ASSIGNMENT 2

DDL (Data Definition Language): Used to define, create, and modify database objects.

1. Tables:

- Store data in rows and columns.
- Example:

```
CREATE TABLE Employee (emp_id INT PRIMARY KEY, emp_name VARCHAR(50) NOT NULL,
emp_age INT CHECK(emp_age > 18), emp_salary DECIMAL(10,2), dept_id INT);
```

2. Constraints:

- Enforce data integrity.
- Types:
- PRIMARY KEY: uniquely identifies rows
- FOREIGN KEY: maintains referential integrity
- UNIQUE: ensures unique values
- NOT NULL: column cannot be empty
- CHECK: enforces a condition
- DEFAULT: sets a default value
- Example:

```
ALTER TABLE Employee ADD CONSTRAINT fk_dept FOREIGN KEY(dept_id) REFERENCES
Department(dept_id);
```

3. Views:

- Virtual tables derived from other tables.
- Example:

```
CREATE VIEW Employee_View AS SELECT emp_name, emp_salary FROM Employee WHERE
emp_salary > 30000;
```

4. Indexes:

- Speed up data retrieval.
- Example:

```
CREATE INDEX idx_emp_name ON Employee(emp_name);
```

5. Sequences:

- Automatically generate numeric values.
- Example:

```
CREATE SEQUENCE emp_seq START WITH 1 INCREMENT BY 1;
```

- Usage:

```
INSERT INTO Employee(emp_id, emp_name) VALUES(emp_seq.NEXTVAL, 'John');
```

6. Synonyms:

- Alias for a database object.
- Example:

```
CREATE SYNONYM emp_syn FOR Employee;
SELECT * FROM emp_syn;
```

———

Part B – SQL DML Concepts

DML (Data Manipulation Language): Used to insert, update, delete, and retrieve data.

1. INSERT:

```
INSERT INTO Department VALUES (1, 'HR', 'Pune');
INSERT INTO Employee VALUES (101, 'Atharva', 20, 35000, 1);
```

2. SELECT:

- Simple: SELECT * FROM Employee;
- With condition: SELECT emp_name, emp_salary FROM Employee WHERE emp_salary > 30000;
- Aggregate: SELECT AVG(emp_salary) FROM Employee;
- Set operator: SELECT emp_name FROM Employee WHERE dept_id=1 UNION SELECT emp_name FROM Employee WHERE dept_id=2;

3. UPDATE:
UPDATE Employee SET emp_salary = emp_salary + 5000 WHERE dept_id = 1;
4. DELETE:
DELETE FROM Employee WHERE emp_age < 20;

———

Operators
- Arithmetic: +, -, *, /
- Comparison: =, <>, >, <, >=, <=
- Logical: AND, OR, NOT
- Set Operators: UNION, INTERSECT, MINUS (or EXCEPT)

———

Functions
- Numeric: ROUND(), CEIL(), FLOOR()
- String: UPPER(), LOWER(), CONCAT()
- Date: SYSDATE, TO_DATE(), MONTH()
- Aggregate: SUM(), AVG(), COUNT(), MAX(), MIN()

Example:
SELECT UPPER(emp_name), ROUND(emp_salary) FROM Employee;

———

Sample 10 SQL Queries
1. SELECT * FROM Employee;
2. SELECT emp_name, emp_salary FROM Employee WHERE emp_salary > 30000;
3. INSERT INTO Department VALUES(2, 'IT', 'Mumbai');
4. UPDATE Employee SET emp_salary = emp_salary + 2000 WHERE dept_id = 2;
5. DELETE FROM Employee WHERE emp_id = 105;
6. SELECT COUNT(*) FROM Employee;
7. SELECT dept_name, COUNT(emp_id) FROM Employee JOIN Department USING(dept_id) GROUP BY dept_name;
8. CREATE VIEW HighSalary AS SELECT emp_name, emp_salary FROM Employee WHERE emp_salary > 40000;
9. SELECT emp_name FROM Employee WHERE dept_id = 1 UNION SELECT emp_name FROM Employee WHERE dept_id = 2;
10. SELECT emp_name, ROUND(emp_salary, 0) FROM Employee;

Assignment 3

1️⃣ SQL Joins

Joins combine rows from multiple tables based on a related column.

- INNER JOIN – returns only matching rows.

Example:
SELECT emp_name, dept_name FROM Employee INNER JOIN Department ON Employee.dept_id = Department.dept_id;

- LEFT JOIN – returns all rows from left table and matching rows from right table.

Example:
SELECT emp_name, dept_name FROM Employee LEFT JOIN Department ON Employee.dept_id = Department.dept_id;

- RIGHT JOIN – returns all rows from right table and matching rows from left table.

Example:
SELECT emp_name, dept_name FROM Employee RIGHT JOIN Department ON Employee.dept_id = Department.dept_id;

- FULL OUTER JOIN – returns all rows from both tables.

Example:
SELECT emp_name, dept_name FROM Employee FULL OUTER JOIN Department ON Employee.dept_id = Department.dept_id;

- CROSS JOIN – returns Cartesian product of tables.

