ZEALEDUCATIONSOCIETY's

ZEALCOLLEGEOFENGINEERINGANDRESEARCH, NARHE, PUNE

DEPARTMENT OF COMPUTER ENGINEERING

SEMESTER-II

[A.Y.:2022-2023]

Database Management Systems (310246)LABORATORY MANUAL

Department Vision and Mission

INSTITUTE VISION	To impart value added technological education through pursuit of academic excellence, research and entrepreneurial attitude.
INSTITUTE MISSION	**M1:** To achieve academic excellence through innovative teaching and learning process. **M2:** To imbibe the research culture for addressing industry and societal needs. **M3:** To provide conducive environment for building the entrepreneurial skills. **M4:** To produce competent and socially responsible professionals with core human values.

DEPARTMENT VISION	To generate competent professionals to become part of the industry and research organizations at the national and international levels.
DEPARTMENT MISSION	**M1:** Providing a strong theoretical and practical background across the computer science discipline with an emphasis on software development.. **M2:** Imparting the skills necessary to continue education to grow as a professional. **M3:** Inculcating professional behavior, strong ethical values, innovative research capabilities and leadership abilities.

Department
Program Educational Objectives(PEOs)

PEO1: To Impart fundamentals in science, mathematics and engineering to cater the needs of society and Industries.
PEO2: Encourage graduates to involve in research, higher studies, and/or to become entrepreneurs.
PEO3: To Work effectively as individuals and as team members in a multi-disciplinary environment with high ethical values for the benefit of society.

Savitribai Phule Pune University Third Year of Computer Engineering(2019Course)310246: Database Management Systems Laboratory		
Teaching Scheme: PR:04Hours/Week	**Credit** 02	**Examination Scheme:** TW:25Marks PR:25Marks

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, student 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

List of Assignments

Sr. No.	Title
	Group A: SQL and PL/SQL
01	**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.
02	**SQL Queries:** a. Design and Develop SQLDDL 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.
03	**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
04	**Unnamed PL/SQL code block: Use of Control structure and Exception handling is mandatory.** 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 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.

05	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.
06	**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.
07	**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**.**
08	**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/SQLblock for all types of Triggers in line with above statement**.**
09	**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
01	**MongoDB Queries:** Design and Develop MongoDB Queries using CRUD operations. (Use CRUD operations, SAVE method, logical operators etc.).
02	**MongoDB – Aggregation and Indexing:** Design and Develop MongoDB Queries using aggregation and indexing with suitable example using MongoDB.
03	**MongoDB – Map-reduces operations:** Implement Map reduces operation with suitable example using MongoDB.
04	**Database Connectivity:** Write a program to implement Mongo DB database connectivity with any front-end language to implement Database navigation operations (add, delete, edit etc.)
	Group C: Mini Project
	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 through out 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

Group A

Assignment No 1

Problem Statement:

ER Modelling 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.

Objective: Able to develop Database programming skills

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

Theory:

- 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)

Entity Types, Entity Sets, Attributes and Keys, Relationship

- Entity:

Real-world object or thing with an independent existence and which is distinguishable from other objects.

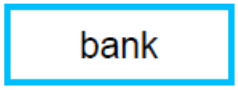
Examples are a person, car, customer, product, gene, book etc.

Types of entities:

1. Strong Entity

   The strong entity has a primary key. Weak entities are dependent on strong entity. Its existence is not dependent on any other entity. It is represented by rectangle.

   E.g.



2. Weak Entity

The weak entity in DBMS do not have a primary key and are dependent on the other strong entity. It is represented by double rectangle.

E.g.

- Entity Set:

Collection of entities of a particular entity type at any point in time; entity set is typically referred to using the same name as entity type.

- Attributes:

An entity is represented by a set of attributes (its descriptive properties), 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.

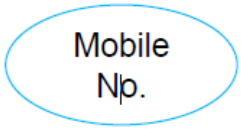
With each attribute a domain is associated, i.e., a set of permitted values for an attribute. Possible domains are integer, string, dates, etc.

Types of Attributes:

1. Simple attribute

Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.

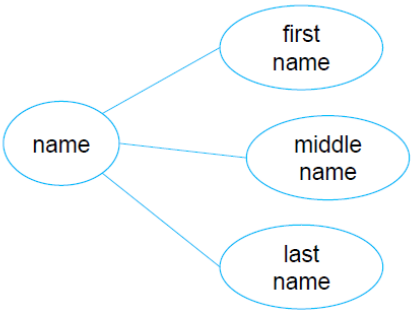
E.g.,



2. Composite

Composite attributes are made of more than one simple attribute.

E.g., a student's complete name may have first_name, middle_name and last_name.



3. Single Valued
   An attribute which has only one value.E.g., Name, gender, address, roll_no

4. Multi-valued

   An attribute which has many values. It represented by double oval.

   E.g., Mobile No, Email Id

5. Derived
   An attribute which can derive its value from another attribute.It represented by dashed oval.

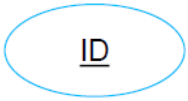
   E.g.,

6. Key Attribute

   An attribute which has all unique values for all records. It represents a primary key.

   The key attribute is represented by an oval with the text underlined.

   E.g., Roll No, AdhaarNo, ID, Pan No



• Relationship:

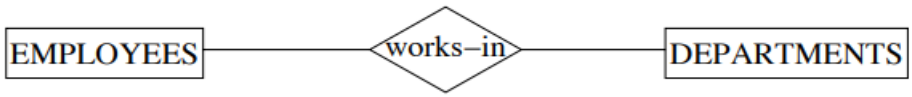
It is association among two or more entities, e.g., "customer 'Smith' orders product 'PC42'"

• Constraint:

Mapping Constraint-

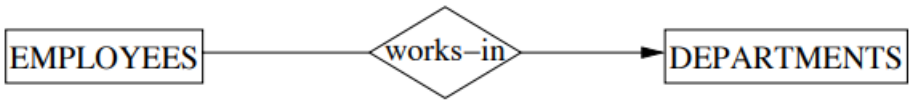
From one entity set how many entities are associated with other entity sets through relationship set.

1. Many-To-Many (default)



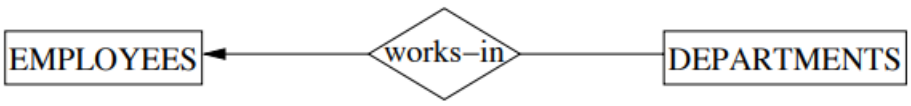
Meaning: An employee can work in many departments (≥ 0),and a department can have several employees

2. Many-To-One



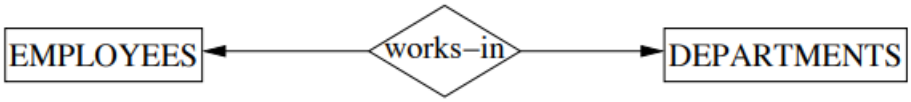
Meaning: An employee can work in at most one department (≤ 1), and a department can have several employees.

3. One-To-Many



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

4. One-To-One

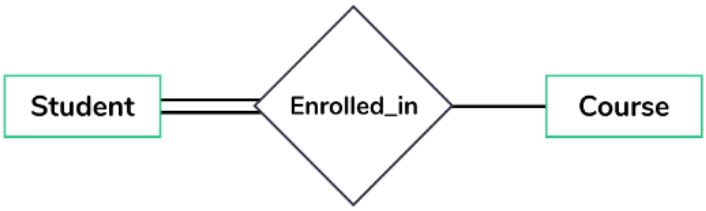


Meaning: An employee can work in at most one department, and a department can have at most one employee.

Participation constraint:

It specifies the presence of an entity when it is related to another entity in a relationship type. It is also called the minimum cardinality constraint.
This constraint specifies the number of instances of an entity that are participating in the relationship type.
There are two types of Participation constraint:

1. Total participation
2. Partial participation



1. Total participation constraint

It specifies that each entity present in the entity set must mandatorily participate in at least one relationship instance of that relationship set, for this reason, it is also called as mandatory participation. It is represented using a double line between the entity set and relationship set.

Example of total participation constraint

- It specifies that each student must be enrolled in at least one course where the “student” is the entity set and relationship “enrolled in” signifies total participation
- It means that every student must have enrolled at least in one course

2. Partial participation constraint

It specifies that each entity in the entity set may or may not participate in the relationship instance of the relationship set, is also called as optional participation. It is represented using a single line between the entity set and relationship set in the ER diagram.

Example of partial participation

- A single line between the entities i.e., courses and enrolled in a relationship signifies the partial participation, which means there might be some courses where enrollments are not made i.e., enrollments are optional in that case.

Example of an Entity-Relationship Diagram

University database

The university database stores details about university students, courses, the semester a student took a particular course (and his mark and grade if he completed it), and what degree program each student is enrolled in.

Consider the following requirements list:

- The university offers one or more programs.
- A program is made up of one or more courses.
- A student must enroll in a program.
- A student takes the courses that are part of her program.
- A program has a name, a program identifier, the total credit points required to graduate, and the year it commenced.
- A course has a name, a course identifier, a credit point value, and the year it commenced.
- Students have one or more given names, a surname, a student identifier, a date of birth, and the year they first enrolled. We can treat all given names as a single object—for example, “John Paul.”
- When a student takes a course, the year and semester he attempted it are recorded. When he finishes the course, a grade (such as A or B) and a mark (such as 60 percent) are recorded.
- Each course in a program is sequenced into a year (for example, year 1) and a semester (for example, semester 1).

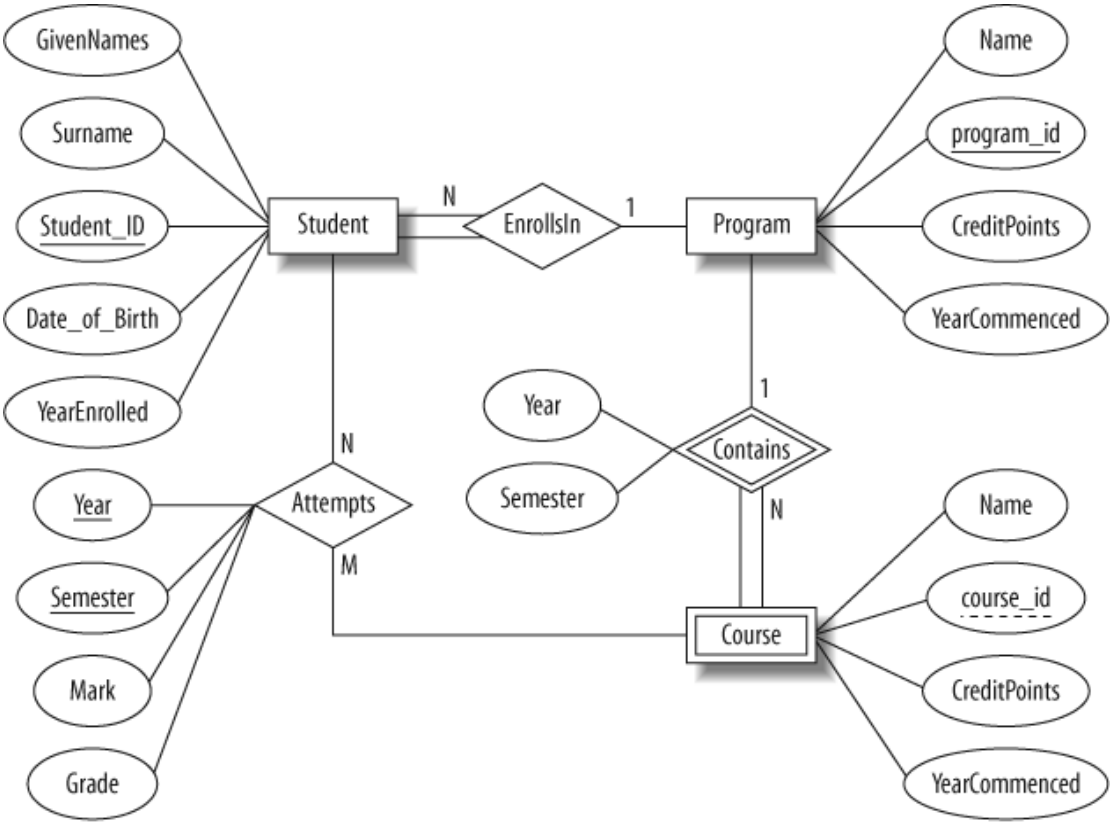


Figure 1. The ER diagram of the university database

- Student is a strong entity, with an identifier, student_id, created to be the primary key used to distinguish between students.
- Program is a strong entity, with the identifier program_id as the primary key used to distinguish between programs.
- Each student must be enrolled in a program, so the Student entity participates totally in the many-to-one EnrollsIn relationship with Program. A program can exist without having any enrolled students, so it participates partially in this relationship.
- A Course has meaning only in the context of a Program, so it's a weak entity, with course_id as a weak key. This means that a Course is uniquely identified using its course_id and the program_id of its owning program.
- As a weak entity, Course participates totally in the many-to-one identifying relationship with its owning Program. This relationship has Year and Semester attributes that identify its sequence position.

