"Techno – Social Excellence"

Marathwada Mitramandal's

INSTITUTE OF TECHNOLOGY (MMIT)

Lohgaon, Pune-411047

"Towards Ubiquitous Computing Technology"

DEPARTMENT OF COMPUTER ENGINEERING

Savitribai Phule Pune University

Third Year of Computer Engineering (2019 Course)

310257: DBMS Lab

LAB MANUAL

Database Management System Lab

TE COMPUTER

(2019 Pattern)

Prepared By :- Dr.S.G.Rathod

Ms. M.S.Jagtap

"Techno-social Excellence"

Marathwada Mitra Mandal's

INSTITUTE OF TECHNOLOGY

Lohgaon, Pune-411047

"येथे बहुतांचे हित"

VISION

"Techno-Social Excellence"

MISSION

- Enhance Technology Transfer
- Implement entrepreneurship
- Promote global competency
- Integrate innovative pedagogy
- Create excellent human resource

CORE VALUES

- Teamwork
- Value Based Ethics
- Societal Trust
- Pleasant Environment
- Industrial Approach
- Committed Faculty
- Standard Report Writing
- Adaptive Research
- Lifelong Learning

"Techno-social Excellence"

Marathwada Mitra Mandal's
INSTITUTE OF TECHNOLOGY
Lohgaon, Pune-411047

"येथे बहुतांचे हित"

"Towards ubiquitous Computing Technology"

Department of Computer Engineering

VISION

"Towards Ubiquitous Computing Technology"

MISSION

- **Creating Industry Oriented Competency**
- **Innovating Computing Studies**
- **Inspiring Students to Endorse New Technologies**
- **Virtualizing Upcoming World for Implementation**

Savitribai Phule Pune University

Third Year of Computer Engineering (2019 Course)

(With effect from Academic Year 2021-22)

Table of Contents

Savitribai Phule Pune University

Third Year of Computer Engineering (2019 Course)

(With effect from Academic Year 2021-22)

Semester V

Course Code	Course Name	Teaching Scheme (Hours/ week)			Examination Scheme and Marks						Credit Scheme			
		Lecture	Practical	Tutorial	Mid-Sem	End-Sem	Term work	Practical	Oral	Total	Lecture	Practical	Tutorial	Total
310241	Database Management Systems	03	-	-	30	70	-	-	-	100	03	-	-	03
310242	Theory of Computation	03	-	-	30	70	-	-	-	100	03	-	-	03
310243	Systems Programming and Operating System	03	-	-	30	70	-	-	-	100	03	-	-	03
310244	Computer Networks and Security	03	-	-	30	70	-	-	-	100	03	-	-	03
310245	Elective I	03	-	-	30	70	-	-	-	100	03	-	-	03
310246	Database Management Systems Laboratory	-	04	-	-	-	25	25	-	50	-	02	-	02
310247	Computer Networks and Security Laboratory	-	02	-	-	-	25	-	25	50	-	01	-	01
310248	Laboratory Practice I	-	04	-	-	-	25	25	-	50	-	02	-	02
310249	Seminar and Technical Communication	-	01	-	-	-	50	-	-	50	-	01	-	01
	Total	**15**	**11**	**-**	**150**	**350**	**125**	**50**	**25**	**700**	**15**	**06**	**-**	**21**
310250	Audit Course 5												**Grade**	
										Total Credit	**15**	**06**	**-**	**21**

Elective I

- Internet of Things and Embedded Systems
- Human Computer Interface
- Distributed Systems
- Software Project Management

Audit Course 5

- Cyber Security
- Professional Ethics and Etiquettes
- MOOC- Learn New Skills
- Engineering Economics
- Foreign Language

Laboratory Practice I

Assignments from **Systems Programming and Operating System** and **Elective I**

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

Home

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

Companion Course: Database Management Systems (310241)

Course Objectives:

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

Course Outcomes:

On completion of the course, learners will be able to

CO1: Design E-R Model for given requirements and convert the same into database tables
CO2: Design schema in appropriate normal form considering actual requirements
CO3: Implement SQL queries for given requirements , using different SQL concepts
CO4: Implement PL/SQL Code block for given requirements
CO5: Implement NoSQL queries using MongoDB
CO6: Design and develop application considering actual requirements and using database concepts

Guidelines for Instructor's Manual

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

Guidelines for Student's Laboratory Journal

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

Guidelines for Laboratory /Term Work Assessment

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

Guidelines for Practical Examination

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

Guidelines for Laboratory Conduction

The instructor is expected to frame the assignments by understanding the prerequisites, technological aspects, utility and recent trends related to the topic. The assignment framing policy need to address the average students and inclusive of an element to attract and promote the intelligent students. Use of open source software is encouraged. Based on the concepts learned. Instructor may also set one assignment or mini-project that is suitable to respective branch beyond the scope of syllabus.
Operating System recommended :- 64-bit Open source Linux or its derivative
Programming tools recommended: - MYSQL/Oracle, MongoDB, ERD plus, ER Win

Virtual Laboratory:

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

Suggested List of Laboratory Experiments/Assignments

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

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

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

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

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

Assignment No .1

Title of Assignment :- ER Modeling and Normalization:

Decide a case study related to real time application in group of 2-3 students and formulate a problem statement for application to be developed. Propose a Conceptual Design using ER features using tools like ERD plus, ER Win etc. (Identifying entities, relationships between entities, attributes, keys, cardinalities, generalization, specialization etc.) Convert the ER diagram into relational tables and normalize Relational data model. Note: Student groups are required to continue same problem statement throughout all the assignments in order to design and develop an application as a part Mini Project. Further assignments will be useful for students to develop a backend for system. To design front end interface students should use the different concepts learnt in the other subjects also.

Course Objective:

- To learn and understand the concept offer Model and steps to Design ER Model and to ER Model into table.

Software Required: - ERD Plus,ER win etc

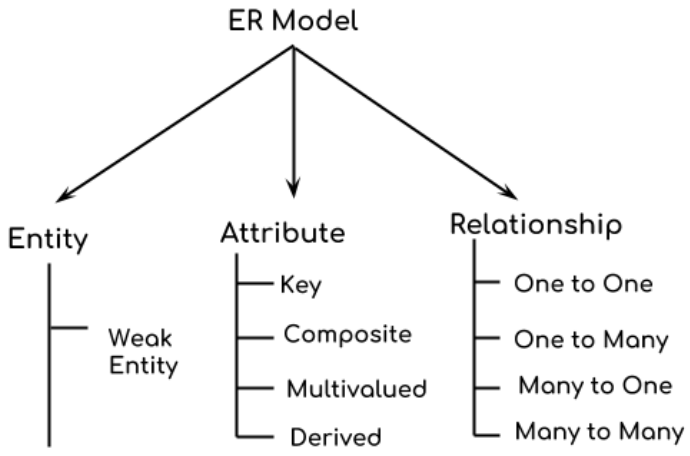
Prerequisite: Basics of RDBMS.

Theory:-

New Concepts:

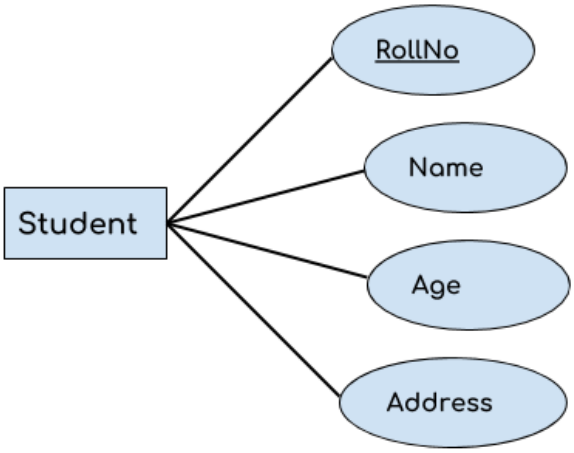
Entity-Relationship model is used in the conceptual design of a database (☞ conceptual level, conceptual schema). A database schema in the ER model can be represented pictorially (Entity-Relationship diagram)

Components of ER Diagram:

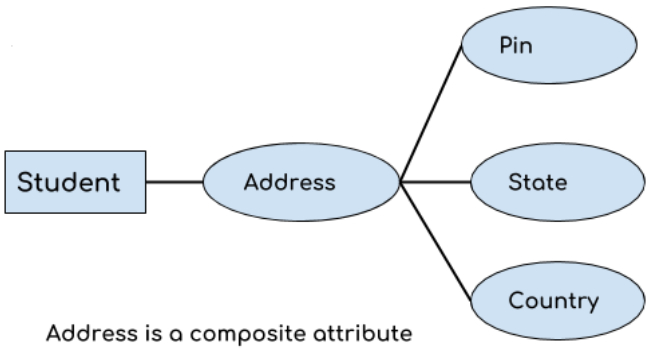


1> Entity: real-world object or thing with an independent existence and which is distinguishable from other objects. An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.
Examples are a person, car, customer, product, gene, book etc.
2> Attributes: an entity is represented by a set of attributes . An attribute describes the property of an entity. e.g., name, age, salary, price etc. Attribute values that describe each entity become a major part of the data eventually stored in a database.
An attribute is represented as Oval in an ER diagram. There are four types of attributes:
1. Key attribute
2. Composite attribute
3. Multivalued attribute
4. Derived attribute

1. Key attribute:
ER diagram key attribute: A key attribute can uniquely identify an entity from an entity set. For example, student roll number can uniquely identify a student from a set of students. Key attribute is represented by oval same as other attributes however the text of key attribute is underlined.



2. Composite attribute:
ER diagram composite attribute: An attribute that is a combination of other attributes is known as composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.

