

4.Default constraint

In a MySQL table, while inserting data into a table, if no value is supplied to a column, then the column gets the value set as **DEFAULT**.

Syntax

CREATE TABLE table_name(col_name data_type(size) DEFAULT 'default_value');

Example

CREATE TABLE Stud (rno int ,name varchar2(20), addr varchar(30) DEFAULT 'Pune');

5.Check constraint

In a MySQL table, A CHECK constraint controls the values in the associated column. The **CHECK** constraint determines whether the value is valid or not.

Syntax

CREATE TABLEtable_name(col_name data_type(size) Check (condition));

Example

CREATE TABLE Stud(rollno int CHECK (rollno BETWEEN 1 AND 60));

CREATE TABLE Stud(age int CHECK (age>18));

6.Foreign key

A **FOREIGN** KEY in MySQL creates a link between two tables by one specific column of both tables. The specified column in one table must be a PRIMARY KEY and referred by the column of another table known as FOREIGN KEY.

Syntax

Create table table_name(col_name data_type(size)references table_name(col_name));

Example

Create table stud1 (rollno number(4) references stud(rno));

You can also add constraint after table creation using alter table option

Example

Alter table stud add constraint prk1 primary key(rollno);

You can also drop constraint using Drop command & name of constraint

Example

Drop constraint prk1;

View

View is a logical table. It is a physical object which stores data logically. View just refers to data that is stored in base tables.

A view is a logical entity. It is a SQL statement stored in the database in the system tablespace. Data for a view is built in a table created by the database engine in the TEMP tablespace.

Create View Syntax

Create View view_name as select col_name1,col_name2 from table_name [where <condition>]

Example

Create view v1 as select name from stud;

Create view v2 as select name from stud where addr='Nashik';

Show View Syntax

Select col_name1,.. from View_name [where condition]

Example

Select * from v1;

Drop View Syntax

Drop View view_name

Example

Drop View v1;

Index

Database index, or just index, helps speed up the retrieval of data from tables. When you query data from a table, first MySQL checks if the indexes exist, then MySQL uses the indexes to select exact physical corresponding rows of the table instead of scanning the whole table.

Create Index Syntax

Create Index index_name on table_name(column_name)

Alter table table_name add index index_name (column_name)

Example

Create Index n1 on Stud(Name)

Alter table Stud add Index n1 (name)

Show index Syntax

Show Index from table_name

Example

Show Index from Stud;

Drop Index Syntax

Alter table table_name drop Index index_name

Example

Alter table Stud drop Index n1;

Sequence

You can create a column that contains a **sequence** of numbers (1, 2, 3, and so on) by using the AUTO_INCREMENT attribute.

Syntax

CREATE TABLE table_name (column1 datatype NOT NULL AUTO_INCREMENT,….);

***ALTER TABLE table_name AUTO_INCREMENT = start_value*;**

Example

CREATE TABLE Bills (Bill_No INT(11) NOT NULL AUTO_INCREMENT, name varchar2(20));

ALTER TABLE Bills AUTO_INCREMENT = 1001

Synonym

A synonym is an alternative name for objects such as tables, views, sequences, stored procedures, and other database objects.

Eg. Nickname or short name of any person

Note: Synonyms are not possible in MySQL but possible with oracle.

ALTER Command

The ALTER TABLE statement in SQL is used to modify an existing table's structure. You can use it to add, modify, or drop columns, change data types, add constraints, and perform other structural changes to a table.

Adding a New Column:

```
ALTER TABLE employees ADD email VARCHAR(255);
```

Modifying a Column:

```
ALTER TABLE products MODIFY COLUMN price float;
```

Dropping a Column:

```
ALTER TABLE customers DROP COLUMN phone_number;
```

Renaming a Table:

```
ALTER TABLE employees RENAME TO staff;
```

Add constraints with alter command:

ALTER TABLE stud1 ALTER COLUMN name **SET DEFAULT** 'ABC';

ALTER TABLE t1 modify COLUMN name varchar(15) **not null**;

DROP Command

The DROP command is used to delete database objects such as tables, indexes, views, or databases themselves.

Drop a Table:

To delete a table and all of its data, you can use the DROP TABLE command.

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE customers;
```

Drop an Index:

To remove an index that you no longer need, you can use the DROP INDEX command.

```
DROP INDEX index_name ON table_name;
```

Example:

```
DROP INDEX idx_product_code ON products;
```

Drop a View:

To delete a view, you can use the DROP VIEW command.

```
DROP VIEW view_name;
```

Example:

```
DROP VIEW sales_report;
```

Drop a Database:

To remove an entire database and all of its objects, you can use the DROP DATABASE command. Be extremely cautious when using this command, as it permanently deletes all data in the database.

```
DROP DATABASE database_name;
```

Example:

```
DROP DATABASE mydb;
```

Drop Other Objects:

Depending on the DBMS, you can use the DROP command to delete other objects like stored procedures, functions, or triggers.

```
DROP PROCEDURE procedure_name;
DROP FUNCTION function_name;
DROP TRIGGER trigger_name;
```

Example:

```
DROP PROCEDURE calculate_sales;
```

TRUNCATE Command

The TRUNCATE statement is used to quickly delete all rows from a table, effectively resetting the table to its original empty state.

Unlike the DELETE statement, which removes rows one by one and generates a large number of individual delete statements in the transaction log, TRUNCATE is a more efficient and faster way to remove all rows from a table because it does not generate individual delete operations. It should be used with caution because it cannot be rolled back, and it may not be allowed under certain conditions (e.g., if the table is referenced by a foreign key constraint).

Syntax:

```
TRUNCATE TABLE table_name;
```

Example:

```
TRUNCATE TABLE orders;
```

Data Manipulation Language (DML)

It is a subset of SQL (Structured Query Language) that focuses on interacting with and manipulating data stored in a relational database management system (RDBMS). DML commands are used to perform operations such as inserting, updating, retrieving, and deleting data within database tables. Here are examples of DML commands in MySQL:

INSERT

The INSERT statement is used to add new rows of data to a table.

- Used to add new rows of data into a table.

- You can specify the values for each column in the table.

1.Using a Single INSERT Statement with Multiple Value Sets:

INSERT INTO table_name (column1, column2, ...)
VALUES
(value1_1, value1_2, ...),
(value2_1, value2_2, ...),
...;

```
INSERT INTO employees (first_name, last_name, salary)
VALUES
  ('John', 'Doe', 50000),
  ('Jane', 'Smith', 60000),
  ('Bob', 'Johnson', 55000);
```

2. Using Multiple INSERT Statements

```
INSERT INTO employees (first_name, last_name, salary) VALUES ('John', 'Doe', 50000);

INSERT INTO employees (first_name, last_name, salary)
VALUES ('Jane', 'Smith', 60000);

INSERT INTO employees (first_name, last_name, salary)
VALUES ('Bob', 'Johnson', 55000);
```

UPDATE

The UPDATE statement is used to modify existing records in a database table. It allows you to change the values of one or more columns in existing rows based on specified conditions.

Syntax of the UPDATE statement:

UPDATE table_name
SET column1 = new_value1, column2 = new_value2, ...
WHERE condition;

Example:

```
UPDATE employees
SET salary = 55000
WHERE first_name = 'John';
```

DELETE

This statement is used to remove one or more rows from a table based on specified conditions. It allows you to selectively delete data from a table without deleting the entire table.

DELETE FROM table_name

WHERE condition;

Example:

```
DELETE FROM employees
WHERE salary < 40000;
```

Operators in SQL:

SQL (Structured Query Language) includes a variety of operators that are used in queries and expressions to perform operations on data.

They are used in SQL statements like `SELECT`, `WHERE`, `UPDATE`, `INSERT`, and `DELETE` to filter, manipulate, and query data in a relational database.

Fundamental operators in SQL are:

1. Arithmetic Operators:
   - `+` (Addition)
   - `-` (Subtraction)
   - `*` (Multiplication)
   - `/` (Division)
   - `%` (Modulo, returns the remainder of a division)

   Example:

   ```
   SELECT salary * 1.1 FROM employees; -- Increases the salary by 10%
   ```

2. Comparison Operators:
   - `=` (Equal to)
   - `!=` or `<>` (Not equal to)
   - `<` (Less than)
   - `>` (Greater than)
   - `<=` (Less than or equal to)
   - `>=` (Greater than or equal to)

   Example:

   ```
   SELECT * FROM products WHERE price > 50;
   ```

3. Logical Operators:
   - `AND` (Logical AND)
   - `OR` (Logical OR)
   - `NOT` (Logical NOT)

   Example:

   ```
   SELECT * FROM orders WHERE status = 'Shipped' AND (total_amount > 1000 OR customer_id = 123);
   ```

4. Concatenation Operator:
   - `||` (Double pipe) or `CONCAT()` (function in some databases) is used to concatenate strings.

   Example:

   ```
   SELECT first_name || ' ' || last_name AS full_name FROM employees;
   ```

5. Wildcard Operators:
- `%` (Percent sign): Matches any sequence of characters (in `LIKE` and `ILIKE` clauses).
- `_` (Underscore): Matches any single character (in `LIKE` and `ILIKE` clauses).

Example:
SELECT * FROM products WHERE product_name LIKE 'Apple%';

6. IN Operator:
- Checks if a value matches any value in a list of values.

Example:
SELECT * FROM orders WHERE status IN ('Processing', 'Shipped');

7. BETWEEN Operator:
- Checks if a value falls within a specified range.

Example:
SELECT * FROM products WHERE price BETWEEN 20 AND 50;

8. IS NULL Operator:
- Checks if a value is NULL (or IS NOT NULL).

Example:
SELECT * FROM customers WHERE email IS NULL;

DISTINCT

In SQL, the DISTINCT keyword is used in conjunction with the SELECT statement to retrieve unique values from a specified column or a combination of columns. It ensures that the result set only contains distinct (unique) values, eliminating duplicate values.

SELECT DISTINCT column1, column2, ...
FROM table_name
WHERE condition;

Example:
SELECT DISTINCT country
FROM customers;

Functions in SQL:

SQL provides a wide range of functions that allow you to perform various operations on data within your database queries.

1. Aggregate Functions:

- Aggregate functions operate on sets of values and return a single value.
- Examples include SUM, AVG, COUNT, MAX, and MIN.

Example:

```
SELECT AVG(salary) FROM employees;
```

2. String Functions:

- String functions are used for manipulating and working with character data.
- Examples include CONCAT, SUBSTRING, LENGTH, UPPER, LOWER, and TRIM.

Example:

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name FROM employees;
```

3. Date and Time Functions:

- Date and time functions are used for working with date and time data types.
- Examples include NOW(), CURDATE(), CURTIME(), DATEDIFF(), DATE_ADD(), and DATE_FORMAT().

Example:

```
SELECT curdate();
+------------+
| curdate()  |
+------------+
| 2023-09-11 |
+------------+
```

4. Numeric Functions:

- Numeric functions are used for mathematical operations on numeric data.
- Examples include ABS, ROUND, CEIL, FLOOR, and POWER.

Example:

```
SELECT ROUND(2.999, 2) AS rounded_price;
+---------------+
| rounded_price |
+---------------+
|          3.00 |
+---------------+
```

5. Mathematical Functions:

- Mathematical functions perform various mathematical operations.
- Examples include SQRT, EXP, LOG, and SIN.

Example:

```
SELECT SQRT(9) AS square_root;
+-------------+
| square_root |
+-------------+
|           3 |
```

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

Set operators in SQL

They are used to perform operations on the result sets of two or more SELECT queries. These operators allow you to combine, intersect, or subtract rows from different queries.
The common set operators in SQL include:

1. **UNION**:
The UNION operator combines the result sets of two or more SELECT statements into a single result set, eliminating duplicate rows. It returns all unique rows present in any of the SELECT queries.

```
SELECT column1 FROM table1
UNION
SELECT column1 FROM table2;
```

2. **UNION ALL:**
UNION ALL is similar to UNION but does not eliminate duplicate rows. It combines all rows from the SELECT statements, including duplicates.

```
SELECT column1 FROM table1
UNION ALL
SELECT column1 FROM table2;
```

3. **INTERSECT:**
The INTERSECT operator returns only the rows that are common to the result sets of two SELECT queries. It returns distinct rows that exist in both result sets.

```
SELECT column1 FROM table1
INTERSECT
SELECT column1 FROM table2;
```

4. **EXCEPT :**
The EXCEPT operator returns rows that are in the first result set but not in the second result set. It subtracts the rows of the second query from the first query.

```
SELECT column1 FROM table1
EXCEPT
SELECT column1 FROM table2;
```

These set operators are especially useful when you need to combine or compare data from different tables or result sets, and they can help simplify complex queries by allowing you to work with smaller, reusable components of your SQL code.

SQL Clauses:

SQL consists of various clauses that are used within SQL statements to perform different operations on a database. Each clause serves a specific purpose and is often used in combination with other clauses to construct SQL statements. Here are some of the most common SQL clauses:

1. SELECT Clause:
   - Used to retrieve data from one or more tables.
   - Specifies the columns to be included in the result set.
   - Can also include expressions and calculations.

   SELECT column1, column2 FROM table_name;

2. FROM Clause:
   - Specifies the table(s) from which data is retrieved in a `SELECT` statement.

   SELECT column1 FROM table_name;

3. WHERE Clause:
   - Used to filter rows based on a specified condition.
   - It narrows down the result set to include only rows that meet the specified criteria.

   SELECT column1 FROM table_name WHERE condition;

4. GROUP BY Clause:
   - Groups rows with the same values in one or more columns into summary rows.
   - Used in combination with aggregate functions to perform calculations on each group.

   SELECT department, AVG(salary) FROM employees GROUP BY department;

5. HAVING Clause:
   - Filters grouped rows based on aggregate values.
   - Similar to the `WHERE` clause but operates on groups created by `GROUP BY`.

   SELECT department, AVG(salary) FROM employees GROUP BY department HAVING AVG(salary) > 50000;

6. ORDER BY Clause:
   - Sorts the result set based on one or more columns, in ascending (default) or descending order.

   SELECT column1 FROM table_name ORDER BY column1 DESC;

7. LIMIT Clause:
   - Limits the number of rows returned in the result set.

   SELECT column1 FROM table_name LIMIT 10;

Assignment No 2 Part A & B: Solution

mysql> create database **bank**;
Query OK, 1 row affected (1.63 sec)

mysql> show databases;

```
+--------------------+
| Database           |
+--------------------+
| bank               |
| batchc             |
| bvcoel             |
| db1                |
| db2                |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

10 rows in set (1.19 sec)

mysql> create table **Branch**(branch_name varchar(25) primary key,branch_city varchar(20) not null,assets integer);
Query OK, 0 rows affected (16.41 sec)

mysql> desc Branch;

```
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| branch_name | varchar(25) | NO   | PRI | NULL    |       |
| branch_city | varchar(20) | NO   |     | NULL    |       |
| assets      | int         | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

3 rows in set (2.46 sec)

mysql> create table **Account**(acc_no bigint auto_increment primary key,branch_name varchar(25) not null,balance integer default 500 check (balance >= 500), foreign key (branch_name) references Branch(branch_name));
Query OK, 0 rows affected (2.65 sec)

mysql> alter table account auto_increment=1;
Query OK, 0 rows affected (0.36 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> desc account;

Field	Type	Null	Key	Default	Extra
acc_no	bigint	NO	PRI	NULL	auto_increment
branch_name	varchar(25)	NO	MUL	NULL	
balance	int	YES		500	

3 rows in set (0.00 sec)

mysql> create table **customer**(cust_name varchar(25) primary key, cust_city varchar(15) not null);
Query OK, 0 rows affected (2.51 sec)

mysql> desc customer;

Field	Type	Null	Key	Default	Extra
cust_name	varchar(25)	NO	PRI	NULL	
cust_city	varchar(15)	NO		NULL	

2 rows in set (0.01 sec)

mysql> create table **depositor**(cust_name varchar(25), acc_no bigint,foreign key (cust_name) references customer(cust_name),foreign key (acc_no) references account(acc_no));
Query OK, 0 rows affected (3.29 sec)

mysql> desc depositor;

Field	Type	Null	Key	Default	Extra

```
| cust_name | varchar(25) | YES  | MUL | NULL    |       |
| acc_no    | bigint      | YES  | MUL | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
```

mysql> create table **loan**(loan_no bigint auto_increment primary key, branch_name varchar(25),loan_amount int, foreign key (branch_name) references branch(branch_name));
Query OK, 0 rows affected (1.62 sec)

mysql> alter table loan auto_increment=202201;
Query OK, 0 rows affected (0.26 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> desc loan;

```
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| loan_no     | bigint      | NO   | PRI | NULL    | auto_increment |
| branch_name | varchar(25) | YES  | MUL | NULL    |                |
| loan_amount | int         | YES  |     | NULL    |                |
+-------------+-------------+------+-----+---------+----------------+
```

3 rows in set (0.01 sec)

mysql> create table **borrower**(cust_name varchar(25), loan_no bigint, foreign key (cust_name) references customer(cust_name), foreign key (loan_no) references loan(loan_no));
Query OK, 0 rows affected (4.07 sec)

mysql> desc borrower;

```
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| cust_name | varchar(25) | YES  | MUL | NULL    |       |
| loan_no   | bigint      | YES  | MUL | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
```

2 rows in set (0.18 sec)

INSERT VALUES

mysql> desc branch;

```
+-------------+-------------+------+-----+---------+-------+
```

```
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| branch_name | varchar(25) | NO   | PRI | NULL    |       |
| branch_city | varchar(20) | NO   |     | NULL    |       |
| assets      | int         | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> insert into branch values('Katraj','Pune',100000000);
Query OK, 1 row affected (0.13 sec)

mysql> insert into branch values('Swargate','Pune',1055000000);
Query OK, 1 row affected (0.24 sec)

mysql> insert into branch values('Kothrud','Pune',1250000000);
Query OK, 1 row affected (0.17 sec)

mysql> insert into branch values('Hadapsar','Pune',1950000000);
Query OK, 1 row affected (0.14 sec)

mysql> select * from branch;
+-------------+-------------+------------+
| branch_name | branch_city | assets     |
+-------------+-------------+------------+
| Hadapsar    | Pune        | 1950000000 |
| Katraj      | Pune        |  100000000 |
| Kothrud     | Pune        | 1250000000 |
| Swargate    | Pune        | 1055000000 |
+-------------+-------------+------------+
4 rows in set (0.00 sec)


mysql> desc account;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| acc_no      | bigint      | NO   | PRI | NULL    | auto_increment |
| branch_name | varchar(25) | NO   | MUL | NULL    |                |
| balance     | int         | YES  |     | 500     |                |
```

```
+-------------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> insert into account(branch_name, balance) values('Hadapsar',1000);
Query OK, 1 row affected (0.15 sec)
mysql> insert into account(branch_name, balance) values('Swargate',50000);
Query OK, 1 row affected (1.55 sec)
mysql> insert into account(branch_name, balance) values('Katraj',15000);
Query OK, 1 row affected (0.52 sec)
mysql> insert into account(branch_name, balance) values('Kothrud',28000);
Query OK, 1 row affected (0.13 sec)
mysql> insert into account(branch_name, balance) values('Kothrud',90000);
Query OK, 1 row affected (0.18 sec)
mysql> insert into account(branch_name, balance) values('Swargate',100000);
Query OK, 1 row affected (0.23 sec)
mysql> insert into account(branch_name, balance) values('Kothrud',50000);
Query OK, 1 row affected (0.13 sec)
mysql> insert into account(branch_name, balance) values('Katraj',250000);
Query OK, 1 row affected (0.25 sec)
mysql> insert into account(branch_name, balance) values('Swargate',1100000);
Query OK, 1 row affected (0.13 sec)
mysql> insert into account(branch_name, balance) values('Hadapsar',50000);
Query OK, 1 row affected (0.20 sec)
mysql> insert into account(branch_name, balance) values('Pune');
ERROR 1136 (21S01): Column count doesn't match value count at row 1
mysql> insert into account(branch_name) values('Pune');
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`bank`.`account`, CONSTRAINT `account_ibfk_1` FOREIGN KEY (`branch_name`) REFERENCES `branch` (`branch_name`))
mysql> insert into account(branch_name) values('Katraj');
Query OK, 1 row affected (0.17 sec)
mysql> insert into account(branch_name,balance) values('Katraj',400);
ERROR 3819 (HY000): Check constraint 'account_chk_1' is violated.

mysql> select * from account;
+--------+-------------+---------+
| acc_no | branch_name | balance |
+--------+-------------+---------+
|      1 | Hadapsar    |    1000 |
```

```
|     2 | Swargate    |   50000 |
|     3 | Katraj      |   15000 |
|     4 | Kothrud     |   28000 |
|     5 | Kothrud     |   90000 |
|     6 | Swargate    |  100000 |
|     7 | Kothrud     |   50000 |
|     8 | Katraj      |  250000 |
|     9 | Swargate    | 1100000 |
|    10 | Hadapsar    |   50000 |
|    12 | Katraj      |     500 |
+--------+-------------+---------+
11 rows in set (0.00 sec)

mysql> desc customer;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| cust_name | varchar(25) | NO   | PRI | NULL    |       |
| cust_city | varchar(15) | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> insert into customer values('Ram','Pune');
Query OK, 1 row affected (0.27 sec)
mysql> insert into customer values('Suraj','Mumbai');
Query OK, 1 row affected (0.15 sec)
mysql> insert into customer values('Priya','Mumbai');
Query OK, 1 row affected (0.18 sec)
mysql> insert into customer values('Pooja','Nasik');
Query OK, 1 row affected (0.28 sec)
mysql> insert into customer values('Karan','Pune');
Query OK, 1 row affected (0.10 sec)
mysql> insert into customer values('Raman','Nagpur');
Query OK, 1 row affected (0.51 sec)
mysql> insert into customer values('Divya','Pune');
Query OK, 1 row affected (0.11 sec)
mysql> insert into customer values('Pihu','Nasik');
Query OK, 1 row affected (0.13 sec)
```

```
mysql> insert into customer values('Suman','Nagpur');
Query OK, 1 row affected (0.11 sec)
mysql> insert into customer values('Deepika','Pune');
Query OK, 1 row affected (0.08 sec)

mysql> Select * from customer;
+-----------+-----------+
| cust_name | cust_city |
+-----------+-----------+
| Deepika   | Pune      |
| Divya     | Pune      |
| Karan     | Pune      |
| Pihu      | Nasik     |
| Pooja     | Nasik     |
| Priya     | Mumbai    |
| Ram       | Pune      |
| Raman     | Nagpur    |
| Suman     | Nagpur    |
| Suraj     | Mumbai    |
+-----------+-----------+
10 rows in set (0.00 sec)

mysql> desc depositor;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| cust_name | varchar(25) | YES  | MUL | NULL    |       |
| acc_no    | bigint      | YES  | MUL | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> insert into depositor values('Suman',2);
Query OK, 1 row affected (0.13 sec)
mysql> insert into depositor values('Divya',6);
Query OK, 1 row affected (0.12 sec)
mysql> insert into depositor values('Karan',8);
Query OK, 1 row affected (0.17 sec)
mysql> insert into depositor values('Ram',9);
Query OK, 1 row affected (0.10 sec)
```

```
mysql> insert into depositor values('Deepika',10);
Query OK, 1 row affected (0.11 sec)

mysql> select * from depositor;
+-----------+--------+
| cust_name | acc_no |
+-----------+--------+
| Suman     |      2 |
| Divya     |      6 |
| Karan     |      8 |
| Ram       |      9 |
| Deepika   |     10 |
+-----------+--------+
5 rows in set (0.00 sec)

mysql> desc loan;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| loan_no     | bigint      | NO   | PRI | NULL    | auto_increment |
| branch_name | varchar(25) | YES  | MUL | NULL    |                |
| loan_amount | int         | YES  |     | NULL    |                |
+-------------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

//Inserting multiple rows in a table
mysql> insert into loan(branch_name,loan_amount) values('Swargate',200000), ('Katraj',50000),
('Katraj',25000),('Hadapsar',500000), ('Swargate',100000), ('Swargate',20000),('Katraj',250000),
('Swargate',80000),('Hadapsar',2500),('Katraj',1500);

mysql> select * from loan;
+---------+-------------+-------------+
| loan_no | branch_name | loan_amount |
+---------+-------------+-------------+
|  202201 | Swargate    |      200000 |
|  202202 | Katraj      |       50000 |
|  202203 | Katraj      |       25000 |
|  202204 | Hadapsar    |      500000 |
|  202205 | Swargate    |      100000 |
```

```
| 202206 | Swargate   |      18000 |
| 202207 | Katraj     |     250000 |
| 202208 | Swargate   |      80000 |
| 202209 | Hadapsar   |       2500 |
| 202210 | Katraj     |       1500 |
+---------+------------+------------+
10 rows in set (0.00 sec)
mysql> desc borrower;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| cust_name | varchar(25) | YES  | MUL | NULL    |       |
| loan_no   | bigint      | YES  | MUL | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
2 rows in set (0.18 sec)

mysql> insert into borrower values('Deepika',202201);
Query OK, 1 row affected (0.32 sec)
mysql> insert into borrower values('Divya',202205);
Query OK, 1 row affected (0.18 sec)
mysql> insert into borrower values('Karan',202206);
Query OK, 1 row affected (0.19 sec)
mysql> insert into borrower values('Ram',202208);
Query OK, 1 row affected (0.12 sec)
mysql> insert into borrower values('Priya',202204);
Query OK, 1 row affected (0.31 sec)
mysql> select * from borrower;
+-----------+---------+
| cust_name | loan_no |
+-----------+---------+
| Deepika   |  202201 |
| Divya     |  202205 |
| Karan     |  202206 |
| Ram       |  202208 |
| Priya     |  202204 |
+-----------+---------+
5 rows in set (0.00 sec)
```