- Student and Course are related through the many-to-many Attempts relationships; a course can exist without a student, and a student can be enrolled without attempting any courses, so the participation is not total.

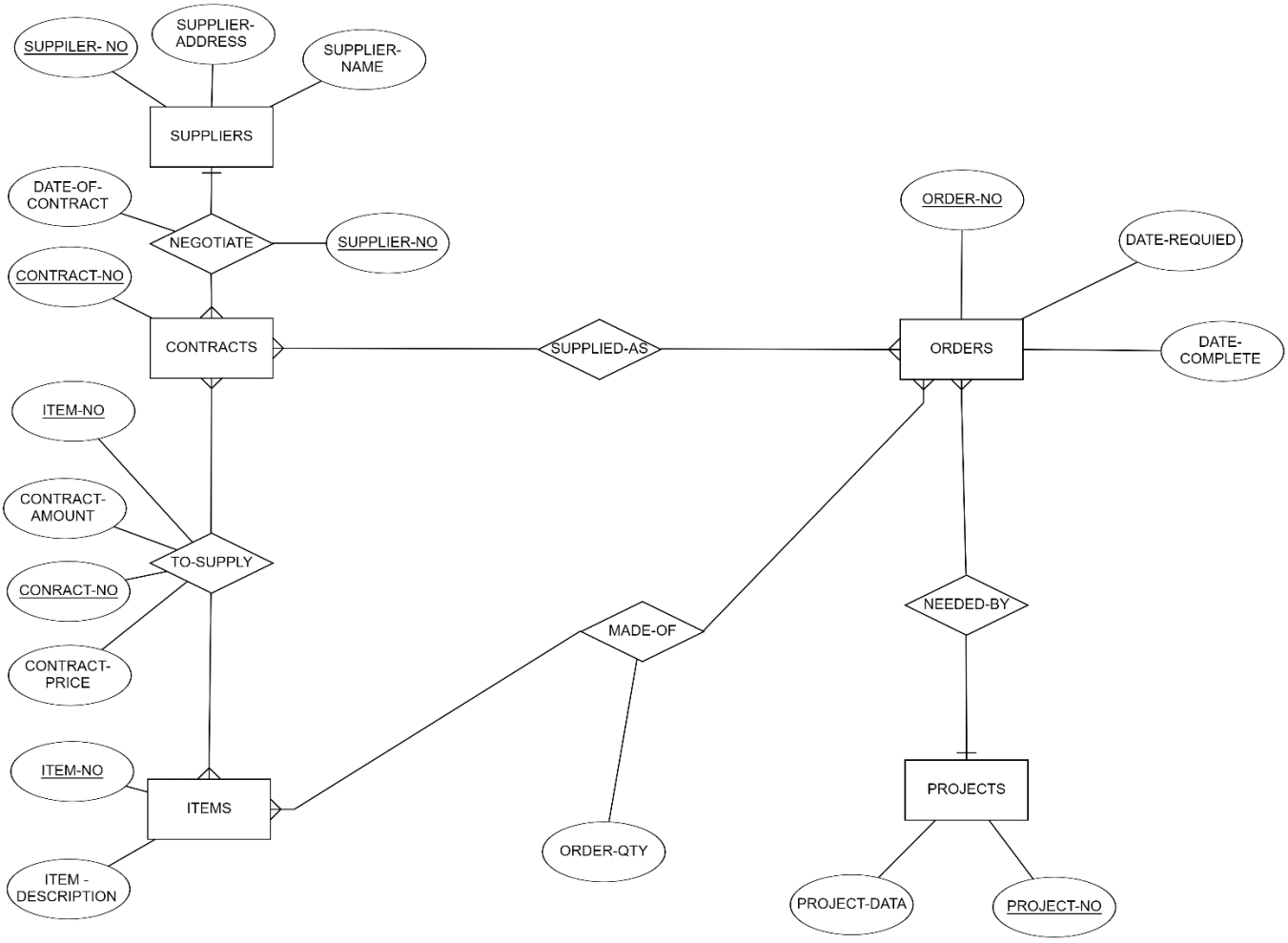
- When a student attempts a course, there are attributes to capture the Year and Semester, and the Mark and Grade.

ERD plus

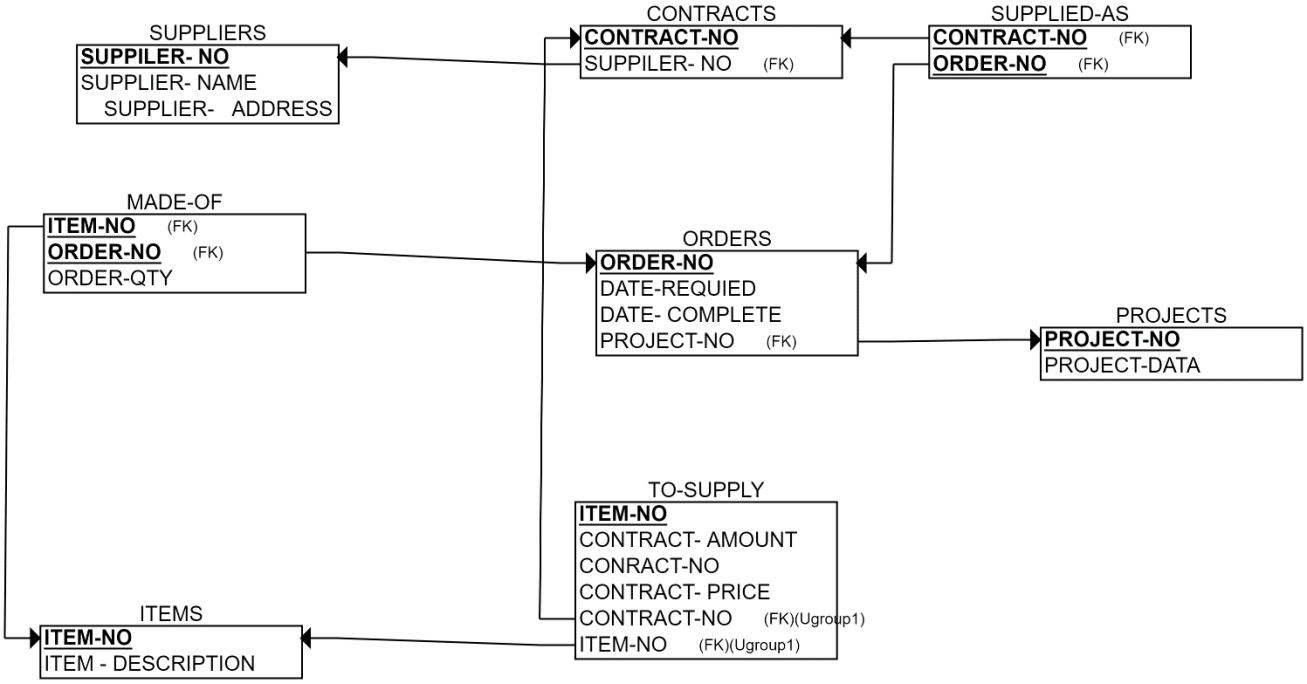
Web-based database modeling tool that offers relational schemas, SQL generation, data export, ER diagram conversion, and more.Designed for business analysts, a database modeling tool that helps with drawing entity relationship diagram components, converting ER Diagrams to relational schemas, exporting SQL, and more.

The notation supports drawing regular and weak entities, various types of attributes (regular, unique, multi-valued, derived, composite, and optional), and all possible cardinality constraints of relationships (mandatory-many, optional-many, mandatory-one and optional-one).

ER diagram for Library Management System:



Relational schema for Library Management System:



Normalization

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

- Eliminating redundant(useless) data.
- Ensuring data dependencies make sense i.e., data is logically stored.

Normalization rules are divided into the following normal forms:

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. BCNF

1) First Normal Form (1NF):

If a relation contains a composite or multi-valued attribute, it violates the first normal form, or the relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is singled valued attribute.

There are only Single Valued Attributes.

- Attribute Domain does not change.
- There is a unique name for every Attribute/Column.
- The order in which data is stored does not matter.

Consider the example given below.

ID Name Courses

---

1 A c1, c2

2 E c3

3 M C2, c3

In the above table, Course is a multi-valued attribute so it is not in 1NF.

Below Table is in 1NF as there is no multi-valued attribute:

ID Name Course

---

1 A c1

1 A c2

2 E c3

3 M c2

3 M c3

2. Second Normal Form(2NF)

- The second step in Normalization is 2NF.
- A table is in 2NF, only if a relation is in 1NF and meet all the rules, and every non-key attribute is fully dependent on primary key.
- The Second Normal Form eliminates partial dependencies on primary keys.

Consider the example given below.

Table Name: <StudentProject>

StudentID	ProjectID	StudentName	ProjectName
S89	P09	Olivia	Geo Location
S76	P07	Jacob	Cluster Exploration
S56	P03	Ava	IoT Devices
S92	P05	Alexandra	Cloud Deployment

In the above table, we have partial dependency; let us see how −

- The prime key attributes are StudentID and ProjectID.
- As stated, the non-prime attributes i.e. StudentName and ProjectName should be functionally dependent on part of a candidate key, to be Partial Dependent.
- The StudentName can be determined by StudentID, which makes the relation Partial Dependent.
- The ProjectName can be determined by ProjectID, which makes the relation Partial Dependent.
- Therefore, the <StudentProject> relation violates the 2NF in Normalization and is considered a bad database design.

Example (Table converted to 2NF)

To remove Partial Dependency and violation on 2NF, decompose the above tables –

<StudentInfo>

StudentID	ProjectID	StudentName
S89	P09	Olivia
S76	P07	Jacob
S56	P03	Ava
S92	P05	Alexandra

<ProjectInfo>

ProjectID	ProjectName
P09	Geo Location
P07	Cluster Exploration
P03	IoT Devices
P05	Cloud Deployment

3. Second Normal Form(3NF)

- A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.
- A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency X –> Y:
- X is a super key.
- Y is a prime attribute (each element of Y is part of some candidate key).
- In other words,

A relation that is in First and Second Normal Form and in which no non-primary-key attribute is transitively dependent on the primary key, then it is in Third Normal Form (3NF).

Note – If A->B and B->C are two FDs then A->C is called transitive dependency.

Example: Let's say a company wants to store the complete address of each employee, they create a table named Employee_Details that looks like this:

Emp_Id	Emp_Name	Emp_Zip	Emp_State	Emp_City	Emp_District
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {Emp_Id}, {Emp_Id, Emp_Name}, {Emp_Id, Emp_Name, Emp_Zip}…so on

Candidate Keys: {Emp_Id}

Non-prime attributes: all attributes except Emp_Id are non-prime as they are not part of any candidate keys.

Here, Emp_State, Emp_City & Emp_District dependent on Emp_Zip. Further Emp_zip is dependent on Emp_Id that makes non-prime attributes (Emp_State, Emp_City & Emp_District) transitively dependent on super key (Emp_Id). This violates the rule of 3NF.

To make this table complies with 3NF we have to disintegrate the table into two tables to remove the transitive dependency:

Employee Table:

Emp_Id	Emp_Name	Emp_Zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

Employee_Zip table:

Emp_Zip	Emp_State	Emp_City	Emp_District
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City

282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

4. Boyce Codd Normal Form (BCNF)

It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.

Example: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

EMPLOYEE table:

Emp_Id	Emp_Nationality	Emp_Dept	Dept_Type	Dept_No_Of_Emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

In the above table Functional dependencies are as follows:

EMP_ID → EMP_COUNTRY

EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

EMP_ID → EMP_COUNTRY

EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID

For the second table: EMP_DEPT

For the third table: {EMP_ID, EMP_DEPT}

Conclusion-

In this assignment, we have studied and demonstrated various ER diagrams

Viva Questions:

- What is database?
- What is ER modeling?
- Explain the attribute and its types?
- What is entity?
- Draw ER diagram for student database system?