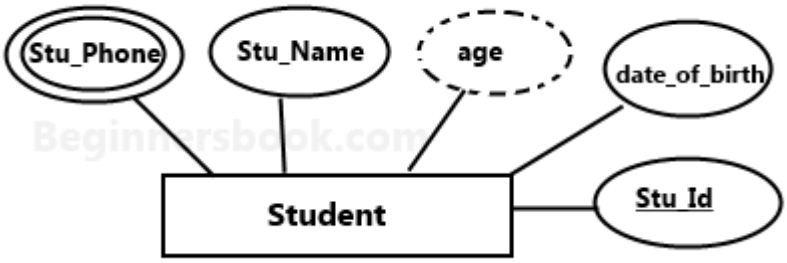


3. Multivalued attribute:

An attribute that can hold multiple values is known as multivalued attribute. It is represented with double ovals in an ER Diagram. For example – A person can have more than one phone numbers so the phone number attribute is multivalued.

4. Derived attribute:
A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by dashed oval in an ER Diagram. For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).

Multivalued and derived attribute
E-R diagram with multivalued and derived attributes:

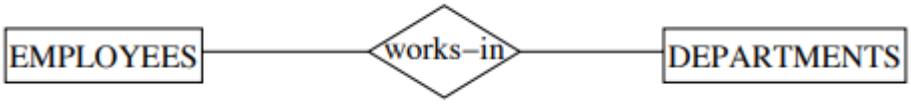


- Rectangles represent entity types
- Ellipses represent attributes
- Diamonds represent relationship types
- Lines link attributes to entity types and entity types to relationship types
- Primary key attributes are underlined
- Empty Circle at the end of a line linking an attribute to an entity type represents an optional (null) attribute
- Double Ellipses represent multi-valued attributes

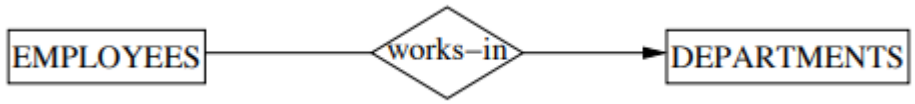
3> Relationship

A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities. There are four types of relationships:
1. One to One
2. One to Many
3. Many to One
4. Many to Many

Many-To-Many (default)
Meaning: An employee can work in many departments (≥ 0),and a department can have several employees

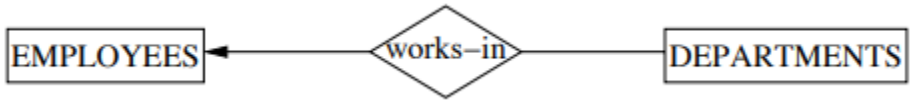


Many-To-One
Meaning: An employee can work in at most one department (≤ 1), and a department can have several employees.



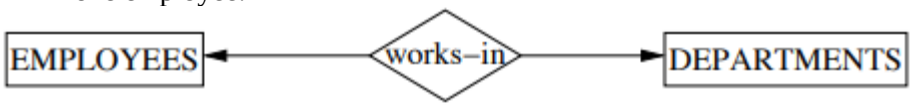
One-To-Many

Meaning: An employee can work in many departments (≥ 0), but a department can have at most one employee.



One-To-One

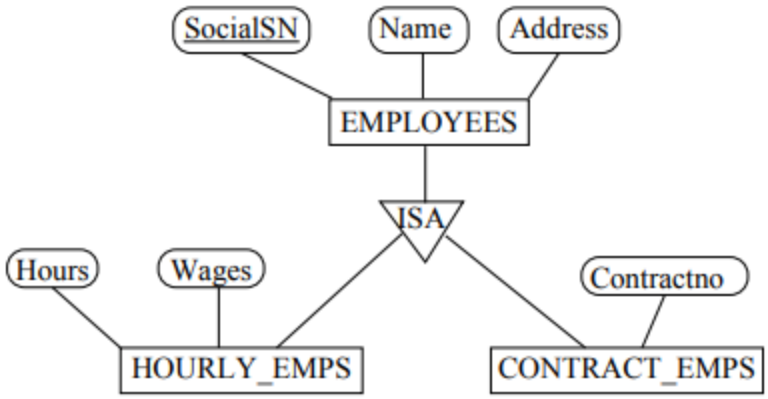
Meaning: An employee can work in at most one department, and a department can have at most one employee.



Specialization

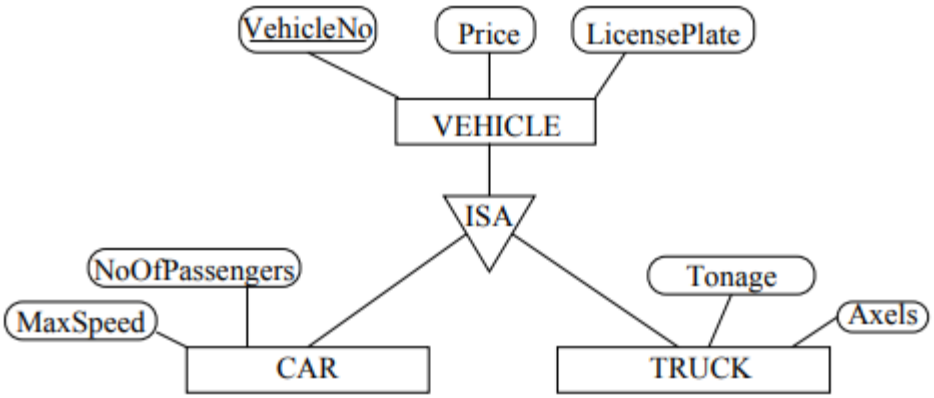
• Process of defining a set of subclasses of an entity type (top-down)

HOURLY EMPS is a subclass of EMPLOYEES and thus inherits its attributes and relationships (same for CONTRACT EMPS).



Generalization:

• Reverse process of specialization (bottom-up); identify common features of entity types and generalize them into single superclass (including primary key!)



Steps in Designing an Entity-Relationship Schema

[Step 1] Identify entity types (entity type vs. attribute)
[Step 2] Identify relationship types
[Step 3] Identify and associate attributes with entity and relationship types
[Step 4] Determine attribute domains
[Step 5] Determine primary key attributes for entity types
[Step 6] Associate (refined) cardinality ratio(s) with relationship types
[Step 7] Design generalization/specialization hierarchies including constraints (includes natural language statements as well)

Translation of ER Schema into Tables

• An ER schema can be represented by a collection of tables which represent contents of the database (instance).
• Primary keys allow entity types and relationship types to be expressed uniformly as tables.
• For each entity and relationship type, a unique table can be derived which is assigned the name of the corresponding entity or relationship type.
• Each table has a number of columns that correspond to the attributes and which have unique names. An attribute of a table has the same domain as the attribute in the ER schema.
• Translating an ER schema into a collection of tables is the basis for deriving a relational database schema from an ER diagram.

ERD Plus: A database modeling tool for creating Entity Relationship Diagrams, Relational Schemas, Star Schemas, and SQL DDL statements.
https://erdplus.com/standalone

Assignment No .2

Title of Assignment: 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 DMLstatements. Note: Instructor will design the queries which demonstrate the use of concepts like Insert, Select, Update, Delete with operators, functions, and set operator etc.

Course Objective:

Implement SQL queries for given requirements, using different SQL concepts

Software Required: - Mysql

Theory: -

Database: - a structured set of data held in a computer, especially one that is accessible in various ways. **Database System**: - A database management system (DBMS) is a computer software application that interacts with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases.
A database management system (DBMS) is a software tool that makes it possible to organize data in a database. **A database** is a collection of information that is organized so that it can be easily accessed, managed and updated.

SQL: The Language of Database Access: -

Structured Query Language (SQL) is a standardized programming language for accessing and manipulating databases. In an RDBMS like MySQL, Sybase, Oracle, or IMB DM2, SQL writes programming that can manage data and stream data processing. SQL is like a database's own version of a server-side script and is responsible for:

- Executing queries, which are "questions" asked of the database
- Retrieving data
- Editing data: inserting, updating, deleting, or creating new records
- Creatingviews
- Setting permissions
- Creating new databases

SQL is a standard programming language, but has a number of variations—including some databases' own proprietary SQL extensions.

Top 7 open source relational databases: -

CUBRID, Firebird, MariaDB, MySQL, Postgre SQL, SQLite

RDBMS Terminology

Before we proceed to explain the MySQL database system, let us revise a few definitions related to the database.

- **Database** − A database is a collection of tables, with related data.
- **Table** − A table is a matrix with data. A table in a database looks like a simple spreadsheet.
- **Column** − One column (data element) contains data of one and the same kind, for example the column postcode.
- **Row** − A row (= tuple, entry or record) is a group of related data, for example the data of one subscription.
- **Redundancy** − Storing data twice, redundantly to make the system faster.
- **Primary Key** − A primary key is unique. A key value can not occur twice in one table. With a key, you can only find one row.
- **Foreign Key** − A foreign key is the linking pin between two tables.
- **Compound Key** − A compound key (composite key) is a key that consists of multiple columns, because one column is not sufficiently unique.
- **Index** − An index in a database resembles an index at the back of a book.
- **Referential Integrity** − Referential Integrity makes sure that a foreign key value always points to an existing row.