Example:
SELECT emp_name, dept_name FROM Employee CROSS JOIN Department;

- SELF JOIN – joins table with itself.

Example:
SELECT E1.emp_name AS Employee, E2.emp_name AS Manager FROM Employee E1 LEFT JOIN Employee E2 ON E1.manager_id = E2.emp_id;

———

2️⃣ Sub-Queries

A query inside another query.

- Single-row sub-query – returns one value.

SELECT emp_name FROM Employee WHERE emp_salary = (SELECT MAX(emp_salary) FROM Employee);

- Multiple-row sub-query – returns multiple values.

SELECT emp_name FROM Employee WHERE dept_id IN (SELECT dept_id FROM Department WHERE location='Pune');

- Correlated sub-query – inner query depends on outer query.

SELECT emp_name FROM Employee E WHERE emp_salary > (SELECT AVG(emp_salary) FROM Employee WHERE dept_id = E.dept_id);

———

3️⃣ Views

- Virtual table based on a SELECT query.
- Simplifies complex queries and improves security.

Example:
CREATE VIEW HighSalary AS SELECT emp_name, emp_salary FROM Employee WHERE emp_salary > 40000;
SELECT * FROM HighSalary;

———

4️⃣ 10 Sample SQL Queries (DML + Joins + Sub-Queries + View)

1. SELECT emp_name, dept_name FROM Employee INNER JOIN Department ON Employee.dept_id = Department.dept_id;
2. SELECT emp_name, dept_name FROM Employee LEFT JOIN Department ON Employee.dept_id = Department.dept_id;
3. SELECT emp_name, dept_name FROM Employee RIGHT JOIN Department ON Employee.dept_id = Department.dept_id;
4. SELECT emp_name, dept_name FROM Employee FULL OUTER JOIN Department ON Employee.dept_id = Department.dept_id;
5. SELECT E1.emp_name AS Employee, E2.emp_name AS Manager FROM Employee E1 LEFT JOIN Employee E2 ON E1.manager_id = E2.emp_id;
6. SELECT emp_name FROM Employee WHERE emp_salary = (SELECT MAX(emp_salary) FROM Employee);
7. SELECT emp_name FROM Employee WHERE dept_id IN (SELECT dept_id FROM Department WHERE location='Pune');
8. SELECT emp_name FROM Employee E WHERE emp_salary > (SELECT AVG(emp_salary) FROM Employee WHERE dept_id = E.dept_id);
9. CREATE VIEW HighSalary AS SELECT emp_name, emp_salary FROM Employee WHERE emp_salary > 40000;
10. SELECT * FROM HighSalary;

Assignment 4

1. PL/SQL Overview

PL/SQL is a procedural extension of SQL. It allows combining SQL queries with procedural constructs like loops, conditions, and exception handling. A PL/SQL block consists of three main parts:

- DECLARE section: for declaring variables, constants, and user-defined exceptions.
- BEGIN section: contains executable statements like calculations, conditional logic, loops, and SQL operations.
- EXCEPTION section: handles runtime errors gracefully.

PL/SQL is useful because it allows automation of tasks, error handling, and batch processing of SQL statements.

———

2. Control Structures in PL/SQL

- IF-ELSE: Executes statements based on a condition. It is used when different actions are required depending on certain logical conditions.
- ELSIF: Provides an alternative condition if the first condition fails.
- ELSE: Executes a default action when none of the conditions are met.
- LOOPS: PL/SQL supports FOR loops and WHILE loops. FOR loops are useful for repeating a task a known number of times, while WHILE loops continue until a certain condition is false.
- Importance: Control structures allow decision-making and repetition in procedural tasks, such as calculating fines based on the number of days or calculating areas for multiple values.

———

3. Exception Handling in PL/SQL

- Purpose: To manage errors that occur during the execution of PL/SQL code.
- Predefined exceptions: Examples include NO_DATA_FOUND, TOO_MANY_ROWS, VALUE_ERROR. These are automatically raised by PL/SQL when specific runtime conditions occur.
- User-defined exceptions: Programmers can create their own exceptions to handle specific application conditions, such as invalid input or logical errors.
- Usage: Exception handling ensures that the program does not terminate abruptly and provides meaningful messages to the user.

———

4. Library Fine Calculation (Conceptual)

In a library scenario:

- The system accepts the borrower's roll number and the book name.
- The number of days since the book was issued is calculated.
- If the number of days is between 15 and 30, a fine of Rs 5 per day is applied.
- If the number of days is greater than 30, the fine is Rs 50 per day.
- If the number of days is less than 15, the fine is Rs 5 per day.
- When a book is returned, the status is updated from issued to returned.
- If a fine is applicable, the details are recorded in the fine system.
- All operations are safeguarded using exception handling to manage missing or invalid records and other runtime errors.

———

5. Circle Area Calculation (Conceptual)

- The radius varies from 5 to 9.

- For each radius value, the area of the circle is calculated using the formula for area.
- The radius and calculated area are stored conceptually in a results structure or log.
- Exception handling is applied to catch unexpected errors during calculation or storage.