---

1. Find the names of all branches in loan relation.

```
mysql> select distinct branch_name from loan;
+-------------+
| branch_name |
+-------------+
| Hadapsar    |
| Katraj      |
| Swargate    |
+-------------+
3 rows in set (0.01 sec)
```

2. Find all loan numbers for loans made at Swargate Branch with loan amount > 25000.

```
mysql> select loan_no from loan where branch_name = 'Swargate' and loan_amount > 25000;
+---------+
| loan_no |
+---------+
|  202201 |
|  202205 |
|  202208 |
+---------+
3 rows in set (0.10 sec)
```

3. Find all customers who have an account or loan or both at bank.

```
mysql> (select cust_name from depositor) union (select cust_name from borrower);
+-----------+
| cust_name |
+-----------+
| Deepika   |
| Divya     |
| Karan     |
| Ram       |
| Suman     |
| Priya     |
+-----------+
6 rows in set (0.14 sec)
```

4. Find all customers who have both account and loan at bank.

```
mysql> select distinct cust_name from borrower where cust_name in (select cust_name from
depositor);
+-----------+
| cust_name |
+-----------+
```

| Deepika |
| Divya |
| Karan |
| Ram |
+----------+
4 rows in set (0.12 sec)

5. Calculate total loan amount given by bank.

mysql> select sum(loan_amount) from loan;
+------------------+
| sum(loan_amount) |
+------------------+
| 1225000 |
+------------------+
1 row in set (0.06 sec)

6. Calculate number of accounts at a given bank.
mysql> select count(*) from account;
+----------+
| count(*) |
+----------+
| 11 |
+----------+
1 row in set (0.24 sec)

7. Change the value of loan amount to 18000 whose loan_no is 202206.
mysql> select * from loan;
+---------+-------------+-------------+
| loan_no | branch_name | loan_amount |
+---------+-------------+-------------+
202201	Swargate	200000
202202	Katraj	50000
202203	Katraj	25000
202204	Hadapsar	500000
202205	Swargate	100000
202206	Swargate	20000
202207	Katraj	250000
202208	Swargate	80000
+---------+-------------+-------------+
8 rows in set (0.03 sec)
mysql> update loan set loan_amount = 18000 where loan_no = 202206;
Query OK, 1 row affected (0.72 sec)
Rows matched: 1 Changed: 1 Warnings: 0

```
mysql> select * from loan;
+---------+-------------+-------------+
| loan_no | branch_name | loan_amount |
+---------+-------------+-------------+
|  202201 | Swargate    |      200000 |
|  202202 | Katraj      |       50000 |
|  202203 | Katraj      |       25000 |
|  202204 | Hadapsar    |      500000 |
|  202205 | Swargate    |      100000 |
|  202206 | Swargate    |       18000 |
|  202207 | Katraj      |      250000 |
|  202208 | Swargate    |       80000 |
+---------+-------------+-------------+
8 rows in set (0.41 sec)
```

8. List all customers in alphabetical order who have loan from Swargate branch.

```
mysql> select cust_name  from borrower, loan where borrower.loan_no = loan.loan_no and branch_name = "Swargate" order by cust_name;
+-----------+
| cust_name |
+-----------+
| Deepika   |
| Divya     |
| Karan     |
| Ram       |
+-----------+
4 rows in set (0.22 sec)
```

9. Find the average account balance at each branch

```
mysql> select branch_name, avg (balance) from account group by branch_name ;
+-------------+---------------+
| branch_name | avg (balance) |
+-------------+---------------+
| Hadapsar    |    25500.0000 |
| Katraj      |    88500.0000 |
| Kothrud     |    56000.0000 |
| Swargate    |   416666.6667 |
+-------------+---------------+
4 rows in set (0.47 sec)
```

10. Delete all loans with loan amount between 1200 and 1500.

```
mysql> select * from loan;
+---------+-------------+-------------+
| loan_no | branch_name | loan_amount |
+---------+-------------+-------------+
|  202201 | Swargate    |      200000 |
|  202202 | Katraj      |       50000 |
|  202203 | Katraj      |       25000 |
|  202204 | Hadapsar    |      500000 |
|  202205 | Swargate    |      100000 |
|  202206 | Swargate    |       20000 |
|  202207 | Katraj      |      250000 |
|  202208 | Swargate    |       80000 |
|  202209 | Hadapsar    |        2500 |
|  202210 | Katraj      |        1500 |
+---------+-------------+-------------+
10 rows in set (0.00 sec)
mysql> delete from loan where loan_amount >= 1200 and loan_amount <= 1500;
Query OK, 1 row affected (0.01 sec)

mysql> select * from loan;
+---------+-------------+-------------+
| loan_no | branch_name | loan_amount |
+---------+-------------+-------------+
|  202201 | Swargate    |      200000 |
|  202202 | Katraj      |       50000 |
|  202203 | Katraj      |       25000 |
|  202204 | Hadapsar    |      500000 |
|  202205 | Swargate    |      100000 |
|  202206 | Swargate    |       20000 |
|  202207 | Katraj      |      250000 |
|  202208 | Swargate    |       80000 |
|  202209 | Hadapsar    |        2500 |
+---------+-------------+-------------+
9 rows in set (0.00 sec)
```

Conclusion:

In this assignment, we have studied and demonstrated various DDL and DML statements in SQL. By using Mysql we have implemented all DDL and DML commands for given problem statement.

ASSIGNMENT NO: 3

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

Write at least10 SQL queries for suitable database application using SQL DML statements. (Demonstrate the use of concepts like all types of Join , Sub-Query and View)

Problem Statement:

Part 1

Consider following Database Schema

1. **Branch (branch_name, branch_city, assets)**
2. **Account (acc_no, branch_name, balance)**
3. **Customer (cust_name, cust_street, cust_city)**
4. **Depositor (cust_name, acc_no)**
5. **Loan (loan_no, branch_name, amount)**
6. **Borrower (cust_name, loan_no)**

Que. Implement all types of joins in MySQL.

Que. Find the list of all depositors along with details such as account no, customer name and branch.

Part 2

Consider following Database Schema (EMPLOYEE Database schema):

dept(dept_id ,dept_name, location, budget)

emp (emp_id, name, dept_id, salary, DOB, phone_no, city)

Constraints:

(default salary **5000,** default budget **10000,** phone_no must be unique, city cannot be NULL)

EMPLOYEE Database

emp_id	name	dept_id	salary	DOB	phone_no	city
101	ABC	401	7000.00	1995-07-24	8908908908	Delhi
102	ABCD	401	9000.00	1996-07-14	8908358918	Mumbai
103	PQR	402	12000.00	1990-11-01	7234358722	Mumbai
104	ABD	403	20000.00	1988-12-31	7456758722	Chennai
105	XYZ	404	6000.00	1997-02-09	7482538719	Bangalore
106	PQRS	405	21000.00	1995-04-09	7682538510	Pune
107	DEF	405	25000.00	1992-07-26	7682513596	Kolkata

dept_id	dept_name	location	budget
401	Computer	2nd Floor North	15000
402	Mechanical	2nd Floor South	12000
403	I.T.	3rd Floor South	30000
404	Electronics	3rd Floor East	17000
405	Human Resources	Ground Floor	14000

Solve the following sub queries:

1. **Display the information of employees, paid more than 'ABCD' from emp table.**
2. **List the name of the employees, who live in the same city as of 'ABCD'.**
3. **Display the information of employees, paid less salary than average salary throughout the company.**
4. **Display the information of employees having maximum salary in company.**
5. **Display the employee name, salary and department no of those employees whose salary is the minimum salary of that department.**

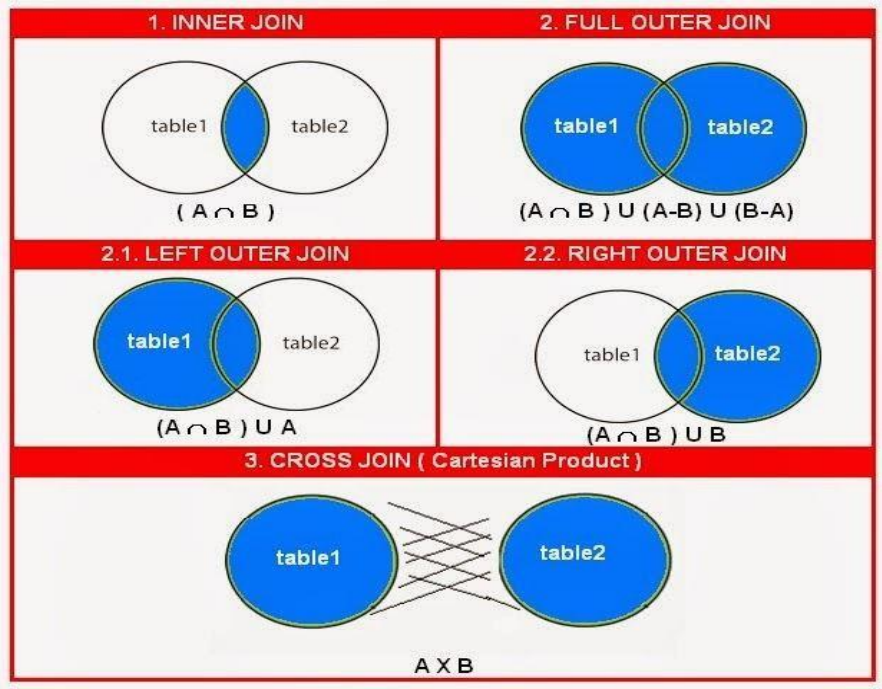
Theory:

Introduction:

SQL JOINS

In MySQL and other SQL-based database systems, joins are used to combine rows from two or more tables based on a related column between them. Joins are fundamental to relational databases as they allow you to retrieve and manipulate data from multiple tables as if they were a single entity.

MySQL supports following types of joins:



The choice of which join to use depends on your specific data retrieval requirements and the relationships between the tables in your database. Joining tables allows you to query and analyze data from multiple sources in a flexible and powerful way.

1. INNER JOIN:

- Returns only the rows where there is a match in both tables.
- Discards rows from both tables that don't have corresponding matches in the other table.
-Thus MySQL INNER JOIN is used to return all rows from multiple tables where the join condition is satisfied. It is the most common type of join.

- **Syntax 1:**

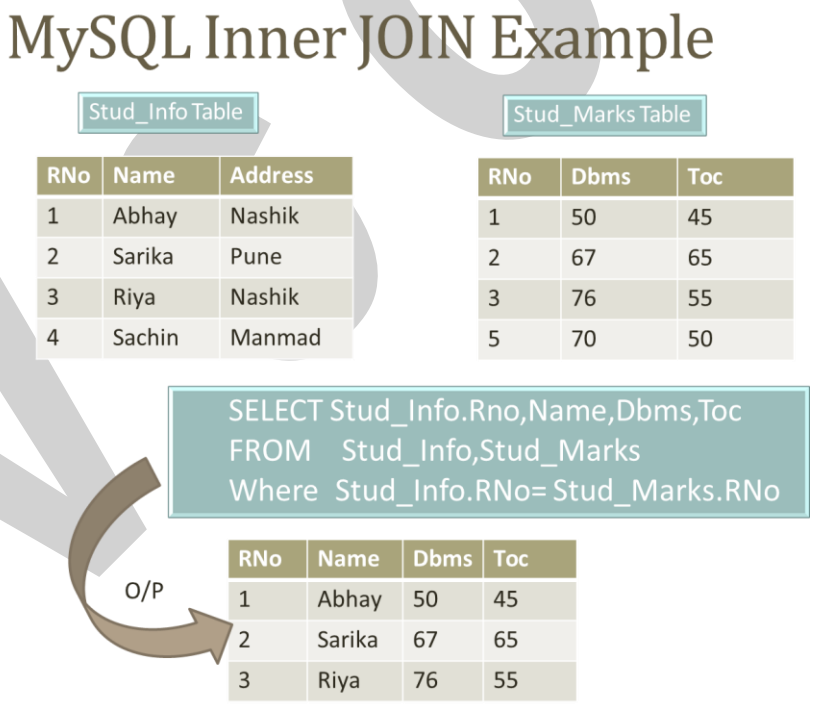
```
SELECT columns  FROM table1 ,table2
Where table1.column = table2.column;
```

- **Syntax 2:**

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

Example:

SELECT orders.order_id, customers.customer_name FROM orders INNER JOIN customers ON orders.customer_id = customers.customer_id;



2. LEFT JOIN (or LEFT OUTER JOIN:

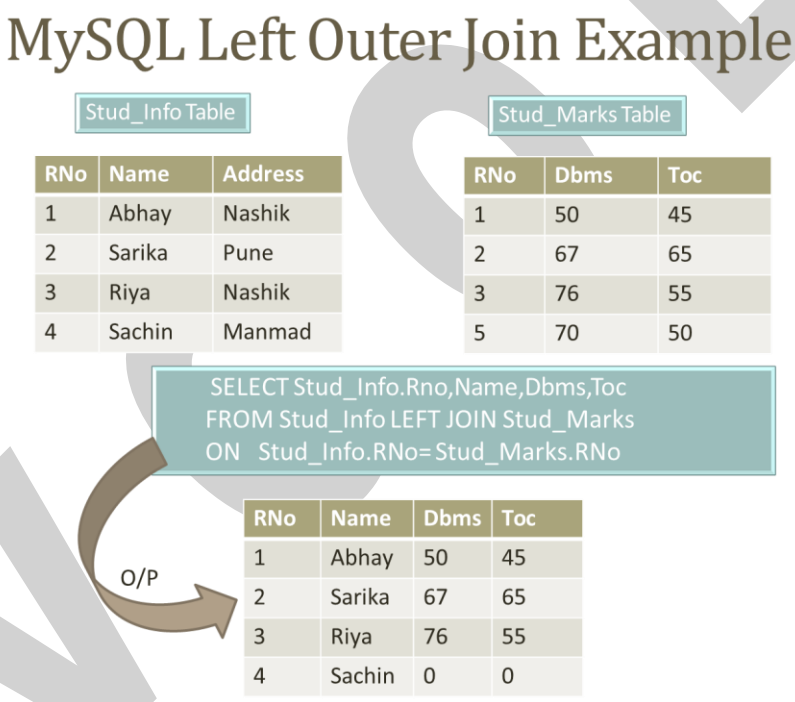
- Returns all rows from the left table and the matched rows from the right table.
- If there's no match in the right table, it returns NULL values for columns from the right table.

Syntax:

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

Example:

```
SELECT employees.employee_id, orders.order_date
FROM employees
LEFT JOIN orders ON employees.employee_id = orders.employee_id;
```



3. RIGHT JOIN (or RIGHT OUTER JOIN):

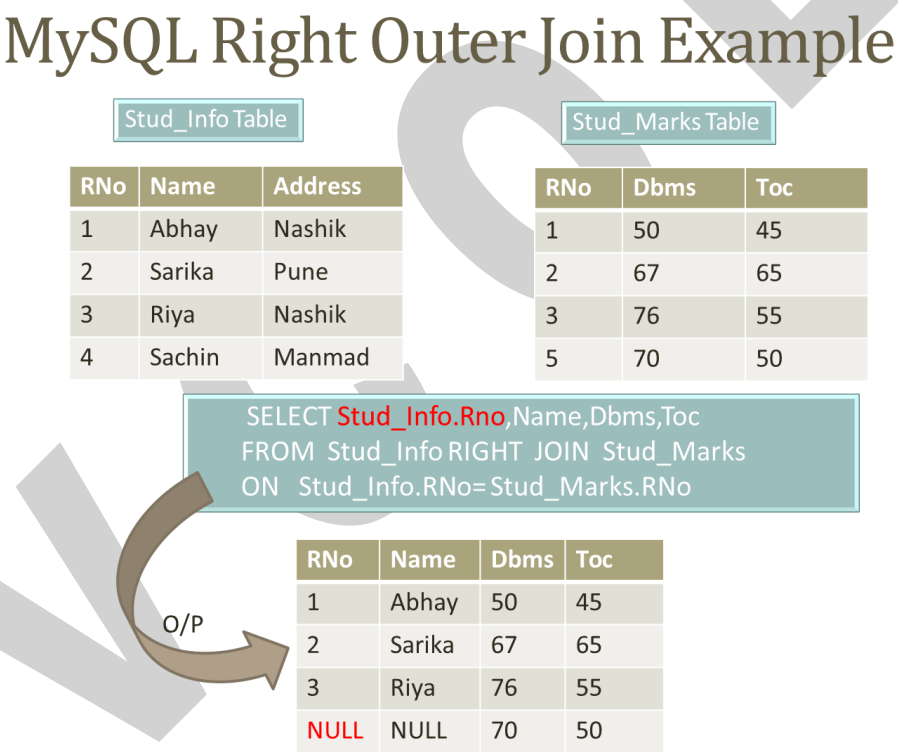
- Returns all rows from the right table and the matched rows from the left table.
- If there's no match in the left table, it returns NULL values for columns from the left table.

Syntax:

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

Example:

```
SELECT customers.customer_name, orders.order_date
FROM customers
RIGHT JOIN orders ON customers.customer_id = orders.customer_id;
```

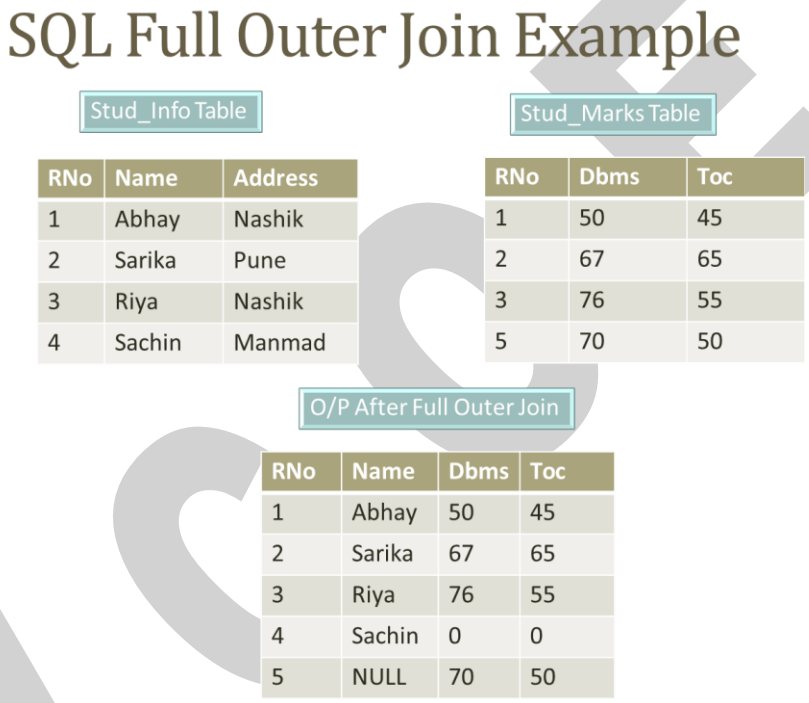


4. FULL JOIN (or FULL OUTER JOIN):

- Returns all rows from both tables.
- If there's no match in one of the tables, it returns NULL values for columns from the table without a match.

Example:

```
SELECT employees.employee_name, orders.order_date
FROM employees
FULL JOIN orders ON employees.employee_id = orders.employee_id;
```



5. SELF JOIN:

- Joins a table with itself, often used for hierarchical data or comparing rows within the same table.

Example:

```
SELECT e1.employee_name, e2.supervisor_name
FROM employees e1
LEFT JOIN employees e2 ON e1.supervisor_id = e2.employee_id;
```

6. CROSS JOIN:

- Returns the Cartesian product of two tables, resulting in all possible combinations of rows.
- Be cautious with this type of join, as it can generate a large result set.