MySQL Database

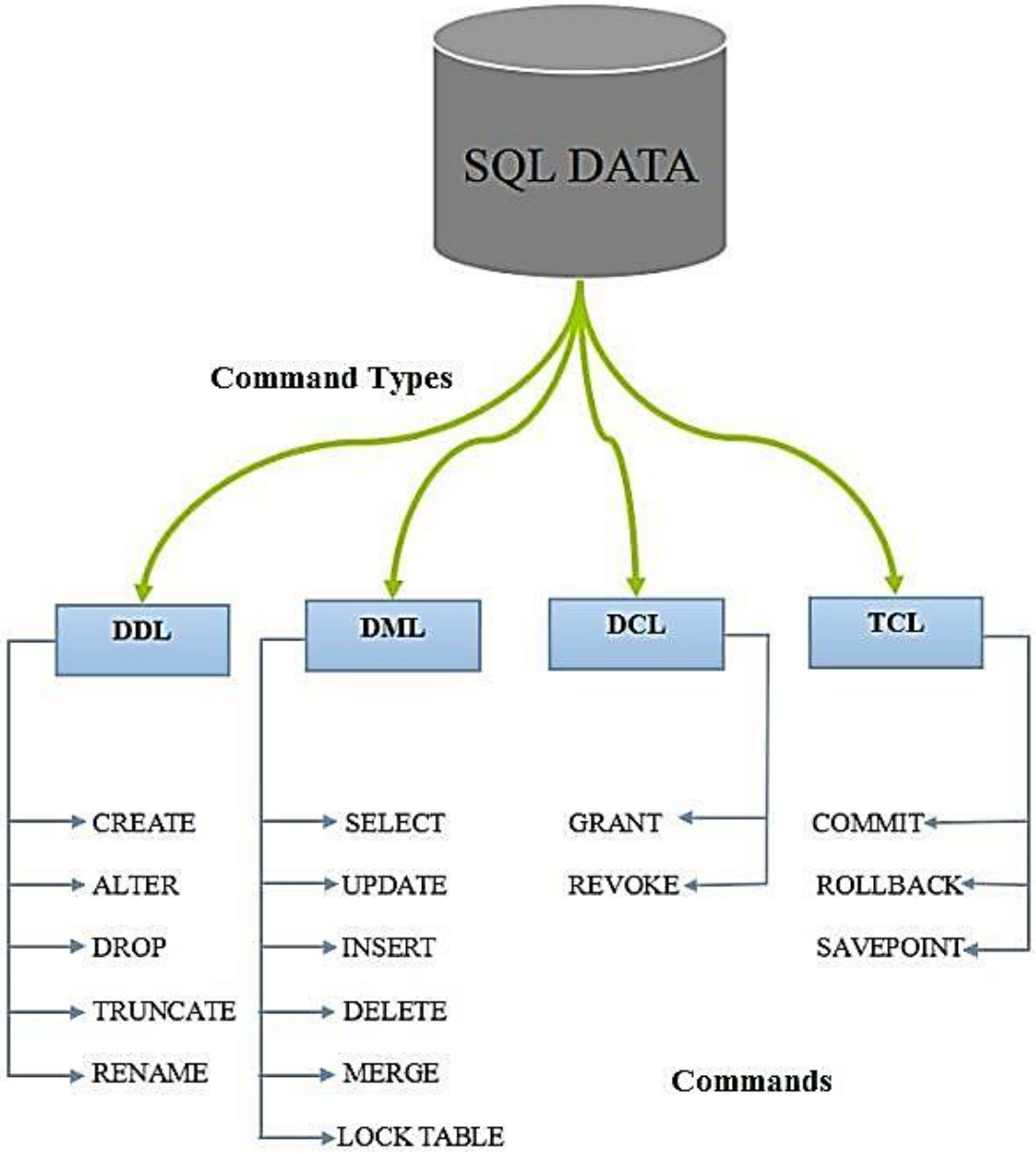
MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons −

- MySQL is released under an open-source license. So you have nothing to pay to use it.
- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets.
- MySQL is very friendly to PHP, the most appreciated language for web development.
- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
- MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

Administrative MySQL Command

Here is the list of the important MySQL commands, which you will use time to time to work with MySQL database −

- **USE Databasename** − this will be used to select a database in the MySQL work area.
- **SHOW DATABASES** − Lists out the databases that are accessible by the MySQL DBMS.
- **SHOW TABLES** − Shows the tables in the database once a database has been selected with the use command.
- **SHOW COLUMNS FROM *tablename:*** Shows the attributes, types of attributes, key information, whether NULL is permitted, defaults, and other information for a table.
- **SHOW INDEX FROM tablename** − Presents the details of all indexes on the table, including the PRIMARY KEY.



MySQL Connection Using MySQL Binary

You can establish the MySQL database using the **mysql** binary at the command prompt.

Example

Here is a simple example to connect to the MySQL server from the command prompt − [root@host]#mysql-u root -p
Enter password:******

This will give you the mysql> command prompt where you will be able to execute any SQL command. Following is the result of above command −
The following code block shows the result of above code − Welcome to the MySQL monitor. Commands end with ; or \g. Your MySQL connection id is 2854760 to server version: 5.0.9

Type 'help;' or '\h' for help.Type '\c' to clear the buffer.

In the above example, we have used **root** as a user but you can use any other user as well. Any user will be able to perform all the SQL operations, which are allowed to that user.
You can disconnect from the MySQL database any time using the **exit** command at mysql> prompt. mysql> exit
Bye

USE DatabaseName;

Always the database name should be unique within the RDBMS.

DROP DATABASE databaseName -- Delete the database (irrecoverable!)
DROP DATABASE IF EXISTS databaseName -- Delete if it exists
CREATE DATABASE databaseName -- Create a new database

CREATE DATABASE IF NOT EXISTS databaseName -- Create only if it does not exists SHOW DATABASES -- Show all the databases in this server
USE databaseName -- Set the default (current) database SELECT DATABASE() -- Show the default database
SHOW CREATE DATABASE databaseName -- Show the CREATE DATABASE statement

-- Table-Level

DROP TABLE [IF EXISTS] tableName, ... CREATE TABLE [IF NOT EXISTS] tableName (
columnName columnType columnAttribute, ... PRIMARY KEY(columnName),
FOREIGN KEY (columnNmae) REFERENCES tableName (columnNmae)

)

SHOW TABLES -- Show all the tables in the default databaseDESCRIBE|DESC tableName -- Describe the details for a table

ALTER TABLE tableName ... -- Modify a table, e.g., ADD COLUMN and DROP COLUMNALTER TABLE tableName ADD columnDefinition ALTER TABLE tableName DROP columnName
ALTER TABLE tableName ADD FOREIGN KEY (columnNmae) REFERENCES tableName (columnNmae)ALTER TABLE tableName DROP FOREIGN KEY constraintName
SHOW CREATE TABLE tableName -- Show the CREATE TABLE statement for this tableName

Data Manipulation Language (DML)

Allows you to modify the database instance by inserting, modifying, and deleting its data. It is responsible for performing all types of data modification in a database. There are three basic constructs which allow database program and user to enter data and information are:
Here are some important DML commands in SQL:
INSERT UPDATE DELETE

INSERT:

This is a statement is a SQL query. This command is used to insert data into the row of a table. Syntax:

INSERT INTO TABLE_NAME (col1, col2, col3, col N)
VALUES (value1, value2, value3, valueN);
Or
INSERT INTO TABLE_NAME
VALUES (value1, value2, value3, valueN);
For example:

INSERT INTO students (RollNo, FIrstName, LastName) VALUES ('60', 'Tom', Erichsen');

UPDATE:

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

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

UPDATE students
SET FirstName = 'Jhon', LastName= 'Wick' WHERE StudID = 3;

DELETE:

This command is used to remove one or more rows from a table. Syntax:

DELETE FROM table_name [WHERE condition]; For example:

DELETE FROM students WHERE FirstName = 'Jhon';

I.Set Operators

Set operators combine the results of two component queries into a single result. Queries containing set operators are called compound queries. The lists of SQL set operators are "UNION [ALL], INTERSECT, MINUS Operators".You can combine multiple queries usingthe set operators UNION, UNION ALL, INTERSECT, and MINUS. All set operators have

equal precedence. If a SQL statement contains multiple set operators, then Oracle Databaseevaluates them from the left to right unless parentheses explicitly specify another order.Thecorresponding expressions in the select lists of the component queries of a compound querymust match in number and must be in the same datatype group (such as numeric or character).

If component queries select character data, then the datatype of the return values aredetermined as follows:

- If both queries select values of datatype CHAR of equal length, then the returned values have datatype CHAR of that length. If the queries select values of CHAR with different lengths, then the returned value is VARCHAR2 with the length of the larger CHAR value.
- If either or both of the queries select values of datatype VARCHAR2, then the returned values have datatype VARCHAR2.

If component queries select numeric data, then the datatype of the return values is determinedby numeric precedence:

- If any query selects values of type BINARY_DOUBLE, then the returned values have datatype BINARY_DOUBLE.
- If no query selects values of type BINARY_DOUBLE but any query selects values of type BINARY_FLOAT, then the returned values have datatype BINARY_FLOAT.
- If all queries select values of type NUMBER, then the returned values have datatype NUMBER.

In queries using set operators, Oracle does not perform implicit conversion across datatype groups. Therefore, if the corresponding expressions of component queries resolve to both character data and numeric data, Oracle returns an error to fetch only unique records insteadof fetching duplicate records.

II. SQL Functions

SQL functions are built into Oracle Database and are available for use in various appropriate SQL statements. Do not confuse SQL functions with user-defined functions written in PL/SQL.If you call a SQL function with an argument of a datatype other than the datatype expected by the SQL function, then Oracle attempts

to convert the argument to the expected datatype before performing the SQL function. If you call a SQL function with a null argument, then the SQL function automatically returns null.

1. Single-Row Functions

Single-row functions return a single result row for every row of a queried table or view. These functions can appear in select lists, WHERE clauses, STARTWITH and CONNECT BY clauses, and HAVING clauses.

2. Character Functions Returning Character Value

Character functions that return character values return values of the following datatypesunless otherwise documented:

- If the input argument is CHAR or VARCHAR2, then the value returnedis VARCHAR2.
- If the input argument is NCHAR or NVARCHAR2, then the value returnedis NVARCHAR2.

The length of the value returned by the function is limited by the maximum length of thedatatype returned.

- For functions that return CHAR or VARCHAR2, if the length of the return value exceeds the limit, then Oracle Database truncates it and returns the result without an error message.
- For functions that return CLOB values, if the length of the return values exceeds the limit, then Oracle raises an error and returns no data.

3. Datetime Functions

Datetime functions operateon date (DATE), timestamp(TIMESTAMP, TIMESTAMP WITH TIME ZONE,and TIMESTAMP WITH L
E ZONE), and interval (INTERVAL DAY TO SECOND, INTERVAL YEAR values.
Some of the datetime functions were designed for the Oracle DATE dat atype (ADD_MONTHS,
CURRENT_DATE, LAST_DAY, NEW_TIME, and NEXT_DAY). If you

provide a timestamp value as their argument, Oracle Database internally converts the input type toa DATE value and returns a DATE value. The exceptions are the MONTHS_BETWEEN function,which returns a number, and the ROUND and TRUNC functions, which do not accept timestamp or interval values at all. The

remaining datetime functions were designed to accept any of the three types of data (date, timestamp, and interval) and to return a value of one of these types.