———

6. Important Viva Questions and Answers

1. What is PL/SQL?

PL/SQL is a procedural extension of SQL that supports variables, loops, conditions, and exception handling.

2. What are the sections of a PL/SQL block?

DECLARE, BEGIN, and EXCEPTION.

3. What is the purpose of control structures in PL/SQL?

To execute statements conditionally or repeatedly, such as calculating fines or performing repeated calculations.

4. What is an exception in PL/SQL?

An exception is an error condition during execution that can be handled to prevent program termination.

5. What are user-defined exceptions?

Exceptions created by the programmer to handle specific conditions not covered by predefined exceptions.

6. What is the purpose of exception handling?

To catch errors gracefully and provide meaningful messages without terminating the program abruptly.

7. How is a fine calculated in a library scenario?

Based on the number of days since issue: a lower rate for 1–15 days, higher for 15–30, and maximum for above 30 days.

8. How is the status of a book updated?

Conceptually, the status changes from issued to returned after submission.

9. Why use loops for area calculation?

Loops allow calculation for multiple radius values efficiently without repeating code manually.

10. How do you ensure calculations or updates only happen when conditions are met?

By using conditional statements within PL/SQL, such as IF-ELSE blocks, and validating inputs before performing operations.

Assignment 5

PL/SQL Named Block – Stored Procedure and Function

1. Overview of Stored Procedures and Functions

- Stored Procedure:

A named PL/SQL block that performs a specific task. Procedures can accept parameters, perform operations, and optionally return results through OUT parameters. Procedures do not return a single value like functions but can modify data in the database.

- Stored Function:

A named PL/SQL block that performs a task and returns a single value. Functions are useful when a calculated or derived value is needed in SQL statements or PL/SQL code.

- Advantages:
- Reusability: Procedures and functions can be called multiple times.
- Modularity: Break complex tasks into smaller units.
- Maintainability: Updates to the procedure or function automatically reflect wherever it is used.
- Performance: Stored in the database, reducing network traffic and execution time.

———

2. Problem Requirement – Student Categorization

- The task is to categorize students based on marks.
- Rules for classification:
- Marks between 990 and 1500 → Distinction
- Marks between 900 and 989 → First Class
- Marks between 825 and 899 → Higher Second Class
- Marks below 825 → No specific category mentioned (can be considered Pass or Fail)
- The procedure is named proc_Grade and accepts student name and total marks as input.
- The procedure determines the category and stores the result (Roll, Name, Class) in the result structure.
- A PL/SQL block calls this procedure for multiple students and handles any exceptions that may occur, such as invalid marks or missing student names.

———

3. Conceptual Steps in the Procedure

1. Accept input parameters: student name and total marks.
2. Check marks against defined ranges.
3. Determine the grade category: Distinction, First Class, Higher Second Class.
4. Store the results in the result structure with student roll number, name, and assigned category.
5. Handle exceptions, for example, if marks are invalid (less than 0 or greater than maximum) or name is missing.
6. Output a confirmation message indicating that the grade has been assigned successfully.

———

4. Conceptual Use of Stored Procedure in PL/SQL Block

- A PL/SQL block is written to:

1. Call the stored procedure for each student.
2. Pass the student's name and total marks as parameters.
3. Catch and display any errors using exception handling.
4. Display confirmation messages for successful grade assignment.

- Benefits of Using Procedure in Block:
- Avoids rewriting categorization logic multiple times.
- Ensures consistent grading across all students.

- Makes it easy to update grading rules in a single procedure without changing the calling code.

---

5. Exception Handling Considerations
- Handle invalid marks: If marks are negative or exceed maximum possible, raise a custom exception.
- Handle missing student name: If the name is null, raise an exception to indicate invalid input.
- Use generic error handling to catch unforeseen runtime errors.
- Output descriptive messages for all exceptions so the user knows what went wrong.

---

6. Viva Questions and Answers – Named PL/SQL Block

1. Q: What is a stored procedure?
A: A named PL/SQL block that performs a task and can accept parameters, but does not return a single value like a function.

2. Q: What is a stored function?
A: A named PL/SQL block that performs a task and returns a single value. Functions can be used in SQL statements.

3. Q: Difference between procedure and function?
A: Functions return a single value and can be used in SQL, procedures do not return values but can modify data.

4. Q: Why use a stored procedure for student grading?
A: To ensure reusability, modularity, maintainability, and consistent grading logic.

5. Q: What are named exceptions?
A: Programmer-defined exceptions to handle specific conditions like invalid marks or missing student name.

6. Q: How do you determine the category of a student?
A: Compare marks with predefined ranges and assign Distinction, First Class, or Higher Second Class accordingly.

7. Q: Can procedures handle multiple students at once?
A: Yes, by calling the procedure multiple times in a PL/SQL block or using loops.

8. Q: What happens if invalid input is provided?
A: The procedure raises an exception which is caught in the PL/SQL block, and a descriptive error message is displayed.

9. Q: Why separate logic into a procedure instead of writing directly in the PL/SQL block?
A: It improves code reusability, readability, and simplifies maintenance.