Example:

```
SELECT products.product_name, categories.category_name
FROM products
CROSS JOIN categories;
```

SUBQUERY

A **subquery**, also known as a nested query or inner query, is a query embedded within another SQL query. Subqueries are used to retrieve data that will be used as a condition in the main query. They can be placed in various parts of a SQL statement, including the SELECT, FROM, WHERE, and HAVING clauses. Subqueries are powerful and versatile tools for querying and manipulating data in SQL. They allow you to break down complex problems into smaller, more manageable parts and are an essential part of SQL query writing.

Syntax for subqueries:

```
SELECT column1, column2, ...
FROM table1
WHERE columnN operator (SELECT columnX FROM table2 WHERE condition);
```

Types of subqueries:

1. Scalar Subquery:

A subquery that returns a single value.
It can be used in places where a single value is expected, such as in the SELECT clause or a WHERE clause.

```
SELECT column1, (SELECT MAX(column2) FROM table2) AS max_value
FROM table1;
```

2. Subquery in WHERE Clause:

A subquery that is used to filter rows in the main query based on a condition derived from another table or subquery.

```
SELECT customer_name
FROM customers
WHERE customer_id IN (SELECT customer_id FROM orders WHERE order_total > 1000);
```

3. Subquery in FROM Clause (Derived Table):

A subquery that is used in the FROM clause to create a temporary table (derived table) that can be used in the main query.

```
SELECT department, AVG(salary) AS avg_salary
FROM (SELECT employee_id, department, salary FROM employees) AS subquery
GROUP BY department;
```

4. Correlated Subquery:

A subquery where the inner query references columns from the outer query.
Correlated subqueries are executed for each row in the outer query.

```
SELECT employee_name
FROM employees e
WHERE salary > (SELECT AVG(salary) FROM employees WHERE department = e.department);
```

5. Subquery with EXISTS:

A subquery used with the EXISTS or NOT EXISTS operators to check for the existence of rows that meet specific conditions.

```
SELECT customer_name
FROM customers c
WHERE EXISTS (SELECT 1 FROM orders o WHERE o.customer_id = c.customer_id);
```

View

View is a logical table. It is a physical object which stores data logically. View just refers to data that is stored in base tables.
A view is a logical entity. It is a SQL statement stored in the database in the system tablespace. Data for a view is built in a table created by the database engine in the TEMP tablespace.

Create View Syntax

Create View view_name as select col_name1,col_name2 from table_name [where <condition>]

Example

```
Create view v1 as select name from stud;
Create view v2 as select name from stud where addr='Nashik';
```

Assignment No 3 : Solution

INNER JOIN

```
mysql> desc account;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| acc_no      | bigint      | NO   | PRI | NULL    | auto_increment |
| branch_name | varchar(25) | NO   | MUL | NULL    |                |
| balance     | int         | YES  |     | 500     |                |
+-------------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
mysql> desc depositor;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| cust_name | varchar(25) | YES  | MUL | NULL    |       |
| acc_no    | bigint      | YES  | MUL | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
mysql> select * from loan;
+---------+-------------+-------------+
| loan_no | branch_name | loan_amount |
+---------+-------------+-------------+
|  202201 | Swargate    |      200000 |
|  202202 | Katraj      |       50000 |
|  202203 | Katraj      |       25000 |
|  202204 | Hadapsar    |      500000 |
|  202205 | Swargate    |      100000 |
|  202206 | Swargate    |       20000 |
|  202207 | Katraj      |      250000 |
|  202208 | Swargate    |       80000 |
|  202209 | Hadapsar    |        2500 |
+---------+-------------+-------------+
9 rows in set (0.00 sec)

mysql> select * from depositor;
+-----------+--------+
| cust_name | acc_no |
+-----------+--------+
| Suman     |      2 |
| Divya     |      6 |
```

Karan	8
Ram	9
Deepika	10
+-----------+--------+
5 rows in set (0.00 sec)

Find the list of all depositors along with details such as account no, customer name and branch.

INNER JOIN

mysql> select account.acc_no, branch_name, cust_name from account, depositor where account.acc_no = depositor.acc_no;

+--------+-------------+-----------+
| acc_no | branch_name | cust_name |
+--------+-------------+-----------+
2	Swargate	Suman
6	Swargate	Divya
8	Katraj	Karan
9	Swargate	Ram
10	Hadapsar	Deepika
+--------+-------------+-----------+
5 rows in set (0.00 sec)

LEFT OUTER JOIN

mysql> select account.acc_no, branch_name, cust_name from account left join depositor on account.acc_no = depositor.acc_no order by acc_no;

+--------+-------------+-----------+
| acc_no | branch_name | cust_name |
+--------+-------------+-----------+
1	Hadapsar	NULL
2	Swargate	Suman
3	Katraj	NULL
4	Kothrud	NULL
5	Kothrud	NULL
6	Swargate	Divya
7	Kothrud	NULL
8	Katraj	Karan
9	Swargate	Ram
10	Hadapsar	Deepika
12	Katraj	NULL
+--------+-------------+-----------+
11 rows in set (0.00 sec)

RIGHT OUTER JOIN

```
mysql> select account.acc_no, branch_name, cust_name from account right join depositor on account.acc_no
= depositor.acc_no order by acc_no;
+--------+-------------+-----------+
| acc_no | branch_name | cust_name |
+--------+-------------+-----------+
|      2 | Swargate    | Suman     |
|      6 | Swargate    | Divya     |
|      8 | Katraj      | Karan     |
|      9 | Swargate    | Ram       |
|     10 | Hadapsar    | Deepika   |
+--------+-------------+-----------+
5 rows in set (0.00 sec)
```

EMPLOYEE Database:

```
mysql> create table dept(dept_id varchar(10) primary key ,dept_name varchar(20) NOT NULL, location
varchar(20) NOT NULL, budget int default 10000);
Query OK, 0 rows affected (0.03 sec)

mysql> desc dept;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| dept_id   | varchar(10) | NO   | PRI | NULL    |       |
| dept_name | varchar(20) | NO   |     | NULL    |       |
| location  | varchar(20) | NO   |     | NULL    |       |
| budget    | int         | YES  |     | 10000   |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
mysql> insert into dept(dept_id,dept_name,location,budget) values ('402','Mechanical','2nd Floor
South',12000);
Query OK, 1 row affected (0.01 sec)
mysql> insert into dept(dept_id,dept_name,location,budget) values ('403','I.T.','3rd Floor South',30000);
Query OK, 1 row affected (0.01 sec)
mysql>  insert into dept(dept_id,dept_name,location,budget) values ('401','Computer','2nd Floor
North',15000);
Query OK, 1 row affected (0.00 sec)
mysql>  insert into dept(dept_id,dept_name,location,budget) values ('404','Electronics','3rd Floor East',17000);
Query OK, 1 row affected (0.01 sec)
mysql> insert into dept(dept_id,dept_name,location,budget) values ('405','Human Resources','Ground
Floor',14000);
Query OK, 1 row affected (0.01 sec)
mysql> select * from dept;
+---------+-----------------+-----------------+--------+
```

```
| dept_id | dept_name      | location       | budget |
+---------+----------------+----------------+--------+
| 401     | Computer       | 2nd Floor North |  15000 |
| 402     | Mechanical     | 2nd Floor South |  12000 |
| 403     | I.T.           | 3rd Floor South |  30000 |
| 404     | Electronics    | 3rd Floor East  |  17000 |
| 405     | Human Resources | Ground Floor    |  14000 |
+---------+----------------+----------------+--------+
5 rows in set (0.00 sec)
```

```
mysql> create table emp (emp_id int primary key ,name varchar(15) NOT NULL, dept_id varchar(10), salary
decimal(18,2) default 5000, DOB date , phone_no varchar(10) UNIQUE, city varchar(20) NOT NULL, foreign
key(dept_id) references dept(dept_id));
Query OK, 0 rows affected (0.06 sec)
mysql> desc emp;
+----------+---------------+------+-----+---------+-------+
| Field    | Type          | Null | Key | Default | Extra |
+----------+---------------+------+-----+---------+-------+
| emp_id   | int           | NO   | PRI | NULL    |       |
| name     | varchar(15)   | NO   |     | NULL    |       |
| dept_id  | varchar(10)   | YES  | MUL | NULL    |       |
| salary   | decimal(18,2) | YES  |     | 5000.00 |       |
| DOB      | date          | YES  |     | NULL    |       |
| phone_no | varchar(10)   | YES  | UNI | NULL    |       |
| city     | varchar(20)   | NO   |     | NULL    |       |
+----------+---------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

```
mysql> insert into emp(emp_id,name,dept_id,salary,DOB,phone_no,city) values
('101','ABC','401','7000.00','1995-07-24','8908908908','Delhi');
Query OK, 1 row affected (0.01 sec)
```

```
mysql> insert into emp(emp_id,name,dept_id,salary,DOB,phone_no,city) values
('102','ABCD','401','9000.00','1996-07-14','8908358918','Mumbai');
Query OK, 1 row affected (0.01 sec)
```

```
mysql>  insert into emp(emp_id,name,dept_id,salary,DOB,phone_no,city) values
('103','PQR','402','12000.00','1990-11-01','7234358722','Mumbai');
Query OK, 1 row affected (0.01 sec)
```

```
mysql> insert into emp(emp_id,name,dept_id,salary,DOB,phone_no,city) values
('104','ABD','403','20000.00','1988-12-31','7456758722','Chennai');
Query OK, 1 row affected (0.01 sec)
```

```
mysql>  insert into emp(emp_id,name,dept_id,salary,DOB,phone_no,city) values
('105','XYZ','404','6000.00','1997-02-09','7482538719','Bangalore');
Query OK, 1 row affected (0.01 sec)

mysql> insert into emp(emp_id,name,dept_id,salary,DOB,phone_no,city) values
('106','PQRS','405','21000.00','1995-04-09','7682538510','Pune');
Query OK, 1 row affected (0.01 sec)

mysql> insert into emp(emp_id,name,dept_id,salary,DOB,phone_no,city) values
('107','DEF','405','25000.00','1992-07-26','7682513596','Kolkata');
Query OK, 1 row affected (0.01 sec)

mysql> select * from emp;
+--------+------+---------+----------+------------+------------+-----------+
| emp_id | name | dept_id | salary   | DOB        | phone_no   | city      |
+--------+------+---------+----------+------------+------------+-----------+
|   101 | ABC  | 401     |  7000.00 | 1995-07-24 | 8908908908 | Delhi     |
|   102 | ABCD | 401     |  9000.00 | 1996-07-14 | 8908358918 | Mumbai    |
|   103 | PQR  | 402     | 12000.00 | 1990-11-01 | 7234358722 | Mumbai    |
|   104 | ABD  | 403     | 20000.00 | 1988-12-31 | 7456758722 | Chennai   |
|   105 | XYZ  | 404     |  6000.00 | 1997-02-09 | 7482538719 | Bangalore |
|   106 | PQRS | 405     | 21000.00 | 1995-04-09 | 7682538510 | Pune      |
|   107 | DEF  | 405     | 25000.00 | 1992-07-26 | 7682513596 | Kolkata   |
+--------+------+---------+----------+------------+------------+-----------+
7 rows in set (0.00 sec)
```

SUBQUERIES:

Display the information of employees, paid more than 'ABCD from emp table

```
mysql> select * from emp where salary > (select salary from emp where name = 'ABCD');
+--------+------+---------+----------+------------+------------+---------+
| emp_id | name | dept_id | salary   | DOB        | phone_no   | city    |
+--------+------+---------+----------+------------+------------+---------+
|   103 | PQR  | 402     | 12000.00 | 1990-11-01 | 7234358722 | Mumbai  |
|   104 | ABD  | 403     | 20000.00 | 1988-12-31 | 7456758722 | Chennai |
|   106 | PQRS | 405     | 21000.00 | 1995-04-09 | 7682538510 | Pune    |
|   107 | DEF  | 405     | 25000.00 | 1992-07-26 | 7682513596 | Kolkata |
+--------+------+---------+----------+------------+------------+---------+
4 rows in set (0.01 sec)
```

List the name of the employees, who live in the same city as of 'ABCD'.

```
mysql> select * from emp where city = (select city from emp where name = 'ABCD');
+--------+------+---------+----------+------------+------------+--------+
```

```
| emp_id | name | dept_id | salary   | DOB       | phone_no   | city   |
+--------+------+---------+----------+-----------+------------+--------+
|   102 | ABCD | 401     |  9000.00 | 1996-07-14 | 8908358918 | Mumbai |
|   103 | PQR  | 402     | 12000.00 | 1990-11-01 | 7234358722 | Mumbai |
+--------+------+---------+----------+-----------+------------+--------+
2 rows in set (0.00 sec)
```

Display the information of employees, paid less salary than average salary throughout the company.

```
mysql> Select * from emp where salary <(select avg(salary)from emp ) ;
+--------+------+---------+----------+-----------+------------+-----------+
| emp_id | name | dept_id | salary   | DOB       | phone_no   | city      |
+--------+------+---------+----------+-----------+------------+-----------+
|   101 | ABC  | 401     |  7000.00 | 1995-07-24 | 8908908908 | Delhi     |
|   102 | ABCD | 401     |  9000.00 | 1996-07-14 | 8908358918 | Mumbai    |
|   103 | PQR  | 402     | 12000.00 | 1990-11-01 | 7234358722 | Mumbai    |
|   105 | XYZ  | 404     |  6000.00 | 1997-02-09 | 7482538719 | Bangalore |
+--------+------+---------+----------+-----------+------------+-----------+
4 rows in set (0.01 sec)
```

Display the information of employees having maximum salary in company.

```
mysql> Select * from emp where salary =(select max(salary)from emp );
+--------+------+---------+----------+-----------+------------+---------+
| emp_id | name | dept_id | salary   | DOB       | phone_no   | city    |
+--------+------+---------+----------+-----------+------------+---------+
|   107 | DEF  | 405     | 25000.00 | 1992-07-26 | 7682513596 | Kolkata |
+--------+------+---------+----------+-----------+------------+---------+
1 row in set (0.00 sec)
```

Display the employee name ,salary and department no of those employees whose salary is the minimum salary of that department.

```
mysql> SELECT name, salary, dept_id FROM EMP  WHERE salary IN ( SELECT MIN(salary) FROM emp GROUP BY
dept_id );
+------+----------+---------+
| name | salary   | dept_id |
+------+----------+---------+
| ABC  |  7000.00 | 401     |
| PQR  | 12000.00 | 402     |
| ABD  | 20000.00 | 403     |
| XYZ  |  6000.00 | 404     |
| PQRS | 21000.00 | 405     |
+------+----------+---------+
5 rows in set (0.01 sec)
```

VIEWS:

```
mysql> Create view v1 as select name,salary from emp;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from v1;
+------+----------+
| name | salary   |
+------+----------+
| ABC  |  7000.00 |
| ABCD |  9000.00 |
| PQR  | 12000.00 |
| ABD  | 20000.00 |
| XYZ  |  6000.00 |
| PQRS | 21000.00 |
| DEF  | 25000.00 |
+------+----------+
7 rows in set (0.01 sec)

mysql> select * from v1 where salary > 10000;
+------+----------+
| name | salary   |
+------+----------+
| PQR  | 12000.00 |
| ABD  | 20000.00 |
| PQRS | 21000.00 |
| DEF  | 25000.00 |
+------+----------+
4 rows in set (0.00 sec)
mysql> create or replace view v1 as select name, salary, dept_id from emp where salary > 10000;
Query OK, 0 rows affected (0.01 sec)

mysql> desc v1;
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| name    | varchar(15)  | NO   |     | NULL    |       |
| salary  | decimal(18,2) | YES  |     | 5000.00 |       |
| dept_id | varchar(10)  | YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> select * from v1;
```

```
+------+----------+---------+
| name | salary   | dept_id |
+------+----------+---------+
| PQR  | 12000.00 | 402     |
| ABD  | 20000.00 | 403     |
| PQRS | 21000.00 | 405     |
| DEF  | 25000.00 | 405     |
+------+----------+---------+
4 rows in set (0.00 sec)
```

//UPDATE using View

```
mysql> update v1 set salary = 35000.00 where name = 'ABD';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from v1;
+------+----------+---------+
| name | salary   | dept_id |
+------+----------+---------+
| PQR  | 12000.00 | 402     |
| ABD  | 35000.00 | 403     |
| PQRS | 21000.00 | 405     |
| DEF  | 25000.00 | 405     |
+------+----------+---------+
4 rows in set (0.00 sec)

mysql> select * from emp;
+--------+------+---------+----------+------------+------------+-----------+
| emp_id | name | dept_id | salary   | DOB        | phone_no   | city      |
+--------+------+---------+----------+------------+------------+-----------+
|   101 | ABC  | 401     |  7000.00 | 1995-07-24 | 8908908908 | Delhi     |
|   102 | ABCD | 401     |  9000.00 | 1996-07-14 | 8908358918 | Mumbai    |
|   103 | PQR  | 402     | 12000.00 | 1990-11-01 | 7234358722 | Mumbai    |
|   104 | ABD  | 403     | 35000.00 | 1988-12-31 | 7456758722 | Chennai   |
|   105 | XYZ  | 404     |  6000.00 | 1997-02-09 | 7482538719 | Bangalore |
|   106 | PQRS | 405     | 21000.00 | 1995-04-09 | 7682538510 | Pune      |
|   107 | DEF  | 405     | 25000.00 | 1992-07-26 | 7682513596 | Kolkata   |
+--------+------+---------+----------+------------+------------+-----------+
7 rows in set (0.00 sec)
```

//**DELETE**

```
mysql> delete from v1 where dept_id = 402;
Query OK, 1 row affected (0.01 sec)
mysql> select * from v1;
+------+----------+---------+
```

```
| name | salary   | dept_id |
+------+----------+---------+
| ABD  | 35000.00 | 403     |
| PQRS | 21000.00 | 405     |
| DEF  | 25000.00 | 405     |
+------+----------+---------+
3 rows in set (0.00 sec)
mysql> select * from emp;
+--------+------+---------+----------+------------+------------+-----------+
| emp_id | name | dept_id | salary   | DOB        | phone_no   | city      |
+--------+------+---------+----------+------------+------------+-----------+
|   101 | ABC  | 401     |  7000.00 | 1995-07-24 | 8908908908 | Delhi     |
|   102 | ABCD | 401     |  9000.00 | 1996-07-14 | 8908358918 | Mumbai    |
|   104 | ABD  | 403     | 35000.00 | 1988-12-31 | 7456758722 | Chennai   |
|   105 | XYZ  | 404     |  6000.00 | 1997-02-09 | 7482538719 | Bangalore |
|   106 | PQRS | 405     | 21000.00 | 1995-04-09 | 7682538510 | Pune      |
|   107 | DEF  | 405     | 25000.00 | 1992-07-26 | 7682513596 | Kolkata   |
+--------+------+---------+----------+------------+------------+-----------+
6 rows in set (0.00 sec)
```

CONCLUSION:

In this assignment we have studied concepts like types of Join , Sub-Query and Views with MySQL and implemented the same for given problem statement.

ASSIGNMENT NO: 4

Title: Unnamed PL/SQLcode block: Use of Control structure

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. Write PL/SQL block in line with above statement.

Theory:

PL/SQL

PL/SQL stands for "Procedural Language/Structured Query Language." It is a powerful and efficient extension of SQL (Structured Query Language), designed for Oracle databases but also used in some other relational database management systems (RDBMS). PL/SQL is widely used for creating stored procedures, functions, triggers, and other database objects that encapsulate business logic and facilitate data manipulation within databases. It's an essential tool for database developers and administrators in Database environments.

PL/SQL serves several key purposes:

- Procedural Programming: PL/SQL is a procedural language, which means it allows you to write blocks of code, including loops, conditionals, and exception handling. This is particularly useful for building complex, business-critical applications and automating tasks within a database.
- Database Interaction: PL/SQL enables you to interact with the database by embedding SQL queries and statements within your code. This tight integration allows you to create, retrieve, update, and delete data in the database.
- Modularity: You can create stored procedures, functions, triggers, and packages in PL/SQL. These database objects promote code reusability, maintainability, and security.
- Error Handling: PL/SQL provides robust error handling mechanisms, making it easier to catch and manage exceptions that may occur during database operations.

PL/SQL is a combination of SQL along with the procedural features of programming languages.

Basic Syntax of PL/SQL which is a block-structured language; this means that the PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts. Every PL/SQL statement ends with a semicolon (;).

Following is the basic structure of a PL/SQL block –

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

Sections	Description
Declarations	• This section starts with the keyword DECLARE. • It is an optional section and defines all variables, cursors, and other elements to be used in the program.
Executable Commands	• This section is enclosed between the keywords BEGIN and END and it is a mandatory section. • It consists of the executable PL/SQL statements of the program. • It should have at least one executable line of code.
Exception Handling	• This section starts with the keyword EXCEPTION. • This optional section contains exception(s) that handle errors in the program.

Unnamed PL/SQL

In PL/SQL, "unnamed PL/SQL" typically refers to anonymous PL/SQL blocks or code blocks that are not stored as named database objects (such as procedures, functions, or triggers). Anonymous PL/SQL blocks are often used for ad-hoc tasks or one-time operations, and they are not saved in the database for reuse.

- 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 a database objects.
- Since they need not to store in the database, they need no compilation steps.
- They are written and executed directly, and compilation and execution happen in a single process.
- These blocks don't have any reference name specified for them.
- These blocks start with the keyword ‘DECLARE’ or 'BEGIN'.
- These blocks can have all three sections of the block, in which execution section is mandatory, the other two sections are optional.
- Not possible in MySQL but possible with oracle SQL.

Here's an example of an anonymous PL/SQL block:

```
BEGIN
 -- Your PL/SQL code here
END;
```

Anonymous PL/SQL blocks are often used for various purposes, including:

- Data Manipulation: You can use them to perform data updates, inserts, or deletes within a single transaction.
- Temporary Testing: You can quickly test a piece of PL/SQL code without creating a permanent database object.

The anonymous PL/SQL blocks are useful for these temporary or ad-hoc tasks, they don't offer the reusability, maintainability, and organization benefits that named PL/SQL objects provide.

A basic PL/SQL block typically includes declarations, executable statements, and optional exception handlers. Here are examples of a simple PL/SQL block

The 'Hello World' Example:

```
DECLARE
msg varchar2(20):= 'Hello, World!';  BEGIN
dbms_output.put_line(msg);
END; /
```

PL/SQL data types:

1. Each value in PL/SQL such as a constant, variable and parameter has a data type that determines the storage format, valid values, and allowed operations.
2. PL/SQL has two kinds of data types: scalar and composite. The scalar types are types that store single values such as number, Boolean, character, and datetime whereas the composite types are types that store multiple values, for example, record and collection.
3. The scalar data types that store values with no internal components.
4. PL/SQL divides the scalar data types into four families:
   - Number
   - Boolean
   - Character
   - Datetime

PL/SQL Variables:

- In PL/SQL, a variable is named storage location that stores a value of a particular data type.
- The value of the variable changes through the program.
- Before using a variable, you must declare it in the declaration section of a block.

- Declaring variables:
  The syntax for a variable declaration is as follows:
  variable_name datatype [NOT NULL] [:= initial_value];
- PL/SQL allows you to set a default value for a variable at the declaration time. To assign a default value to a variable, you use the assignment operator (:=) or the DEFAULT keyword.
- Example

```
DECLARE
 l_product_name VARCHAR2( 100 ) := 'Laptop';
BEGIN
 NULL;
END;
```

PL/SQL IF Statement

The IF statement allows you to either execute or skip a sequence of statements, depending on a condition.

The IF statement has the three forms:

- – IF THEN
- – IF THEN ELSE
- – IF THEN ELSIF
- You can nest an IF statement within another IF statement.
- Example:

```
DECLARE
 n_sales NUMBER := 300000;
 n_commission NUMBER( 10, 2 ) := 0;
BEGIN
 IF n_sales > 200000 THEN
  n_commission := n_sales * 0.1;
 ELSIF n_sales <= 200000 AND n_sales > 100000 THEN
  n_commission := n_sales * 0.05;
 ELSIF n_sales <= 100000 AND n_sales > 50000 THEN
  n_commission := n_sales * 0.03;
 ELSE
  n_commission := n_sales * 0.02;
 END IF;
END;
```

PL/SQL LOOP:

- You can use basic PL/SQL LOOP statement to repeat a block of code until a condition is met.
- The PL/SQL LOOP statement has the following structure:

<<label>> LOOP
statements;
END LOOP loop_label;

- This structure is the most basic of all the loop constructs including FOR LOOP and WHILE LOOP.
- This basic LOOP statement consists of a LOOP keyword, a body of executable code, and the END LOOP keywords.
- The LOOP statement executes the statements in its body and returns control to the top of the loop.
- Typically, the body of the loop contains at least one EXIT or EXIT WHEN statement for terminating the loop. Otherwise, the loop becomes an infinite loop.
- The LOOP statement can have an optional label that appears at the beginning and the end of the statement.

Examples:

// For loop

```
declare
a number:=1;
begin
for a in 1..10 loop
dbms_output.put_line(a);
end loop;
end;
```

//Simple loop

```
declare
    a number:=1;
    begin
loop
    dbms_output.put_line(a);
a:=a+1;
exit when a>10;
end loop;
end;
```

//While loop

```
declare
    a number:=1;
    begin
```

```
while a<11 loop
    dbms_output.put_line(a);
    a:=a+1;
end loop;
end;
```

// if-else

```
declare
a number(4);
begin
for a in 5..15 loop
if mod(a,5)=0 then
dbms_output.put_line(a);
else
dbms_output.put_line('value'||a);
end if;
end loop;
end;
```

PL/SQL block that calculates the sum of two numbers:

```
DECLARE
 x NUMBER := 10;
 y NUMBER := 20;
 result NUMBER;
BEGIN
 -- Perform calculation
 result := x + y;
  -- Display the result
 DBMS_OUTPUT.PUT_LINE('The sum is ' || result);
END;
```

In this code:

-We declare variables x, y, and result.

-In the BEGIN block, we perform the calculation and display the result using the DBMS_OUTPUT.PUT_LINE function.

-The DECLARE, BEGIN, and END keywords define the structure of the PL/SQL block.

Assignment No 4: Solution

create table areas(radius float not null,area float not null);
Table created.

desc areas
TABLE AREAS

Column	Null?	Type
RADIUS	NOT NULL	FLOAT(126)
AREA	NOT NULL	FLOAT(126)

```
declare
  pi constant float:=3.14;
  radius float;
  area float;
  begin
  radius :=5;
  while radius <10
  loop
            area := pi*radius*radius;
            insert into areas  values(radius,area);
            radius := radius+1;
  end loop;
end;
```

select * from areas

RADIUS	AREA
5	78.5
6	113.04
7	153.86
8	200.96
9	254.34

Conclusion:

In this assignment we have studied about unnamed PLSQL block and implemented the same using Oracle SQL.

ASSIGNMENT NO: 5

Title: Named PL/SQL code block: Use of Control structure and Exception handling.

Problem Statement:

Consider Tables:

1. Borrower(Roll_no, Name, Date of Issue, Name of Book, Status)
2. Fine(Roll_no, Date, Amt)
    - Accept Roll_no and Name of Book 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.

Theory:

Named PL/SQL code block:

A named PL/SQL code block is a structured and named unit of code in PL/SQL that can be stored in a database and executed multiple times. Named PL/SQL code blocks offer advantages such as reusability, maintainability, and better organization of code. They can be called and executed multiple times from various applications and scripts. Additionally, named code blocks are stored in the database, which provides a central location for managing and maintaining the code.

- Named blocks are having 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 in the server.
- The compilation process for named blocks happens separately while creating them as a database objects.
- These blocks can be called from other blocks.
- The block structure is same as an anonymous block, except it will never start with the keyword 'DECLARE'. Instead, it will start with the keyword 'CREATE' which instruct the compiler to create it as a database object.
- These blocks can be nested within other blocks. It can also contain nested blocks.
- The main types of named PL/SQL code blocks include Stored Procedure and Function.

Stored Procedure:

A stored procedure is a named block of code that can accept parameters, perform actions, and return results. Stored procedures are commonly used for encapsulating business logic, making it reusable and maintainable.

Example of creating a stored procedure:

```
CREATE OR REPLACE PROCEDURE my_procedure(p_param1 NUMBER, p_param2 NUMBER)
AS
BEGIN
   -- PL/SQL code here
END my_procedure;
```

Stored Procedure- Parameters:

IN – is the default mode. When you define an IN parameter in a stored procedure, the calling program has to pass an argument to the stored procedure.

OUT – the value of an OUT parameter can be changed inside the stored procedure and its new value is passed back to the calling program

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

Exception handling

- PL/SQL treats all errors that occur in an anonymous block, procedure, or function as exceptions. The exceptions can have different causes such as coding mistakes, bugs, even hardware failures.
- It is not possible to anticipate all potential exceptions, however, you can write code to handle exceptions to enable the program to continue running as normal.
- The code that you write to handle exceptions is called an exception handler.
- A PL/SQL block can have an exception-handling section, which can have one or more exception handlers.
- Here is the basic syntax of the exception-handling section:

```
BEGIN
   -- executable section
   ...
   -- exception-handling section
   EXCEPTION
      WHEN e1 THEN
         -- exception_handler1
```

```
        WHEN e2 THEN
          -- exception_handler1
        WHEN OTHERS THEN
          -- other_exception_handler
    END;
```

Declaring a handler

- To declare a handler, you use the 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:

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

Exception handling-simple example:

```
Mysql>delimiter //
Mysql>Create procedure Eh()
begin
DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'Table not found';
SELECT * FROM abc;
end;
//
Mysql>delimiter ;
Mysql>Call Eh();
```

Example 1:

```
DELIMITER //
CREATE PROCEDURE display_message()
BEGIN
  SELECT 'Hello, World!' AS message;
END;
//

DELIMITER ;
CALL display_message();

+--------------+
| message      |
+--------------+
```

```
| Hello, World!|
+--------------+
```

Example 2: Named procedure with parameters

```
DELIMITER //
CREATE PROCEDURE calculate_product10 (    IN a INT,    IN b INT,    OUT product INT )
BEGIN
   SET product = a * b;
END;
//

DELIMITER ;
-- Call the procedure with input parameters
CALL calculate_product(5, 7, @result);
-- Display the result
SELECT @result;
+--------+
| @result|
+--------+
| 35     |
          +--------+
```

ASSIGNMENT NO-5 : Solution

```
mysql> CREATE TABLE  borrower(roll_no INT , name VARCHAR(25), dateofissue DATE,name_of_book
VARCHAR(25), status VARCHAR(20));
Query OK, 0 rows affected (0.03 sec)

mysql> show tables;
+----------------+
| Tables_in_plsql |
+----------------+
| borrower       |
+----------------+
1 row in set (0.00 sec)

mysql> CREATE TABLE fine(roll_no INT,date_of_return DATE,amt INT);
Query OK, 0 rows affected (0.02 sec)

mysql> show tables;
+----------------+
| Tables_in_plsql |
+----------------+
| borrower       |
| fine           |
+----------------+
2 rows in set (0.00 sec)

mysql> CREATE TABLE fine(roll_no INT,date_of_return DATE,amt INT);
Query OK, 0 rows affected (0.02 sec)

mysql> show tables;
+----------------+
| Tables_in_plsql |
+----------------+
| borrower       |
| fine           |
+----------------+
2 rows in set (0.00 sec)

mysql> INSERT INTO borrower VALUES(45,'RAM','2022-8-1','HARRY POTTER','I');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO borrower VALUES(46,'ARYAN','2022-10-15','DARK MATTER', 'I');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO borrower VALUES(47,'ROHAN','2022-10-24','SILENT HILL', 'I');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO borrower VALUES(48,'SANKET','2022-9-26','GOD OF WAR', 'I');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO borrower VALUES(49,'SARTHAK','2022-10-9','SPIDER-MAN', 'I');
Query OK, 1 row affected (0.00 sec)

mysql> select * from borrower;
```

```
+---------+---------+-------------+--------------+--------+
| roll_no | name    | dateofissue | name_of_book | status |
+---------+---------+-------------+--------------+--------+
|      45 | RAM     | 2022-08-01  | HARRY POTTER | I      |
|      46 | ARYAN   | 2022-10-15  | DARK MATTER  | I      |
|      47 | ROHAN   | 2022-10-24  | SILENT HILL  | I      |
|      48 | SANKET  | 2022-09-26  | GOD OF WAR   | I      |
|      49 | SARTHAK | 2022-10-09  | SPIDER-MAN   | I      |
+---------+---------+-------------+--------------+--------+
5 rows in set (0.00 sec)

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

mysql> Create procedure assignment5(In rno1 int(3),name1 varchar(30))
   -> begin
   -> Declare i_date date;
   -> Declare diff int;
   -> Declare fine_amt int;
   -> DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'Table not found';
   -> select dateofissue into i_date from borrower where roll_no=rno1 and name=name1;
   -> select datediff(curdate(), i_date) into diff;
   -> If (diff>=15 and diff<=30)then
   -> SET fine_amt=diff*5;
   -> insert into fine values(rno1,curdate(),fine_amt);
   -> elseif (diff>30) then
   -> SET fine_amt=diff*50;
   -> insert into fine values(rno1,curdate(),fine_amt);
   -> End if;
   -> Update borrower set status='R' where roll_no=rno1 and name=name1;
   -> End;
   -> //
Query OK, 0 rows affected, 1 warning (0.01 sec)
mysql> call assignment5(45,'RAM');
   -> //
Query OK, 1 row affected (0.01 sec)

mysql> select * from fine;
   -> //
+---------+---------------+------+
| roll_no | date_of_return | amt  |
+---------+---------------+------+
|      45 | 2022-10-30    | 4500 |
+---------+---------------+------+
1 row in set (0.00 sec)

mysql> call assignment5(46,'ARYAN');
   -> //
Query OK, 1 row affected (0.01 sec)

mysql> select * from fine;
   -> //
+---------+---------------+------+
| roll_no | date_of_return | amt  |
+---------+---------------+------+
```

```
|     45 | 2022-10-30     | 4500 |
|     46 | 2022-10-30     |   75 |
+---------+----------------+------+
2 rows in set (0.00 sec)

mysql> call assignment5(47,'ROHAN');
   -> //
Query OK, 1 row affected (0.01 sec)

mysql> select * from fine;
   -> //
+---------+----------------+------+
| roll_no | date_of_return | amt  |
+---------+----------------+------+
|     45 | 2022-10-30     | 4500 |
|     46 | 2022-10-30     |   75 |
+---------+----------------+------+
2 rows in set (0.00 sec)

mysql> call assignment5(48,'SANKET');
   -> //
Query OK, 1 row affected (0.01 sec)

mysql> select * from fine;
   -> //
+---------+----------------+------+
| roll_no | date_of_return | amt  |
+---------+----------------+------+
|     45 | 2022-10-30     | 4500 |
|     46 | 2022-10-30     |   75 |
|     48 | 2022-10-30     | 1700 |
+---------+----------------+------+
3 rows in set (0.00 sec)

mysql> select * from borrower;;
   -> //
+---------+---------+-------------+--------------+--------+
| roll_no | name    | dateofissue | name_of_book | status |
+---------+---------+-------------+--------------+--------+
|     45 | RAM     | 2022-08-01  | HARRY POTTER | R      |
|     46 | ARYAN   | 2022-10-15  | DARK MATTER  | R      |
|     47 | ROHAN   | 2022-10-24  | SILENT HILL  | R      |
|     48 | SANKET  | 2022-09-26  | GOD OF WAR   | R      |
|     49 | SARTHAK | 2022-10-09  | SPIDER-MAN   | I      |
+---------+---------+-------------+--------------+--------+
5 rows in set (0.00 sec)

mysql> call assignment5(49,'SARTHAK');
   -> //
Query OK, 1 row affected (0.01 sec)

mysql> select * from borrower;
   -> //
+---------+---------+-------------+--------------+--------+
| roll_no | name    | dateofissue | name_of_book | status |
```

```
+---------+---------+-------------+--------------+--------+
|     45 | RAM     | 2022-08-01  | HARRY POTTER | R      |
|     46 | ARYAN   | 2022-10-15  | DARK MATTER  | R      |
|     47 | ROHAN   | 2022-10-24  | SILENT HILL  | R      |
|     48 | SANKET  | 2022-09-26  | GOD OF WAR   | R      |
|     49 | SARTHAK | 2022-10-09  | SPIDER-MAN   | R      |
+---------+---------+-------------+--------------+--------+
5 rows in set (0.00 sec)


mysql> select * from fine;
   -> //
+---------+---------------+------+
| roll_no | date_of_return | amt  |
+---------+---------------+------+
|     45 | 2022-10-30    | 4500 |
|     46 | 2022-10-30    |   75 |
|     48 | 2022-10-30    | 1700 |
|     49 | 2022-10-30    |  105 |
+---------+---------------+------+
4 rows in set (0.00 sec)
```

Conclusion:

In this assignment we have studied about PLSQL block and implemented the named PLSQL block for given problem statement using MySQL.

ASSIGNMENT NO: 6

Title: Named PL/SQL Block: PL/SQL Stored Procedure and Stored Function.

Problem Statement:

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 marks899and 825 category is Higher Second Class.
Write a PL/SQLblock to use procedure created with above requirement.
Stud_Marks(name, total_marks)
Result(Roll,Name, Class)
Write stored procedure and Function in line with above statement.

Theory:

Function:

A function is similar to a stored procedure but returns a value. Functions can be used in SQL statements as expressions. Functions is a standalone PL/SQL subprogram. Like PL/SQL procedure, functions has a unique name by which it can be referred. These are stored as PL/SQL database objects. Below are some of the characteristics of functions.

- Functions are a standalone block that is mainly used for calculation purpose.
- Function use RETURN keyword to return the value, and the datatype of this is defined at the time of creation.
- Function should either return a value or raise the exception, i.e. return is mandatory in functions.
- Function with no DML statements can be directly called in SELECT query whereas the function with DML operation can only be called from other PL/SQL blocks.
- It can have nested blocks, or it can be defined and nested inside the other blocks or packages.
- It contains declaration part (optional), execution part, exception handling part (optional).
- The values can be passed into the function or fetched from the procedure through the parameters.
- These parameters should be included in the calling statement.
- Function can also return the value through OUT parameters other than using RETURN.
- Since it will always return the value, in calling statement it always accompany with assignment operator to populate the variables.

Syntax:

```
CREATE FUNCTION function_name(param1,param2,…)
RETURNS datatype
```

```
BEGIN
    Statements
END
```

Example of creating a function:

```
CREATE OR REPLACE FUNCTION my_function(p_param1 NUMBER, p_param2 NUMBER)
RETURNS NUMBER
BEGIN
  -- PL/SQL code here
END my_function;
```

Example:1

```
mysql> create function fun() returns float
  -> deterministic
  -> begin
  -> DECLARE c FLOAT;
  -> SET c = 5 * 2;
  -> return c;
  -> end;
  -> //
Query OK, 0 rows affected (0.35 sec)

mysql> select fun();
  -> //
+-------+
| fun() |
+-------+
|    10 |
+-------+
1 row in set (0.01 sec)
```

Example:2

```
DELIMITER //
CREATE FUNCTION CalculateRectangleArea(length float, width float )RETURNS float
DETERMINISTIC
BEGIN
  DECLARE area float;
  SET area = length * width;
  RETURN area;
END;
//
```

```
mysql> select CalculateRectangleArea(2,5)
   -> //
+-----------------------------+
| CalculateRectangleArea(2,5) |
+-----------------------------+
|                          10 |
+-----------------------------+
```

Example:3

```
DELIMITER //
CREATE FUNCTION CalculateSquare(number INT) RETURNS INT
DETERMINISTIC
NO SQL
BEGIN
   DECLARE result INT;
   SET result = number * number;
   RETURN result;
END;
Query OK, 0 rows affected (0.39 sec)

mysql> SELECT CalculateSquare(5);
   -> //
+--------------------+
| CalculateSquare(5) |
+--------------------+
|                 25 |
+--------------------+
1 row in set (0.00 sec)
```

Similarities between Procedure and Function

- Both can be called from other PL/SQL blocks.
- If the exception raised in the subprogram is not handled in the subprogram exception handling section, then it will propagate to the calling block.
- Both can have as many parameters as required.
- Both are treated as database objects in PL/SQL.

Difference between Procedure and Function

Procedure	Function

• Used mainly to execute certain process	• Used mainly to perform some calculation
• Cannot called in SELECT statement	• Function that contain no DML statements can be called in SELECT statement
• Use OUT parameter to return the value	• Use RETURN to return the value
• It is not mandatory to return the value	• It is mandatory to return the value
• RETURN will simply exit the control from subprogram.	• RETURN will exit the control from subprogram and also returns the value
• Return datatype will not be specified at the time of creation	• Return datatype is mandatory at the time of creation

Assignment 6: Solution

Procedure

```
mysql> create database marks;
Query OK, 1 row affected (0.41 sec)
mysql> use marks;
Database changed

mysql> create table studmarks(name varchar(20),total_marks integer);
Query OK, 0 rows affected (1.81 sec)

mysql> create table result(roll_no integer,name varchar(20),class varchar(25));
Query OK, 0 rows affected (0.77 sec)

mysql> select * from studmarks;
Empty set (0.05 sec)

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

mysql> delimiter //
mysql>  create procedure proc_grade(in rollno tinyint, in name varchar(15), in marks int)
   ->     begin
   ->     declare class varchar(25);
   ->     if marks>=990 and marks<=1500 then set class="Distinction";
   ->     elseif marks<=989 and marks>=900 then set class="First Class";
   ->     elseif marks<=899 and marks>=825 then set class="Second Class";
   ->     elseif marks<=824 and marks>=700 then set class="Pass";
   ->     else
   ->     set class="Fail";
   ->     end if;
   ->     insert into studmarks values(name,marks);
   ->     insert into result values(rollno,name,class);
   ->     end
   -> //
Query OK, 0 rows affected (0.27 sec)
mysql> call proc_grade(1,"Aryan",850);
   -> //
Query OK, 1 row affected (0.24 sec)
```

```
mysql> select * from studmarks;
   -> //
+-------+-------------+
| name  | total_marks |
+-------+-------------+
| Aryan |         850 |
+-------+-------------+
1 row in set (0.00 sec)

mysql> select * from result;
   -> //
+---------+-------+--------------+
| roll_no | name  | class        |
+---------+-------+--------------+
|       1 | Aryan | Second Class |
+---------+-------+--------------+
1 row in set (0.00 sec)

mysql> call proc_grade(2,"Peter",1000);//
Query OK, 1 row affected (0.19 sec)

mysql> select * from studmarks;//
+-------+-------------+
| name  | total_marks |
+-------+-------------+
| Aryan |         850 |
| Peter |        1000 |
+-------+-------------+
2 rows in set (0.00 sec)

mysql> select * from result;//
+---------+-------+--------------+
| roll_no | name  | class        |
+---------+-------+--------------+
|       1 | Aryan | Second Class |
|       2 | Peter | Distinction  |
+---------+-------+--------------+
2 rows in set (0.00 sec)
```

Function:

```
DELIMITER //
CREATE FUNCTION grade(rollno TINYINT, name VARCHAR(15), marks INT) RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
   DECLARE class VARCHAR(20);
   IF marks >= 990 AND marks <= 1500 THEN
      SET class = 'Distinction';
   ELSEIF marks <= 989 AND marks >= 900 THEN
      SET class = 'First Class';
   ELSEIF marks <= 899 AND marks >= 825 THEN
      SET class = 'Second Class';
   ELSEIF marks <= 824 AND marks >= 700 THEN
      SET class = 'Pass';
   ELSE
      SET class = 'Fail';
   END IF;
   RETURN class;
END;
//
Query OK, 0 rows affected (0.06 sec)

mysql>
mysql> DELIMITER ;
mysql> select grade(101,'John',1300);
+------------------------+
| grade(101,'John',1300) |
+------------------------+
| Distinction            |
+------------------------+
1 row in set (0.00 sec)
```

Conclusion

In this assignment we have studied how to create PL/SQL Functions and Procedures and implemented the same to solve given problem statement.

ASSIGNMENT NO: 7

Title: Cursors.

Problem Statement:

Write a PL/SQL block of code using parameterized Cursor that will merge the data available in the newly created table N_Roll Call 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. Write PL/SQL block using all types of Cursors in line with above statement**.**

Theory:

Cursor:

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor.
A cursor holds the rows (one or more) returned by a SQL statement.
The set of rows the cursor holds is referred to as the active set.
To handle a result set inside a stored procedure, you use a cursor.
A cursor allows you to iterate a set of rows returned by a query and process each row accordingly.

Implicit Cursors

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

information in it.

- Implicit Cursor is associated with following DML Statements
  - **INSERT,**
  - **UPDATE and**
  - **DELETE**
- In PL/SQL, you can refer to the most recent implicit cursor as the

SQL cursor, which always has attributes such as

- **%FOUND,**
- **%ISOPEN,**
- **%NOTFOUND**, and
- **%ROWCOUNT**.

Explicit Cursors:

- Explicit cursors are programmer-defined cursors for gaining more control over the **context area**.

- An explicit cursor should be defined in the declaration section of the PL/SQL Block.
- It is created on a **SELECT Statement** which returns more than one row.
- Working with an explicit cursor includes the following steps –
    - ***Declaring the cursor for initializing the memory***
    - ***Opening the cursor for allocating the memory***
    - ***Fetching the cursor for retrieving the data***
    - ***Closing the cursor to release the allocated memory***

Step for Using Cursor

- Declare cursor
- Open cursor Loop
- Fetch data from cursor Exit loop
- Close cursor

1. DECLARE statement

- Syntax

    DECLARE cursor_name CURSOR FOR SELECT_statement;

- Explanation
    - The cursor declaration must be after any variable declaration.
    - A cursor must always be associated with a SELECT statement.

2. OPEN statement

- Syntax

    OPEN cursor_name;

- Explanation
    - The OPEN statement initializes the result set for the cursor, therefore, you must call the OPEN statement before fetching rows from the result set.

3. LOOP statement:

- Syntax

    [label_name :] **LOOP**
    statement_list
    END LOOP ***[label_name]***

- Explanation
    - The LOOP loop depends on the careful placement of the LEAVE statement to terminate iteration.

4. FETCH statement:

- Syntax

FETCH cursor_name INTO variables list;

- Explanation
  - Then, you use the FETCH statement to retrieve the next row pointed by the cursor and move the cursor to the next row in the result set.

5. CLOSE statement:

- Syntax

  CLOSE cursor_name;

- Explanation
  - Finally, you call the CLOSE statement to deactivate the cursor and release the memory associated with it

NOT FOUND handler:

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.

- Syntax

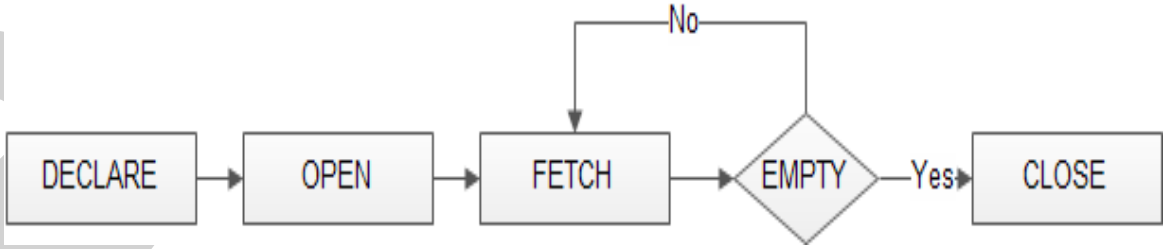
  DECLARE CONTINUE HANDLER FOR NOT FOUND

  SET exit_loop = TRUE

- Explanation

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.

MySQL cursor working:



Example:

```
DELIMITER //

CREATE PROCEDURE CursorExample11()
```

```
BEGIN
   -- Declare variables to hold account data
   DECLARE done INT DEFAULT 0;
   DECLARE no INT;
   DECLARE b_name VARCHAR(255);

   -- Declare a cursor for selecting employee data
   DECLARE cur CURSOR FOR SELECT acc_no, branch_name FROM account;

   -- Handler to detect the end of the result set
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

   -- Open the cursor
   OPEN cur;

   -- Start looping through the result set
   read_loop: LOOP
      FETCH cur INTO no, b_name;
      IF done = 1 THEN
         LEAVE read_loop; -- Exit the loop when there are no more rows
      END IF;

      -- Process the data, e.g., print the details
      SELECT no,b_name;
   END LOOP;

   -- Close the cursor
   CLOSE cur;
END;
//
```

ASSIGNMENT NO 7 : Solution

IMPLICIT CURSOR (Using Oracle SQL)

```
SQL> create table o_rollcall(rno int primary key,name varchar(20),addr varchar2  (20));
Table created.

SQL> insert into o_rollcall values(1,'ppp','Pune');
1 row created.
SQL> insert into o_rollcall values(2,'qqq','Pune');
1 row created.
SQL> insert into o_rollcall values(3,'rrr','Nasik');
1 row created.

SQL> select * from O_rollcall;

      RNO NAME                 ADDR
---------- -------------------- --------------------
         1 ppp                  Pune
         2 qqq                  Pune
         3 rrr                  Nasik



SQL> set serveroutput on;
SQL> DECLARE
  2  total_rows number(2);
  3  BEGIN
  4  UPDATE o_rollcall
  5  SET addr = 'Delhi';
  6  IF sql%notfound THEN
  7  dbms_output.put_line('no records selected');
  8  ELSIF sql%found THEN
  9  total_rows := sql%rowcount;
 10  dbms_output.put_line( total_rows || ' student records selected ');
 11    END IF;
 12  END;
 13  /
3 student records selected

PL/SQL procedure successfully completed.
```

SQL> select * from O_rollcall;

```
     RNO NAME            ADDR
---------- -------------------- --------------------
         1 ppp              Delhi
         2 qqq              Delhi
         3 rrr              Delhi
```

EXPLICIT CURSOR (Using MySQL)

```
mysql> create database cursordb;
Query OK, 1 row affected (0.01 sec)
mysql> use cursordb;
Database changed
mysql> create table o_rollcall(rno int primary key,name varchar(20),addr varchar(20));
Query OK, 0 rows affected (0.02 sec)

mysql> insert into o_rollcall values(1,'AAA','Pune');
Query OK, 1 row affected (0.01 sec)

mysql> insert into o_rollcall values(2,'BBB','Pune');
Query OK, 1 row affected (0.01 sec)

mysql> insert into o_rollcall values(3,'CCC','Nasik');
Query OK, 1 row affected (0.00 sec)

mysql>  create table n_rollcall(rno int primary key,name varchar(20),addr varchar (20));
Query OK, 0 rows affected (0.02 sec)

mysql> insert into n_rollcall values(1,'AAA','Pune');
Query OK, 1 row affected (0.01 sec)

mysql> select * from o_rollcall;
+-----+------+-------+
| rno | name | addr  |
+-----+------+-------+
|   1 | AAA  | Pune  |
|   2 | BBB  | Pune  |
|   3 | CCC  | Nasik |
+-----+------+-------+
3 rows in set (0.00 sec)

mysql> select * from n_rollcall;
+-----+------+------+
```

```
| rno | name | addr |
+-----+------+------+
|   1 | AAA  | Pune |
+-----+------+------+
1 row in set (0.00 sec)

mysql> delimiter //
mysql> create procedure Assignment7(IN rno1 int)
   -> begin
   -> DECLARE c1 CURSOR FOR SELECT rno from o_rollcall where rno=rno1;
   -> OPEN c1;
   -> FETCH c1 into rno1;
   -> if not exists(select * from n_rollcall where rno=rno1) then insert into n_rollcall
   -> select * from o_rollcall where rno=rno1;
   -> end if;
   -> CLOSE c1;
   -> END
   -> ;
   -> //
Query OK, 0 rows affected (0.01 sec)

mysql> call Assignment7(2);
   -> //
Query OK, 1 row affected (0.01 sec)

mysql> select * from n_rollcall;
   -> //
+-----+------+------+
| rno | name | addr |
+-----+------+------+
|   1 | AAA  | Pune |
|   2 | BBB  | Pune |
+-----+------+------+
2 rows in set (0.00 sec)

mysql> call Assignment7(1);
   -> //
Query OK, 0 rows affected (0.00 sec)

mysql> select * from n_rollcall;
   -> //
+-----+------+------+
| rno | name | addr |
```

```
+-----+------+------+
|   1 | AAA  | Pune |
|   2 | BBB  | Pune |
+-----+------+------+
2 rows in set (0.00 sec)

mysql> call Assignment7(3);
   -> //
Query OK, 1 row affected (0.01 sec)

mysql> select * from n_rollcall;
   -> //
+-----+------+-------+
| rno | name | addr  |
+-----+------+-------+
|   1 | AAA  | Pune  |
|   2 | BBB  | Pune  |
|   3 | CCC  | Nasik |
+-----+------+-------+
3 rows in set (0.00 sec)
```

ASSIGNMENT NO: 8

Title: Database Trigger.

Problem Statement:

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.

Write PL/SQLblock for all types of Triggers in line with above statement.

Theory:

Trigger:

A trigger is a named PL/SQL block that is automatically executed in response to a specific event, such as an INSERT, UPDATE, or DELETE operation on a table. Triggers are used to enforce data integrity and implement complex rules.

- A SQL trigger is a set of SQL statements stored in the database catalog.
- A SQL trigger is executed or fired whenever an event associated with a table occurs e.g., insert, update or delete.
- A SQL trigger is a special type of stored procedure.
- It is special because it is not called directly like a stored procedure.
- The main difference between a trigger and a stored procedure is that a trigger is called automatically when a data modification event is made against a table whereas a stored procedure must be called explicitly.

Advantages oftriggers:

- SQL triggers provide an alternative way to check the integrity of data.
- SQL triggers can catch errors in business logic in the database layer. SQL triggers provide an alternative way to run scheduled tasks.
- By using SQL triggers, you don't have to wait to run the scheduled tasks because the triggers are invoked automatically *before* or *after* a change is made to the data in the tables.
- SQL triggers are very useful to audit the changes of data in tables.

Disadvantages of triggers:

- SQL triggers only can provide an extended validation and they cannot replace all the validations.
- SQL triggers are invoked and executed invisible from the client applications, therefore, it is difficult to figure out what happen in the database layer.
- SQL triggers may increase the overhead of the database server.

Oracle Triggers types:

1. Row-Level Triggers

- Row-level triggers execute once for each row in a transaction.
- Row-level triggers are created using the for each row clause in the create trigger command.
- For instance if we insert in a single transaction 20 rows to the table EMPLOYEE, the trigger is executed 20 times.

2. Statement-Level Triggers

- Statement-level triggers execute once for each transaction.
- When we insert in one transaction 20 rows to EMPLOYEE table , then statement-level trigger is executed only once.

MySQL Triggers types:

- BEFORE INSERT – activated before data is inserted into the table.
- AFTER INSERT – activated after data is inserted into the table.
- BEFORE UPDATE – activated before data in the table is updated.
- AFTER UPDATE – activated after data in the table is updated.
- BEFORE DELETE – activated before data is removed from the table.
- AFTER DELETE – activated after data is removed from the table.

Trigger Syntax:

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

Example of creating a trigger:

```
CREATE OR REPLACE TRIGGER my_trigger
BEFORE INSERT ON my_table
FOR EACH ROW
```

```
BEGIN
   -- PL/SQL code here
END my_trigger;
```

Assignment No 8: Solution

```
mysql> CREATE TABLE Library(BookId INT NOT NULL, BookName VARCHAR(30) NOT NULL, BookEdition INT NOT NULL, BookQuantity INT NOT NULL, PRIMARY KEY(BookId));
Query OK, 0 rows affected (2.88 sec)

mysql> INSERT INTO Library VALUES (1, "Wings of Fire", 2, 15);
Query OK, 1 row affected (0.25 sec)

mysql> INSERT INTO Library VALUES (2, "Three Men in a Boat", 3, 10);
Query OK, 1 row affected (0.08 sec)

mysql> CREATE TABLE Library_Audit(BookId INT NOT NULL, BookName VARCHAR(30) NOT NULL, BookEdition INT NOT NULL, BookQuantity INT NOT NULL, TypeOfOperation VARCHAR(10), ModifiedBy VARCHAR(30), ModifiedDateTime DATETIME);
Query OK, 0 rows affected (0.47 sec)
```

Trigger 1:

```
mysql> CREATE TRIGGER AfterUpdateTrigger
   -> AFTER UPDATE ON Library FOR EACH ROW
   -> BEGIN
   -> INSERT INTO Library_Audit VALUES (OLD.BookId, OLD.BookName, OLD.BookEdition,
OLD.BookQuantity, "Update", CURRENT_USER(), CURRENT_TIMESTAMP());
   -> END //
Query OK, 0 rows affected (0.11 sec)

mysql> SELECT * FROM Library;
   -> //
+--------+--------------------+-------------+--------------+
| BookId | BookName           | BookEdition | BookQuantity |
+--------+--------------------+-------------+--------------+
|      1 | Wings of Fire      |           2 |           15 |
|      2 | Three Men in a Boat |           3 |           10 |
+--------+--------------------+-------------+--------------+
2 rows in set (0.00 sec)

mysql> delimiter ;
```

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

mysql> UPDATE Library SET BookQuantity = 10 WHERE BookId = 1;
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM Library_Audit;
+--------+--------------+-------------+--------------+-----------------+---------------+---------------------+
| BookId | BookName     | BookEdition | BookQuantity | TypeOfOperation | ModifiedBy    | ModifiedDateTime    |
+--------+--------------+-------------+--------------+-----------------+---------------+---------------------+
|      1 | Wings of Fire |           2 |           15 | Update          | root@localhost | 2023-09-29 10:45:50 |
+--------+--------------+-------------+--------------+-----------------+---------------+---------------------+
1 row in set (0.00 sec)
```

Trigger 2:

```
mysql> delimiter //
mysql> CREATE TRIGGER BeforeDeleteTrigger
   -> BEFORE DELETE
   -> ON Library FOR EACH ROW
   -> BEGIN
   -> INSERT INTO Library_Audit VALUES (OLD.BookId, OLD.BookName, OLD.BookEdition, OLD.BookQuantity, "Delete", CURRENT_USER(), CURRENT_TIMESTAMP());
   -> END //
Query OK, 0 rows affected (0.13 sec)

mysql> SELECT * FROM Library;
   -> //
+--------+--------------------+-------------+--------------+
| BookId | BookName           | BookEdition | BookQuantity |
+--------+--------------------+-------------+--------------+
|      1 | Wings of Fire      |           2 |           10 |
|      2 | Three Men in a Boat |           3 |           10 |
+--------+--------------------+-------------+--------------+
2 rows in set (0.03 sec)

mysql> SELECT * FROM Library_Audit;
```

```
   -> //
+--------+---------------+-------------+--------------+-----------------+---------------+---------------------+
| BookId | BookName      | BookEdition | BookQuantity | TypeOfOperation | ModifiedBy    | ModifiedDateTime    |
+--------+---------------+-------------+--------------+-----------------+---------------+---------------------+
|      1 | Wings of Fire |           2 |           15 | Update          | root@localhost | 2023-09-29 10:45:50 |
+--------+---------------+-------------+--------------+-----------------+---------------+---------------------+
1 row in set (0.00 sec)

mysql> DELETE FROM Library WHERE BookId = 1;//
Query OK, 1 row affected (0.14 sec)

mysql> SELECT * FROM Library_Audit;
   -> //
+--------+---------------+-------------+--------------+-----------------+---------------+---------------------+
| BookId | BookName      | BookEdition | BookQuantity | TypeOfOperation | ModifiedBy    | ModifiedDateTime    |
+--------+---------------+-------------+--------------+-----------------+---------------+---------------------+
|      1 | Wings of Fire |           2 |           15 | Update          | root@localhost | 2023-09-29 10:45:50 |
|      1 | Wings of Fire |           2 |           10 | Delete          | root@localhost | 2023-09-29 10:56:22 |
+--------+---------------+-------------+--------------+-----------------+---------------+---------------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM Library;//
+--------+--------------------+-------------+--------------+
| BookId | BookName           | BookEdition | BookQuantity |
+--------+--------------------+-------------+--------------+
|      2 | Three Men in a Boat |           3 |           10 |
+--------+--------------------+-------------+--------------+
1 row in set (0.00 sec)
```

ASSIGNMENT NO: 9

Title: Database Connectivity.

Problem Statement:

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

Theory:

Database connectivity in a software application refers to the ability of the application to connect to a database, perform operations like reading, writing, updating, and deleting data, and manage the interaction with the database. The database can be of various types, such as relational databases (e.g., MySQL, PostgreSQL, SQL Server), NoSQL databases (e.g., MongoDB, Cassandra), or other data storage systems.

Here are the general steps to achieve database connectivity in a typical software application:

- Select a Database
- Database Driver/Connector
- Connection String/Configuration
- Establish a Connection
- Execute SQL or Query
- Handle Exceptions
- Close the Connection

Arguments required to connect

You need to know the following detail of the MySQL server to perform the connection fromPython.

Argument	Description
Username	The username that you use to work with MySQL Server. The default username for the MySQL database is a **root**.
Password	Password is given by the user at the time of installing the MySQL server. If you are using root then you won't need the password.
Host name	The server name or Ip address on which MySQL is running. if you are running on localhost, then you can use **localhost** or its IP 127.0.0.0

Databas ename	The name of the database to which you want to connect and perform the operations.

How to Connect to MySQL Database in Python

1. **Install MySQL connector module**

   Use the pip command to install MySQL connector Python.pip install mysql-connector-python

2. **Import MySQL connector module**

   Import using a import mysql.connector statement so you can use this module's methods tocommunicate with the MySQL database.

3. **Use the connect() method**

   Use the connect() method of the MySQL Connector class with the required arguments to connect MySQL. It would return a MySQLConnection object if the connection establishedsuccessfully

4. **Use the cursor() method**

   Use the cursor() method of a MySQLConnection object to create a cursor object to performvarious SQL operations.

5. **Use the execute() method**

   The execute() methods run the SQL query and return the result.

6. **Extract result using fetchall()**

   Use cursor.fetchall() or fetchone() or fetchmany() to read query result.

7. **Close cursor and connection objects**

   use cursor.clsoe() and connection.clsoe() method to close open connections after your workcompletes

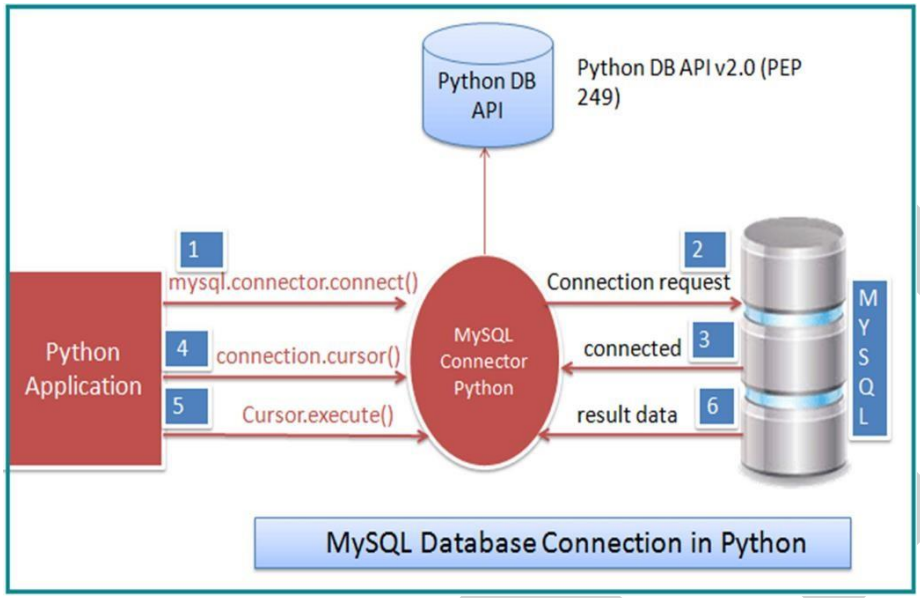


Figure 1: MySQL database connection in Python

Assignment No 9 : Solution

```
pip install mysql-connector-python
import mysql.connector
from mysql.connector import Error
try:
   connection =
mysql.connector.connect(host='localhost',database='library_system',user='root',password='root')
   if connection.is_connected():
      db_Info = connection.get_server_info()
      print("Connected to MySQL Server version ", db_Info)
      cursor = connection.cursor()
      cursor.execute("select database();")
      record = cursor.fetchone()
      print("You're connected to database: ", record)
      cursor.execute("select * from library;")
      record = cursor.fetchall()
      print("Details of library table are: ")
      print("b_id | title | author | edition | noc")
      for x in record:
         print(x)
      f = True
      while(f == True):
         print("\n1.Add\n2.Update\n3.Delete\n4.display\n")
         choice = int(input("Enter your choice: "))
         if choice == 1:
            print("\nEnter b_id | title | author | edition | noc")
            b_id = int(input("Enter b_id "))
            title = input("Enter title ")
            author = input("Enter author ")
            edition = int(input("Enter edition "))
            noc = int(input("Enter noc "))
            sql = "INSERT INTO library (b_id,title,author,edition,noc) VALUES (%s, %s,%s,%s,%s)"
            val = (b_id,title,author,edition,noc)
            cursor.execute(sql,val)
            connection.commit()
            print("Insert Successful!!")
            print(cursor.rowcount, "record(s) inserted!")

         elif choice == 2:
            title = input("Enter title of the book ...")
            noc = int(input("Enter noc "))
            val=(noc,title)
```

```
            sql = "UPDATE library SET noc = %s WHERE title = %s"
            cursor.execute(sql,val)
            connection.commit()
            print("Update Successful!!")
            print(cursor.rowcount, "record(s) affected")

        elif choice == 3:
            b_id = int(input("Enter ID of the book to be deleted..."))
            sql = "DELETE FROM library WHERE b_id = %s"
            val = (b_id,)
            cursor.execute(sql,val)
            connection.commit()
            print("Delete Successful!!")
            print(cursor.rowcount, "record(s) deleted!")


        elif choice == 4:
            cursor.execute("select * from library;")
            record = cursor.fetchall()
            print("Details of library table are: ")
            print("b_id | title | author | edition | noc")
            for x in record:
                print(x)
        ch =input("\n Do you want to continue?? (Y / N)")
        if ch != 'Y':
            f = False
except Error as e:
    print("Error while connecting to MySQL", e)
finally:
    if connection.is_connected():
        cursor.close()
        connection.close()
        print("MySQL connection is closed")
```

Output :

```
Connected to MySQL Server version  8.0.30
You're connected to database:  ('library_system',)
Details of library table are:
b_id | title | author | edition | noc
(1, 'TOC', 'Auth1', 2, 5)
(2, 'DBMS', 'Auth2', 3, 10)
(3, 'CN', 'Auth3', 5, 8)
(4, 'SPOS', 'Auth4', 5, 8)
```

(5, 'IOT', 'Auth5', 1, 4)
(6, 'DS', 'Auth6', 8, 35)

1.Add
2.Update
3.Delete
4.display

Enter your choice: 1

Enter b_id | title | author | edition | noc
Enter b_id 7
Enter title ML
Enter author Auth7
Enter edition 10
Enter noc 25
Insert Successful!!
1 record(s) inserted!