Conclusion:

Students are able to implement SQL queries for given requirements, using different SQL concepts

Activity to be Submitted by Students

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

Here is the structure of the table jobs;

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

2. Write a SQL statement to create a table employees including columns employee_id, first_name, last_name, job_id, salary and make sure that, the employee_id column does not contain any duplicate value at the time of insertion, and the foreign key column job_id, referenced by the column job_id of jobs table, can contain only those values which are exists in the jobs table. The InnoDB Engine have been used to create the tables. The specialty of the statement is that, The ON DELETE NO ACTION and the ON UPDATE NO ACTION actions will reject the deletion and any updates.

3. Consider the following schema for a LibraryDatabase: BOOK (Book_id, Title, Publisher_Name, Pub_Year) BOOK_AUTHORS (Book_id, Author_Name) PUBLISHER (Name, Address, Phone) BOOK_COPIES (Book_id, Branch_id, No-of_Copies) BOOK_LENDING (Book_id, Branch_id, Card_No, Date_Out, Due_Date) LIBRARY_BRANCH (Branch_id, Branch_Name, Address) Write SQL queries to

1. Retrieve details of all books in the library – id, title, name of publisher, authors, number of copies in each branch,etc.

2. Get the particulars of borrowers who have borrowed more than 3 books, but from

Jan 2017 to Jun2017

3. Delete a book in BOOK table. Update the contents of other tables to reflect this data manipulationoperation.

4. Partition the BOOK table based on year of publication. Demonstrate its working with a simple query.

5. Create a view of all books and its number of copies that are currently available in the Library.

Assignment No .3

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

Write at least10 SQL queries for suitable database application using SQL DML statements. Note: Instructor will design the queries which demonstrate the use of concepts like all types of Join, Sub-Query and View

Implement SQL queries for given requirements, using different SQL concepts

Software Required: - Mysql

Theory: -

Join in SQL

SQL Join is used to fetch data from two or more tables, which is joined to appear as single set of data. SQL Join is used for combining column from two or more tables by using values common to both tables. **Join** Keyword is used in SQL queries for joining two or more tables. Minimum required condition for joining table, is **(n-1)** where **n**, is number of tables. A table can also join to itself known as, **Self Join**.

Types of Join:-

The following are the types of JOIN that we can use in SQL.

- Inner
- Outer
- Left
- Right

Cross JOIN or Cartesian Product

This type of JOIN returns the Cartesian product of rows from the tables in Join. It will return a table which consists of records which combines each row from the first table with each row of the second table.
Cross JOIN Syntax is, SELECT column-name-list from *table-name1*
CROSS JOIN

table-name2;

INNER Join or EQUI Join

This is a simple JOIN in which the result is based on matched data as per the equality condition

specified in the query. Inner Join Syntax is, SELECT column-name-list from *table-name1*
INNER JOIN

table-name2

WHERE table-name1.column-name = table-name2.column-name;

Natural JOIN

Natural Join is a type of Inner join which is based on column having same name and same datatype present in both the tables to be joined.
Natural Join Syntax is,
SELECT *

from *table-name1*

NATURAL JOIN

table-name2;

Outer JOIN

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

- Left Outer Join
- Right Outer Join
- Full Outer Join

Left Outer Join

The left outer join returns a result table with the **matched data** of two tables then remaining rows of the **left** table and null for the **right** table's column.
Left Outer Join syntax is, SELECT column-name-list from *table-name1*
LEFT OUTER JOIN

table-name2

on table-name1.column-name = table-name2.column-name; Left outer Join Syntax for **Oracle** is,
select column-name- listfrom *table-name1*, *table-name2*

on table-name1.column-name = table-name2.column-name(+);

Left Outer Join query will be,

SELECT * FROM class LEFT OUTER JOIN class_info ON (class.id=class_info.id); The result table will look like,

ID	NAME	ID	ADDRESS
1	Abhi	1	DELHI
2	Adam	2	MUMBAI
3	Alex	3	CHENNAI

4	Anu	Null	Null

5	Ashish	Null	Null

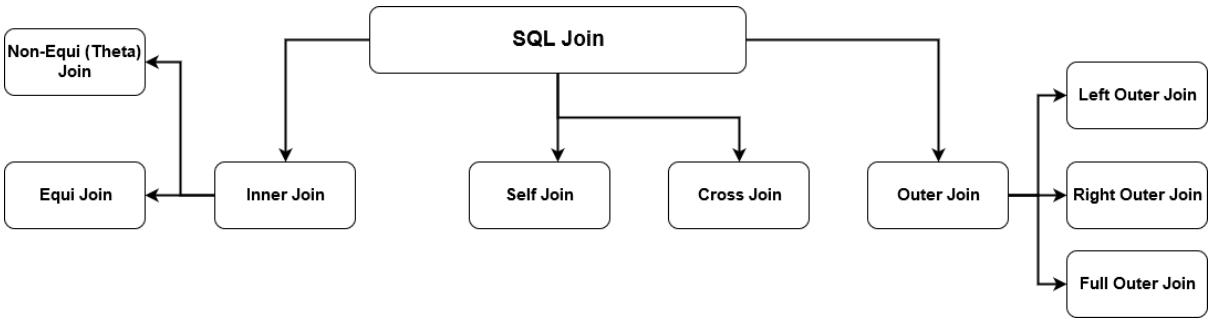
Right Outer Join

The right outer join returns a result table with the **matched data** of two tables then remaining rows of the **right table** and null for the **left** table's columns.
select column-name-list from *table-name1* **RIGHT OUTER JOIN**

table-name2

on table-name1.column-name = table-name2.column-name; Right outer Join Syntax for **Oracle** is,
select column-name- listfrom *table-name1*, *table-name2*

on table-name1.column-name(+) = table-name2.column-name;



SQL Subquery

Subquery or **Inner query** or **Nested query** is a query in a query. SQL subquery is usually added in the WHERE Clause of the SQL statement. Most of the time, a subquery is used when you know how to search for a value using a SELECT statement, but do not know the exact value in the database.
Subqueries are an alternate way of returning data from multiple tables.
Subqueries can be used with the following SQL statements along with the comparision operators like =, <, >, >=, <= etc.

- SELECT
- INSERT
- UPDATE
- DELETE
- SQL Subquery Example:

- 1) Usually, a subquery should return only one record, but sometimes it can also return multiple records when used with operators LIKE IN, NOT IN in the where clause. The query syntax would be like,
- SELECT first_name, last_name, subject FROM student_details WHERE games NOT IN ('Cricket', 'Football');
- **Subquery** output would be similar to:

first_name	last_name	subject
Shekar	Gowda	Badminton
Priya	Chandra	Chess

SQL Subquery; INSERT Statement

3. Subquery can be used with INSERT statement to add rows of data from one or more tables to another table. Lets try to group all the students who study Maths in a table 'maths_group'. INSERT INTO maths_group(id, name)
SELECT id, first_name || ' ' || last_name
FROM student_details WHERE subject= 'Maths'

SQL Subquery; SELECT Statement

4. A subquery can be used in the SELECT statement as follows. Lets use the product and order_items table defined in the sql_joins section.
select p.product_name, p.supplier_name, (select order_id from order_items where product_id = 101) as order_id from product p where p.product_id = 101

product_name	supplier_name	order_id
Television	Onida	5103

Correlated Subquery

A query is called correlated subquery when both the inner query and the outer query are interdependent. For every row processed by the inner query, the outer query is processed as well. The inner query depends on the outer query before it can be processed.

SELECT p.product_name FROM product p

WHERE p.product_id = (SELECT o.product_id FROM order_items o WHERE o.product_id = p.product_id);

Subquery Notes Nested Subquery

1. You can nest as many queries you want but it is recommended not to nest more than 16 subqueries in oracle

Non-Corelated Subquery

2. If a subquery is not dependent on the outer query it is called a non-correlated subquery

Subquery Errors

3. Minimize subquery errors: Use drag and drop, copy and paste to avoid running subqueries with spelling and database typos. Watch your multiple field SELECT comma use, extra or to few getting SQL error message "Incorrect syntax".

SQL Subquery Comments

Adding SQL Subquery comments are good habit (/* your command comment */) which can saveyou time, clarify your previous work .. results in less SQL headaches.

SQL Views

A VIEW is a virtual table, through which a selective portion of the data from one or more tablescan be seen. Views do not contain data of their own. They are used to restrict access to the database or to hide data complexity. A view is stored as a SELECT statement in the database.

DML operations on a view like INSERT, UPDATE, DELETE affects the data in the originaltable upon which the view is based.

The Syntax to create a sql view is

CREATE VIEW view_nameAS SELECT column_list

FROM table_name [WHERE condition];

- ***view_name*** is the name of the VIEW.
- The SELECT statement is used to define the columns and rows that you want to displayin the view.
- **For Example:** to create a view on the product table the sql query would be like,CREATE VIEW view_product

AS SELECT product_id, product_nameFROM product;

Conclusion:

Students are able to implement SQL queries all types of Join, Sub-Query and View for given requirements, using different SQL concepts

Activity to be Submitted by Students

1. Consider the schema for MovieDatabase:
ACTOR (Act_id, Act_Name, Act_Gender) DIRECTOR (Dir_id, Dir_Name, Dir_Phone)
MOVIES (Mov_id, Mov_Title, Mov_Year, Mov_Lang, Dir_id) MOVIE_CAST (Act_id, Mov_id, Role) RATING (Mov_id,
Rev_Stars) Write SQL queries to

1. List the titles of all movies directed by‘Hitchcock’.
2. Find the movie names where one or more actors acted in two or moremovies.
3. List all actors who acted in a movie before 2000 and also in a

movieafter 2015 (use JOINoperation).

4. Find the title of movies and number of stars for each movie that has at least one rating and find the highest number of stars that movie received. Sort the result by movie title.
5. Update rating of all movies directed by ‘Steven Spielberg’ to5.