10. Q: How can the procedure be updated if grading rules change?
A: Only the procedure needs to be modified; all calling blocks automatically use the updated logic.

Assignment 6

PL/SQL Cursors – Plain Text Explanation

1. Overview of Cursors
- A cursor in PL/SQL is a pointer to a result set of a query.
- Cursors are used to process multiple rows returned by a query one at a time.
- There are four types of cursors:

1. Implicit Cursor
- Automatically created by PL/SQL for single-row queries like INSERT, UPDATE, DELETE, or SELECT INTO.
- Developers do not need to explicitly declare it.
- Useful for operations that return exactly one row.
2. Explicit Cursor
- Declared by the programmer to handle queries that return multiple rows.
- Allows opening, fetching rows one by one, and closing the cursor explicitly.
3. Cursor FOR Loop
- A simplified version of explicit cursor.
- Automatically opens, fetches, and closes the cursor while looping over each row.
4. Parameterized Cursor
- A type of explicit cursor that accepts parameters to filter data dynamically.
- Useful when the same cursor logic needs to operate on different subsets of data.

———

2. Conceptual Problem – Merging Roll Call Data
- Goal: Merge data from N_RollCall (new data) into O_RollCall (existing data).
- Rule:
- If a record from N_RollCall already exists in O_RollCall, it should be skipped.
- Only non-duplicate records should be inserted.
- Solution Approach Using Parameterized Cursor:

1. Declare a parameterized cursor that accepts criteria to identify each row from the new data.
2. Open the cursor to fetch each row from N_RollCall.
3. For each row fetched, check if the same data exists in O_RollCall.
4. If it does not exist, insert the row into O_RollCall.
5. If it exists, skip to the next row.
6. Close the cursor after all rows are processed.
- Why parameterized cursor?
- It allows passing filter criteria dynamically for each record.
- Reduces redundancy and allows processing a dynamic set of rows efficiently.

———

3. Conceptual Steps for PL/SQL Block
1. Declare a parameterized cursor to select new records from N_RollCall.
2. Open the cursor to start fetching rows one by one.
3. Loop through each row using either a cursor FOR loop or explicit FETCH.
4. Check for existence in O_RollCall for the current row.
5. Insert the row into O_RollCall if it does not exist.
6. Skip duplicates automatically.
7. Close the cursor after processing all rows.
8. Handle exceptions for runtime errors like duplicate keys, null values, or invalid data.

———

4. Key Points to Remember

- Cursors are necessary for row-by-row processing of multiple rows in PL/SQL.
- Implicit cursors are simple but limited to single-row operations.
- Explicit and parameterized cursors provide flexibility and control.
- Cursor FOR loops simplify the code because opening, fetching, and closing are handled automatically.
- Parameterized cursors are particularly useful when you need dynamic filtering of data.
- Always include exception handling to manage errors gracefully.

———

5. Important Viva Questions and Answers

1. Q: What is a cursor in PL/SQL?

A: A cursor is a pointer to a result set of a query that allows processing rows one by one.

2. Q: Difference between implicit and explicit cursor?

A: Implicit cursors are automatically created for single-row queries, explicit cursors are programmer-defined for multiple-row queries.

3. Q: What is a parameterized cursor?

A: An explicit cursor that accepts parameters to filter or customize the data it processes.

4. Q: Why use a cursor instead of a single SQL statement?

A: To process multiple rows one at a time, especially when conditional logic or row-level checks are needed.

5. Q: What is a cursor FOR loop?

A: A simplified loop structure that automatically opens, fetches, and closes a cursor while iterating over each row.

6. Q: How can duplicates be avoided when merging data?

A: By checking the existence of each row before inserting and skipping duplicates in the cursor loop.

7. Q: When should you use a parameterized cursor?

A: When the same logic must operate on different subsets of data, or dynamic filtering is required.

8. Q: How do you handle exceptions with cursors?

A: Use the EXCEPTION block to catch errors like NO_DATA_FOUND, DUP_VAL_ON_INDEX, or other runtime errors.

9. Q: What is the advantage of using cursors over multiple SQL statements?

A: Cursors allow efficient, row-by-row processing and centralized logic for complex operations.

10. Q: Can a cursor update or delete data?

A: Yes, inside a loop, after fetching rows, you can perform UPDATE or DELETE operations conditionally.

Ass 7

1. Overview of Triggers

- A trigger is a stored PL/SQL code that is automatically executed (fired) in response to certain events on a table or view.
- Triggers are used to enforce business rules, audit changes, maintain integrity, and automate tasks.
- Key characteristics of triggers:
- Executed automatically when the event occurs.
- Cannot be called directly by users.
- Can affect multiple tables or maintain logs of changes.

———

2. Types of Triggers

1. Based on Timing
   - Before Trigger: Executes before the triggering event (INSERT, UPDATE, DELETE). Useful for validation or modifying data before it is saved.
   - After Trigger: Executes after the triggering event. Useful for logging changes, auditing, or performing dependent actions.
2. Based on Scope
   - Row-Level Trigger: Executes once for each row affected by the event. Used when you need to act on individual rows.
   - Statement-Level Trigger: Executes once for the entire SQL statement, regardless of the number of rows affected. Used for bulk actions or aggregate operations.