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	

Group A

Assignment No: 2

Title of the 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 DML statements.

Objective of the Assignment: To understand and demonstrate DDL statements on various SQL objects..

Outcome:

1) Students will be able to learn and understand various DDL queries like create, drop, truncate.

2) Students will be able to demonstrate creating and dropping SQL objects like table, view, sequence, index etc

Theory:

DDL-

Data Definition Language (DDL) statements are used to define the database structure or schema. Data Definition Language understanding with database schemas and describes how the data should consist in the database, therefore language statements like CREATE TABLE or ALTER TABLE belongs to the DDL. DDL is about "metadata".

DDL includes commands such as CREATE, ALTER and DROP statements.DDL is used to CREATE, ALTER OR DROP the database objects (Table, Views, Users).

Data Definition Language (DDL) are used different statements :

1. CREATE - to create objects in the database
2. ALTER - alters the structure of the database
3. DROP - delete objects from the database
4. TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
5. RENAME - rename an object

1. CREATE

The CREATE TABLE statement is used to create a new table in a database.

Syntax

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

Example

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

2. ALTER

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

1. **ALTER TABLE - ADD Column**
   To add a column in a table
   Syntax:
   ALTER TABLE table_name ADD column_name datatype;
   Example
   ALTER TABLE Customers ADD Email varchar(255);

2. **ALTER TABLE - DROP COLUMN**
   To delete a column in a table,(notice that some database systems don't allow deleting a column):
   syntax
   ALTER TABLE table_name DROP COLUMN column_name;
   Example
   ALTER TABLE Customers DROP COLUMN Email;

3. **ALTER TABLE - MODIFY COLUMN**
   To change the data type of a column in a table
   syntax:
   ALTER TABLE table_name MODIFY COLUMN column_name datatype;
   Example
   ALTER TABLE Persons ADD DateOfBirth date;

3. Drop

The DROP TABLE statement is used to drop an existing table in a database.
Syntax
DROP TABLE table_name;
Example
DROP TABLE Shippers;

4. TRUNCATE

The TRUNCATE statement is used to delete the data inside a table, but not the table itself.
Syntax
TRUNCATE TABLE table_name;
Example
TRUNCATE TABLE Shippers;

5. RENAME
RENAME statement you can rename a table.

Syntax
RENAME TABLE (tbl_name) TO (new_tbl_name);
Example
RENAME table student to info;

DML

DML is short name of Data Manipulation Language which deals with data manipulation and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE, etc., and it is used to store, modify, retrieve, delete and update data in a database.

1. SELECT - retrieve data from a database
2. INSERT - insert data into a table
3. UPDATE - updates existing data within a table
4. DELETE - Delete all records from a database table

1. SELECT
The SELECT statement is used to select data from a database.
Syntax
SELECT column1, column2, ...FROM table_name;
Example
SELECT CustomerName, City, Country FROM Customers;
Syntax
SELECT * FROM table_name;
Example
SELECT * FROM Customers;

2. INSERT
The INSERT INTO statement is used to insert new records in a table. It is possible to write the INSERT INTO statement in two ways:
1. Specify both the column names and the values to be inserted:
Syntax
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
Example
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

Syntax
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
Example
INSERT INTO Customers (CustomerName, City, Country)VALUES ('Cardinal', 'Stavanger', 'Norway');

3. **UPDATE**
The UPDATE statement is used to modify the existing records in a table.
Syntax
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
Example
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City = 'Frankfurt'
WHERE CustomerID = 1;

4. **DELETE**
The DELETE statement is used to delete existing records in a table.
Syntax
DELETE FROM table_name WHERE condition;
Example
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

Views

- a view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

1. **CREATE VIEW**

Syntax
CREATE VIEW view_name AS SELECT column1, column2, ... FROM table_name WHERE condition;
Examples
CREATE VIEW [Brazil Customers] AS SELECT CustomerName, ContactName FROM Customers
WHERE Country = 'Brazil';
SELECT * FROM [Brazil Customers];

2. **CREATE OR REPLACE VIEW**
Syntax
CREATE OR REPLACE VIEW view_name AS SELECT column1, column2, ... FROM table_name
WHERE condition;
Examples
CREATE OR REPLACE VIEW [Brazil Customers] AS SELECT CustomerName, ContactName, City
FROM Customers WHERE Country = 'Brazil';

3. **Dropping a View**
   A view is deleted with the DROP VIEW statement.
   Syntax
   DROP VIEW view_name;
   Example
   DROP VIEW Brazil-Customers

Sequence

- Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.
- Often this is the primary key field that we would like to be created automatically every time a new record is inserted.
  Example
  CREATE TABLE Persons (
  Personid int NOT NULL AUTO_INCREMENT,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,
  PRIMARY KEY (Personid)
  );
- By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

- To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:
  Example
  ALTER TABLE Persons AUTO_INCREMENT=100;
  INSERT INTO Persons (FirstName,LastName)VALUES ('Lars','Monsen');

SYNONYM

A SYNONYM provides another name for database object, referred to as original object, that may exist on a local or another server.

Syntax

Call sys.create_synonym_db(‘old database name’,’new database name’);

Example

Call sys.create_synonym_db(‘student,’information);

Constraints

- Constraints are used to specify rules for data in a table.
- Constraints are used to limit the type of data that can go into a table.
- This ensures the accuracy and reliability of the data in the table.
- If there is any violation between the constraint and the data action, the action is aborted.

Types of constraints

1) NOT NULL Constraint
2) Unique Constraint
3) Primary key
4) Foreign key
5) Check constraint
6) Default constraint

1) NOT NULL Constraint

- By default, a column can hold NULL values.
- The NOT NULL constraint enforces a column to NOT accept NULL values.
- This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

NOT NULL on CREATE TABLE

Example

CREATE TABLE Persons (ID int NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255) NOT NULL, Age int);

NOT NULL on ALTER TABLE

Example

ALTER TABLE Persons
MODIFY COLUMN Age int NOT NULL;

2) UNIQUE Constraint

- The UNIQUE constraint ensures that all values in a column are different.
- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint.
- However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

UNIQUE Constraint on CREATE TABLE(One column)

Example

```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   UNIQUE (ID)
);
```

UNIQUE Constraint on CREATE TABLE(Multiple column)

Example

```
CREATE TABLE Persons (
   ID int NOT NULL,
```

```
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

UNIQUE Constraint on ALTER TABLE(One column)
Example
ALTER TABLE Persons ADD UNIQUE (ID);

UNIQUE Constraint on ALTER TABLE(Multiple column)
Example
ALTER TABLE Persons ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);

DROP a UNIQUE Constraint
Example
ALTER TABLE Persons DROP INDEX UC_Person;

3) PRIMARY KEY Constraint

- The PRIMARY KEY constraint uniquely identifies each record in a table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

PRIMARY KEY on CREATE TABLE
Example

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
PRIMARY KEY (ID)
);
```

PRIMARY KEY on ALTER TABLE
Example
ALTER TABLE Persons ADD PRIMARY KEY (ID);

DROP a PRIMARY KEY Constraint
Example
ALTER TABLE Persons DROP PRIMARY KEY;

4) Foreign key constraint

- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.
- The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

FOREIGN KEY on CREATE TABLE

Example

```
CREATE TABLE Orders (
 OrderID int NOT NULL,
OrderNumber int NOT NULL,
 PersonID int,
 PRIMARY KEY (OrderID),
 FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

FOREIGN KEY on ALTER TABLE

Example

```
ALTER TABLE Orders ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

DROP a FOREIGN KEY Constraint

Example

```
ALTER TABLE Orders DROP FOREIGN KEY FK_PersonOrder;
```

5) Check Constraint

- The CHECK constraint is used to limit the value range that can be placed in a column.
- If you define a CHECK constraint on a column it will allow only certain values for this column.
- If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

CHECK on CREATE TABLE (One column)

Example

```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   CHECK (Age>=18)
);
```

CHECK on CREATE TABLE (Multiple column)

Example

```
 CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
 );
```

CHECK on ALTER TABLE (One column)

```
ALTER TABLE Persons ADD CHECK (Age>=18);
```

CHECK on ALTER TABLE (Multiple columns)

ALTER TABLE Persons ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');

DROP a CHECK Constraint
ALTER TABLE Persons DROP CONSTRAINT CHK_PersonAge;

6) Default constraint

- The DEFAULT constraint is used to set a default value for a column.
- The default value will be added to all new records, if no other value is specified.

DEFAULT on CREATE TABLE
Example

```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   City varchar(255) DEFAULT 'Sandnes'
);
```

DEFAULT on ALTER TABLE
Example
ALTER TABLE Persons ALTER City SET DEFAULT 'Sandnes';

DROP a DEFAULT Constraint
Example
ALTER TABLE Persons ALTER City DROP DEFAULT;

Conclusion: In this assignment, we have studied and demonstrated various DDL,DML statements in SQL.

Viva Questions:

- What is DDL & DML with examples?
- What is difference between drop and truncate?
- What mean by constraints and explain its types.
- What difference between unique and primary key constraints?
- What is mean by view & how to create a view?

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	

Group A

Assignment No 3

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

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

b. design the queries which demonstrate the use of concepts like all types of Join, Sub-Query

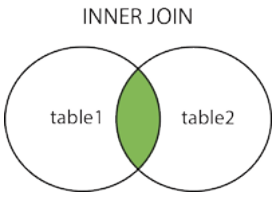
Objective of the Assignment: To understand and demonstrate DDL statements and joins and its types on various SQL object

Outcome: Students will be able to learn and understand various joins queries and subqueries.

Theory:
A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
Different Types of SQL JOINs

1) (INNER) JOIN: Returns records that have matching values in both tables
2) LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
3) RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
4) FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table

1) **INNER JOIN**
The INNER JOIN keyword selects records that have matching values in both tables.



Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

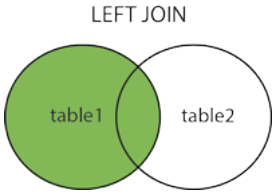
Example

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

2) **LEFT (OUTER) JOIN:** Returns all records from the left table, and the matched records from the right table

Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

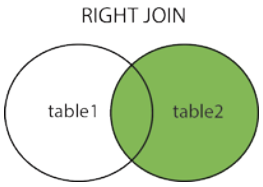


Example

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

3) **RIGHT JOIN**

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.



Syntax

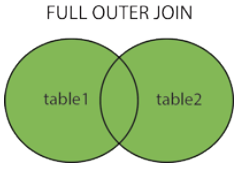
```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Example

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

4) **Full Join**

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.



Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Example

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

Subquery

- A subquery is a SQL query nested inside a larger query.
- A subquery may occur in :
  - A SELECT clause
  - A FROM clause
  - A WHERE clause
- The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.

Subqueries with the SELECT Statement

Syntax :

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
(SELECT column_name [, column_name ]
FROM table1 [, table2 ]
[WHERE])
```

Example

```
SELECT *  FROM CUSTOMERS WHERE ID IN (SELECT ID
FROM CUSTOMERS
 WHERE SALARY > 4500) ;
```

Conclusion: In this assignment, we have studied and demonstrated various types of joins and its types and subquery.

Viva Questions:
What is joins and its types?
What is subquery?
What is full join and write syntax?
What is left join and write syntax?
Write any one subquery with example?

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	

Group A

Assignment No 4

Title of the Assignment: 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.

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

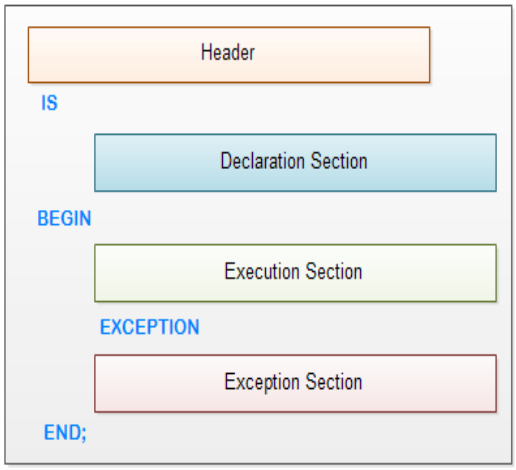
Outcome: Students will be able to learn and understand various control structure and exception handling.