2. Apply Self Join to Student(Roll,name,address,branch,Class)

3. From the following table, create a view for all salespersons. Return salesperson ID, name, and city.

Sample table: salesman

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5007	Paul Adam	Rome	0.13
5003	Lauson Hen	San Jose	0.12

Assignment No .4

Title of Assignment: Unnamed PL/SQL code block: Use of Control structure and Exception handling ismandatory.

Suggested 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 perday. 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.

OR

Write a PL/SQL code block to calculate the area of a circle for a value of radius varying from 5 to 9. Store the radius and the corresponding values of calculated area in an empty table named areas, consisting of two columns, radius and area.

Note: Instructor will frame the problem statement for writing PL/SQL block in line with above statement

Course Objective:

Implement PL/SQL Code block for given requirements

Course Outcome:

C306.4 Implement PL/SQL Code block for given requirements

Software Required: - Mysql

Theory: -

A. **Control Structures:** - PL/SQL allows the use of an IF statement to control the execution of a block of code. In PL/SQL, the IF -THEN - ELSIF - ELSE - END IF construct in code blocks allow specifying certain conditions under which a specific block of code should be execute PL/SQL Control Structures are used to control flow of execution. PL/SQL provides different kinds of statements to provide such type of procedural capabilities. These statements are almost same as that of provided by other

languages. The flow of control statements can be classified into the following categories:

- Conditional Control
- Iterative Control
- Sequential Control

ConditionalControl:

PL/SQL allows the use of an IF statement to control the execution of a block of code. In PL/SQL, the IF -THEN - ELSIF - ELSE - END IF construct in code blocks allow specifying certain conditions under which a specific block of code should be executed.

Iterative Control :

Iterative control indicates the ability to repeat or skip sections of a code block. A **loop** marks a sequence of statements that has to be repeated. The keyword loop has to be placed before the first statement in the sequence of statements to be repeated, while the keyword end loop is placed immediately after the last statement in the sequence. Once a loop begins to execute, it will go on forever. Hence a conditional statement that controls the number of times a loop is executed always accompanies loops. PL/SQL supports the following structures for iterative control:

Simple loop : In simple loop, the key word loop should be placed before the first statement in the sequence and the keyword end loop should be written at the end of the sequence to end the loop.

Syntax:

1. **WHILE loop**

The while loop executes commands in its body as long as the condition remains true

2. **The FOR Loop**

The FOR loop can be used when the number of iterations to be executed are known.

Syntax :

The variable in the For Loop need not be declared. Also the increment value cannot be specified. The For Loop variable is always incremented by 1.

3. Sequential Control :

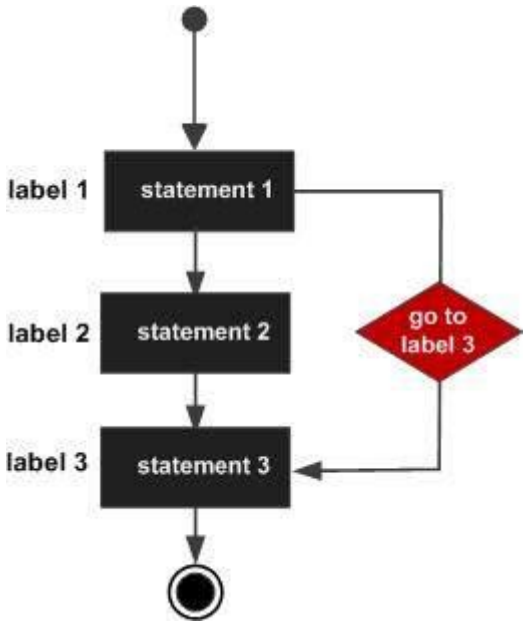
The GOTO Statement

The GOTO statement changes the flow of control within a PL/SQL block. This statement allows execution of a section of code, which is not in the normal flow of control. The entry point into such a block of code is marked using the tags «userdefined name». The GOTO statement can then make use of this user-defined name to jump into that block of code for execution.

Syntax :

```
GOTO label;
..
..
<< label >>
statement;
```

Flow Diagram:-



B. Exceptions

An Exception is an error situation, which arises during program execution. When an error occurs exception is raised, normal execution is stopped and

control transfers to exception handling part. Exception handlers are routines written to handle the exception. The exceptions can be internally defined (system-defined or pre-defined) or User-defined exception.

Syntax:

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling goes here >
  WHEN exception1 THEN
    exception1-handling-statements
  WHEN exception2  THEN
    exception2-handling-statements
  WHEN exception3 THEN
    exception3-handling-statements
  ........
  WHEN others THEN
    exception3-handling-statements
END;
```

Predefined exception:

Predefined exception is raised automatically whenever there is a violation of Oracle coding rules. Predefined exceptions are those like ZERO_DIVIDE, which is raised automatically when we try to divide a number by zero. Other built-in exceptions are given below. You can handle unexpected Oracle errors using OTHERS handler. It can handle all raised exceptions that are not handled by any other handler. It must always be written as the last handler in exception block.

Exception	**Raised when....**
DUP_VAL_ON_INDEX	When you try to insert a duplicate value into a unique column.
INVALID_CURSOR	It occurs when we try accessing an invalid cursor.

INVALID_NUMBER	On usage of something other than number in place of number value.

LOGIN_DENIED	At the time when user login is denied.
TOO_MANY_ROWS	When a select query returns more than one row and the destination variable can take only single value.
VALUE_ERROR	When an arithmetic, value conversion, truncation, or constraint error occurs.
CURSOR_ALREADY_OPEN	Raised when we try to open an already open cursor.

Predefined exception handlers are declared globally in package STANDARD. Hence we need not have to define them rather just use them. The biggest advantage of exception handling is it improves readability and reliability of the code. Errors from many statements of code can be handles with a single handler. Instead of checking for an error at every point we can just add an exception handler and if any exception is raised it is handled by that. For checking errors at a specific spot it is always better to have those statements in a separate begin – end block.

User Defined Exception Handling:

To trap business rules being violated the technique of raising user-defined exceptions and then handling them, is used. User-defined error conditions must be declared in the declarative part of any PL/SQL block. In the executable part, a check for the condition that needs special attention is made. If that condition exists, the call to the user-defined exception is made using a RAISE statement. The exception once raised is then handled in the Exception handling section of the PL/SQL code block.

Syntax:

```
DECLARE
  my-exception EXCEPTION;
```

Conclusion:

Students are able to Implement PL/SQL Code block for given requirements

Activity to be Submitted by Students

1. Write a PL/SQL block which accepts a cleaner number and returns the cleaners name and salary and update this cleaner's details by increasing salary by 10%. Use Exception Handling.

2. Write a PL/SQL block to find Largest of three numbers

Assignment No .5

Title of Assignment: 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)

Note: Instructor will frame the problem statement for writing stored procedure and Function in line with above statement**.**

Course Objective:

Implement PL/SQL Code block for given requirements

Course Outcome:

C306.4 Implement PL/SQL Code block for given requirements

Software Required: - Mysql

Theory: -

Stored Procedures : A stored procedure or in simple a proc is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages. A procedure has a header anda body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists or declaration section, execution section and exception section similar to a general PL/SQL Block. A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

Procedures: Passing Parameters

We can pass parameters to procedures in three ways.

1. IN-parameters
2. OUT-parameters
3. IN OUT-parameters

PL/SQL Create Procedure

Syntax for creating procedure:

```
1. CREATE [OR REPLACE] PROCEDURE procedure_name
2.     [ (parameter [,parameter]) ]
3. IS
4.     [declaration_section]
5. BEGIN
```

```
executable_section
[EXCEPTION
    exception_section]
END [procedure_name];
```

Create procedure example

In this example, we are going to insert record in user table. So you need to create user table first.

Table creation:

```
create table user(id number(10) primary key,name varchar2(100));
```

Now write the procedure code to insert record in user table.

Procedure Code:

```
create or replace procedure "INSERTUSER"
(id IN NUMBER,
name IN VARCHAR2)
is
begin
insert into user values(id,name);
end;
/
```

Output:

```
Procedure created.
```

Pl sql function:

The pl sql function is a named PL/SQL block which performs one or more specific tasks and must returns a value.

How to pass parameter in a function?

We can use the below modes to pass the parameters in a function:

IN-parameters: These parameters are the read-only parameters. Function cannot change the value of IN parameters.

OUT-parameters: These parameters are the write-only parameters and used to return values back to the calling program. Function can change the value of OUT parameters.

IN OUT-parameters: These parameters are read and write parameters i.e. a function can reads and change the IN OUT parameter value and return it back to the calling program.

Syntax of pl sql function:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
RETURN return_datatype;
IS|AS
        //Declaration block
BEGIN
        //Execution_block
        Return return_variable;
EXCEPTION
        //Exception block
        Return return_variable;
END;
 /
```

How to create a function?

```
create or replace function getMultiple(num1 in number, num2 in number)
return number
is
  num3 number(8);
begin
```

```
 num3 :=num1*num2;
 return num3;
end;
/
```

How to execute a function?

A functions return value can be assign to a variable.

```
result :=  getMultiple(4, 5);
```

As a part of a SELECT statement:

```
SELECT getMultiple(4, 5) FROM dual;
```

In a PL/SQL Statement:

```
dbms_output.put_line(getMultiple(4, 5));
```

How to drop a function?

```
DROP FUNCTION function_name;
```

Conclusion:

Students are able to PL/SQL Stored Procedure and Stored Function

Activity to be Submitted by Students

1. Implement Stored Procedure to Check if a given a year is a leap year. The condition is:- year should be (divisible by 4 and not divisible by 100) or (divisible by 4 and divisible by 400.) Display the output on the screen using dbms_output.put_line. The year should be input by the user.

2. Implement Stored Procedure or Function to read in a number and print it out digit by digit, as a series of words. For example, the number 523 would be printed as "five two three". Use decode function within a for loop. Display the

results on the screen using dbms_output.put_line.

Assignment No .6

Title of Assignment: Cursors :(All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor) Write a PL/SQL block of code using parameterized Cursor that will merge the data available in the newly created table N_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.