———

3. Conceptual Problem – Library Audit Trigger

- Goal: Track all changes (updates or deletions) in the Library table.
- Requirements:

1. Whenever a record in Library is updated or deleted, the old values should be captured.
2. Old values should be stored in a separate audit structure (Library_Audit) for tracking purposes.
3. The trigger should automatically fire without manual intervention.

- Solution Approach (Conceptual):
- Create row-level triggers for UPDATE and DELETE on the Library table.
- Use a before or after trigger depending on requirement:
- Before trigger can be used if you want to validate or transform data before the change.
- After trigger is commonly used for auditing, since the operation has already occurred.
- Store all old values, along with relevant metadata such as the date of change and type of operation, in Library_Audit.
- Ensure that exceptions are handled to avoid disrupting normal database operations.

———

4. Conceptual Steps for the Trigger

1. Identify the events to track: UPDATE and DELETE.
2. Decide the scope: row-level (to capture each changed row).
3. Capture the old values before they are overwritten or deleted.
4. Insert the old values along with metadata into the audit tracking structure (Library_Audit).

5. Optionally, include user information, timestamp, or operation type for better tracking.
6. Handle exceptions to ensure that any errors in logging do not prevent the original operation.

---

5. Key Points to Remember
- Before vs After Triggers:
- Before triggers can modify the data before the operation.
- After triggers are ideal for logging and auditing, because the original operation has completed.
- Row-Level vs Statement-Level:
- Row-level triggers act on each row individually, perfect for detailed auditing.
- Statement-level triggers act once per SQL statement, suitable for bulk operations.
- Advantages of Using Triggers:
- Automates auditing and enforcement of business rules.
- Ensures data integrity without modifying application code.
- Reduces manual intervention and errors.

---

6. Important Viva Questions and Answers
1. Q: What is a trigger in PL/SQL?
A: A trigger is a stored PL/SQL block that automatically executes in response to certain events like INSERT, UPDATE, or DELETE.
2. Q: What are the types of triggers based on timing?
A: Before triggers and After triggers.
3. Q: What are the types of triggers based on scope?
A: Row-level triggers and Statement-level triggers.
4. Q: When should you use a row-level trigger?
A: When you need to act on each individual row affected by the operation.
5. Q: When should you use a statement-level trigger?
A: When the action should occur once per SQL statement, regardless of the number of rows affected.
6. Q: What is the purpose of auditing with triggers?
A: To automatically capture old values of updated or deleted records for tracking and accountability.
7. Q: Why use an after trigger for auditing?
A: Because it fires after the operation, ensuring that the change has been applied and can be logged safely.
8. Q: Can a trigger prevent an operation from happening?
A: Yes, a before trigger can raise an exception to prevent invalid data changes.
9. Q: How do triggers handle multiple rows in bulk operations?
A: Row-level triggers execute for each row, while statement-level triggers execute only once for the entire statement.
10. Q: What precautions should be taken when using triggers?
A: Ensure that triggers do not cause recursive loops, maintain performance, and handle exceptions to avoid blocking normal operations.

Ass 8

1. Overview

Database connectivity refers to the process of connecting a front-end application to a database system (like MySQL or Oracle) so that the application can store, retrieve, update, and delete data. This allows creating interactive applications with a persistent backend.

- Front-end language examples: Java, Python, C#, PHP, JavaScript.
- Back-end database examples: MySQL, Oracle, SQL Server.
- Purpose: Enable applications to perform CRUD operations (Create, Read, Update, Delete) on a database.

———

2. Steps for Database Connectivity

1. Load Database Driver:
   - The front-end program needs to load the specific database driver for MySQL or Oracle.
   - The driver acts as a bridge between the application and the database.
2. Establish Connection:
   - Use a connection string or URL containing the database server address, port, database name, username, and password.
   - A connection object is created which allows communication with the database.
3. Create Statements:
   - Use SQL statements (SELECT, INSERT, UPDATE, DELETE) to perform operations on the database.
   - Statements can be simple statements or prepared statements for dynamic data.
4. Execute SQL Statements:
   - Execute queries using the connection object.
   - For SELECT queries, a result set is returned, which can be navigated row by row.
   - For INSERT, UPDATE, DELETE, the number of affected rows is returned.
5. Handle Exceptions:
   - Use exception handling to manage errors such as invalid SQL syntax, connection failure, or data type mismatch.
6. Close Connection:
   - Always close the connection after operations to free resources and avoid memory leaks.

———

3. CRUD Operations Concept

1. Create (Add Data):
   - Insert new records into the database table.
   - Ensure data is validated before insertion.
2. Read (View Data):
   - Retrieve records using SELECT statements.
   - Use loops or navigation mechanisms to display data row by row.
3. Update (Edit Data):
   - Modify existing records based on specific conditions.
   - Ensure that only the intended rows are updated to prevent data loss.
4. Delete (Remove Data):
   - Remove records based on specific criteria.
   - Typically, a confirmation mechanism is used to prevent accidental deletion.