Do you want to continue?? (Y / N)Y

1.Add
2.Update
3.Delete
4.display

Enter your choice: 4
Details of library table are:
b_id | title | author | edition | noc
(1, 'TOC', 'Auth1', 2, 5)
(2, 'DBMS', 'Auth2', 3, 10)
(3, 'CN', 'Auth3', 5, 8)
(4, 'SPOS', 'Auth4', 5, 8)
(5, 'IOT', 'Auth5', 1, 4)
(6, 'DS', 'Auth6', 8, 35)
(7, 'ML', 'Auth7', 10, 25)

Do you want to continue?? (Y / N)Y

1.Add
2.Update
3.Delete
4.display

Enter your choice: 2

```
Enter title of the book ...ML
Enter noc 40
Update Successful!!
1 record(s) affected

 Do you want to continue?? (Y / N)Y

1.Add
2.Update
3.Delete
4.display

Enter your choice: 4
Details of library table are:
b_id | title | author | edition | noc
(1, 'TOC', 'Auth1', 2, 5)
(2, 'DBMS', 'Auth2', 3, 10)
(3, 'CN', 'Auth3', 5, 8)
(4, 'SPOS', 'Auth4', 5, 8)
(5, 'IOT', 'Auth5', 1, 4)
(6, 'DS', 'Auth6', 8, 35)
(7, 'ML', 'Auth7', 10, 40)

 Do you want to continue?? (Y / N)Y

1.Add
2.Update
3.Delete
4.display

Enter your choice: 3
Enter ID of the book to be deleted...6
Delete Successful!!
1 record(s) deleted!

 Do you want to continue?? (Y / N)Y

1.Add
2.Update
3.Delete
4.display

Enter your choice: 4
Details of library table are:
b_id | title | author | edition | noc
```

```
(1, 'TOC', 'Auth1', 2, 5)
(2, 'DBMS', 'Auth2', 3, 10)
(3, 'CN', 'Auth3', 5, 8)
(4, 'SPOS', 'Auth4', 5, 8)
(5, 'IOT', 'Auth5', 1, 4)
(7, 'ML', 'Auth7', 10, 40)

 Do you want to continue?? (Y / N)N
MySQL connection is closed
```

Conclusion :

In this assignment we have studied about database connectivity and successfully connected MySQL database with Python application.

ASSIGNMENT NO: 10

Title: MongoDB Queries.

Problem Statement:

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

Theory:

What is NoSQL?

NoSQL is a non-relational DBMS, that does not require a fixed schema, avoids joins, and is easy to scale. The purpose of using a NoSQL database is for distributed data stores with humongous data storage needs. NoSQL is used for Big data and real-time web apps. For example, companies like Twitter, Facebook, Google collect terabytes of user data every single day.

NoSQL database stands for "Not Only SQL" or "Not SQL." Though a better term would be "NoREL", NoSQL caught on. Carl Strozz introduced the NoSQL concept in 1998.

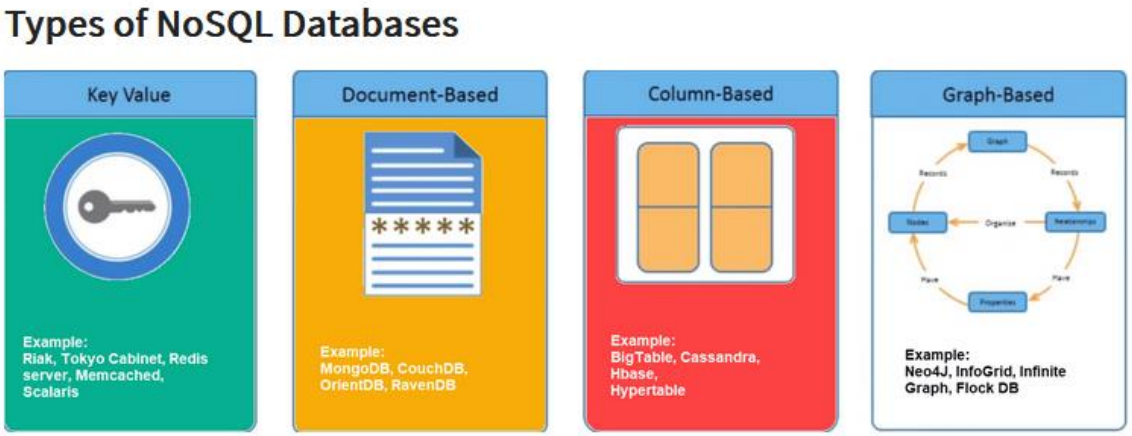
Why NoSQL?

The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data. The system response time becomes slow when you use RDBMS for massive volumes of data.

To resolve this problem, we could "scale up" our systems by upgrading our existing hardware. This process is expensive.

Difference Between SQL and NoSQL

SQL	NOSQL
Relational Database management system	Distributed Database management system
Vertically Scalable	Horizontally Scalable
Fixed or predifined Schema	Dynamic Schema
Not suitable for hierarchical data storage	Best suitable for hierarchical data storage
Can be used for complex queries	Not good for complex queries



Document-Oriented:

Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document. The document is stored in JSON or XML formats. The value is understood by the DB and can be queried.

MongoDB

Scalable High-Performance Open-source, Document-orientated database.

- Built for Speed
- Rich Document based queries for Easy readability.
- Full Index Support for High Performance.
- Replication and Failover for High Availability.
- Auto Sharding for Easy Scalability.
- Map / Reduce for Aggregation**.**

Advantages of MongoDB

- Schema less : Number of fields, content and size of the document can be differ from one document to another.
- No complex joins
- Data is stored as JSON style
- Index on any attribute
- Replication and High availability

Mongo DB Terminologies for RDBMS concepts

RDBMS	**MongoDB**
Database	**Database**

Table, View	Collection
Row	**Document (JSON, BSON)**
Column	**Field**
Index	**Index**
Join	**Embedded Document**
Foreign Key	**Reference**
Partition	**Shard**

Data Types of MongoDB

- String : This is most commonly used datatype to store the data. String in mongodb must be UTF-8 valid.
- Integer : This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- Boolean : This type is used to store a boolean (true/ false) value.
- Double : This type is used to store floating point values.
- Min/ Max keys : This type is used to compare a value against the lowest and highest BSON elements.
- Arrays : This type is used to store arrays or list or multiple values into one key.
- Timestamp : ctimestamp. This can be handy for recording when a document has been modified or added.
- Object : This datatype is used for embedded documents.
- Null : This type is used to store a Null value.
- Symbol : This datatype is used identically to a string however, it's generally reserved for languages that use a specific symbol type.
- Date : This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- Object ID : This datatype is used to store the document's ID.
- Binary data : This datatype is used to store binay data.
- Code : This datatype is used to store javascript code into document.
- Regular expression : This datatype is used to store regular expression

Basic Database Operations

- use *<database name>*
  switched to database provided with command

- db
  To check currently selected database use the command db
- show dbs
  Displays the list of databases
- db.dropDatabase()
  To Drop the database

- db.createCollection (name)
- Ex:- db.createCollection(Stud)
  - To create collection
- >show collections
  - List out all names of collection in current database
- db.*databasename*.insert
- ({Key : Value})
- Ex:- db.Stud.insert({{Name:"Jiya"})
  - In mongodb you don't need to create collection. MongoDB creates collection automatically, when you insert some document.
- db.collection.drop() Example:- db.Stud.drop()

  MongoDB's db.collection.drop() is used to drop a collection from the database.

CRUD Operations:

- Insert
- Find
- Update
- Delete

CRUD Operations – Insert

The insert() Method:- To insert data into MongoDB collection, you need to use MongoDB's insert() or save()method.

Syntax

>db.COLLECTION_NAME.insert(document)

Example

>db.stud.insert({name: "Jiya", age:15})

_id Field

- If the document does not specify an *_id* field, then MongoDB will add the _id field and assign a unique *ObjectId* for the document before inserting.
- The _id value must be unique within the collection to avoid duplicate key error.

Insert a Document without Specifying an _id Field

- db.stud.insert({ Name : “Reena", Rno: 15 })
- db.stud.find()
  { "_id" : "5063114bd386d8fadbd6b004”, “Name” : “Reena", “Rno”: 15 }

Insert a Document Specifying an _id Field

- db.stud.insert({ _id: 10, Name : “Reena", Rno: 15 })
- db.stud.find()

{ "_id" : 10, “Name” : “Reena", “Rno”: 15 }

Insert Single Documents

db.stud.insert ({Name: “Ankit”, Rno:1, Address: “Pune”})

Insert Multiple Documents

```
db.stud.insert ( [
{ Name: “Ankit”, Rno:1, Address: “Pune”} ,
{ Name: “Sagar”, Rno:2},
{ Name: “Neha”, Rno:3}
] )
```

Insert Multicolumn attribute

```
db.stud.insert( {
   Name: “Ritu",
   Address: { City: “Pune", State: “MH” },
  Rno: 6
 })
```

Insert Multivalued attribute

```
db.stud.insert( {
    Name : “Sneha",
    Hobbies: [“Singing”, “Dancing” , “Cricket”] ,
    Rno:8
 })
```

Insert Multivalued with Multicolumn attribute

```
db.stud.insert(  {
    Name : “Sneha",
    Awards: [ { Award : “Dancing”, Rank: “1st”, Year: 2008 },
                {Award : “Drawing”, Rank: “3rd”, Year: 2010 } ,
                {Award : “Singing”, Rank: “1st”, Year: 2015 } ],
   Rno: 9  })
```

CRUD Operations – **Find**

The find() Method- To display data from MongoDB collection. Displays all the documents in a non structured way.

Syntax

>db.COLLECTION_NAME.find()

The pretty() Method- To display the results in a formatted way, you can use **pretty()** method.

Syntax

>db. COLLECTION_NAME.find().pretty()

Specify Equality Condition

use the query document { <field>: <value> }

Examples:

- db.stud.find(name: “Jiya" })
- db.stud.find({ _id: 5 })

Comparison Operators

Operator	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	values that are greater than or equal to a specified value.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$in	Matches any of the values specified in an array.
$nin	Matches none of the values specified in an array.

Find Examples with comparison operators

- db.stud.find({ rno: { $gt:5} }) *Shows all documents whose rno>5*
- db.stud.find({ rno: { $gt: 0, $lt: 5} }) *Shows all documents whose rno greater than 0 and less than 5*

Examples to show only particular columns

- db.stud.find({name: “Jiya”},{Rno:1}) *To show the rollno of student whose name is equal to Jiya (by default _id is also shown)*
- db.stud.find({name: “jiya”},{_id:0,Rno:1}) *show the rollno of student whose name is equal to Jiya (_id is not shown)*

Examples for Sort function

- db.stud.find().sort({ Rno: 1 })
  Sort on age field in Ascending order (1)
- db.stud.find().sort({ Rno: -1 })
  Sort on age field in Ascending order(-1)

Examples of Count functions

- db.stud.find().count()
  Returns no of documents in the collection

Examples of limit and skip

- db.stud.find().limit(2)
  Returns only first 2 documents
- db.stud.find().skip(5)
  Returns all documents except first 5 documents

CRUD Operations – Update

Syntax

db.*CollectionName*.update (
<query/Condition>,
<update with $set or $unset>,
{
upsert: <boolean>,
multi: <boolean>,
})

upsert

- If set to *True*, creates new document if no matches found.

multi

- If set to *True*, updates multiple documents that matches the query criteria

CRUD Operations – Update Examples

1**>** Set age = 25 where id is 100, First Whole document is replaced where condition is matched and only one field is remained as age:25

db.stud.update(
{ _id: 100 },

{ age: 25})

2> Set age = 25 where id is 100, Only the age field of one document is updated where condition is matched .

```
db.stud.update(
{ _id: 100 },
{ $set:{age: 25}})
```

3> To remove a age column from single document where id=100

```
db.stud.update(
{ _id: 100 },
{ $unset:{age: 1}})
```

CRUD Operations – Remove

- **Remove All Documents**
  - db.inventory.remove({})
- **Remove All Documents that Match a Condition**
  - db.inventory.remove ({ type : "food" })
- **Remove a Single Document that Matches a Condition**
  - db.inventory.remove ({ type : "food" }, 1)

db.collection.save()

The save() returns an object that contains the status of the operation. It returns: A WriteResult object that contains the status of the operation.

The save() method has the following form:

```
db.collection.save(
  <document>,
  {
    writeConcern: <document>
  }
)
```

Parameter	Type	Description
document	document	A document to save to the collection.
writeConcern	document	Optional. A document

		expressing the write concern.

Write Concern

The save() method uses either the insert or the update command, which use the default write concern. To specify a different write concern, include the write concern in the options parameter.

Insert

If the document does not contain an _id field, then the save() method calls the insert() method. During the operation, the mongo shell will create an ObjectId and assign it to the _id field.

Update

If the document contains an _id field, then the save() method is equivalent to an update with the upsert option set to true and the query predicate on the _id field.

db.collection.save() can be used inside multi-document transactions.

Save a New Document Specifying an _id Field

Example:

In the following example, save() performs an update with upsert:true since the document contains an _id field:

db.products.save({ _id: 100, item: "water", qty: 30 })

Because the _id field holds a value that does not exist in the collection, the update operation results in an insertion of the document. The results of these operations are identical to an update() method with the upsert option set to true.

The operation results in the following new document in the products collection:

{ "_id" : 100, "item" : "water", "qty" : 30 }

Starting in MongoDB 4.2, the db.collection.save() method is deprecated. Use db.collection.insertOne() or db.collection.replaceOne() instead.

Assignment No 10 :Solution

> show dbs
MyDB 0.000GB
admin 0.000GB
config 0.000GB
local 0.000GB
stud 0.000GB

1. Create database Institute.

> use Institute
switched to db Institute
> show dbs
MyDB 0.000GB
admin 0.000GB
config 0.000GB
local 0.000GB
stud 0.000GB

2. Create collection Students.

> db.createCollection("Students")
{ "ok" : 1 }
> show dbs
Institute 0.000GB
MyDB 0.000GB
admin 0.000GB
config 0.000GB
local 0.000GB
stud 0.000GB

3. Insert 10 documents with above mentioned structure.

> try {
... db.Students.insertMany
... ([
... {RollNo:1, StudName:"Ram", Age:20, Branch:"Computer", Address:{City: "Pune",State: " Maharashtra"}, Hobbies: ["Singing", "Dancing" , "Cricket"] },
... {RollNo:2, StudName:"Priya", Age:21, Branch:"Computer", Address:{City: "Pune",State: " Maharashtra"}, Hobbies: ["Singing", "Dancing"] }, {RollNo:3, StudName:"Raj", Age:22, Branch:"Civil", Address:{City: "Pune",State: " Maharashtra"}, Hobbies: ["Dancing" , "Cricket"] },{RollNo:4, StudName:"Muskan", Age:20, Branch:"E&TC", Address:{City: "Nasik",State: " Maharashtra"}, Hobbies: ["Painting", "Dancing"] }, {RollNo:5, StudName:"Reya", Age:24, Branch:"Computer", Address:{City: "Pune",State: " Maharashtra"}, Hobbies: ["Dancing" , "Reading"] }, {RollNo:6, StudName:"Pooja", Age:20, Branch:"Computer", Address:{City: "Pune",State: " Maharashtra"}, Hobbies: ["Singing", "Dancing"] }, {RollNo:7, StudName:"Teena", Age:19,

Branch:"Computer", Address:{City: "Pune",State: " Maharashtra"}, Hobbies: ["Dancing" , "Drawing"] }, {RollNo:8, StudName:"Roma", Age:20, Branch:"Computer", Address:{City: "Pune",State: " Maharashtra"}, Hobbies: ["Singing", "Cooking"] }, {RollNo:9, StudName:"Rohan", Age:23, Branch:"Computer", Address:{City: "Pune",State: " Maharashtra"}, Hobbies: ["Singing","Cricket"] }, {RollNo:10, StudName:"Aveer", Age:22, Branch:"Mechanical", Address:{City: "Pune",State: " Maharashtra"}, Hobbies: ["Dancing" , "Cricket"] }]);

... } catch (e) {

... print (e);

... }

```
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("636033bb521583da91f7c021"),
        ObjectId("636033bb521583da91f7c022"),
        ObjectId("636033bb521583da91f7c023"),
        ObjectId("636033bb521583da91f7c024"),
        ObjectId("636033bb521583da91f7c025"),
        ObjectId("636033bb521583da91f7c026"),
        ObjectId("636033bb521583da91f7c027"),
        ObjectId("636033bb521583da91f7c028"),
        ObjectId("636033bb521583da91f7c029"),
        ObjectId("636033bb521583da91f7c02a")
    ]
}
```

4. Display all students' information.

> db.Students.find().pretty()

```
{
    "_id" : ObjectId("636033bb521583da91f7c021"),
    "RollNo" : 1,
    "StudName" : "Ram",
    "Age" : 20,
    "Branch" : "Computer",
    "Address" : {
        "City" : "Pune",
        "State" : " Maharashtra"
    },
    "Hobbies" : [
        "Singing",
        "Dancing",
        "Cricket"
    ]
```

```
}
{
    "_id" : ObjectId("636033bb521583da91f7c022"),
    "RollNo" : 2,
    "StudName" : "Priya",
    "Age" : 21,
    "Branch" : "Computer",
    "Address" : {
        "City" : "Pune",
        "State" : " Maharashtra"
    },
    "Hobbies" : [
        "Singing",
        "Dancing"
    ]
}
{
    "_id" : ObjectId("636033bb521583da91f7c023"),
    "RollNo" : 3,
    "StudName" : "Raj",
    "Age" : 22,
    "Branch" : "Civil",
    "Address" : {
        "City" : "Pune",
        "State" : " Maharashtra"
    },
    "Hobbies" : [
        "Dancing",
        "Cricket"
    ]
}
{
    "_id" : ObjectId("636033bb521583da91f7c024"),
    "RollNo" : 4,
    "StudName" : "Muskan",
    "Age" : 20,
    "Branch" : "E&TC",
    "Address" : {
        "City" : "Nasik",
        "State" : " Maharashtra"
    },
    "Hobbies" : [
        "Painting",
```

```
            "Dancing"
        ]
}
{
        "_id" : ObjectId("636033bb521583da91f7c025"),
        "RollNo" : 5,
        "StudName" : "Reya",
        "Age" : 24,
        "Branch" : "Computer",
        "Address" : {
            "City" : "Pune",
            "State" : " Maharashtra"
        },
        "Hobbies" : [
            "Dancing",
            "Reading"
        ]
}
{
        "_id" : ObjectId("636033bb521583da91f7c026"),
        "RollNo" : 6,
        "StudName" : "Pooja",
        "Age" : 20,
        "Branch" : "Computer",
        "Address" : {
            "City" : "Pune",
            "State" : " Maharashtra"
        },
        "Hobbies" : [
            "Singing",
            "Dancing"
        ]
}
{
        "_id" : ObjectId("636033bb521583da91f7c027"),
        "RollNo" : 7,
        "StudName" : "Teena",
        "Age" : 19,
        "Branch" : "Computer",
        "Address" : {
            "City" : "Pune",
            "State" : " Maharashtra"
        },
```

```
        "Hobbies" : [
                "Dancing",
                "Drawing"
        ]
}
{
        "_id" : ObjectId("636033bb521583da91f7c028"),
        "RollNo" : 8,
        "StudName" : "Roma",
        "Age" : 20,
        "Branch" : "Computer",
        "Address" : {
                "City" : "Pune",
                "State" : " Maharashtra"
        },
        "Hobbies" : [
                "Singing",
                "Cooking"
        ]
}
{
        "_id" : ObjectId("636033bb521583da91f7c029"),
        "RollNo" : 9,
        "StudName" : "Rohan",
        "Age" : 23,
        "Branch" : "Computer",
        "Address" : {
                "City" : "Pune",
                "State" : " Maharashtra"
        },
        "Hobbies" : [
                "Singing",
                "Cricket"
        ]
}
{
        "_id" : ObjectId("636033bb521583da91f7c02a"),
        "RollNo" : 10,
        "StudName" : "Aveer",
        "Age" : 22,
        "Branch" : "Mechanical",
        "Address" : {
                "City" : "Pune",
```

```
            "State" : " Maharashtra"
        },
        "Hobbies" : [
            "Dancing",
            "Cricket"
        ]
}
```

> db.Students.find({ Age: { $gt:22} })

```
> db.Students.find( { Age: { $gt:22} } )
{ "_id" : ObjectId("636033bb521583da91f7c025"), "RollNo" : 5, "StudName" : "Reya", "Age" : 24, "Branch" :
"Computer", "Address" : { "City" : "Pune", "State" : " Maharashtra" }, "Hobbies" : [ "Dancing", "Reading" ] }
{ "_id" : ObjectId("636033bb521583da91f7c029"), "RollNo" : 9, "StudName" : "Rohan", "Age" : 23, "Branch" :
"Computer", "Address" : { "City" : "Pune", "State" : " Maharashtra" }, "Hobbies" : [ "Singing", "Cricket" ] }
>
```

6. Display Student information sorted on name field

> db. Students.find().sort({ StudName: 1 })

```
{ "_id" : ObjectId("636033bb521583da91f7c02a"), "RollNo" : 10, "StudName" : "Aveer", "Age" : 22, "Branch" :
"Mechanical", "Address" : { "City" : "Pune", "State" : " Maharashtra" }, "Hobbies" : [ "Dancing", "Cricket" ] }
{ "_id" : ObjectId("636033bb521583da91f7c024"), "RollNo" : 4, "StudName" : "Muskan", "Age" : 20, "Branch" :
"E&TC", "Address" : { "City" : "Nasik", "State" : " Maharashtra" }, "Hobbies" : [ "Painting", "Dancing" ] }
{ "_id" : ObjectId("636033bb521583da91f7c026"), "RollNo" : 6, "StudName" : "Pooja", "Age" : 20, "Branch" :
"Computer", "Address" : { "City" : "Pune", "State" : " Maharashtra" }, "Hobbies" : [ "Singing", "Dancing" ] }
{ "_id" : ObjectId("636033bb521583da91f7c022"), "RollNo" : 2, "StudName" : "Priya", "Age" : 21, "Branch" :
"Computer", "Address" : { "City" : "Pune", "State" : " Maharashtra" }, "Hobbies" : [ "Singing", "Dancing" ] }
{ "_id" : ObjectId("636033bb521583da91f7c023"), "RollNo" : 3, "StudName" : "Raj", "Age" : 22, "Branch" :
"Civil", "Address" : { "City" : "Pune", "State" : " Maharashtra" }, "Hobbies" : [ "Dancing", "Cricket" ] }
{ "_id" : ObjectId("636033bb521583da91f7c021"), "RollNo" : 1, "StudName" : "Ram", "Age" : 20, "Branch" :
"Computer", "Address" : { "City" : "Pune", "State" : " Maharashtra" }, "Hobbies" : [ "Singing", "Dancing",
"Cricket" ] }
{ "_id" : ObjectId("636033bb521583da91f7c025"), "RollNo" : 5, "StudName" : "Reya", "Age" : 24, "Branch" :
"Computer", "Address" : { "City" : "Pune", "State" : " Maharashtra" }, "Hobbies" : [ "Dancing", "Reading" ] }
{ "_id" : ObjectId("636033bb521583da91f7c029"), "RollNo" : 9, "StudName" : "Rohan", "Age" : 23, "Branch" :
"Computer", "Address" : { "City" : "Pune", "State" : " Maharashtra" }, "Hobbies" : [ "Singing", "Cricket" ] }
{ "_id" : ObjectId("636033bb521583da91f7c028"), "RollNo" : 8, "StudName" : "Roma", "Age" : 20, "Branch" :
"Computer", "Address" : { "City" : "Pune", "State" : " Maharashtra" }, "Hobbies" : [ "Singing", "Cooking" ] }
{ "_id" : ObjectId("636033bb521583da91f7c027"), "RollNo" : 7, "StudName" : "Teena", "Age" : 19, "Branch" :
"Computer", "Address" : { "City" : "Pune", "State" : " Maharashtra" }, "Hobbies" : [ "Dancing", "Drawing" ] }
```