Note: Instructor will frame the problem statement for writing PL/SQL block using all types of Cursors in line with above statement

Course Objective:

Implement PL/SQL Code block for given requirements

Course Outcome:

C306.4 Implement PL/SQL Code block for given requirements

Software Required: - Mysql

Theory: - PL/SQL Cursor

Summary: in this tutorial, we will introduce you to **PL/SQL cursor**. You will learn step by step how to use a cursor to loop through a set of rows and process each row individually.

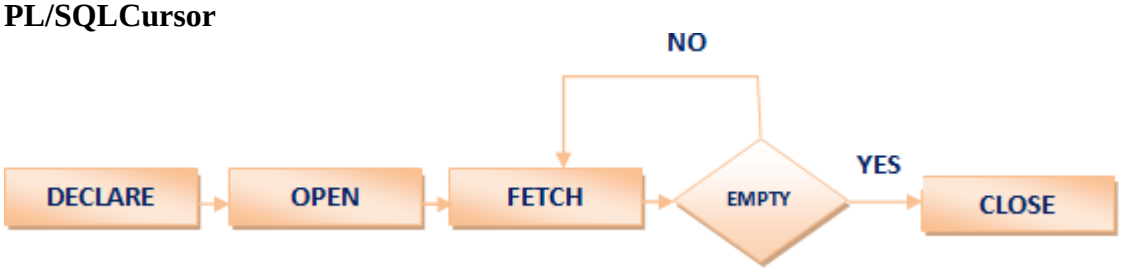
Introducing to PL/SQL Cursor

When you work with Oracle database, you work with a complete set of rows returned from an SQL SELECT statement. However the application in some cases cannot work effectively with the entire result set, therefore, the database server needs to provide a mechanism for the application to work with one row or a subset of the result set at a time. As the result, Oracle created PL/SQL cursor to provide these extensions.

A PL/SQL cursor is a pointer that points to the result set of an SQL query against database tables. Working with PL/SQL Cursor

The following picture describes steps that you need to follow when you work with a PL/SQL cursor:

PL/SQLCursor



Declaring PL/SQL Cursor

To use PL/SQL cursor, first you must declare it in the declaration section of PL/SQL block or in a package as follows:

```
CURSOR cursor_name
IS
  SELECT_statement;
```

Example

For example, you could define a cursor called c1 as below.

```
CURSOR c1
IS
  SELECT course_number
  FROM courses_tbl
  WHERE course_name = name_in;
```

The result set of this cursor is all course_numbers whose course_name matches the variable called name_in.

Below is a function that uses this cursor.

```
CREATE OR REPLACE Function FindCourse
   ( name_in IN varchar2 )
   RETURN number
IS
   cnumber number;

   CURSOR c1
   IS
     SELECT course_number
     FROM courses_tbl
     WHERE course_name = name_in;

BEGIN

   OPEN c1;
   FETCH c1 INTO cnumber;
```

```
  if c1%notfound then
     cnumber := 9999;
  end if;

  CLOSE c1;

RETURN cnumber;

END;
```

Cursor with parameters

As we get more complicated, we can declare cursors with parameters.

Syntax

The syntax for a **cursor with parameters** in Oracle/PLSQL is:

```
CURSOR cursor_name (parameter_list)
IS
  SELECT_statement;
```

Example

For example, you could define a cursor called c2 as below.

```
CURSOR c2 (subject_id_in IN varchar2)
IS
  SELECT course_number
  FROM courses_tbl
  WHERE subject_id = subject_id_in;
```

The result set of this cursor is all course_numbers whose subject_id matches the subject_id passed to the cursor via the parameter.

Cursor with return clause

Finally, we can declare a cursor with a return clause.

Syntax

The syntax for a cursor with a return clause in Oracle/PLSQL is:

```
CURSOR cursor_name
RETURN field%ROWTYPE
IS
  SELECT_statement;
```

Example

For example, you could define a cursor called c3 as below.

```
CURSOR c3
RETURN courses_tbl%ROWTYPE
IS
  SELECT *
  FROM courses_tbl
  WHERE subject = 'Mathematics';
```

Types of Cursors

Cursors are classified depending on the circumstances in which they are opened.

- **Implicit Cursor:** If the Oracle engine opened a cursor for its internal processing it is known as an Implicit Cursor. It is created "automatically" for the user by Oracle when a query is executed and is simpler to code.
- **Explicit Cursor:** A Cursor can also be opened for processing data through a PL/SQL block, on demand. Such a user-defined cursor is known as an Explicit Cursor.

Explicit cursor

An explicit cursor is defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row. A suitable name for the cursor.

General syntax for creating a cursor:

CURSOR cursor_name IS select_statement;

cursor_name – A suitable name for the cursor.

select_statement – A select query which returns multiple rows

How to use Explicit Cursor?
There are four steps in using an Explicit Cursor.

1. DECLARE the cursor in the Declaration section.
2. OPEN the cursor in the Execution Section.
3. FETCH the data from the cursor into PL/SQL variables or records in the Execution Section.
4. CLOSE the cursor in the Execution Section before you end the PL/SQL Block.

Syntax:

```
DECLARE variables;
 records;
 create a cursor;
 BEGIN
OPEN cursor;
FETCH cursor;
 process the records;
 CLOSE cursor;
END;
```

Conclusion: We have implemented all types of Cursors successfully.

Activity to be Submitted by Students

Write PL/SQL code to display Employee details using Explicit Cursors Write PL/SQL code in Cursor to display employee names and salary.

Assignment No .7

Title of Assignment: Database Trigger (All Types: Row level and Statement level triggers, Before and AfterTriggers).

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

Note: Instructor will Frame the problem statement for writing PL/SQLblock for all types of Triggers in line with above statement**.**

Course Objective:

Implement PL/SQL Code block for given requirements

Course Outcome:

C306.4 Implement PL/SQL Code block for given requirements

Software Required: - Mysql

PL/SQL Trigger

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.Trigger is stored intodatabase and invoked repeatedly, when specific condition match.

Triggers are stored programs, which are automatically executed or fired when some event occurs.Triggers are written to be executed in response to any of the following events.

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN). Triggers could be defined on the table, view, schema, or database with which the event is associated. Advantages of Triggers

These are the following advantages of Triggers:

- Trigger generates some derived column values automatically
- Enforces referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions Creating a trigger:

Syntax for creating trigger:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
```

```
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Where,

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

Example

To start with, we will be using the CUSTOMERS table we had created and used in the previous chapters −

```
Select * from customers;

+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

The following program creates a **row-level** trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values −

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
  sal_diff number;
BEGIN
  sal_diff := :NEW.salary  - :OLD.salary;
  dbms_output.put_line('Old salary: ' || :OLD.salary);
  dbms_output.put_line('New salary: ' || :NEW.salary);
  dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

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

```
Trigger created.
```

The following points need to be considered here −

- OLD and NEW references are not available for table-level triggers, rather you can use them for record-level triggers.
- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.
- The above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using the DELETE operation on the table.

Triggering a Trigger

Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table −

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00 );
```

When a record is created in the CUSTOMERS table, the above create trigger, **display_salary_changes** will be fired and it will display the following result −

```
Old salary:
New salary: 7500
Salary difference:
```

Because this is a new record, old salary is not available and the above result comes as null. Let us now perform one more DML operation on the CUSTOMERS table. The UPDATE statement will update an existing record in the table −

```
UPDATE customers
SET salary = salary + 500
WHERE id = 2;
```

When a record is updated in the CUSTOMERS table, the above create trigger, **display_salary_changes** will be fired and it will display the following result −

```
Old salary: 1500
New salary: 2000
Salary difference: 500
```

PL/SQL: Types of Triggers

The above diagram clearly indicated that Triggers can be classified into three categories:

1. Level Triggers
2. Event Triggers
3. Timing Triggers

which are further divided into different parts.

Level Triggers

There are 2 different types of level triggers, they are:

1. **ROW LEVEL TRIGGERS**
    - It fires for every record that got affected with the execution of DML statements like INSERT, UPDATE, DELETE etc.
    - It always use a FOR EACH ROW clause in a triggering statement.
2. **STATEMENT LEVEL TRIGGERS**
    - It fires once for each statement that is executed.

Event Triggers

There are 3 different types of event triggers, they are:

1. **DDL EVENT TRIGGER**

- It fires with the execution of every DDL statement(CREATE, ALTER, DROP, TRUNCATE).

2. **DML EVENT TRIGGER**

    - It fires with the execution of every DML statement(INSERT, UPDATE, DELETE).

3. **DATABASE EVENT TRIGGER**

    - It fires with the execution of every database operation which can be LOGON, LOGOFF, SHUTDOWN, SERVERERROR etc.

Timing Triggers

There are 2 different types of timing triggers, they are:

- **BEFORE TRIGGER**
    - It fires before executing DML statement.
    - Triggering statement may or may not executed depending upon the before condition block.
- **AFTER TRIGGER**
    - It fires after executing DML statement.

Conclusion: We have implemented all types of Triggers successfully.

Activity to be Submitted by Students

1. Write pl/sql code in Trigger not to accept the existing Empno (Unique no)
2.Write pl/sql code using Trigger to salary with more than old salary

Assignment No .8

Title of Assignment: 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.)

Course Objective:

Implement PL/SQL Code block for given requirements

Course Outcome:

C306.6

Design and develop application considering actual requirements and using database concepts

Software Required: - Mysql/eclipse/ wampserver/php/oracle

Why use JDBC

Before JDBC, ODBC API was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

JDBC Driver

1. JDBC Drivers

1. JDBC-ODBC bridge driver
2. Native-API driver
3. Network Protocol driver
4. Thin driver

JDBC Driver is a software component that enables java application to interact with the database.There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1. JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBCmethod calls into the ODBC function calls. This is now discouraged because of thin driver.

Advantages:

- easy to use.
- can be easily connected to any database. Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

2. Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into

native callso f the database API. It is not written entirely in java. Advantage:

- performance upgraded than JDBC-ODBC bridge driver.Disadvantage:
- The Native driver needs to be installed on the each client machine.
-

- The Vendor client library needs to be installed on client machine.

1. Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.
Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be donein the middle tier.

2. Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side. Disadvantage:
- Drivers depends on the Database. **5 Steps to connect to the database in java**They are as follows:
- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection

1. Register the driver class

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.
Syntax of forName() method

1. public static void forName(String className)throws ClassNotFoundExceptionExample to register the OracleDriver class
1. Class.forName("oracle.jdbc.driver.OracleDriver");

2. Create the connection object

The getConnection() method of DriverManager class is used to establish connection with the database.Syntax of getConnection() method

3. public static Connection getConnection(String url)throws SQLException

4. public static Connection getConnection(String url,String name,String password) throws SQLExceptionExample to establish connection with the Oracle database
5. Connection con=DriverManager.getConnection(

6. "jdbc:oracle:thin:@localhost:1521:xe","system","password");

7. Create the Statement object
The createStatement() method of Connection interface is used to create statement. The object of statement is responsibleto execute queries with the database.
Syntax of createStatement() method public Statement createStatement()throws

SQLExceptionExample to create the statement object Statement stmt=con.createStatement();

8. Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns theobject of ResultSet that can be used to get all the records of a table.
Syntax of executeQuery() method
public ResultSet executeQuery(String sql)throws SQLExceptionExample to execute query
ResultSet rs=stmt.executeQuery("select * from emp"); while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}

9. Close the connection object
By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection. Syntax of close() method
public void close()throws

SQLExceptionExample to close connection con.close();
import java.sql.*; class
Oracle Con{ public static void main(String args[]){try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system", "oracle");Statement stmt=con.createStatement(); ResultSet rs=stmt.executeQuery("select * from emp");

```
while(rs.next())
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
con.close();
}catch(Exception e){ System.out.println(e);} } }
```

PHP Connectivity

PHP will require that mysqli is enabled (it is on most PHP set ups).

```
$db = mysqli_connect('localhost','username','password','database_name') or die('Error connecting to MySQL server.');
```

The variable $db is created and assigned as the connection string, it will be used in future steps. If there is a failure then anerror message will be displayed on the page. If it is successful you will see PHP connect to MySQL.

Performing a database query

The mysql query is actually performed in the body of the html page, so additional php opening and closing tags will be required. For the query we are going to specify a read of all fields from a given table. The $query variable selects all rows in the table. You just need to use your table name.

```
<?php
$query = "SELECT * FROM table_name"; mysqli_query($db, $query) or die('Error querying database.');
```

Again the returned page in the browser should be blank and error free, if you do receive the error – 'Error queryingdatabase..' check the table name is correct.

```
<?php
//Step1
$query = "SELECT * FROM table_name"; mysqli_query($db, $query) or die('Error querying database.');
$result = mysqli_query($db, $query);
$row = mysqli_fetch_array($result);

while ($row = mysqli_fetch_array($result)) {
echo $row['first_name'] . ' ' . $row['last_name'] . ': ' . $row['email'] . ' ' . $row['city'] .'<br />';}?>
</body>
</html>
```

Closing off the connection

Closing the connection will require another set off opening and closing php tags after the closing html tag.

```
<?php
//Step1
$db = mysqli_connect('localhost','root','root','database_name') or die('Error connecting to MySQL server.');
?>
<html>
<head>
```

```
</head>
<body>
<h1>PHP connect to MySQL</h1>
  <?php

  //Step2
  $query = "SELECT * FROM table_name"; mysqli_query($db, $query) or die('Error
  querying database.');

  //Step3
  $result = mysqli_query($db, $query);
  $row = mysqli_fetch_array($result);
  while ($row = mysqli_fetch_array($result)) {
echo $row['first_name'] . ' ' . $row['last_name'] . ': ' . $row['email'] . ' ' . $row['city']
.'<br />';}

  //Step 4
   mysqli_close( $db);
  ?>
  </body>
  </html>
```

Conclusion: We have implemented database connectivity

Activity to be Submitted by Students

1. Login page in PHP

Assignment No .09

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

Course Objective:
To acquire the skills to use a powerful, flexible, and scalable general-purpose databases to handle Big Data

Course Outcome:

Implement NoSQL queries using MongoDB

Software Required: - Mongodb

The use Command use DATABASE_NAME

MongoDB **use DATABASE_NAME** is used to create database. The command will create a new database, if it doesn't existotherwise it will return the existing database.**Syntax:**

Example: If you want to create a database with name **<mydb>**, then **use DATABASE** statement would be as follows:

```
>use mydb
```

To check your currently selected database use the command **db**

```
>db;
```

The dropData

MongoDB **db.dropDatabase()** command is used to drop a existing database.

Syntax: Basic syntax of **dropDatabase()** command is as follows:

```
db.dropDatabase()
```

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database

MongoDB Create Collection The createCollection() Method: MongoDB **db.createCollection(name, options)** is used to create collection. db.createCollection(name, options)

Syntax:

In the command, **name** is name of collection to be created. **Options** is a document and used to specify configuration of collection

The drop() Method

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

Syntax:

MongoDB - Insert Document The insert() Method

The insert() method has the following syntax:

```
db.collection.insert(
<document or array of documents>,
{
writeConcern: <document>,
ordered: <boolean>
}
)
```

Example

```
>db.stud1.insert({name:'abc',age:20})
```

Query Documents (Find)

In MongoDB, the db.collection.find() method retrieves documents from a collection. The db.collection.find() method returns a *cursor* to the retrieved documents. The db.collection.findOne() method also performs a read operation to return a single document. Internally, the db.collection.findOne() method is the db.collection.find() method with a limit of

1. **Select All Documents in a Collection**

An empty query document ({}) selects all documents in the collection:

```
db.inventory.find( {} )
```

Not specifying a query document to the find() is equivalent to specifying an empty query document. Therefore thefollowing operation is equivalent to the previous operation:

```
db.inventory.find()
```

Specify Equality Condition

To specify equality condition, use the query document { <field>: <value> } to select all documents that contain the <field> with the specified <value>. The following example retrieves from the inventory collection all documents where the type field has the value snacks:

```
db.inventory.find( { type: "snacks" } )
```

Specify Conditions Using Query Operators

A query document can use the *query operators* to specify conditions in a MongoDB query.

The following example selects all documents in the inventory collection where the value of the type field is either 'food' or'snacks':

```
db.inventory.find( { type: { $in: [ 'food', 'snacks' ] } } )
```

Although you can express this query using the $or operator, use the $in operator rather than the $or operator whenperforming equality checks on the same field.

Comparison Operators and Logical Operators : For comparison of different BSON type values, see the specified BSON comparison order.

Comparison OperatorName Description
$eq Matches values that are equal to a specified value.

$gt Matches values that are greater than a specified value.

$gte Matches values that are greater than or equal to a specified value.
$in Matches any of the values specified in an array.

$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.
$nin Matches none of the values specified in an array.
Logical Operators Name Description
$and Joins query clauses with a logical AND returns all documents that match the conditions of both clauses

$not Inverts the effect of a query expression and returns documents that do not match the query expression.

$nor Joins query clauses with a logical NOR returns all documents that fail to match both clauses.
$or Joins query clauses with a logical OR returns all documents that match the conditions of either clause.

SAVE METHOD
The save() method has the following form:

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

<document> })The products collection contains the following document:
{ "_id" : 100, "item" : "water", "qty" : 30 }
The save() method performs an update with upsert:true since the document contains an

_id field:db.products.save({ _id : 100, item : "juice" })

MongoDB Logical Query Operator - $and & $not
Logical Operator - $and

The MongoDB $and operator performs a logical AND operation on an array of two or more expressions and retrieves the documents which satisfy all the expressions in the array. The $and operator uses short-circuit evaluation. If the first expression (e.g. <expression1>) evaluates to false, MongoDB will not evaluate the remaining expressions.
Syntax:
{ $and: [{ <exp1> }, { <exp2> } , ... , { <expN> }] }

Our database name is 'myinfo' and our collection name is 'student'. Here, is the collection bellow.Example of MongoDB Logical Operator - $and
If we want to select all documents from the collection "student" which satisfying the condition -

1. sex of student is Female and
2. class of the student is VI and
3. grd_point of the student is greater than equal to 31 the following mongodb command can be used :

>db.student.find({$and:[{"sex":"Male"},{"grd_point":{ $gte: 31 }},{"class":"VI"}]}).pretty();

Logical Operator - $not

The MongoDB $not operator performs a logical NOT operation on the given expression and fetches selected documentsthat do not match the expression and the document that do not contain the field as well, specified in the expression.

Syntax:

{ field: { $not: { <expression> } } }

Example of MongoDB Logical Operator - $not

If we want to select all documents from the collection "student" which satisfying the condition -age of the student is at least 12 the following mongodb command can be used :

>db.student.find({"age": { $not: {$lt : 12}}}).pretty();The following mongodb command can be used :

>db.student.find({"sex": { $not: /^M.*/}}).pretty();

Conclusion: We have implemented CRUD operations using Mongodb

Activity to be Submitted by Students

1. Collection “orderinfo“ which contains the documents given as below(Perform on Mongo Terminal)

{
cust_id:123 cust_name:”abc”, status:”A”, price:250
}

i.find the average price for each customers having status 'A'
ii.Display the status of the customers whose amount/price lie between 100 and 1000
iii.Display the customers information without “_id” .
iv.create a simple index on onderinfo collection and fire the queries.

2. Collection “movies“ which contains the documents given as below(Perform on Mongo Terminal)

{
name: “Movie1”, type: “action”, budget:1000000 producer:{

name: “producer1”, address:”PUNE”
}
}

i. Find the name of the movie having budget greater than 1,00,000.
ii. Find the name of producer who lives in Pune
iii. Update the type of movie “action” to “horror”
iv. Find all the documents produced by name “producer1” with their address

Assignment No .10

Title of Assignment: MongoDB – Aggregation and Indexing:
Design and Develop MongoDB Queries using aggregation and indexing with suitable exampleusing MongoDB.
Course Objective:
To acquire the skills to use a powerful, flexible, and scalable general-purpose databases to handle Big Data

Course Outcome:

Implement NoSQL queries using MongoDB

Software Required: - Mongodb

INDEXING

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. This scan is highly inefficient and requires the **mongod** to process a large volume of data. “Indexes are special data structures that store a small portion of the data set in an easy to traverse form.”
The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in index.