———

4. Best Practices

- Use prepared statements to prevent SQL injection.
- Always close connections and resources after use.

- Implement exception handling to manage errors gracefully.
- Validate data on the front-end before sending it to the database.
- Use transaction management for multiple operations to ensure data integrity.

---

5. Conceptual Example Flow
- A front-end program accepts input from the user, such as student details.
- The program connects to the database using a driver and connection URL.
- The program executes an INSERT statement to add new records.
- The program executes a SELECT statement to display all records.
- The program executes an UPDATE statement to edit existing records.
- The program executes a DELETE statement to remove records.
- All operations are logged or displayed to the user, and exceptions are handled.
- The connection is closed after all operations.

---

6. Important Viva Questions and Answers

1. Q: What is database connectivity?

A: It is the process of linking a front-end application to a database to perform CRUD operations.

2. Q: What is the purpose of a database driver?

A: The driver acts as a bridge between the application and the database, enabling communication.

3. Q: What is the difference between a statement and a prepared statement?

A: A statement is used for static SQL queries, while a prepared statement allows dynamic parameters and prevents SQL injection.

4. Q: What are CRUD operations?

A: Create (insert), Read (select), Update (edit), and Delete (remove) data from the database.

5. Q: Why is exception handling important in database connectivity?

A: To handle errors like connection failure or invalid queries and ensure the application does not crash.

6. Q: How do you navigate records in a result set?

A: By using loops to iterate through each row of the result set.

7. Q: Why close the database connection after use?

A: To free resources, prevent memory leaks, and maintain database performance.

8. Q: What is the role of transaction management?

A: To ensure multiple database operations either succeed together or fail together, maintaining data integrity.

9. Q: Can front-end validation replace database validation?

A: No, front-end validation improves user experience, but database validation ensures actual data integrity.

10. Q: Give examples of front-end languages for database connectivity.

A: Java, Python, C#, PHP, JavaScript (Node.js).

Ass. 9

1. Overview of MongoDB

- MongoDB is a NoSQL document-oriented database.
- Stores data in JSON-like documents called BSON.
- Does not require a fixed schema, allowing flexible and scalable data storage.
- Commonly used for modern applications like web apps, real-time analytics, and big data solutions.

———

2. CRUD Operations in MongoDB

CRUD stands for Create, Read, Update, and Delete. These are the basic operations used to manipulate documents in MongoDB collections.

1. Create (Insert/Save)
   - Adds new documents to a collection.
   - The save method can be used to insert a new document or overwrite an existing document with the same identifier.
   - Allows adding multiple documents at once or a single document.
   - Flexible data structure allows storing documents with different fields in the same collection.
2. Read (Find/Query)
   - Retrieves documents from a collection.
   - Queries can filter documents using logical operators like AND, OR, NOT.
   - Supports projection to select specific fields.
   - Can sort and limit results.
   - Example query logic: Find all students with marks greater than 80 AND enrolled in a specific course.
3. Update
   - Modifies existing documents in a collection.
   - Can update a single document or multiple documents at once.
   - Supports operators like $set to modify fields, $inc to increment values, $push to add elements to arrays.
   - Conditional updates are possible using filters.
4. Delete
   - Removes documents from a collection.
   - Can delete a single document or multiple documents using conditions.
   - Logical operators help in selecting the exact documents to delete.

———

3. Use of Logical Operators

Logical operators are used in MongoDB queries to filter documents based on multiple conditions:

- $and: All conditions must be true.
- $or: At least one condition must be true.
- $not: Negates a condition.
- $nor: True only if none of the conditions are true.

These operators are used in combination with query filters to perform complex searches.

———

4. Conceptual Example Flow

- Suppose you have a student collection.
- Insert operation: Add new student documents with fields like name, roll number, course, and marks.

- Read operation: Retrieve students based on marks greater than 80 AND enrolled in “Computer Science”.
- Update operation: Increase marks for students in a specific course or update their course field.
- Delete operation: Remove students who have graduated or left the course.
- Save operation: Insert a new student or overwrite an existing student document with updated data.

Logical operators are used to ensure only the correct documents are retrieved, updated, or deleted.

———

5. Important Viva Questions and Answers

1. Q: What is MongoDB?

A: MongoDB is a NoSQL document-oriented database that stores data in flexible JSON-like documents.

2. Q: What is CRUD in MongoDB?

A: CRUD stands for Create, Read, Update, and Delete – the basic operations for managing documents.

3. Q: What is the difference between insert and save?

A: Insert adds new documents, while save can insert new documents or overwrite existing ones based on the document ID.

4. Q: What are logical operators in MongoDB?

A: Operators like AND, OR, NOT, NOR used to filter documents based on multiple conditions.

5. Q: How do you retrieve specific fields in a query?

A: By using projection to select only the required fields from the documents.

6. Q: Can MongoDB handle multiple documents at once?

A: Yes, bulk insert, update, and delete operations are supported.

7. Q: How do you update multiple documents at once?

A: Use an update operation with a filter condition and the multi-document option.