7. Update student branch Computer of RollNo 3.

> db. Students.update({ RollNo: 3 },{ $set:{ Branch: "Computer"}})

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.Students.find().pretty()
{
	"_id" : ObjectId("636033bb521583da91f7c021"),
	"RollNo" : 1,
	"StudName" : "Ram",
	"Age" : 20,
	"Branch" : "Computer",
	"Address" : {
		"City" : "Pune",
		"State" : " Maharashtra"
	},
	"Hobbies" : [
		"Singing",
		"Dancing",
		"Cricket"
	]
}
{
	"_id" : ObjectId("636033bb521583da91f7c022"),
	"RollNo" : 2,
	"StudName" : "Priya",
	"Age" : 21,
	"Branch" : "Computer",
	"Address" : {
		"City" : "Pune",
		"State" : " Maharashtra"
	},
	"Hobbies" : [
		"Singing",
		"Dancing"
	]
}
{
	"_id" : ObjectId("636033bb521583da91f7c023"),
	"RollNo" : 3,
	"StudName" : "Raj",
	"Age" : 22,
	"Branch" : "Computer",
	"Address" : {
		"City" : "Pune",
		"State" : " Maharashtra"
```

```
        },
        "Hobbies" : [
            "Dancing",
            "Cricket"
        ]
}
{
        "_id" : ObjectId("636033bb521583da91f7c024"),
        "RollNo" : 4,
        "StudName" : "Muskan",
        "Age" : 20,
        "Branch" : "E&TC",
        "Address" : {
            "City" : "Nasik",
            "State" : " Maharashtra"
        },
        "Hobbies" : [
            "Painting",
            "Dancing"
        ]
}
{
        "_id" : ObjectId("636033bb521583da91f7c025"),
        "RollNo" : 5,
        "StudName" : "Reya",
        "Age" : 24,
        "Branch" : "Computer",
        "Address" : {
            "City" : "Pune",
            "State" : " Maharashtra"
        },
        "Hobbies" : [
            "Dancing",
            "Reading"
        ]
}
{
        "_id" : ObjectId("636033bb521583da91f7c026"),
        "RollNo" : 6,
        "StudName" : "Pooja",
        "Age" : 20,
        "Branch" : "Computer",
        "Address" : {
```

```
        "City" : "Pune",
        "State" : " Maharashtra"
    },
    "Hobbies" : [
        "Singing",
        "Dancing"
    ]
}
{
    "_id" : ObjectId("636033bb521583da91f7c027"),
    "RollNo" : 7,
    "StudName" : "Teena",
    "Age" : 19,
    "Branch" : "Computer",
    "Address" : {
        "City" : "Pune",
        "State" : " Maharashtra"
    },
    "Hobbies" : [
        "Dancing",
        "Drawing"
    ]
}
{
    "_id" : ObjectId("636033bb521583da91f7c028"),
    "RollNo" : 8,
    "StudName" : "Roma",
    "Age" : 20,
    "Branch" : "Computer",
    "Address" : {
        "City" : "Pune",
        "State" : " Maharashtra"
    },
    "Hobbies" : [
        "Singing",
        "Cooking"
    ]
}
{
    "_id" : ObjectId("636033bb521583da91f7c029"),
    "RollNo" : 9,
    "StudName" : "Rohan",
    "Age" : 23,
```

```
        "Branch" : "Computer",
        "Address" : {
                "City" : "Pune",
                "State" : " Maharashtra"
        },
        "Hobbies" : [
                "Singing",
                "Cricket"
        ]
}
{
        "_id" : ObjectId("636033bb521583da91f7c02a"),
        "RollNo" : 10,
        "StudName" : "Aveer",
        "Age" : 22,
        "Branch" : "Mechanical",
        "Address" : {
                "City" : "Pune",
                "State" : " Maharashtra"
        },
        "Hobbies" : [
                "Dancing",
                "Cricket"
        ]
}
>
```

8. Remove document with RollNo 1

```
> db. Students.remove  ( { RollNo: 1} )
WriteResult({ "nRemoved" : 1 })
> db.Students.find().pretty()
{
        "_id" : ObjectId("636033bb521583da91f7c022"),
        "RollNo" : 2,
        "StudName" : "Priya",
        "Age" : 21,
        "Branch" : "Computer",
        "Address" : {
                "City" : "Pune",
                "State" : " Maharashtra"
        },
        "Hobbies" : [
```

```
            "Singing",
            "Dancing"
        ]
}
{
    "_id" : ObjectId("636033bb521583da91f7c023"),
    "RollNo" : 3,
    "StudName" : "Raj",
    "Age" : 22,
    "Branch" : "Computer",
    "Address" : {
            "City" : "Pune",
            "State" : " Maharashtra"
    },
    "Hobbies" : [
            "Dancing",
            "Cricket"
    ]
}
{
    "_id" : ObjectId("636033bb521583da91f7c024"),
    "RollNo" : 4,
    "StudName" : "Muskan",
    "Age" : 20,
    "Branch" : "E&TC",
    "Address" : {
            "City" : "Nasik",
            "State" : " Maharashtra"
    },
    "Hobbies" : [
            "Painting",
            "Dancing"
    ]
}
{
    "_id" : ObjectId("636033bb521583da91f7c025"),
    "RollNo" : 5,
    "StudName" : "Reya",
    "Age" : 24,
    "Branch" : "Computer",
    "Address" : {
            "City" : "Pune",
            "State" : " Maharashtra"
```

```
        },
        "Hobbies" : [
            "Dancing",
            "Reading"
        ]
}
{
        "_id" : ObjectId("636033bb521583da91f7c026"),
        "RollNo" : 6,
        "StudName" : "Pooja",
        "Age" : 20,
        "Branch" : "Computer",
        "Address" : {
            "City" : "Pune",
            "State" : " Maharashtra"
        },
        "Hobbies" : [
            "Singing",
            "Dancing"
        ]
}
{
        "_id" : ObjectId("636033bb521583da91f7c027"),
        "RollNo" : 7,
        "StudName" : "Teena",
        "Age" : 19,
        "Branch" : "Computer",
        "Address" : {
            "City" : "Pune",
            "State" : " Maharashtra"
        },
        "Hobbies" : [
            "Dancing",
            "Drawing"
        ]
}
{
        "_id" : ObjectId("636033bb521583da91f7c028"),
        "RollNo" : 8,
        "StudName" : "Roma",
        "Age" : 20,
        "Branch" : "Computer",
        "Address" : {
```

```
        "City" : "Pune",
        "State" : " Maharashtra"
    },
    "Hobbies" : [
        "Singing",
        "Cooking"
    ]
}
{
    "_id" : ObjectId("636033bb521583da91f7c029"),
    "RollNo" : 9,
    "StudName" : "Rohan",
    "Age" : 23,
    "Branch" : "Computer",
    "Address" : {
        "City" : "Pune",
        "State" : " Maharashtra"
    },
    "Hobbies" : [
        "Singing",
        "Cricket"
    ]
}
{
    "_id" : ObjectId("636033bb521583da91f7c02a"),
    "RollNo" : 10,
    "StudName" : "Aveer",
    "Age" : 22,
    "Branch" : "Mechanical",
    "Address" : {
        "City" : "Pune",
        "State" : " Maharashtra"
    },
    "Hobbies" : [
        "Dancing",
        "Cricket"
    ]
}
>
```


9. Display Student information whose name starts with A

> db. Students.find({StudName:/^A/})

{ "_id" : ObjectId("636033bb521583da91f7c02a"), "RollNo" : 10, "StudName" : "Aveer", "Age" : 22, "Branch" : "Mechanical", "Address" : { "City" : "Pune", "State" : " Maharashtra" }, "Hobbies" : ["Dancing", "Cricket"] }
>

10. Display the total numbers of documents available in collection.

> db. Students.find().count()
9

Conclusion:

In this assignment we have studied about NoSQL MongoDB database and implemented the same to solve given problem statement.

ASSIGNMENT NO: 11

Title: MongoDB – Aggregation and Indexing.

Problem Statement:

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

Theory:

Indexing: Indexes support the efficient execution of queries in MongoDB

Indexing Types

- **Single field index** only includes data from a single field of the Single Field Indexes documents in a collection.
- **Compound index** includes more than one field of the documents in Compound Indexes a collection.
- **Multikey index** is an index on an array field, adding an index key for Multikey each value in the array. Indexes
- **Geospatial indexes** support location-based searches. Geospatial Indexes and Queries Text Indexes
- **Text indexes** support search of string content in documents.
- **Hashed Index** -Hashed indexes maintain entries with hashes of the values of the indexed field and are used with sharded clusters to support hashed shard keys.

Index Properties:

Index Properties -The properties you can specify when building indexes.

1. TTL Indexes The TTL index is used for TTL collections, which expire data after a period of time
2. Unique Indexes A unique index causes MongoDB to reject all documents that contain a duplicate value for the indexed field.
3. Sparse Indexes A sparse index does not index documents that do not have the indexed field.

Index Creation:

Syntax:

db.CollectionName.createIndex({ KeyName: 1 or -1})

- 1 for Ascending Sorting
- -1 for Descending Sorting

Index Creation Example:

- Single: db.stud.createIndex({ zipcode: 1})
- Compound: db.stud.createIndex({ dob: 1, zipcode: -1 })
- Unique: db.stud.createIndex({ rollno: 1 }, { unique: true })
- Sparse: db.stud.createIndex({ age: 1 }, { sparse: true })

Index Display

db.collection.getIndexes()

Returns an array that holds a list of documents that identify and describe the existing indexes on the collection.

Index Drop

Syntax:

1. db.collection.dropIndex()
2. db.collection.dropIndex(index)

Example:

1. db.stud.dropIndex()
2. db.stud.dropIndex({ "name" : 1 })

Indexing and Querying

create an ascending index on the field name for a collection records:

- db.records.createIndex({ name: 1 })
  This index can support an ascending sort on name :
- db.records.find().sort({ name: 1 })
  The index can also support descending sort
- db.records.find().sort({ a: -1 })
- db.stud.findOne({rno:2}), using index {rno:1}

Indexing with Unique:

- db.collectionname.ensureIndex ({x:1}, {unique:true})
- Don't allow {_id:10,x:2} and {_id:11,x:2}

- Don’t allow {_id:12} and {_id:13} (both match {x:null}

Aggregation:

Aggregations operations process data records and return computed results.

Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data

For aggregation in mongodb use aggregate() method.

Syntax:

• >db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)

aggregate() method

Expression	Description
$sum	Sums up the defined value from all documents in the collection.
$avg	Calculates the average of all given values from all documents in the collection.
$min	Gets the minimum of the corresponding values from all documents in the collection.
$max	Gets the maximum of the corresponding values from all documents in the collection.
$first	Gets the first document from the source documents according to the grouping.
$last	Gets the last document from the source documents according to the grouping.

Possible stages in aggregation

- $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.

Assignment No 11: Output

> **use assignment11**
switched to db assignment11

> **db.createCollection("teachers");**
{ "ok" : 1 }

> **db.teachers.insert({tname: "Aliya", dname: "Comp", salary:11000,exp:2});**
WriteResult({ "nInserted" : 1 })
> db.teachers.insert({tname: "Jinal", dname: "Comp", salary:12000,exp:3});
WriteResult({ "nInserted" : 1 })
> db.teachers.insert({tname: "Isha", dname: "Comp", salary:10000,exp:1});
WriteResult({ "nInserted" : 1 })
> db.teachers.insert({tname: "Tejal", dname: "Mech", salary:10500,exp:2});
WriteResult({ "nInserted" : 1 })
> db.teachers.insert({tname: "Harshi", dname: "Mech", salary:14000,exp:2});
WriteResult({ "nInserted" : 1 })
> db.teachers.insert({tname: "Urvi", dname: "Mech", salary:24000,exp:5});
WriteResult({ "nInserted" : 1 })
> db.teachers.insert({tname: "Rutu", dname: "IT", salary:10000,exp:1});
WriteResult({ "nInserted" : 1 })
> db.teachers.insert({tname: "Trupti", dname: "IT", salary:20000,exp:4});
WriteResult({ "nInserted" : 1 })
> db.teachers.insert({tname: "Surbhi", dname: "IT", salary:12000,exp:2});
WriteResult({ "nInserted" : 1 })
> db.teachers.insert({tname: "Shweta", dname: "Civil", salary:10000,exp:1});
WriteResult({ "nInserted" : 1 })
> db.teachers.insert({tname: "Athira", dname: "Civil", salary:12000,exp:2});
WriteResult({ "nInserted" : 1 })
> db.teachers.insert({tname: "Tejaswee", dname: "Civil", salary:15000,exp:3});
WriteResult({ "nInserted" : 1 })

> **db.teachers.find({});**
{ "_id" : ObjectId("6360a3f81be64b3c2d938ce2"), "tname" : "Aliya", "dname" : "Comp", "salary" : 11000, "exp" : 2 }
{ "_id" : ObjectId("6360a4061be64b3c2d938ce3"), "tname" : "Jinal", "dname" : "Comp", "salary" : 12000, "exp" : 3 }
{ "_id" : ObjectId("6360a40d1be64b3c2d938ce4"), "tname" : "Isha", "dname" : "Comp", "salary" : 10000, "exp" : 1 }
{ "_id" : ObjectId("6360a4141be64b3c2d938ce5"), "tname" : "Tejal", "dname" : "Mech", "salary" : 10500, "exp" : 2 }
{ "_id" : ObjectId("6360a41c1be64b3c2d938ce6"), "tname" : "Harshi", "dname" : "Mech", "salary" : 14000, "exp" : 2 }
{ "_id" : ObjectId("6360a4231be64b3c2d938ce7"), "tname" : "Urvi", "dname" : "Mech", "salary" : 24000, "exp" : 5 }
{ "_id" : ObjectId("6360a42c1be64b3c2d938ce8"), "tname" : "Rutu", "dname" : "IT", "salary" : 10000, "exp" : 1 }
{ "_id" : ObjectId("6360a4341be64b3c2d938ce9"), "tname" : "Trupti", "dname" : "IT", "salary" : 20000, "exp" :

4 }
{ "_id" : ObjectId("6360a43a1be64b3c2d938cea"), "tname" : "Surbhi", "dname" : "IT", "salary" : 12000, "exp" : 2 }
{ "_id" : ObjectId("6360a4421be64b3c2d938ceb"), "tname" : "Shweta", "dname" : "Civil", "salary" : 10000, "exp" : 1 }
{ "_id" : ObjectId("6360a44a1be64b3c2d938cec"), "tname" : "Athira", "dname" : "Civil", "salary" : 12000, "exp" : 2 }
{ "_id" : ObjectId("6360a4691be64b3c2d938ced"), "tname" : "Tejaswee", "dname" : "Civil", "salary" : 15000, "exp" : 3 }

> **db.teachers.find({},{_id:0});**
{ "tname" : "Aliya", "dname" : "Comp", "salary" : 11000, "exp" : 2 }
{ "tname" : "Jinal", "dname" : "Comp", "salary" : 12000, "exp" : 3 }
{ "tname" : "Isha", "dname" : "Comp", "salary" : 10000, "exp" : 1 }
{ "tname" : "Tejal", "dname" : "Mech", "salary" : 10500, "exp" : 2 }
{ "tname" : "Harshi", "dname" : "Mech", "salary" : 14000, "exp" : 2 }
{ "tname" : "Urvi", "dname" : "Mech", "salary" : 24000, "exp" : 5 }
{ "tname" : "Rutu", "dname" : "IT", "salary" : 10000, "exp" : 1 }
{ "tname" : "Trupti", "dname" : "IT", "salary" : 20000, "exp" : 4 }
{ "tname" : "Surbhi", "dname" : "IT", "salary" : 12000, "exp" : 2 }
{ "tname" : "Shweta", "dname" : "Civil", "salary" : 10000, "exp" : 1 }
{ "tname" : "Athira", "dname" : "Civil", "salary" : 12000, "exp" : 2 }
{ "tname" : "Tejaswee", "dname" : "Civil", "salary" : 15000, "exp" : 3 }
>
>
db.teachers.aggregate([{$group:{_id:"$dname",avg_sal:{$avg:
"$salary"}}}]);
{ "_id" : "IT", "avg_sal" : 14000 }
{ "_id" : "Mech", "avg_sal" : 16166.666666666666 }
{ "_id" : "Civil", "avg_sal" : 12333.333333333334 }
{ "_id" : "Comp", "avg_sal" : 11000 }

> **db.teachers.aggregate([{$group:{_id:"$dname",no_of _emp:{$sum:1}}}]);**
{ "_id" : "IT", "no_of_emp" : 3 }
{ "_id" : "Mech", "no_of_emp" : 3 }
{ "_id" : "Civil", "no_of_emp" : 3 }
{ "_id" : "Comp", "no_of_emp" : 3 }

> **db.teachers.aggregate([{$group:{_id:"$dname",no_of_emp:{$max:"$salary"}}}]);**
{ "_id" : "IT", "no_of_emp" : 20000 }
{ "_id" : "Mech", "no_of_emp" : 24000 }
{ "_id" : "Civil", "no_of_emp" : 15000 }
{ "_id" : "Comp", "no_of_emp" : 12000 }
>
db.teachers.aggregate([{$group:{_id:"$dname",total_sal:{$sum:"$salary"}}},{$match:{total_sal:{$gte:40000}}}]);
{ "_id" : "IT", "total_sal" : 42000 }
{ "_id" : "Mech", "total_sal" : 48500 }
>

db.teachers.aggregate([{$group:{_id:"$dname",total_sal:{$sum:"$salary"}}},{$match:{total_sal:{$gte:45000}}}]);

```
{ "_id" : "Mech", "total_sal" : 48500 }
>
```

db.teachers.aggregate([{$group:{_id:"$dname",total_sal:{$sum:"$salary"}}},{$match:{total_sal:{$gte:15000}}}]);

```
{ "_id" : "IT", "total_sal" : 42000 }
{ "_id" : "Mech", "total_sal" : 48500 }
{ "_id" : "Civil", "total_sal" : 37000 }
{ "_id" : "Comp", "total_sal" : 33000 }
>
```

> db.teachers.createIndex({ tname: -1 })

```
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

> db.teachers.createIndex({ dname: -1 })

```
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 2,
    "numIndexesAfter" : 3,
    "ok" : 1
}
```

> **db.teachers.createIndex({ salary: 1 })**

```
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 3,
    "numIndexesAfter" : 4,
    "ok" : 1
}
```

> db.teachers.getIndexes()

```
[
    {
        "v" : 2,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "assignment11.teachers"
    },
    {
        "v" : 2,
        "key" : {
```

```
                "tname" : -1
            },
            "name" : "tname_-1",
            "ns" : "assignment11.teachers"
        },
        {
            "v" : 2,
            "key" : {
                "dname" : -1
            },
            "name" : "dname_-1",
            "ns" : "assignment11.teachers"
        },
        {
            "v" : 2,
            "key" : {
                "salary" : 1
            },
            "name" : "salary_1",
            "ns" : "assignment11.teachers"
        }
]
>
```


Conclusion:

In this assignment we have studied MongoDB Aggregation and Indexing and solved given Queries using MongoDB.

ASSIGNMENT NO: 12

Title: MongoDB – Map-reduces operations.

Problem Statement:

Implement Map reduces operation with suitable example using MongoDB.

Theory:

Map-Reduce

Map-Reduce is a data processing paradigm used in MongoDB for data aggregation and transformation. It allows you to process large volumes of data and generate aggregated results. MongoDB provides a built-in Map-Reduce feature that enables you to run Map-Reduce operations on your data.

Overview of how Map-Reduce works in MongoDB:

1. **Map Function:**

The Map function processes each document in the input collection and emits key-value pairs. These key-value pairs are stored in temporary storage and will be later grouped by keys for further processing.

2. **Reduce Function:**

The Reduce function processes the grouped data, where documents with the same key are grouped together. It can aggregate and summarize data for each key.

3. **Finalize Function (optional):**

If specified, the Finalize function can further process the results produced by the Reduce function.