Theory:

Introducing PL/SQL block structure and anonymous block PL/SQL program units organize the code into blocks. A block without a name is known as an anonymous block. The anonymous block is the simplest unit in PL/SQL. It is called anonymous block because it is not saved in the Oracle database.An anonymous block is an only one-time use and useful in certain situations such as creating test units. The following illustrates anonymous block syntax:

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

The anonymous block has three basic sections that are the declaration, execution, and exception handling. Only the execution section is mandatory and the others are optional.



- The declaration section allows you to define data types, structures, and variables. You often declare variables in the declaration section by giving those names, data types, and initial values. The execution section is required in a block structure and it must have at least one statement.
- The execution section is the place where you put the execution code or business logic code. You can use both procedural and SQL statements inside the execution section. The exception handling section is starting with the EXCEPTION keyword.
- The exception section is the place that you put the code to handle exceptions. You can either catch or handle exceptions in the exception section.

IF- Statement:

The IF statement executes a sequence of statements depending on the value of a condition.

Syntax:

IFcondition THEN statement1;

ElSEIF condition2 THEN statement2;

ENDIF;

Using the LOOP Statement:

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

LOOP

sequence_of_statements

END LOOP;

With each iteration of the loop, the sequence of statements is executed, then control resumes at the top of the loop. You use an EXIT statement to stop looping and prevent an infinite loop. You can place one or more EXIT statements anywhere inside a loop, but not outside a loop. There are two forms of EXIT statements: EXIT and EXIT-WHEN.

Exception:

PL/SQL supports programmers to catch such conditions using EXCEPTION block in the program and an appropriate action is taken against the error condition. There are two types of exceptions

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

Syntax for Exception Handling:

```
DECLARE
Declaration section
BEGIN
Exception section
EXCEPTION
WHEN ex_name1 THEN
-Error handling statements
WHEN ex_name2 THEN
-Error handling statements
WHEN Others THEN
-Error handling statements
END;
```

Raising Exception:

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

Syntax:

```
DECLARE
Exception_name EXCEPTION;
BEGIN
 IF condition THEN
```

RAISE exception_name;

ENDIF;

EXCEPTION

WHEN exception_name THEN

STATEMENT

User-defined Exceptions:

Steps to be followed to use user-defined exceptions:

- They should be explicitly declared in the declaration section.
- They should be explicitly raised in the Execution Section.
- They should be handled by referencing the user-defined exception name in the exception section.

Program:

```
mysql> use library;
Database changed
mysql> create table borrower(Roll_no int(10) primary key, Name varchar(20) not null, Date_of_iss
ue date not null, Name_of_book varchar(20) not null, Status varchar(1) not null);
Query OK, 0 rows affected, 1 warning (0.03 sec)


mysql> desc borrower;
+---------------+-------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| Roll_no | int | NO | PRI | NULL | |
| Name | varchar(20) | NO | | NULL | |
| Date_of_issue | date | NO | | NULL | |
| Name_of_book | varchar(20) | NO | | NULL | |
| Status | varchar(1) | NO | | NULL | |

5 rows in set (0.00 sec)
mysql> create table fine(Roll_no int(10), Date date not null, Amount int(10), foreign key(Roll_no)
references borrower(Roll_no));
Query OK, 0 rows affected, 2 warnings (0.02 sec)

mysql> desc fine;
+---------+------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+---------+------+------+-----+---------+-------+
| Roll_no | int | YES | MUL | NULL | |
| Date | date | NO | | NULL | |
| Amount | int | YES | | NULL | |
+---------+------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> insert into borrower values(1, 'Amruta', '2022/07/18', 'MySQL', 'I');
Query OK, 1 row affected, 1 warning (0.01 sec)
mysql> insert into borrower values(2, 'Siddeshdesh', '2022/06/02', 'Computer Network', 'I');
```

```
Query OK, 1 row affected, 1 warning (0.02 sec)
mysql> insert into borrower values(3, 'Swarali', '2022/06/22', 'Operating System', 'I');
Query OK, 1 row affected, 1 warning (0.01 sec)
mysql> insert into borrower values(4, 'Bhavesh', '2022/07/17', 'Design of Complier', 'I');
Query OK, 1 row affected, 1 warning (0.02 sec)
mysql> insert into borrower values(5, 'Chaitali', '2022/08/15', 'Internet of Things', 'I');
Query OK, 1 row affected, 1 warning (0.01 sec)


mysql> insert into borrower values(6, 'Omkar', '2022/09/02', 'Mobile Computing', 'I');
Query OK, 1 row affected, 1 warning (0.01 sec)
mysql> select * from borrower;
+---------+-------------+---------------+--------------------+--------+
| Roll_no | Name | Date_of_issue | Name_of_book | Status |
+---------+-------------+---------------+--------------------+--------+
| 1 | Amruta | 2022-07-18 | MySQL | I |
| 2 | Siddeshdesh | 2022-06-02 | Computer Network | I |
| 3 | Swarali | 2022-06-22 | Operating System | I |
| 4 | Bhavesh | 2022-07-17 | Design of Complier | I |
| 5 | Chaitali | 2022-08-15 | Internet of Things | I |
| 6 | Omkar | 2022-09-02 | Mobile Computing | I |
+---------+-------------+---------------+--------------------+--------+
6 rows in set (0.00 sec)


mysql> delimiter //
mysql> create procedure A(IN rollno1 int(10), name1 varchar(20))
 -> begin
 -> declare i_date date;
 -> declare diff int;
 -> declare fine_amt int;
 -> declare EXIT HANDLER FOR SQLEXCEPTION SELECT 'Table not found';
 -> select Date_of_issue into i_date from borrower where Roll_no= rollno1 and Name_of_book =
name1;
 -> select DATEDIFF(CURDATE(), i_date) into diff;
 -> if(diff>=15 and diff<=30) then
 -> set fine_amt= diff*5;
 -> insert into fine values(rollno1, CURDATE(), fine_amt);
 -> elseif(diff>30) then
 -> set fine_amt= diff*50;
 -> insert into fine values(rollno1, CURDATE(), fine_amt);
 -> end if;
 -> update borrower set Status= 'R' where Roll_no= rollno1 and Name_of_book = name1;
 -> end
 -> //

Query OK, 0 rows affected, 1 warning (0.01 sec)
mysql> delimiter ;
mysql> call A(1, 'MySQL');
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from fine;
+---------+------------+--------+
| Roll_no | Date | Amount |
+---------+------------+--------+
| 1 | 2022-09-04 | 2400 |
+---------+------------+--------+
1 row in set (0.00 sec)



mysql> select * from borrower;
+---------+-------------+--------------+-------------------+--------+
| Roll_no | Name | Date_of_issue | Name_of_book | Status |
+---------+-------------+--------------+-------------------+--------+
| 1 | Amruta | 2022-07-18 | MySQL | R |
| 2 | Siddeshdesh | 2022-06-02 | Computer Network | I |
| 3 | Swarali | 2022-06-22 | Operating System | I |
| 4 | Bhavesh | 2022-07-17 | Design of Complier | I |
| 5 | Chaitali | 2022-08-15 | Internet of Things | I |
| 6 | Omkar | 2022-09-02 | Mobile Computing | I |
+---------+-------------+--------------+-------------------+--------+
6 rows in set (0.00 sec)

mysql> call A(2, 'Computer Network');
Query OK, 1 row affected (0.01 sec)
mysql> select * from borrower;
+---------+-------------+--------------+-------------------+--------+
| Roll_no | Name | Date_of_issue | Name_of_book | Status |
+---------+-------------+--------------+-------------------+--------+
| 1 | Amruta | 2022-07-18 | MySQL | R |
| 2 | Siddeshdesh | 2022-06-02 | Computer Network | R |
| 3 | Swarali | 2022-06-22 | Operating System | I |
| 4 | Bhavesh | 2022-07-17 | Design of Complier | I |
| 5 | Chaitali | 2022-08-15 | Internet of Things | I |
| 6 | Omkar | 2022-09-02 | Mobile Computing | I |
+---------+-------------+--------------+-------------------+--------+
6 rows in set (0.00 sec)

mysql> select * from fine;
+---------+------------+--------+
| Roll_no | Date | Amount |
+---------+------------+--------+
| 1 | 2022-09-04 | 2400 |
| 2 | 2022-09-04 | 4700 |
+---------+------------+--------+
2 rows in set (0.00 sec)

mysql> call A(5, 'Internet of Things');
Query OK, 1 row affected (0.00 sec)
mysql> select * from fine;
+---------+------------+--------+
```

```
| Roll_no | Date | Amount |
+---------+------------+--------+
| 1 | 2022-09-04 | 2400 |
| 2 | 2022-09-04 | 4700 |
| 5 | 2022-09-04 | 100 |
+---------+------------+--------+
3 rows in set (0.01 sec)

mysql> select * from borrower;
+---------+-------------+--------------+-------------------+--------+
| Roll_no | Name | Date_of_issue | Name_of_book | Status |
+---------+-------------+--------------+-------------------+--------+
| 1 | Amruta | 2022-07-18 | MySQL | R |
| 2 | Siddeshdesh | 2022-06-02 | Computer Network | R |
| 3 | Swarali | 2022-06-22 | Operating System | I |
| 4 | Bhavesh | 2022-07-17 | Design of Complier | I |
| 5 | Chaitali | 2022-08-15 | Internet of Things | R |
| 6 | Omkar | 2022-09-02 | Mobile Computing | I |
+---------+-------------+--------------+-------------------+--------+
6 rows in set (0.00 sec)

mysql> call A(6, 'Mobile Computing');
Query OK, 1 row affected (0.01 sec)
mysql> select * from borrower;
+---------+-------------+--------------+-------------------+--------+
| Roll_no | Name | Date_of_issue | Name_of_book | Status |
+---------+-------------+--------------+-------------------+--------+
| 1 | Amruta | 2022-07-18 | MySQL | R |
| 2 | Siddeshdesh | 2022-06-02 | Computer Network | R |
| 3 | Swarali | 2022-06-22 | Operating System | I |
| 4 | Bhavesh | 2022-07-17 | Design of Complier | I |
| 5 | Chaitali | 2022-08-15 | Internet of Things | R |
| 6 | Omkar | 2022-09-02 | Mobile Computing | R |
+---------+-------------+--------------+-------------------+--------+
6 rows in set (0.01 sec)

mysql> select * from fine;
+---------+------------+--------+
| Roll_no | Date | Amount |
+---------+------------+--------+
| 1 | 2022-09-04 | 2400 |
| 2 | 2022-09-04 | 4700 |
| 5 | 2022-09-04 | 100 |
+---------+------------+--------+
3 rows in set (0.00 sec)
```

Conclusion:

Successfully implemented the PL/SQL code with proper understanding of different control structure and exception handling.

Viva Questions:

- What is PL/SQL? Explain the structure of PL/SQL with example.
- Explain types of PL/SQL block?
- What is procedure? How to create procedure explain with example?
- What is named block of PL/SQL?
- Explain the if-else structure with example?

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	

Group A

Title of the 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 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)

Objective:

1) To understand the difference between procedure and function

2) To understand commands related to procedure and function

Outcome: Students will be able to learn and understand various stored procedure and Stored Function

Theory:

Procedures and Functions are the subprograms which can be created and saved in the database as database objects. They can be called or referred inside the other blocks also.

Parameter:

The parameter is variable or placeholder of any valid PL/SQL data-type through which the PL/SQL subprogram exchange the values with the main code. This parameter allows to give input to the subprograms and to extract from these subprograms.

- These parameters should be defined along with the subprograms at the time of creation.
- These parameters are included n the calling statement of these subprograms to interact the values with the subprograms.
- The datatype of the parameter in the subprogram and the calling statement should be same.
- The size of the datatype should not mention at the time of parameter declaration, as the size is dynamic for this type.

Based on their purpose parameters are classified as

1. IN Parameter
2. OUT Parameter
3. IN OUT Parameter

IN Parameter:

- This parameter is used for giving input to the subprograms.
- It is a read-only variable inside the subprograms. Their values cannot be changed inside the subprogram.
- In the calling statement, these parameters can be a variable or a literal value or an expression, for example, it could be the arithmetic expression like '5*8' or 'a/b' where 'a' and 'b' are variables.
- By default, the parameters are of IN type

OUT Parameter:

- This parameter is used for getting output from the subprograms.
- It is a read-write variable inside the subprograms. Their values can be changed inside the subprograms.
- In the calling statement, these parameters should always be a variable to hold the value from the current subprograms.

IN OUT Parameter:

- This parameter is used for both giving input and for getting output from the subprograms.
- It is a read-write variable inside the subprograms. Their values can be changed inside the subprograms.
- In the calling statement, these parameters should always be a variable to hold the value from the subprograms.