The ensureIndex() Method

To create an index you need to use ensureIndex() method of mongodb.

SYNTAX:

Basic syntax of **ensureIndex()** method is as follows()

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

In **ensureIndex()** method you can pass multiple fields, to create index on multiple fields.

Drop Index :

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 to return a single result. In SQL count(*) and with group by is an equivalent of MongoDB aggregation.

The aggregate() Method

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

Syntax

Basic syntax of **aggregate()** method is as follows −

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

Example

In the collection you have the following data −

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by_user: 'tutorials point',
  url: 'http://www.tutorialspoint.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100
},
{
  _id: ObjectId(7df78ad8902d)
  title: 'NoSQL Overview',
  description: 'No sql database is very fast',
  by_user: 'tutorials point',
  url: 'http://www.tutorialspoint.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 10
},
{
  _id: ObjectId(7df78ad8902e)
  title: 'Neo4j Overview',
  description: 'Neo4j is no sql database',
  by_user: 'Neo4j',
  url: 'http://www.neo4j.com',
  tags: ['neo4j', 'database', 'NoSQL'],
  likes: 750
},
```

Now from the above collection, if you want to display a list stating how many tutorials are written by each user, then you will use the following **aggregate()** method −

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

Sql equivalent query for the above use case will be **select by_user, count(*) from mycol group by by_user**.

In the above example, we have grouped documents by field **by_user** and on each occurrence of by user previous value of sum is incremented. Following is a list of available aggregation expressions.

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

$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

Pipeline Concept

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

Following are the possible stages in aggregation framework −

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

Example

```
db.posts.aggregate([

 // Stage 1: Only find documents that have more than 1 like

 {

  $match: { likes: { $gt: 1 } }

 },

 // Stage 2: Group documents by category and sum each categories likes

 {
```

```
  $group: { _id: "$category", totalLikes: { $sum: "$likes" } }
 }
])
```

Output:-

```
[ { _id: 'News', totalLikes: 3 }, { _id: 'Event', totalLikes: 8 } ]
Atlas atlas-8iy36m-shard-0 [primary] blog>
```

Conclusion: We have implemented aggregation and indexing using Mongodb

Activity to be Submitted by Students

Perform aggregation and Indexing using mongodb on below database

1. Create a database department
2. Create a collection as teacher with fields as name , department ,experience and salary
3. Display the department wise average salary.
4. Display the no. Of employees working in each department.
5. Display the department wise minimum salary.
6. Apply index and drop index

Assignment No .11

Title of Assignment: MongoDB – Map-reduces operations:
Implement Map reduces operation with suitable example using MongoDB.
Course Objective:
To acquire the skills to use a powerful, flexible, and scalable general-purpose databases to handle Big Data

Course Outcome:

Implement NoSQL queries using MongoDB

Software Required: - Mongodb

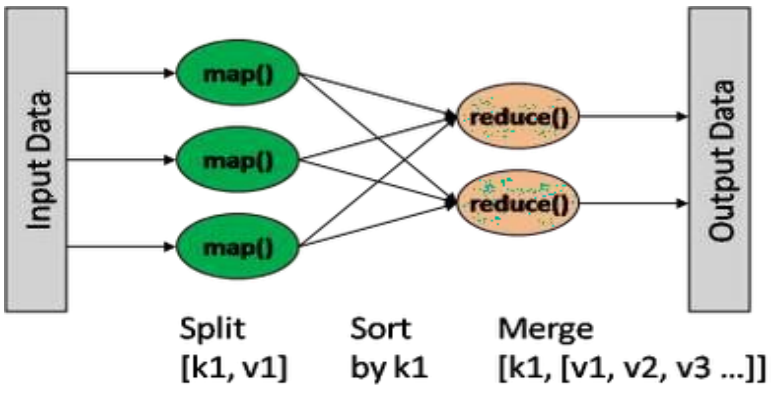
Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results. MongoDB uses **mapReduce** command for map-reduce operations. MapReduce is generally used for processing large data sets. In simple terms, the mapReduce command takes 2 primary inputs, the mapper function and the reducer function .

Working of Mapper and Reducer Function :

MapReduce is a two-step approach to data processing. First you map, and then you reduce. The mapping step transforms the inputted documents and emits a key=>value pair (the key and/or value can be complex). Then, key/value pairs are grouped by key, such that values for the same key end up in an array. The reduce gets a key and the array of values emitted for that key, and produces the final result. The map and reduce functions are written in JavaScript. 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.

MapReduce Command:

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



The map-reduce function first queries the collection, then maps the result documents to emit key-value pairs which is then reduced based on the keys that have multiple values. MapReduce Command:
syntax of the basic mapReduce command:

db.collection.mapReduce(function() {emit(key,value);}, //map function function(key,values) {return reduceFunction}, //reduce function

{out: collection, query: document, sort: document, limit: number}) The map-reduce function first queries the collection, then maps the result documents to emit key-value pairs which is then reduced based on the keys that have multiple values

In the above syntax:

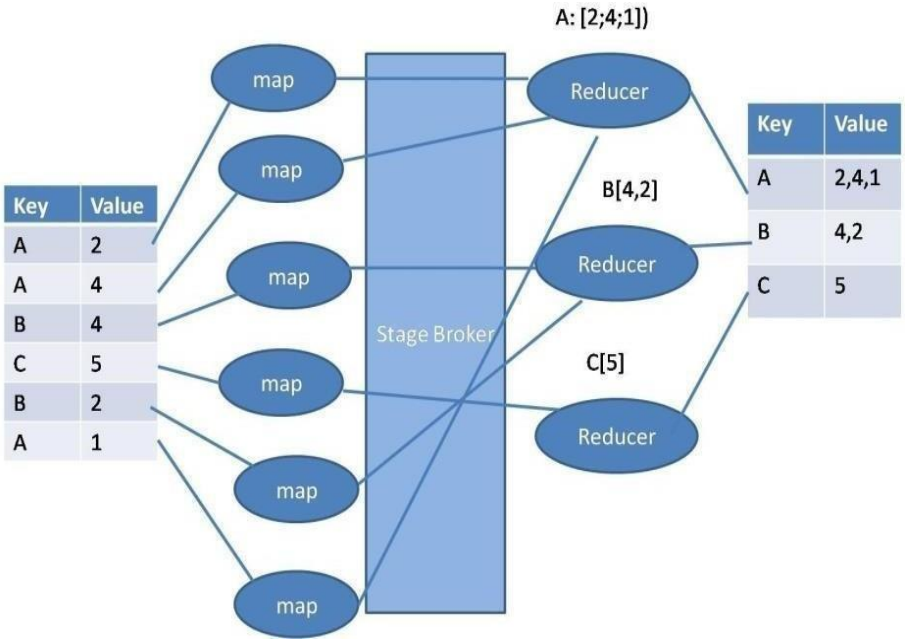
- **map** is a javascript function that maps a value with a key and emits a key-value pair
- **reduce** is a javascript function that reduces or groups all the documents having the same key
- **out** specifies the location of the map-reduce query result
- **query** specifies the optional selection criteria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

Map Reduce Example

The below example is to retrieve the sum of total values related to particular key.

1. Insert data in *mapCollection.*

```
db.mapc.insert({key:"a", value:2})
db.mapc.insert({key:"a", value:4})
```



Conclusion: We have implemented Map reduce using Mongodb Successfully

Activity to be Submitted by Students

Collection “city “ which contains the documents given as below(Perform on Mongo Terminal)

{

city:”pune”,

type:”urban”,

state:”MH”, population:”5600000”

}

-using mapreduce, find statewise population

-using mapreduce, find citywise population

-using mapreduce, find typewise population.

Assignment No .12

Title of Assignment: Database Connectivity:
Write a program to implement MySQL/Oracle database connectivity with any frontend language to implement Database navigation operations (add, delete, edit etc.)

Course Objective:

Implement PL/SQL Code block for given requirements

Course Outcome:

C306.6

Design and develop application considering actual requirements and using database concepts

Software required: - Mongodb/eclipse/ wampserver/php

Mongodb Connectivity Mongo connection

Connect to MongoDB server. For MongoDB version >= 2.10.0, uses MongoClient.
// Old version, uses Mongo
Mongo mongo = new Mongo("localhost", 27017);
// Since 2.10.0, uses MongoClient

MongoClient mongo = new MongoClient("localhost" , 27017);If MongoDB in secure mode, authentication is required.
MongoClient mongoClient = new MongoClient();DB db = mongoClient.getDB("database name");
boolean auth = db.authenticate("username", "password".toCharArray());

Mongo Database

Get database. If the database doesn't exist, MongoDB will create it for you.DB db = mongo.getDB("database name");
Display all collections from selected database.DB db = mongo.getDB("testdb");

Save example

Save a document (data) into a collection (table) named "user".DBCollection table = db.getCollection("user"); BasicDBObject document = new

BasicDBObject();document.put("name", "mkyong"); document.put("age", 30); document.put("createdDate", new Date()); table.insert(document);

Update example

Update a document where "name=mkyong". DBCollection table = db.getCollection("user"); BasicDBObject query = new BasicDBObject(); query.put("name", "mkyong");
BasicDBObject newDocument = new BasicDBObject(); newDocument.put("name", "mkyong-updated"); BasicDBObject updateObj = new BasicDBObject(); updateObj.put("$set", newDocument); table.update(query, updateObj);

Find example

Find document where “name=mkyong”, and display it with DBCursorDBCollection table = db.getCollection("user"); BasicDBObject searchQuery = new BasicDBObject(); searchQuery.put("name", "mkyong");

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

Delete example

Find document where “name=mkyong”, and delete it.DBCollection table = db.getCollection("user"); BasicDBObject searchQuery = new BasicDBObject(); searchQuery.put("name", "mkyong"); table.remove(searchQuery);

Conclusion: We have implemented database connectivity

Activity to be Submitted by Students

1. CRUD operation using Mongodb and JDBC connectivity