8. Q: Why use MongoDB instead of a relational database?

A: For flexible schema design, horizontal scalability, and high-performance handling of large and unstructured data.

9. Q: Can you delete documents conditionally?

A: Yes, delete operations can use filters and logical operators to remove only the selected documents.

10. Q: How does MongoDB ensure data integrity?

A: Through atomic operations on a single document and optional transactions for multi-document operations.

Ass 10

1. Overview of Map-Reduce

- Map-Reduce is a data processing paradigm used in MongoDB for aggregating large amounts of data.
- It allows performing complex aggregation and computation that might be difficult using standard aggregation pipelines.
- Map-Reduce consists of two main functions:

1. Map Function: Processes each document and emits one or more key-value pairs.
2. Reduce Function: Takes all values associated with the same key and reduces them to a single output.

- Map-Reduce can also include a finalize function to further process the reduced results.
- Typically used for large datasets or complex operations like word counts, data summarization, or computing statistics.

———

2. Conceptual Steps for Map-Reduce in MongoDB

1. Identify the requirement:
   - Example: Count the total marks scored by students in each course.
2. Define the Map function:
   - For each document, emit a key-value pair where the key is the course and the value is the marks.
3. Define the Reduce function:
   - Combine all marks for the same course and calculate totals or averages.
4. Optional Finalize function:
   - Process the reduced results further, for example, to calculate averages or percentages.
5. Execute Map-Reduce:
   - The output is stored in a collection or returned inline for analysis.

———

3. Conceptual Example

- Collection: students with fields name, course, marks.
- Task: Find total marks scored by each course using Map-Reduce.
- Steps:

1. Map Function: Emit course as the key and marks as the value for each student.
2. Reduce Function: Sum all marks for the same course.
3. The result gives total marks per course, which can be stored in a separate collection or viewed directly.

———

4. Use Cases of Map-Reduce

- Word count in documents or blogs.
- Calculating total or average sales per region or product.
- Generating complex reports across large datasets.
- Performing multi-step aggregations that are difficult with standard aggregation pipelines.

———

5. Advantages

- Handles very large datasets efficiently.
- Supports custom computation logic beyond standard aggregation.

- Can store results in a collection for further analysis.

---

6. Important Viva Questions and Answers

1. Q: What is Map-Reduce in MongoDB?

A: Map-Reduce is a programming paradigm for processing and aggregating large datasets in MongoDB using map and reduce functions.

2. Q: What are the two main functions in Map-Reduce?

A: Map function (emits key-value pairs) and Reduce function (combines values with the same key).

3. Q: What is the purpose of the finalize function?

A: To further process or format the results after reduction, such as calculating averages.

4. Q: When should Map-Reduce be used?

A: For complex aggregations or computations that are difficult to achieve with the aggregation pipeline.

5. Q: Can Map-Reduce handle large datasets?

A: Yes, it is designed to process very large amounts of data efficiently.

6. Q: How is Map-Reduce different from the aggregation pipeline?

A: Aggregation pipeline is faster and simpler for most standard aggregations, while Map-Reduce is more flexible for custom computations.

7. Q: What type of output can Map-Reduce produce?

A: Inline results or results stored in a separate collection for further use.

8. Q: Give a real-world example of Map-Reduce.

A: Counting the number of students in each course, total sales per region, or word frequency in a blog dataset.

9. Q: Can Map-Reduce results be indexed?

A: Yes, results stored in a collection can have indexes applied for faster querying.

10. Q: Why is Map-Reduce less commonly used now?

A: Because the aggregation pipeline is faster and easier for most operations, but Map-Reduce is still useful for highly customized computations.

Ass 11

MongoDB Map-Reduce Operations – Plain Text Explanation

1. Overview of Map-Reduce
- Map-Reduce is a data processing paradigm used in MongoDB for aggregating large amounts of data.
- It allows performing complex aggregation and computation that might be difficult using standard aggregation pipelines.
- Map-Reduce consists of two main functions:
1. Map Function: Processes each document and emits one or more key-value pairs.
2. Reduce Function: Takes all values associated with the same key and reduces them to a single output.
- Map-Reduce can also include a finalize function to further process the reduced results.
- Typically used for large datasets or complex operations like word counts, data summarization, or computing statistics.

———

2. Conceptual Steps for Map-Reduce in MongoDB
1. Identify the requirement:
- Example: Count the total marks scored by students in each course.
2. Define the Map function:
- For each document, emit a key-value pair where the key is the course and the value is the marks.
3. Define the Reduce function:
- Combine all marks for the same course and calculate totals or averages.
4. Optional Finalize function:
- Process the reduced results further, for example, to calculate averages or percentages.
5. Execute Map-Reduce:
- The output is stored in a collection or returned inline for analysis.

———

3. Conceptual Example
- Collection: students with fields name, course, marks.
- Task: Find total marks scored by each course using Map-Reduce.
- Steps:
1. Map Function: Emit course as the key and marks as the value for each student.
2. Reduce Function: Sum all marks for the same course.
3. The result gives total marks per course, which can be stored in a separate collection or viewed directly.