These parameter types should be mentioned at the time of creating the subprograms.

RETURN

RETURN is the keyword that instructs the compiler to switch the control from the subprogram to the calling statement. In subprogram RETURN simply means that the control needs to exit from the subprogram. Once the controller finds RETURN keyword in the subprogram, the code after this will be skipped.

Normally, parent or main block will call the subprograms, and then the control will shift from those parent blocks to the called subprograms. RETURN in the subprogram will return the control back to their parent block. In the case of functions RETURN statement also returns the value. The datatype of this value is always mentioned at the time of function declaration. The datatype can be of any valid PL/SQL data type.

Procedure in PL/SQL

A Procedure is a subprogram unit that consists of a group of PL/SQL statements. Each procedure in Oracle has its own unique name by which it can be referred. This subprogram unit is stored as a database object. Below are the characteristics of this subprogram unit.

- Procedures are standalone blocks of a program that can be stored in the database.
- Call to these procedures can be made by referring to their name, to execute the PL/SQL statements.
- It is mainly used to execute a process in PL/SQL.
- It can have nested blocks, or it can be defined and nested inside the other blocks or packages.
- It contains declaration part (optional), execution part, exception handling part (optional).
- The values can be passed into the procedure or fetched from the procedure through parameters.
- These parameters should be included in the calling statement.
- Procedure can have a RETURN statement to return the control to the calling block, but it cannot return any values through the RETURN statement.
- Procedures cannot be called directly from SELECT statements. They can be called from another block or through EXEC keyword.

Syntax:

```
CREATE OR REPLACE PROCEDURE
<procedure_name>
(
<parameter IN/OUT <data type>
..
.
)
[ IS | AS ]
<Declaration part>
BEGIN
<Execution part>
EXCEPTION
<Exception handling part>
END;
```

CREATE PROCEDURE instructs the compiler to create new procedure. Keyword 'OR REPLACE' instructs the compile to replace the existing procedure (if any) with the current one.

- Procedure name should be unique.
- Keyword 'IS' will be used, when the procedure is nested into some other blocks. If the procedure is standalone then 'AS' will be used. Other than this coding standard, both have the same meaning.

Procedures: Example

The below example creates a procedure ‘employer_details’ which gives the details of the employee.

```
1> CREATE OR REPLACE PROCEDURE employer_details
2> IS
3> CURSOR emp_cur IS
4> SELECT first_name, last_name, salary FROM emp_tbl;
5> emp_rec emp_cur%rowtype;
6> BEGIN
7> FOR emp_rec in sales_cur
8> LOOP
9> dbms_output.put_line(emp_cur.first_name || ' ' ||emp_cur.last_name
10> || ' ' ||emp_cur.salary);
11> END LOOP;
12>END;
13> /
```

Function:

A function is a standalone PL/SQL subprogram. Like PL/SQL procedure, functions have a unique name by which it can be referred. These are stored as PL/SQL database objects. Below are some of the characteristics of functions.

- Functions are a standalone block that is mainly used for calculation purpose.
- Function use RETURN keyword to return the value, and the datatype of this is defined at the time of creation.
- A Function should either return a value or raise the exception, i.e. return is mandatory in functions
- Function with no DML statements can be directly called in SELECT query whereas the function with DML operation can only be called from other PL/SQL blocks.

- It can have nested blocks, or it can be defined and nested inside the other blocks or packages.
- It contains declaration part (optional), execution part, exception handling part (optional).
- The values can be passed into the function or fetched from the procedure through the parameters.
- These parameters should be included in the calling statement.
- Function can also return the value through OUT parameters other than using RETURN.
- Since it will always return the value, in calling statement it always accompanies with assignment operator to populate the variables.

Syntax:

```
CREATE OR REPLACE FUNCTION
<procedure_name>
        (
        <parameter1 IN/OUT <datatype>
        ..
        .
        )
        RETURN <datatype>
[ IS | AS ]
        <declaration_part>
BEGIN
        <execution part>
EXCEPTION
        <exception handling part>
END;
```

CREATE FUNCTION instructs the compiler to create a new function. Keyword 'OR REPLACE' instructs the compiler to replace the existing function (if any) with the current one.

- The Function name should be unique.
- RETURN datatype should be mentioned.
- Keyword 'IS' will be used, when the procedure is nested into some other blocks. If the procedure is standalone then 'AS' will be used. Other than this coding standard, both have the same meaning

Function: Example

Let's create a frunction called "employer_details_func' similar to the one created in stored proc

```
1> CREATE OR REPLACE FUNCTION employer_details_func
2> RETURN VARCHAR(20);
3> IS
5> emp_name VARCHAR(20);
6> BEGIN
7> SELECT first_name INTO emp_name
8> FROM emp_tbl WHERE empID = '100';
9> RETURN emp_name;
10> END;
11> /
```

Similarities between Procedure and Function

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

```
MariaDB [(none)]> use stud;
Database changed
MariaDB [stud]> create table stud_marks(rollno int(2),name varchar(12),total_marks int(12));
Query OK, 0 rows affected (0.06 sec)
MariaDB [stud]> insert into stud_marks
values(1,"Ravi",933),(2,"sagar",450),(3,"sarita",1300),(4,"avi",250),(5,"raj",675);
Query OK, 5 rows affected (0.02 sec)
Records: 5 Duplicates: 0 Warnings: 0

MariaDB [stud]> select * from stud_marks;
+--------+--------+-------------+
| rollno | name | total_marks |
+--------+--------+-------------+
| 1 | Ravi | 933 |
| 2 | sagar | 450 |
| 3 | sarita | 1300 |
| 4 | avi | 250 |
| 5 | raj | 675 |
+--------+--------+-------------+
5 rows in set (0.00 sec)
MariaDB [stud]> create table new_stud_marks(roll_no int(2),name char(10),grade char(10));
Query OK, 0 rows affected (0.04 sec)

create procedure proc_grade1()
 begin
 declare i int;
 declare n int;
 declare rollno1 int;
 declare name1 varchar(20);
 declare class1 varchar(40);
 declare total1 int;
 declare s1 int;
 declare s2 int;
 declare s3 int;
 declare s4 int;
 declare s5 int;
 select count(*)into n from marks;
 set i=0;
 disp:loop
 set i=i+1;
 select rollno into rollno1 from marks where rollno=i;
 select name into name1 from marks where rollno=i;
 select sub1 into s1 from marks where rollno=i;
 select sub2 into s2 from marks where rollno=i;
 select sub3 into s3 from marks where rollno=i;
 select sub4 into s4 from marks where rollno=i;
 select sub5 into s5 from marks where rollno=i;
 set total1=s1+s2+s3+s4+s5;
 if total1<=1500 and total1>990 then
 set class1='distinction';
 else
 if total1<=989 and total1>900 then
```

```
set class1='first class';
else
if total1<899 and total1>825 then
set class1='higher class';
else
set class1='pass class';
end if;
end if;
end if;
insert into stud_marks values(rollno1,name1,total1);
insert into results values(rollno1,name1,class1);
if i=n then
leave disp;
end if;
end loop;
end
//
```

```
mysql> create table stud_marks(rolllno int,name varchar(90),class varchar(90));
Query OK, 0 rows affected (0.29 sec)

mysql> create table results(rollno int,name varchar(90),class varchar
    -> (90));
Query OK, 0 rows affected (0.27 sec)

mysql> alter table stud_marks drop class;
Query OK, 0 rows affected (0.28 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table stud_marks add total int;
Query OK, 0 rows affected (0.26 sec)
Records: 0  Duplicates: 0  Warnings: 0

 create table marks(rollno int,name varchar(99),sub1 int,sub2 int,sub3 int,sub4 int,sub5 int);
Query OK, 0 rows affected (0.34 sec)

mysql> insert into marks values(1,'komal',254,123,345,234,444);
Query OK, 1 row affected (0.06 sec)

mysql> select * from marks;
+--------+--------+------+------+------+------+------+
| rollno | name   | sub1 | sub2 | sub3 | sub4 | sub5 |
+--------+--------+------+------+------+------+------+
|      1 | komal  |  254 |  123 |  345 |  234 |  444 |
|      2 | shital |  154 |  223 |  325 |  134 |  144 |
|      3 | raj    |  154 |  523 |  245 |  244 |  414 |
+--------+--------+------+------+------+------+------+
3 rows in set (0.01 sec)
```

```
//////////////////////output/////////////////////////////////////
mysql> delimiter ;
mysql> call proc_grades();
Query OK, 1 row affected (0.97 sec)

mysql> select * from stud_marks;
+---------+--------+-------+
| rolllno | name   | total |
+---------+--------+-------+
|       1 | komal  |  NULL |
|       2 | shital |  NULL |
|       3 | raj    |  NULL |
+---------+--------+-------+
3 rows in set (0.00 sec)

mysql> select * from results;
+--------+--------+------------+
| rollno | name   | class      |
+--------+--------+------------+
|      1 | komal  | pass class |
|      2 | shital | pass class |
|      3 | raj    | pass class |
+--------+--------+------------+
3 rows in set (0.00 sec)
```

Conclusion: Performed implementation of procedures and functions in MYSQL successfully.

Viva Questions:

- Explain a for loop with example?
- Explain if-else loop with example?
- Explain In parameter with example?
- Explain while loop with example?
- Explain out parameter with example?

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	

Group A

Assignment No 7

Title of the 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_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.

Objective of the Assignment:

To understand the concept of cursors and its types.

Outcome:

Students will be able to learn and understand All types: Implicit, Explicit, Cursor FOR Loop, and Parameterized Cursor.

Theory:

PL/SQL Cursor

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

There are two types of cursors:

- Implicit Cursors
- Explicit Cursors

1) PL/SQL Implicit Cursors

The implicit cursors are automatically generated by Oracle while an SQL statement is executed, if you don't use an explicit cursor for the statement.

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

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

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

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

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

2) PL/SQL Explicit Cursors

The Explicit cursors are defined by the programmers to gain more control over the context area. These cursors should be defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.

Syntax of explicit cursor

Following is the syntax to create an explicit cursor:

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

Steps:

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

1. Declare the cursor to initialize in the memory.
2. Open the cursor to allocate memory.
3. Fetch the cursor to retrieve data.
4. Close the cursor to release allocated memory.

1) Declare the cursor:

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

Syntax for explicit cursor declaration
CURSOR name IS

1. SELECT statement;

2) Open the cursor:

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

Syntax for cursor open:

OPEN cursor_name;

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

Syntax for cursor fetch:

FETCH cursor_name INTO variable_list;

4) Close the cursor:
It is used to release the allocated memory. The following syntax is used to close the above-opened cursors.

Syntax for cursor close:

Close cursor_name;

/.////////////////////////////////////Program//////////////////

```
create procedure n1(in rno1 int)
begin
declare rno2 int;
declare exit_cond boolean;
declare c1 cursor for select rollno from o_rollcall where rollno>rno1;
declare continue handler for not found set exit_cond=TRUE;
open c1;
l1:loop
fetch c1 into rno2;
if not exists(select * from n_rollcall where rollno=rno2)then
insert into n_rollcall select * from o_rollcall where rollno=rno2;
end if;
if exit_cond then
close c1;
leave l1;
end if;
end loop l1;
end
//
```

```
mysql> create table o_rollcall(rollno int,name varchar(90),address varchar(90));

Query OK, 0 rows affected (0.53 sec)

mysql> create table n_rollcall(rollno int,name varchar(90),address varchar(90));
Query OK, 0 rows affected (0.50 sec)

mysql> insert into o_rollcall values(1,'komal','abc');
Query OK, 1 row affected (0.07 sec)

mysql> insert into o_rollcall values(2,'raj','pbc');
Query OK, 1 row affected (0.06 sec)

mysql> insert into o_rollcall values(3,'rudra','pune');
```

```
Query OK, 1 row affected (0.07 sec)
mysql> insert into n_rollcall values(1,'komal','abc');
Query OK, 1 row affected (0.06 sec)

mysql> insert into n_rollcall values(4,'lina','nashik');
Query OK, 1 row affected (0.08 sec)

mysql> insert into n_rollcall values(5,'sheetal','aaldi');
Query OK, 1 row affected (0.06 sec)

mysql> select * from o_rollcall;
+--------+-------+---------+
| rollno | name  | address |
+--------+-------+---------+
|      1 | komal | abc     |
|      2 | raj   | pbc     |
|      3 | rudra | pune    |
+--------+-------+---------+
3 rows in set (0.00 sec)

mysql> select * from n_rollcall;
+--------+---------+---------+
| rollno | name    | address |
+--------+---------+---------+
|      1 | komal   | abc     |
|      4 | lina    | nashik  |
|      5 | sheetal | aaldi   |
+--------+---------+---------+

////////////////OUTPUT///////////////
mysql> delimiter ;
mysql> call n1(1);
Query OK, 0 rows affected (0.15 sec)

mysql> select * from n_rollcall;
+--------+---------+---------+
| rollno | name    | address |
+--------+---------+---------+
|      1 | komal   | abc     |
|      4 | lina    | nashik  |
|      5 | sheetal | aaldi   |
|      2 | raj     | pbc     |
|      3 | rudra   | pune    |
+--------+---------+---------+
5 rows in set (0.00 sec)
```

```
Mysql> select * from o_rollcall;
+--------+-------+---------+
| rollno | name  | address |
+--------+-------+---------+
|      1 | komal | abc     |
|      2 | raj   | pbc     |
|      3 | rudra | pune    |
+--------+-------+---------+
3 rows in set (0.00 sec)
```

Conclusion: Performed implementation of cursors in MYSQL successfully.