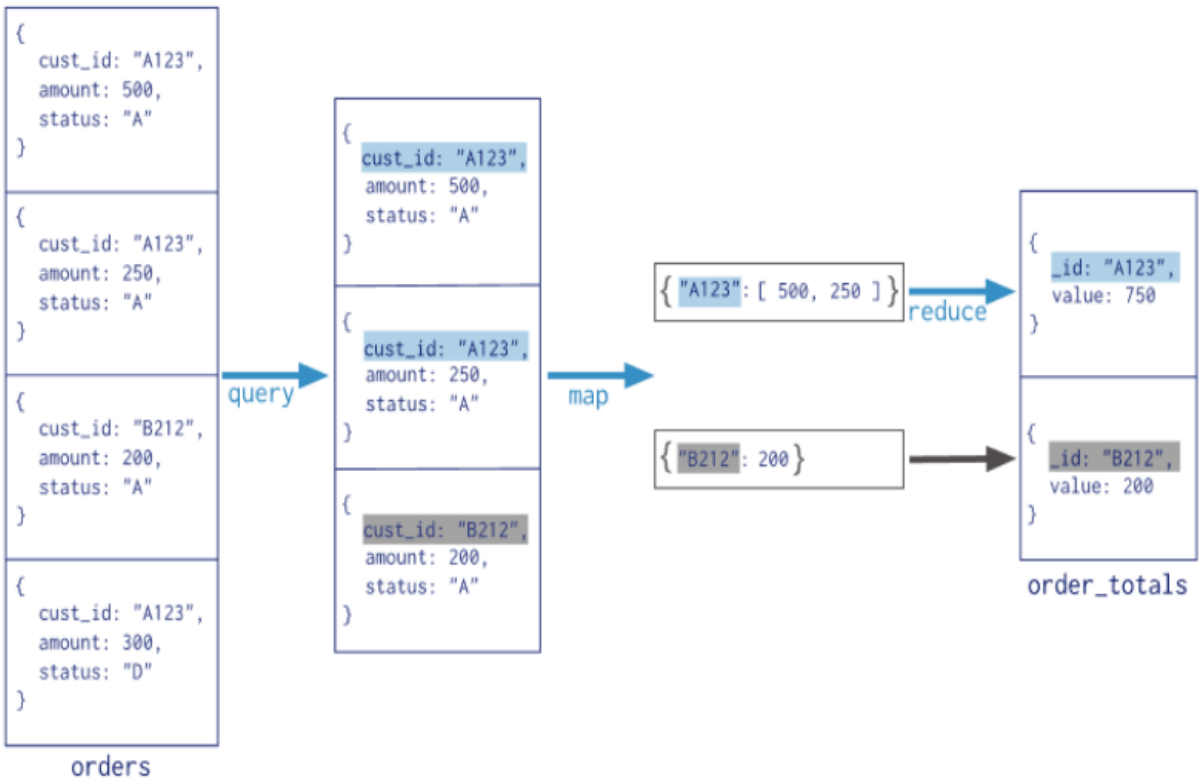
4. **Output:**

The results of the Map-Reduce operation can be stored in a new collection or returned as a cursor.

• **Consider the following map-reduce operation:**

```
Collection
   ↓
db.orders.mapReduce(
        map      ──→  function() { emit( this.cust_id, this.amount ); },
        reduce   ──→  function(key, values) { return Array.sum( values ) },
                      {
        query    ──→    query: { status: "A" },
        output   ──→    out: "order_totals"
                      }
                   )
```

MapReduce:



The mapReduce command takes 2 primary inputs, the mapper function and the reducer function. A Mapper will start off by reading a collection of data and building a Map with only the required fields we wish to process and group them into one array based on the key. And then this key value pair is fed into a Reducer, which will process the values.

Map-Reduce Syntax:

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

Map-Reduce Syntax Explanation:

The above map-reduce function will query the collection, and then map the output documents to the emit key-value pairs. After this, it is reduced based on the keys that have multiple values. Here, we have used the following functions and parameters.

• Map: – It is a JavaScript function. It is used to map a value with a key and produces a key-value pair.
• Reduce: – It is a JavaScript function. It is used to reduce or group together all the documents which have the same key.
• Out: – It is used to specify the location of the map-reduce query output.
• Query: – It is used to specify the optional selection criteria for selecting documents.
• Sort: – It is used to specify the optional sort criteria.
• Limit: – It is used to specify the optional maximum number of documents which are desired to be returned.

.

Assignment 12: Solution

> use author
switched to db author
> db.author.save({"book_title" : "MongoDB Tutorial", "author_name" : "aparajita", "status" : "active", publish_year" : "2016" })
WriteResult({ "nInserted" : 1 })
> db.author.save({ "book_title" : "Software Testing Tutorial", "author_name" : "aparajita", "status" : "active", "publish_year": "2015" })
WriteResult({ "nInserted" : 1 })
> db.author.save({ "book_title" : "Node.js Tutorial", "author_name" : "Kritika", "status" : "active", "publish_year": "2016" })
WriteResult({ "nInserted" : 1 })
> db.author.save({ "book_title" : "PHP7 Tutorial", "author_name" : "aparajita", "status" : "passive", "publish_year": "2016" })
WriteResult({ "nInserted" : 1 })

> db.author.find()
{ "_id" : ObjectId("6360b68c1be64b3c2d938cee"), "book_title" : "MongoDB Tutorial", "author_name" : "aparajita", "status" : "active", "publish_year" : "2016" }
{ "_id" : ObjectId("6360b6961be64b3c2d938cef"), "book_title" : "Software Testing Tutorial", "author_name" : "aparajita", "status" : "active", "publish_year" : "2015" }
{ "_id" : ObjectId("6360bfea1be64b3c2d938cf0"), "book_title" : "Node.js Tutorial", "author_name" : "Kritika", "status" : "active", "publish_year" : "2016" }
{ "_id" : ObjectId("6360c01f1be64b3c2d938cf1"), "book_title" : "PHP7 Tutorial", "author_name" : "aparajita", "status" : "passive", "publish_year" : "2016" }
>

> db.author.mapReduce(function() { emit(this.author_name,1) }, function(key, values) {return Array.sum(values)}, { query:{status:"active"}, out:"author_total" })
```
{
        "result" : "author_total",
        "timeMillis" : 81,
        "counts" : {
                "input" : 3,
                "emit" : 3,
                "reduce" : 1,
                "output" : 2
        },
        "ok" : 1
}
```

db.author.mapReduce(function() { emit(this.author_name,1) },function(key, values) {return Array.sum(values)}, { query:{status:"active"}, out:"author_total" }).find()
{ "_id" : "Kritika", "value" : 1 }
{ "_id" : "aparajita", "value" : 2 }
> db.stud.insert({ Name: "Amit", Marks:80 })
WriteResult({ "nInserted" : 1 })
> db.stud.insert({ Name: "Amit", Marks:90 })

```
WriteResult({ "nInserted" : 1 })
> db.stud.insert({ Name: "Shreya", Marks:40 })
WriteResult({ "nInserted" : 1 })
> db.stud.insert({ Name: "Neha", Marks:80 })
WriteResult({ "nInserted" : 1 })
> db.stud.insert({ Name: "Neha", Marks:35})
WriteResult({ "nInserted" : 1 })
```

> db.stud.find()

```
{ "_id" : ObjectId("6360c43d1be64b3c2d938cf2"), "Name" : "Amit", "Marks" : 80 }
{ "_id" : ObjectId("6360c4431be64b3c2d938cf3"), "Name" : "Amit", "Marks" : 90 }
{ "_id" : ObjectId("6360c4481be64b3c2d938cf4"), "Name" : "Shreya", "Marks" : 40 }
{ "_id" : ObjectId("6360c44f1be64b3c2d938cf5"), "Name" : "Neha", "Marks" : 80 }
{ "_id" : ObjectId("6360c4571be64b3c2d938cf6"), "Name" : "Neha", "Marks" : 35 }
>
```

> db.stud.mapReduce(function(){ emit(this.Name,1)},function(key, values) {return Array.sum(values)}, {out: "Name_Total" })

```
{
      "result" : "Name_Total",
      "timeMillis" : 70,
      "counts" : {
            "input" : 5,
            "emit" : 5,
            "reduce" : 2,
            "output" : 3
      },
      "ok" : 1
}
```

> db.stud.mapReduce(function(){ emit(this.Name,1)},function(key, values) {return Array.sum(values)}, {out: "Name_Total" }).find()

```
{ "_id" : "Amit", "value" : 2 }
{ "_id" : "Neha", "value" : 2 }
{ "_id" : "Shreya", "value" : 1 }
>
```

> db.stud.mapReduce(function() { emit(this.Name,this.Marks) }, function(key, values) {return Array.sum(values)}, {out: "Total_Marks" })

```
{
      "result" : "Total_Marks",
      "timeMillis" : 77,
      "counts" : {
            "input" : 5,
            "emit" : 5,
            "reduce" : 2,
            "output" : 3
      },
      "ok" : 1
}
```

> db.stud.mapReduce(function() { emit(this.Name,this.Marks) }, function(key, values) {return Array.sum(values)}, {out: "Total_Marks" }).find()

{ "_id" : "Amit", "value" : 170 }
{ "_id" : "Neha", "value" : 115 }
{ "_id" : "Shreya", "value" : 40 }

> db.stud.mapReduce(function(){ emit(this.Name,this.Marks)}, function(key, values) {return Array.sum(values)},{out: 'Name_Total'}).find().sort({value:1})

{ "_id" : "Shreya", "value" : 40 }
{ "_id" : "Neha", "value" : 115 }
{ "_id" : "Amit", "value" : 170 }

> db.stud.mapReduce(function(){ emit(this.Name,this.Marks)}, function(key, values) {return Array.sum(values)},{out: 'Name_Total'}).find().limit(1)

{ "_id" : "Amit", "value" : 170 }

> db.stud.mapReduce(function(){ emit(this.Name,1)}, function(key, values) {return Array.sum(values)}, {query:{Marks:{$gt:70}},out: 'Name_Total'}).find()

{ "_id" : "Amit", "value" : 2 }
{ "_id" : "Neha", "value" : 1 }

Conclusion:

In this assignment we have studied about Map reduce operation with suitable example using MongoDB.

ASSIGNMENT NO: 13

Title: Database Connectivity.

Problem Statement:

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

Theory:

Database connectivity in a software application refers to the ability of the application to connect to a database, perform operations like reading, writing, updating, and deleting data, and manage the interaction with the database. The database can be of various types, such as relational databases (e.g., MySQL, PostgreSQL, SQL Server), NoSQL databases (e.g., MongoDB, Cassandra), or other data storage systems.

Here are the general steps to achieve database connectivity in a typical software application:

- Select a Database
- Database Driver/Connector
- Connection String/Configuration
- Establish a Connection
- Execute SQL or Query
- Handle Exceptions
- Close the Connection

To connect to a MongoDB database using Java, you'll need to use the MongoDB Java driver. The following steps outline the process of connecting to a MongoDB database and performing basic operations using Java.

Software Required and Steps

• Eclipse
• JDK 1.6
• MongoDB
• MongoDB-Java-Driver

In Eclipse perform following steps:

1. File - New – Java Project –Give Project Name – ok
2. In project Explorer window- right click on project namenew- class- give Class name- ok
3. In project Explorer window- right click on project nameBuild path- Configure build path- Libraries- Add External

Jar - MongoDB-Java-Driver

5. Start Mongo server before running the program

Steps to Write Code in Java

1. **Import packages**

```
import com.mongodb.*;
```

2. Create connection

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

3. Create Database

```
DB db = mongo.getDB("database name");
```

4. Insert Document

```
BasicDBObject d1 = new BasicDBObject(“rno“,“1”).append(“name", “Monika"). append(“age", “17”)
coll.insert(d1);
```

5. Display document

```
DBCursor cursor = coll.find();
while (cursor.hasNext())
{
System.out.println(cursor.next());
}
```

6. Update Document

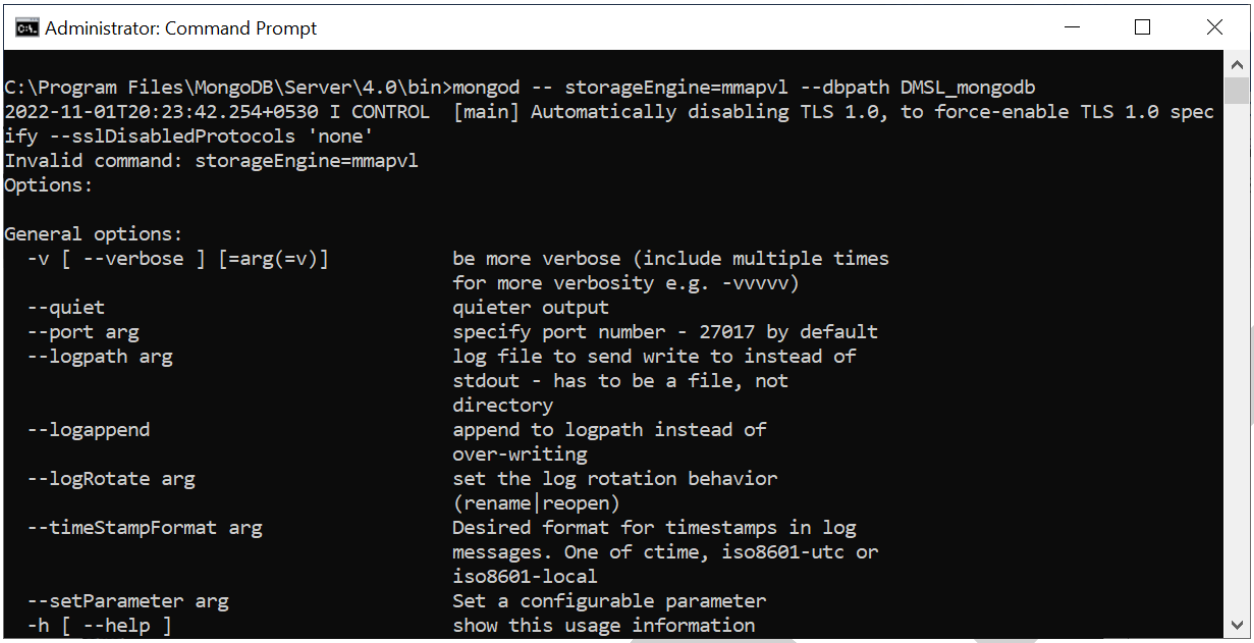
```
• BasicDBObject query = new BasicDBObject();
• query.put("name", “Monika");
• BasicDBObject newDocument = new BasicDBObject();
• newDocument.put("name", “Ragini");
• BasicDBObject updateObj = new BasicDBObject();
• updateObj.put("$set", newDocument);
• Coll.update(query, updateObj);
```

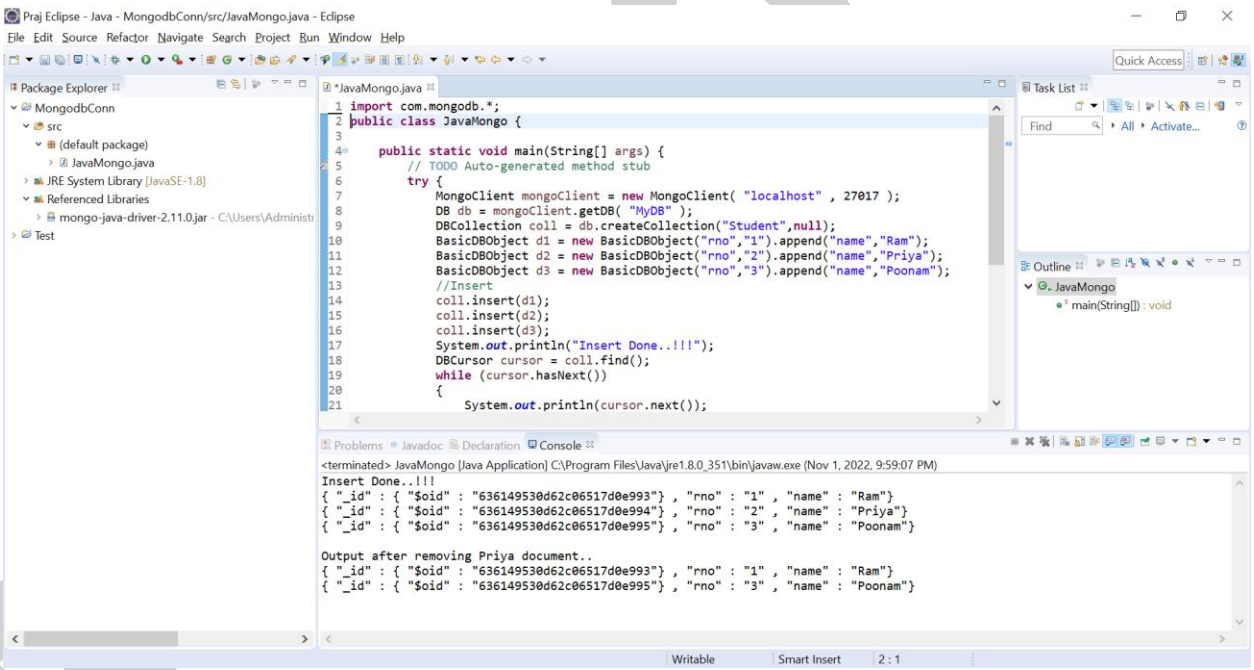
7. Remove document

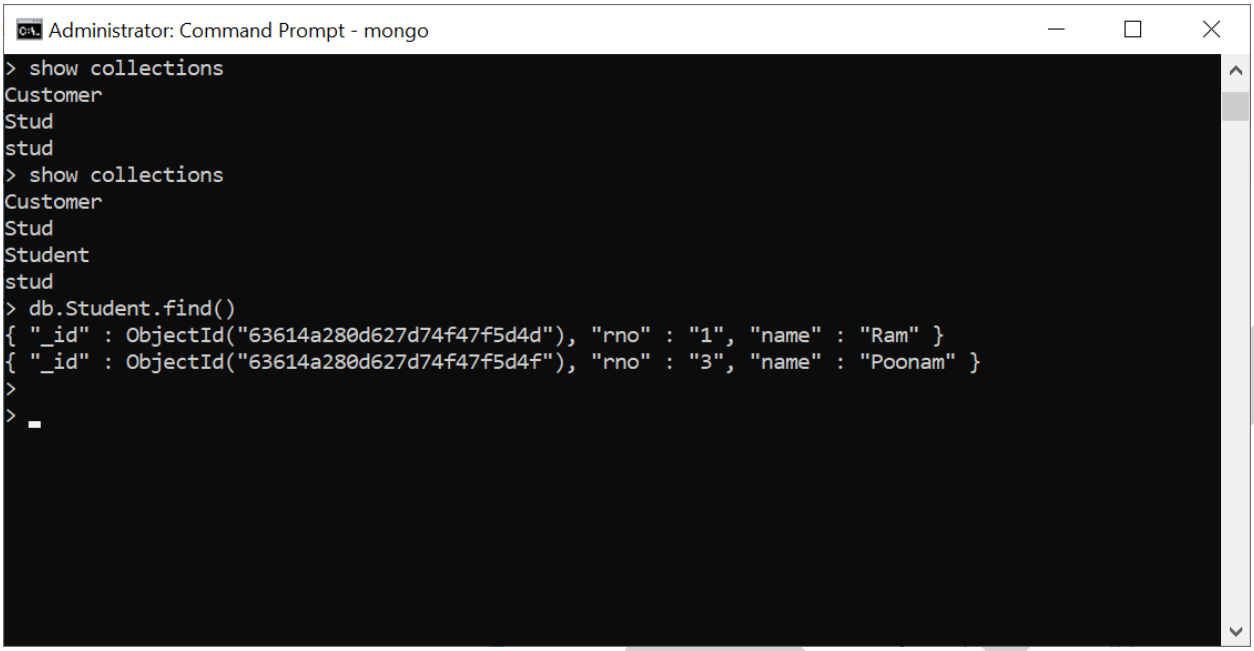
```
BasicDBObject searchQuery = new BasicDBObject("name", “Monika");
Coll.remove(searchQuery);
```

Assignment No: 13 : Solution

```
import com.mongodb.*;
public class JavaMongo
{
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		try {
			MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
			DB db = mongoClient.getDB( "MyDB" );
			DBCollection coll = db.createCollection("Student",null);
			BasicDBObject d1 = new BasicDBObject("rno","1").append("name","Ram");
			BasicDBObject d2 = new
BasicDBObject("rno","2").append("name","Priya");
			BasicDBObject d3 = new
BasicDBObject("rno","3").append("name","Poonam");
			//Insert
			coll.insert(d1);
			coll.insert(d2);
			coll.insert(d3);
			System.out.println("Insert Done..!!!");
			DBCursor cursor = coll.find();
			while (cursor.hasNext())
			{
				System.out.println(cursor.next());
			}
			BasicDBObject R1 = new BasicDBObject("name", "Priya");
			coll.remove(R1);
			System.out.println("\nOutput after removing Priya document..");
			cursor = coll.find();
			while (cursor.hasNext())
			{
			System.out.println(cursor.next());
			}
			}
			catch(Exception e)
			{
			e.printStackTrace();
			}
			}
}
```


Administrator: Command Prompt
C:\Program Files\MongoDB\Server\4.0\bin>mongod -- storageEngine=mmapvl --dbpath DMSL_mongodb
2022-11-01T20:23:42.254+0530 I CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 spec
ify --sslDisabledProtocols 'none'
Invalid command: storageEngine=mmapvl
Options:
General options:
-v [--verbose] [=arg(=v)] be more verbose (include multiple times
for more verbosity e.g. -vvvvv)
--quiet quieter output
--port arg specify port number - 27017 by default
--logpath arg log file to send write to instead of
stdout - has to be a file, not
directory
--logappend append to logpath instead of
over-writing
--logRotate arg set the log rotation behavior
(rename|reopen)
--timeStampFormat arg Desired format for timestamps in log
messages. One of ctime, iso8601-utc or
iso8601-local
--setParameter arg Set a configurable parameter
-h [--help] show this usage information


Praj Eclipse - Java - MongodbConn/src/JavaMongo.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
MongodbConn
src
(default package)
JavaMongo.java
JRE System Library [JavaSE-1.8]
Referenced Libraries
mongo-java-driver-2.11.0.jar
Test
*JavaMongo.java
import com.mongodb.*;
public class JavaMongo {
public static void main(String[] args) {
// TODO Auto-generated method stub
try {
MongoClient mongoClient = new MongoClient("localhost" , 27017);
DB db = mongoClient.getDB("MyDB");
DBCollection coll = db.createCollection("Student",null);
BasicDBObject d1 = new BasicDBObject("rno","1").append("name","Ram");
BasicDBObject d2 = new BasicDBObject("rno","2").append("name","Priya");
BasicDBObject d3 = new BasicDBObject("rno","3").append("name","Poonam");
//Insert
coll.insert(d1);
coll.insert(d2);
coll.insert(d3);
System.out.println("Insert Done..!!!");
DBCursor cursor = coll.find();
while (cursor.hasNext())
{
System.out.println(cursor.next());
Task List
Outline
JavaMongo
main(String[]) : void
Problems Javadoc Declaration Console
<terminated> JavaMongo [Java Application] C:\Program Files\Java\jre1.8.0_351\bin\javaw.exe (Nov 1, 2022, 9:59:07 PM)
Insert Done..!!!
{ "_id" : { "$oid" : "636149530d62c06517d0e993"} , "rno" : "1" , "name" : "Ram"}
{ "_id" : { "$oid" : "636149530d62c06517d0e994"} , "rno" : "2" , "name" : "Priya"}
{ "_id" : { "$oid" : "636149530d62c06517d0e995"} , "rno" : "3" , "name" : "Poonam"}
Output after removing Priya document..
{ "_id" : { "$oid" : "636149530d62c06517d0e993"} , "rno" : "1" , "name" : "Ram"}
{ "_id" : { "$oid" : "636149530d62c06517d0e995"} , "rno" : "3" , "name" : "Poonam"}
Writable
Smart Insert
2 : 1



Conclusion :

In this assignment we have studied about database connectivity and successfully connected MongoDB database with Java application.