———

4. Use Cases of Map-Reduce
- Word count in documents or blogs.
- Calculating total or average sales per region or product.
- Generating complex reports across large datasets.
- Performing multi-step aggregations that are difficult with standard aggregation pipelines.

———

5. Advantages
- Handles very large datasets efficiently.
- Supports custom computation logic beyond standard aggregation.

- Can store results in a collection for further analysis.

---

6. Important Viva Questions and Answers

1. Q: What is Map-Reduce in MongoDB?

A: Map-Reduce is a programming paradigm for processing and aggregating large datasets in MongoDB using map and reduce functions.

2. Q: What are the two main functions in Map-Reduce?

A: Map function (emits key-value pairs) and Reduce function (combines values with the same key).

3. Q: What is the purpose of the finalize function?

A: To further process or format the results after reduction, such as calculating averages.

4. Q: When should Map-Reduce be used?

A: For complex aggregations or computations that are difficult to achieve with the aggregation pipeline.

5. Q: Can Map-Reduce handle large datasets?

A: Yes, it is designed to process very large amounts of data efficiently.

6. Q: How is Map-Reduce different from the aggregation pipeline?

A: Aggregation pipeline is faster and simpler for most standard aggregations, while Map-Reduce is more flexible for custom computations.

7. Q: What type of output can Map-Reduce produce?

A: Inline results or results stored in a separate collection for further use.

8. Q: Give a real-world example of Map-Reduce.

A: Counting the number of students in each course, total sales per region, or word frequency in a blog dataset.

9. Q: Can Map-Reduce results be indexed?

A: Yes, results stored in a collection can have indexes applied for faster querying.

10. Q: Why is Map-Reduce less commonly used now?

A: Because the aggregation pipeline is faster and easier for most operations, but Map-Reduce is still useful for highly customized computations.

Ass 12

MongoDB Database Connectivity with Front-End – Plain Text Explanation

1. Overview
- MongoDB connectivity allows a front-end application to interact with a MongoDB database to perform CRUD operations.
- Front-end applications can be built using languages like Java, Python, C#, PHP, or Node.js.
- Through connectivity, the application can add, edit, delete, or retrieve documents from MongoDB collections.

———

2. Steps for MongoDB Connectivity
1. Install MongoDB Driver:
- Use the official MongoDB driver for the chosen front-end language (e.g., pymongo for Python, mongo-java-driver for Java).
2. Establish Connection:
- Provide MongoDB server address, port, database name, and optional authentication.
- Create a connection object that allows operations on the database.
3. Select Database and Collection:
- Specify which database and collection the operations will apply to.
- Collections in MongoDB are analogous to tables in relational databases.
4. Perform CRUD Operations:
- Add (Insert): Insert new documents into a collection.
- Edit (Update): Modify existing documents based on specific conditions.
- Delete (Remove): Delete documents that match specific criteria.
- View (Read): Retrieve documents using queries and optionally sort or filter results.
5. Handle Exceptions:
- Catch errors such as connection failures, invalid queries, or data type mismatches.
- Ensure the application does not crash if a MongoDB operation fails.
6. Close Connection:
- Properly close the connection after operations to free resources.

———

3. Conceptual Example Flow
- Front-end program collects input from the user, such as student details.
- Connects to the MongoDB database using the driver and server URL.
- Add Operation: Insert a new student document with fields like name, roll number, course, and marks.
- Edit Operation: Update the course or marks of a student identified by roll number.
- Delete Operation: Remove a student document based on roll number.
- View Operation: Display all students or filter by course or marks.
- Connection is closed after completing all operations.

———

4. Best Practices
- Use parameterized operations or validation to prevent injection attacks.
- Always handle exceptions to maintain stability.
- Use indexes in MongoDB for frequently queried fields to improve performance.
- Close the database connection after operations to conserve resources.

———

5. Important Viva Questions and Answers

1. Q: What is MongoDB connectivity?

A: Connecting a front-end application to MongoDB to perform CRUD operations on collections.

2. Q: What is a collection in MongoDB?

A: A collection is a group of documents, similar to a table in relational databases.

3. Q: How do you insert a document into MongoDB?

A: Use the insert operation to add a new document with specified fields into a collection.

4. Q: How do you update a document in MongoDB?

A: Use the update operation with a filter condition to modify specific fields of existing documents.

5. Q: How do you delete documents in MongoDB?

A: Use the delete operation with a filter to remove documents that match specific criteria.

6. Q: How do you retrieve documents from MongoDB?

A: Use queries with optional filters, projections, sorting, and limits.

7. Q: Why is exception handling important in database connectivity?

A: To manage errors like connection failures or invalid queries without crashing the application.

8. Q: What front-end languages can connect to MongoDB?

A: Java, Python, C#, PHP, Node.js, and many others with official MongoDB drivers.

9. Q: How can performance be improved in MongoDB queries?

A: By creating indexes on frequently queried fields.

10. Q: What is the difference between relational and MongoDB connectivity?

A: Relational databases use structured tables and SQL, while MongoDB uses flexible JSON-like documents and queries.