Viva Questions:

- What is cursor? Explain with example?
- Write advantages of cursor?
- Write application of cursor?
- Explain the types of cursor?
- Explain the explicit cursor with example**?**

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	

Group A

Assignment No 8

Title of the Assignment:

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.

Objective of the Assignment: To understand the concept of Triggers and its types.

Outcome:

Students will be able to learn and understand All types: All Types: Row level and Statement level triggers, Before and After Triggers.

Theory:

A PL/SQL trigger is a named database object that encapsulates and defines a set of actions that are to be performed in response to an insert, update, or delete operation against a table. Triggers are created using the PL/SQL CREATE TRIGGER statement.

Types of triggers (PL/SQL)

The data server supports row-level and statement-level triggers within a PL/SQL context.

- A row-level trigger fires once for each row that is affected by a triggering event. For example, if deletion is defined as a triggering event for a particular table, and a single DELETE statement deletes five rows from that table, the trigger fires five times, once for each row.
- A statement-level trigger fires only once for each statement. Using the previous example, if deletion is defined
- as a triggering event for a particular table, and a single DELETE statement deletes five rows from that table, the trigger fires once. Statement-level trigger granularity cannot be specified for BEFORE triggers or INSTEAD OF triggers.

Trigger variables (PL/SQL)

NEW and OLD are special variables that you can use with PL/SQL triggers without explicitly defining them.

- NEW is a pseudo-record name that refers to the new table row for insert and update operations in row-level triggers. Its usage is : NEW.column, where column is the name of a column in the table on which the trigger is defined.
- OLD is a pseudo-record name that refers to the old table row for update and delete operations in row-level triggers. Its usage is :OLD.column, where column is the name of a column in the table on which the trigger is defined.

Trigger event predicates (PL/SQL)

The trigger event predicates, UPDATING, DELETING, and INSERTING can only be used in a trigger to identify the event that activated the trigger.

OR REPLACE

Specifies to replace the definition for the trigger if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog. This option is ignored if a definition for the trigger does not exist at the current server.

trigger-name

Names the trigger. The name, including the implicit or explicit schema name, must not identify a trigger already described in the catalog (SQLSTATE 42710). If a two-part name is specified, the schema name cannot begin with 'SYS' (SQLSTATE 42939).

BEFORE

Specifies that the associated triggered action is to be applied before any changes caused by the actual update of the subject table are applied to the database.

AFTER

Specifies that the associated triggered action is to be applied after the changes caused by the actual update of the subject table are applied to the database.

INSTEAD OF

Specifies that the associated triggered action replaces the action against the subject view.

trigger-event

Specifies that the triggered action associated with the trigger is to be executed whenever one of the events is applied to the subject table. Any combination of the events can be specified, but each event (INSERT, DELETE, and UPDATE) can only be specified once (SQLSTATE 42613).

INSERT

Specifies that the triggered action associated with the trigger is to be executed whenever an INSERT operation is applied to the subject table.

DELETE

Specifies that the triggered action associated with the trigger is to be executed whenever a DELETE operation is applied to the subject table.

UPDATE

Specifies that the triggered action associated with the trigger is to be executed whenever an UPDATE operation is applied to the subject table, subject to the columns specified or implied.

If the optional column-name list is not specified, every column of the table is implied. Therefore, omission of the column-name list implies that the trigger will be activated by the update of any column of the table.

OF column-name,...

Each column-name specified must be a column of the base table (SQLSTATE 42703). If the trigger is a BEFORE trigger, the column-name specified cannot be a generated column other than the identity column (SQLSTATE 42989). No column-name can appear more than once in the column-name list (SQLSTATE 42711). The trigger will only be activated by the update of a column that is identified in the column-name list. This clause cannot be specified for an INSTEAD OF trigger (SQLSTATE 42613).

ON table-name

Designates the subject table of the BEFORE trigger or AFTER trigger definition. The name must specify a base table or an alias that resolves to a base table (SQLSTATE 42704 or 42809). The name must not specify a catalog table (SQLSTATE 42832), a materialized query table (SQLSTATE 42997), a created temporary table, a declared temporary table (SQLSTATE 42995), or a nickname (SQLSTATE 42809).

REFERENCING

Specifies the correlation names for the *transition variables*. Correlation names identify a specific row in the set of rows affected by the triggering SQL operation. Each row affected by the triggering SQL operation is available to the triggered action by qualifying columns with correlation-names specified as follows.

OLD AS correlation-name

Specifies a correlation name that identifies the row state prior to the triggering SQL operation. If the trigger event is INSERT, the values in the row are null values.

NEW AS correlation-name

Specifies a correlation name that identifies the row state as modified by the triggering SQL operation and by any SET statement in a BEFORE trigger that has already executed. If the trigger event is DELETE, the values in the row are null values.

If the REFERENCING clause is not invoked, then trigger variables NEW and OLD can optionally be used without explicitly defining them.

FOR EACH ROW

Specifies that the triggered action is to be applied once for each row of the subject table that is affected by the triggering SQL operation.

FOR EACH STATEMENT

Specifies that the triggered action is to be applied only once for the whole statement.

WHEN

(search-condition)

Specifies a condition that is true, false, or unknown. The search-condition provides a capability to determine whether or not a certain triggered action should be executed. The associated action is performed only if the specified search condition evaluates as true.

declaration

Specifies a variable declaration.

statement or handler-statement

Specifies a PL/SQL program statement. The trigger body can contain nested blocks.

condition

Specifies an exception condition name, such as NO_DATA_FOUND.

program

```
delimiter //
create trigger tr_ins77_Borrower
before insert
on table1
for each row
begin
insert into table2 values(new.rollno,new.name,new.nameofbook);
end
//
```

```
//////////////////////output////////////////////////////////
mysql> insert into table1 values(1,'gg','ggh');
    -> //
Query OK, 1 row affected (0.06 sec)
```

```
mysql> select * from table2;
    -> //
+--------+------+------------+
| rollno | name | nameofbook |
+--------+------+------------+
|      1 | gg   | ggh        |
+--------+------+------------+
1 row in set (0.00 sec)
```

```
//////////////////////////////////////////////////
```

```
delimiter //
create trigger tr_del_Borrower
before delete
on table1
for each row
begin
delete from table2 where rollno=old.rollno;
end
//
```

```
//////////////////output/////////////////////////////
mysql> delete from table1 where rollno=1;
    -> //
Query OK, 1 row affected (0.07 sec)

mysql> select * from table2;
    -> //
```

```
+--------+-------+------------+
| rollno | name  | nameofbook |
+--------+-------+------------+
|      2 | komal | oop        |
+--------+-------+------------+
1 row in set (0.00 sec)
//////////////////////////////////////////////
delimiter //
create trigger tr_up1_Borrower
before update
on table1
for each row
begin
update table2 set rollno=new.rollno where rollno=old.rollno;
end
//

//////////////////output//////////////////

mysql> update table1 set rollno=4 where rollno=2;
    -> //
Query OK, 1 row affected (0.10 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from table2;
    -> //
+--------+-------+------------+
| rollno | name  | nameofbook |
+--------+-------+------------+
|      4 | komal | oop        |
+--------+-------+------------+
1 row in set (0.00 sec)
```

Conclusion: Performed implementation of Triggers in PL/SQL successfully.

Viva Question:

- What is trigger explain with examples?
- Write is advantages of trigger?
- Write is application of trigger?
- What is before and after the event in Triggers?
- What are the types for trigger?

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	

Group A

Assignment No 9

Title of the 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.)

Objective of the Assignment: To understand the concept of MySQL/Oracle database Connectivity.

Outcome:

Students will be able to learn and understand MySQL/Oracle database Connectivity.

Theory:

In Java, we can connect to our database(MySQL) with JDBC(Java Database Connectivity) through the Java code. JDBC is one of the standard APIs for database connectivity, using it we can easily run our query, statement, and also fetch data from the database.

Prerequisite to understand Java Database Connectivity with MySQL:-

1. You have MySQL on your System.

2. You have JDK on your System.

3. To set up the connectivity user should have MySQL Connector to the Java (JAR file), the 'JAR' file must be in classpath while compiling and running the code of JDBC.

Steps to download MySQL Connector:

Search for MySQL community downloads.

- Then, go to the Connector/J.
- Then, select the Operating System platform-independent.
- Then, download the zip file Platform Independent (Architecture Independent), ZIP Archive.
- Then, extract the zip file.
- Get the mysql-connector-java-8.0.20.jar file from the folder.

Java Database Connectivity with MySQL.

To connect Java application with the MySQL database, we need to follow steps.
In this example we are using MySql as the database. So we need to know following information's for the mysql database:

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

Let's first create a table in the mysql database, but before creating table, we need to create database first.

```
create database sonoo;
use sonoo;
create table emp(id int(10),name varchar(40),age int(3));
```

Example to Connect Java Application with mysql database

In this example, sonoo is the database name, root is the username and password both.

1. Connection program.

```
import java.sql.*;
 class MysqlCon{
 public static void main(String args[]){
 try{
 Class.forName("com.mysql.jdbc.Driver");
 Connection con=DriverManager.getConnection(
 "jdbc:mysql://localhost:3306/sonoo","root","root");
 //here sonoo is database name, root is username and password
 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);}
 }
 }
```

2. Insert record

```
package simple;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class insert {

public static void main(String[] args) {
	// TODO Auto-generated method stub
	try{
		Class.forName("com.mysql.cj.jdbc.Driver");
		Connection con=DriverManager.getConnection(
		"jdbc:mysql://localhost:3306/TE","root","123");
		Statement stmt=con.createStatement();
	   // Execute a query
	   System.out.println("Inserting records into the table...");
	   String sql = "INSERT INTO student VALUES (100, 'Zara')";
	   stmt.executeUpdate(sql);
	   sql = "INSERT INTO student VALUES (101, 'Mahnaz')";
	   stmt.executeUpdate(sql);
	   sql = "INSERT INTO student  VALUES (102, 'Zaid')";
	   stmt.executeUpdate(sql);
	   sql = "INSERT INTO student VALUES(103, 'Sumit')";
	   stmt.executeUpdate(sql);
	   System.out.println("Inserted records into the table...");


		}catch(Exception e){ System.out.println(e);
	}
}
}
```

3. Update Record

```
package simple;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.DriverManager;
import java.sql.Statement;

public class update {
 static final String QUERY = "SELECT rollno,name FROM student";
public static void main(String[] args) {
	// TODO Auto-generated method stub
	try{
		Class.forName("com.mysql.cj.jdbc.Driver");
		Connection con=DriverManager.getConnection(
		"jdbc:mysql://localhost:3306/TE","root","Root@1234");
		//here sonoo is database name, root is username and password
```

```
                    Statement stmt=con.createStatement();
                String sql = "UPDATE student " +
                  "SET name = 'om' WHERE rollno in (100)";
                stmt.executeUpdate(sql);
                ResultSet rs = stmt.executeQuery(QUERY);
                while(rs.next()){
                  //Display values
                   System.out.println(rs.getInt(1)+"  "+rs.getString(2));
                }}catch(Exception e){ System.out.println(e);}

          }
}
```

4. Delete record

```
mport java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JDBCExample {
  static final String DB_URL = "jdbc:mysql://localhost/TUTORIALSPOINT";
  static final String USER = "guest";
  static final String PASS = "guest123";
  static final String QUERY = "SELECT id, first, last, age FROM Registration";

  public static void main(String[] args) {
    // Open a connection
    try(Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
      Statement stmt = conn.createStatement();
    ) {
      String sql = "DELETE FROM Registration " +
        "WHERE id = 101";
      stmt.executeUpdate(sql);
      ResultSet rs = stmt.executeQuery(QUERY);
      while(rs.next()){
        //Display values
        System.out.print("ID: " + rs.getInt("id"));
        System.out.print(", Age: " + rs.getInt("age"));
      }
      rs.close();
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }
    }
```

Conclusion: Performed implementation of MySQL/Oracle database connectivity with java platform.

Viva Question:

- What is database connectivity?
- Write is JDBC?
- Write is JDBC Driver?
- What are the steps to connect to the database in java?
- What is the return type of Class.forName() method?

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	

Group B

Assignment No 10

Title of the Assignment: MongoDB Queries

Design and Develop MongoDB Queries using CRUD operations. (Use CRUD operations,

SAVE method, logical operators etc.).

Objective of the Assignment: To understand the concept of CURD operations and logical operators.

Outcome:

Students will be able to learn and understand concept of CURD operations and logical operators.

Theory:

MongoDB Information:

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Database:
Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection:
Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document:
A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field

Features of MongoDB:

- Multiple Servers: It can run over multiple servers.
- Schema-less Database: It is a schema-less database.
- Indexing: Any field in the document can be indexed.
- Rich Object Model: It supports a rich object model.

MongoDB CRUD operations

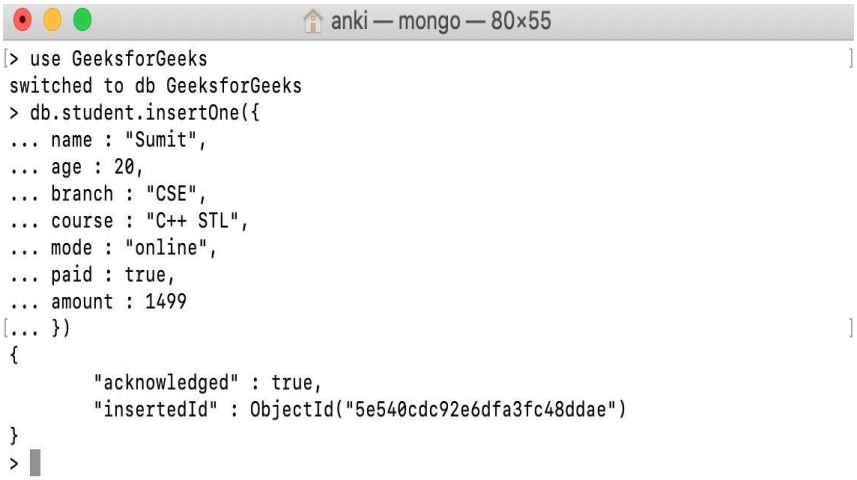
Create Operations –

The create or insert operations are used to insert or add new documents in the collection.
If a collection does not exist, then it will create a new collection in the database.
You can perform, create operations using the following methods provided by the MongoDB:

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

Example 1:

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



Example 2:
In this example, we are inserting details of the multiple students in the form of documents in the student collection using **db.collection.insertMany()** method.

Read Operations –
The Read operations are used to retrieve documents from the collection, or in other words, read operations are used to query a collection for a document.

You can perform read operation using the following method provided by the MongoDB:

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

Example :

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

```
anki — mongo — 80×55
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddb0"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
>
```

Update Operations –

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

Method & Description

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

Example 1:

In this example, we are updating the age of Sumit in the student collection using **db.collection.updateOne()** method.

```
anki — mongo — 80×43
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.updateOne({name: "Sumit"},{$set:{age: 24 }})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 0 }
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
        "name" : "Sumit",
        "age" : 24,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddb0"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
>
```

Example 2:
In this example, we are updating the year of course in all the documents in the student collection using **db.collection.updateMany()** method.

```
anki — mongo — 80×43
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.updateMany({}, {$set: {year: 2020}})
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
        "name" : "Sumit",
        "age" : 24,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddb0"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
>
```

Delete Operations –

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

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

Example 1:

In this example, we are deleting a document from the student collection using **db.collection.deleteOne()** method.

```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
        "name" : "Sumit",
        "age" : 24,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
{
        "_id" : ObjectId("5e54103592e6dfa3fc48ddb1"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
> db.student.deleteOne({name: "Sumit"})
{ "acknowledged" : true, "deletedCount" : 1 }
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
{
        "_id" : ObjectId("5e54103592e6dfa3fc48ddb1"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
```

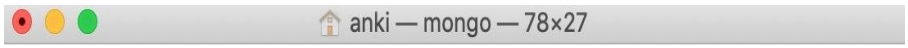
MongoDB – Logical Query Operators

- MongoDB supports logical query operators.
- These operators are used for filtering the data and getting precise results based on the given conditions.
- The following table contains the comparison query operators:

1. $and
   It is used to join query clauses with a logical AND and return all documents that match the given conditions of both clauses.

Example:
Query:

```
db.contributor.find({$and: [{branch: "CSE"}, {joiningYear: 2018}]}).pretty()
```




```
> db.contributor.find({$and: [{branch: "CSE"}, {joiningYear: 2018}]}).pretty()
{
        "_id" : ObjectId("5e6f7a6692e6dfa3fc48ddbe"),
        "name" : "Rohit",
        "branch" : "CSE",
        "joiningYear" : 2018,
        "language" : [
                "C#",
                "Python",
                "Java"
        ],
        "personal" : {
                "contactinfo" : 0,
                "state" : "Delhi",
                "age" : 24,
                "semesterMarks" : [
                        70,
                        73.3,
                        76.5,
                        78.6
                ]
        },
        "salary" : 1000
}
```

2. $or
   It is used to join query clauses with a logical OR and return all documents that match the given conditions of either clause.

Example:
Query:

```
db.contributor.find({$or: [{branch: "ECE"}, {joiningYear: 2017}]}).pretty()
```

anki — mongo — 78×46

```
> db.contributor.find({$or: [{branch: "ECE"}, {joiningYear: 2017}]}).pretty()
{
        "_id" : ObjectId("5e7b9f0a92e6dfa3fc48ddbf"),
        "name" : "Amit",
        "branch" : "ECE",
        "joiningYear" : 2017,
        "language" : [
                "Python",
                "C#"
        ],
        "personal" : {
                "contactinfo" : 234556789,
                "state" : "UP",
                "age" : 25,
                "semesterMarks" : [
                        80,
                        80.1,
                        98,
                        70
                ]
        },
        "salary" : 10000
}
{
        "_id" : ObjectId("5e7b9f0a92e6dfa3fc48ddc0"),
        "name" : "Sumit",
        "branch" : "CSE",
        "joiningYear" : 2017,
        "language" : [
                "Java",
                "Perl"
        ],
        "personal" : {
                "contactinfo" : 2300056789,
                "state" : "MP",
                "age" : 24,
                "semesterMarks" : [
                        89,
                        80.1,
                        78,
                        71
                ]
        }
}
```

3. $not
   It is used to invert the effect of the query expressions and return documents that does not match the query expression

   Example:
   Query:
   db.contributor.find({salary: {$not: {$gt: 2000}}}).pretty()

anki — mongo — 78×47

```
> db.contributor.find({salary: {$not: {$gt: 2000}}}).pretty()
{
        "_id" : ObjectId("5e6f7a6692e6dfa3fc48ddbe"),
        "name" : "Rohit",
        "branch" : "CSE",
        "joiningYear" : 2018,
        "language" : [
                "C#",
                "Python",
                "Java"
        ],
        "personal" : {
                "contactinfo" : 0,
                "state" : "Delhi",
                "age" : 24,
                "semesterMarks" : [
                        70,
                        73.3,
                        76.5,
                        78.6
                ]
        },
        "salary" : 1000
}
{
        "_id" : ObjectId("5e7b9f0a92e6dfa3fc48ddc0"),
        "name" : "Sumit",
        "branch" : "CSE",
        "joiningYear" : 2017,
        "language" : [
                "Java",
                "Perl"
        ],
        "personal" : {
                "contactinfo" : 2300056789,
                "state" : "MP",
                "age" : 24,
                "semesterMarks" : [
                        89,
                        80.1,
                        78,
                        71
                ]
        }
} _
```

4. $nor
   It is used to join query clauses with a logical NOR and return all documents that fail to match both clauses.

   Example:
   Query:
   db.contributor.find({$nor: [{salary: 3000}, {branch: "ECE"}]}).pretty()

```
anki — mongo — 78×46
> db.contributor.find({$nor: [{salary: 3000}, {branch: "ECE"}]}).pretty()
{
        "_id" : ObjectId("5e6f7a6692e6dfa3fc48ddbe"),
        "name" : "Rohit",
        "branch" : "CSE",
        "joiningYear" : 2018,
        "language" : [
                "C#",
                "Python",
                "Java"
        ],
        "personal" : {
                "contactinfo" : 0,
                "state" : "Delhi",
                "age" : 24,
                "semesterMarks" : [
                        70,
                        73.3,
                        76.5,
                        78.6
                ]
        },
        "salary" : 1000
}
{
        "_id" : ObjectId("5e7b9f0a92e6dfa3fc48ddc0"),
        "name" : "Sumit",
        "branch" : "CSE",
        "joiningYear" : 2017,
        "language" : [
                "Java",
                "Perl"
        ],
        "personal" : {
                "contactinfo" : 2300056789,
                "state" : "MP",
                "age" : 24,
                "semesterMarks" : [
                        89,
                        80.1,
                        78,
                        71
                ]
        }
}
```

Conclusion: Performed and implement the CURD operations and logical operators.

Viva Question:

- What is MongoDB?
- What are some of the advantages of MongoDB??
- What is a Document and collection in MongoDB?
- What are the CURD operations?
- Write the all logical operators with examples?

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	

Group B

Assignment No 11

Title of the Assignment: MongoDB – Aggregation and Indexing:

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

Objective:

1) To understand the difference aggregation and indexing in MongoDB.

Outcome: Students will be able to learn and understand MongoDB Queries using aggregation and indexing

Theory:

MongoDB - Aggregation

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In SQL count(*) and with group by is an equivalent of MongoDB aggregation.

Aggregation is a way of processing a large number of documents in a collection by means of passing them through different stages. The stages make up what is known as a pipeline. The stages in a pipeline can filter, sort, group, reshape and modify documents that pass through the pipeline.

One of the most common use cases of Aggregation is to calculate aggregate values for groups of documents. This is similar to the basic aggregation available in SQL with the GROUP BY clause and COUNT, SUM and AVG functions. MongoDB Aggregation goes further though and can also perform relational-like joins, reshape documents, create new and update existing collections, and so on.

While there are other methods of obtaining aggregate data in MongoDB, the aggregation framework is the recommended approach for most work.

There are what are called single purpose methods like estimatedDocumentCount(), count(), and distinct() which are appended to a find() query making them quick to use but limited in scope.

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"}}}])

Program for aggregation:

```
test> use customer
switched to db customer
customer> db.createCollection('Customer');
{ ok: 1 }
customer> db.Customer.insertOne({'CustID':'A123', 'Amount':500,
'Status':'completed'})
{
acknowledged: true,
insertedId: ObjectId("635a9a5d29e81711afa59d0b")
}
customer> db.Customer.insertOne({'CustID':'A124', 'Amount':200,
'Status':'completed'})
{
acknowledged: true,
insertedId: ObjectId("635a9a7c29e81711afa59d0c")
}
customer> db.Customer.insertOne({'CustID':'A150', 'Amount':250,
'Status':'pending'})
{
acknowledged: true,
insertedId: ObjectId("635a9a9a29e81711afa59d0d")
}
customer> db.Customer.insertOne({'CustID':'A150', 'Amount':300,
'Status':'pending'})
{
acknowledged: true,
insertedId: ObjectId("635a9aa629e81711afa59d0e")
}
customer> db.Customer.insertOne({'CustID':'A201', 'Amount':750,
'Status':'pending'})
{
acknowledged: true,
insertedId: ObjectId("635a9aef29e81711afa59d0f")
}

customer> db.Customer.find().pretty()
[
{
_id: ObjectId("635a9a5d29e81711afa59d0b"),
CustID: 'A123',
Amount: 500,
Status: 'completed'
},
{
_id: ObjectId("635a9a7c29e81711afa59d0c"),
CustID: 'A124',
Amount: 200,
Status: 'completed'
},
```

```
{
_id: ObjectId("635a9a9a29e81711afa59d0d"),


CustID: 'A150',
Amount: 250,
Status: 'pending'
},
{
_id: ObjectId("635a9aa629e81711afa59d0e"),
CustID: 'A150',
Amount: 300,
Status: 'pending'
},
{
_id: ObjectId("635a9aef29e81711afa59d0f"),
CustID: 'A201',
Amount: 750,
Status: 'pending'
}
]


customer>
```

1. Find the customer whose status is "completed"

```
customer> db.Customer.aggregate([{$match:{'Status':'completed'}}])
[
{
_id: ObjectId("635a9a5d29e81711afa59d0b"),
CustID: 'A123',
Amount: 500,
Status: 'completed'
},
{
_id: ObjectId("635a9a7c29e81711afa59d0c"),
CustID: 'A124',
Amount: 200,
Status: 'completed'
}
]
```

2. Find the customer whose status is "completed" and whose amount is more than or equal to 250

```
customer> db.Customer.aggregate([{$match:{'Status':'completed',
'Amount':{$gte:250}}}])
[
{
_id: ObjectId("635a9a5d29e81711afa59d0b"),
CustID: 'A123',
Amount: 500,
Status: 'completed'
}
```

```
]
```

3. In above query use match operator twice to acheive same result

```
customer>
db.Customer.aggregate([{$match:{'Status':'completed'}},{$match:{'Amount':
{$gte:250}}}])
[
{
_id: ObjectId("635a9a5d29e81711afa59d0b"),
CustID: 'A123',
Amount: 500,
Status: 'completed'
}
]
```

4. Find sum of amount of customer under status "completed"

```
customer>
db.Customer.aggregate([{$match:{'Status':'completed'}},{$group:{_id:'$Cus
tID','totalAmount':{$sum:'$Amount'}}}])
[
{ _id: 'A123', totalAmount: 500 },
{ _id: 'A124', totalAmount: 200 }
]
```

5. Find the minimum amount of customer under status "pending"

```
customer>
db.Customer.aggregate([{$match:{'Status':'pending'}},{$group:{_id:'$CustI
D','minAmount':{$max:'$Amount'}}}])
[ { _id: 'A150', minAmount: 300 }, { _id: 'A201', minAmount: 750 } ]
```

6. Find the maximum amount of customer under status "pending"

```
customer>
db.Customer.aggregate([{$match:{'Status':'pending'}},{$group:{_id:'$CustI
D','maxAmount':{$max:'$Amount'}}}]).pretty()
[ { _id: 'A150', maxAmount: 300 }, { _id: 'A201', maxAmount: 750 } ]
```

7. Find the average amount of customer under status "pending"

```
customer>
db.Customer.aggregate([{$match:{'Status':'pending'}},{$group:{_id:'$CustI
D','avgAmount':{$avg:'$Amount'}}}]).pretty()
[ { _id: 'A150', avgAmount: 275 }, { _id: 'A201', avgAmount: 750 } ]
```

8. Sort the customer according to CustID in ascending order

```
customer> db.Customer.aggregate([{$sort:{'CustID':1}}]).pretty()
[
{
_id: ObjectId("635a9a5d29e81711afa59d0b"),
CustID: 'A123',
Amount: 500,
Status: 'completed'
},
{
_id: ObjectId("635a9a7c29e81711afa59d0c"),
```

```
CustID: 'A124',

Amount: 200,
Status: 'completed'
},
{
_id: ObjectId("635a9a9a29e81711afa59d0d"),
CustID: 'A150',
Amount: 250,
Status: 'pending'
},
{
_id: ObjectId("635a9aa629e81711afa59d0e"),
CustID: 'A150',
Amount: 300,
Status: 'pending'
},
{
_id: ObjectId("635a9aef29e81711afa59d0f"),
CustID: 'A201',
Amount: 750,
Status: 'pending'
}
]
```

9.Sort the customer according to CustID in descending order

```
customer> db.Customer.aggregate([{$sort:{'CustID':-1}}]).pretty()
[
{
_id: ObjectId("635a9aef29e81711afa59d0f"),
CustID: 'A201',
Amount: 750,
Status: 'pending'
},
{
_id: ObjectId("635a9a9a29e81711afa59d0d"),
CustID: 'A150',
Amount: 250,
Status: 'pending'
},
{
_id: ObjectId("635a9aa629e81711afa59d0e"),
CustID: 'A150',
Amount: 300,
Status: 'pending'
},
{
_id: ObjectId("635a9a7c29e81711afa59d0c"),
CustID: 'A124',
Amount: 200,
Status: 'completed'
},
```

```
{
_id: ObjectId("635a9a5d29e81711afa59d0b"),
CustID: 'A123',
Amount: 500,
Status: 'completed'
}
]
```

10. using skip operator find total amount of customer

```
customer> db.Customer.aggregate([{$skip:1},{$group:{_id:'$CustID',
'totalAmount':{$sum:'$Amount'}}}])
[
{ _id: 'A124', totalAmount: 200 },
{ _id: 'A150', totalAmount: 550 },
{ _id: 'A201', totalAmount: 750 }
]
```

11. using limit operator find total amount of customer

```
customer> db.Customer.aggregate([{$limit:2},{$group:{_id:'$CustID',
'totalAmount':{$sum:'$Amount'}}}])
[
{ _id: 'A123', totalAmount: 500 },
{ _id: 'A124', totalAmount: 200 }
]
```

12. using limit operator find first amount of customer

```
customer> db.Customer.aggregate([{$limit:4},{$group:{_id:'$CustID',
'totalAmount':{$first:'$Amount'}}}])
[
{ _id: 'A123', totalAmount: 500 },
{ _id: 'A124', totalAmount: 200 },
        { _id: 'A150', totalAmount: 250 }
        ]
        13. using limit operator find last amount of customer
        customer> db.Customer.aggregate([{$limit:4},{$group:{_id:'$CustID',
        'totalAmount':{$last:'$Amount'}}}])
        [
        { _id: 'A123', totalAmount: 500 },
        { _id: 'A124', totalAmount: 200 },
        { _id: 'A150', totalAmount: 300 }
        ]
```

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 require MongoDB 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 the index.

1. Ensure Index

```
customer> db.Customer.ensureIndex({'CustID':1})
[ 'CustID_1' ]
```

2.Find Indexes

```
customer> db.Customer.getIndexes()
[
{ v: 2, key: { _id: 1 }, name: '_id_' },
{ v: 2, key: { CustID: 1 }, name: 'CustID_1' }
]
```

3. Create Index

```
customer> db.Customer.createIndex({'Amount':-1})
Amount_-1
customer> db.Customer.getIndexes()
[
{ v: 2, key: { _id: 1 }, name: '_id_' },
{ v: 2, key: { CustID: 1 }, name: 'CustID_1' },
{ v: 2, key: { Amount: -1 }, name: 'Amount_-1' }
]
```

4. Drop Index

```
customer> db.Customer.dropIndex({'Amount':-1})
{ nIndexesWas: 3, ok: 1 }
customer> db.Customer.getIndexes()
[
{ v: 2, key: { _id: 1 }, name: '_id_' },
{ v: 2, key: { CustID: 1 }, name: 'CustID_1' }
]
```

5. sort indexes

a. Descending

```
customer> db.Customer.find().sort({'CustID':-1})
[
{
_id: ObjectId("635a9aef29e81711afa59d0f"),
CustID: 'A201',
Amount: 750,
Status: 'pending'
},
{
_id: ObjectId("635a9aa629e81711afa59d0e"),
CustID: 'A150',
Amount: 300,
Status: 'pending'
},
{
_id: ObjectId("635a9a9a29e81711afa59d0d"),
CustID: 'A150',
Amount: 250,
Status: 'pending'
},
{
```

```
_id: ObjectId("635a9a7c29e81711afa59d0c"),


CustID: 'A124',
Amount: 200,
Status: 'completed'
},
{
_id: ObjectId("635a9a5d29e81711afa59d0b"),
CustID: 'A123',
Amount: 500,
Status: 'completed'
}
]
```

b. Ascending

```
customer> db.Customer.find().sort({'CustID':1})
[
{
_id: ObjectId("635a9a5d29e81711afa59d0b"),
CustID: 'A123',
Amount: 500,
Status: 'completed'
},
{
_id: ObjectId("635a9a7c29e81711afa59d0c"),
CustID: 'A124',
Amount: 200,
Status: 'completed'
},
{
_id: ObjectId("635a9a9a29e81711afa59d0d"),
CustID: 'A150',
Amount: 250,
Status: 'pending'
},
{
_id: ObjectId("635a9aa629e81711afa59d0e"),
CustID: 'A150',
Amount: 300,
Status: 'pending'
},
{
_id: ObjectId("635a9aef29e81711afa59d0f"),
CustID: 'A201',
Amount: 750,
Status: 'pending'
}
]
```

Note:
createIndex() used to create indexes on collections whereas ensureIndex() creates an index on the specified field if the index does not already exist. Moreover when we execute createIndex() twice the second execution will just fail whereas with ensureIndex() you can invoke it multiple times and it will not fail

Conclusion: Performed Aggregation and indexing operations in MongoDB.

Viva Question:

- What is Aggregation in MongoDB?
- What is indexing in MongoDB?
- List out the aggregation expression?
- What are the features of MongoDB?
- Write difference between RDBMS and MongoDB?

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	

Group B

Assignment No 12

Title of the Assignment: MongoDB – Map-reduces operations:

Implement Map reduces operation with suitable example using MongoDB**.**

Objective of the Assignment: To understand the concept of Map reduces operation.

Outcome: Students will be able to learn and understand concept Map reduces operation with examples.

Theory:

MongoDB - Map Reduce

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

MapReduce Command

Following is the 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

Program:

1. create database

```
test> use map
switched to db map
```

2. create collection

```
map> db.createCollection("books")
{ ok: 1 }
```

3. insert documents

```
map> db.books.insertOne({'name':'JAVA','pages':'100'})
{
acknowledged: true,
insertedId: ObjectId("635d28cbeb14641fd96c5c23")
}
map> db.books.insertOne({'name':'PYTHON','pages':'200'})
{
acknowledged: true,
insertedId: ObjectId("635d28dfeb14641fd96c5c24")
}
map> db.books.insertOne({'name':'XML','pages':'300'})
{
acknowledged: true,
insertedId: ObjectId("635d28eeeb14641fd96c5c25")
}
map> db.books.insertOne({'name':'C++','pages':'350'})
{
acknowledged: true,
insertedId: ObjectId("635d28ffeb14641fd96c5c26")
}
map> db.books.insertOne({'name':'JAVASCRIPT','pages':'250'})
{
acknowledged: true,
insertedId: ObjectId("635d2917eb14641fd96c5c27")
}
```

4. display documents

```
map> db.books.find().pretty()
[
{
_id: ObjectId("635d28cbeb14641fd96c5c23"),
name: 'JAVA',
pages: '100'
},
{
_id: ObjectId("635d28dfeb14641fd96c5c24"),
name: 'PYTHON',
pages: '200'
},
{
_id: ObjectId("635d28eeeb14641fd96c5c25"),
name: 'XML',
```

```
pages: '300'
},
{
_id: ObjectId("635d28ffeb14641fd96c5c26"),
name: 'C++',
pages: '350'
},
{
_id: ObjectId("635d2917eb14641fd96c5c27"),
name: 'JAVASCRIPT',
pages: '250'
}
]
```

5.write map function on books

```
map> var map = function()
{
var category;
if(this.pages>=250)
category='Big Books';
else category='Small Books';
emit(category,{name:this.name});
};
```

6. write reduce function on books

```
map> var reduce = function(key, values)
{
var sum =0;
values.forEach(function(doc)
{
sum = sum+1;
});
return{books:sum};};
```

7. write mapreduce function on books

```
map> var count = db.books.mapReduce(map,reduce, {out:"book_results"});
```

8. display the combined result

```
map> db[count.result].find()
[
{ _id: 'Big Books', value: { books: 3 } },
{ _id: 'Small Books', value: { books: 2 } }
]
```

Conclusion: Performed implementation of Map reduces operation.

Viva Question:

- What is MapReduce in MongoDB?
- What is indexing in MongoDB?
- Write reduce function on books?
- Write mapreduce function on books?
- how display the combined result?

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	