CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | |

Internal Assesment Test –III, October 2024

Sub:	Database Management System	Code:	22MCA21

	Answer Any 5 **QUESTIONS**	Marks	OBE	
			CO	RBT
1)	Discuss various Transaction states with a neat diagram and its pros and cons	10	CO3	L2
	OR			
2)	Explain various types of failure that may occur in a system.	10	CO3	L2
3)	Explain types of problems that may occur when two simple transaction run concurrently with examples. Why concurrency control and recovery are needed in DBMS?	10	CO3	L1
	OR			
4)	Explain properties, characteristics, advantages and disadvantages of a transaction in detail.	10	CO3	L3
5)	Explain the select and project operation with syntax and examples.	10	CO4	L2
	OR			
6)	Explain Union, intersection and minus operation with examples	10	CO4	L3
7)	Consider the following COMPANY database EMP(Name,SSN,Salary,address, SuperSSN,Gender,Dno) DEPT(DNum,Dname,MgrSSN) PROJECT(Pname,Pnumber,Plocation,Dnum) Write the relational algebra queries for the following (i) Retrieve the name, address, salary of employees who work for the Research department. (ii) Find the names of employees who work on all projects controlled by department number 4. iii) Retrieve the SSN of all employees who either in department no :4 or directly supervise an employee who work in dno 4	10	CO4	L3
	OR			
8)	Explain briefly about database application development	10	CO4	L2
9)	Explain in detailed about i.)JDBC ii.) Cursors	10	CO4	L3
	OR			
10)	Explain in detailed about i.)Embedded SQL ii.) SQLJ	10	CO4	L3

1

Key Transaction States in DBMS

1. **Active State**:
   - **Description**: The transaction starts here, performing its operations (e.g., read/write).
   - **Pro**: Allows flexibility to perform all operations.
   - **Con**: No guarantee of completion; may end abruptly if an error occurs.
2. **Partially Committed State**:
   - **Description**: Transaction has completed all operations but not yet committed.
   - **Pro**: Ensures that checks can occur before commitment.
   - **Con**: Can lead to a failed state if checks fail.
3. **Committed State**:
   - **Description**: Changes are saved permanently; the transaction is successful.
   - **Pro**: Guarantees durability of transaction results.
   - **Con**: No changes can be reverted without a new transaction.
4. **Failed State**:
   - **Description**: An error occurred; transaction cannot continue.
   - **Pro**: Prevents inconsistent data from entering the database.
   - **Con**: Requires re-execution if a failure occurs due to recoverable issues.
5. **Aborted State**:
   - **Description**: Transaction has been rolled back, undoing all changes.
   - **Pro**: Restores database consistency by discarding changes.
   - **Con**: May involve performance overhead during rollback.

2

Failure Type Impact Recovery Mechanism

Transaction Failure: Incomplete transaction Rollback using logs

System Failure Lost in-memory data Checkpoints, transaction logs

Media Failure Permanent data loss Restore from backups or replication

Application Failure Inconsistent or incomplete transactions Rollback incomplete transactions

Network Failure Distributed transaction inconsistency Distributed recovery protocols

Buffer Overflow Possible data corruption Memory management adjustments, rollback

3

1. Lost Update Problem

- **Description**: Occurs when two transactions update the same data item simultaneously, causing one update to overwrite the other.
- **Example**:
  - Assume a bank account balance is initially $500.
  - Transaction T1T_1T1 reads the balance, adds $100, and sets it to $600.
  - Before T1T_1T1 writes $600 back, Transaction T2T_2T2 reads the same balance of $500, subtracts $50, and sets it to $450.
  - Both transactions complete, but T1T_1T1's update to $600 is overwritten by T2T_2T2's update to $450, causing the addition of $100 to be "lost."

2. Dirty Read (Uncommitted Dependency Problem)

- **Description**: Occurs when a transaction reads data modified by another transaction that has not yet been committed. If the modifying transaction rolls back, the reading transaction has read invalid data.
- **Example**:
  - T1T_1T1 updates an account balance from $500 to $600 but does not commit.
  - T2T_2T2 reads this uncommitted balance of $600 and uses it for calculations.
  - If T1T_1T1 rolls back, the original balance of $500 is restored, but T2T_2T2 has already used the invalid $600 balance.

3. Unrepeatable Read (Inconsistent Retrieval)

- **Description**: Occurs when a transaction reads the same data multiple times and gets different results due to another transaction's updates.
- **Example**:
  - T1T_1T1 reads the balance of an account as $500.
  - T2T_2T2 updates the balance to $600 and commits.
  - T1T_1T1 reads the balance again and sees $600, which is inconsistent with the initial read.

4. Phantom Read

- **Description**: Occurs when a transaction reads a set of rows based on a condition, but another transaction inserts, deletes, or updates rows that affect the result of the original query.
- **Example**:
  - T1T_1T1 reads a list of accounts with a balance greater than $500, finding three accounts.
  - Meanwhile, T2T_2T2 inserts a new account with a balance of $700.
  - If T1T_1T1 re-reads the list, it will find four accounts, leading to inconsistencies.

Why Concurrency Control and Recovery are Needed in DBMS

1. **Concurrency Control**:
   - Concurrency control mechanisms manage the execution of transactions to ensure data integrity and isolation. They prevent conflicts like lost updates, dirty reads, unrepeatable reads, and phantom reads by coordinating access to shared data.
   - **Reason**: Concurrency control is essential to maintain the ACID properties (especially isolation and consistency) and avoid data inconsistencies. It enables multiple users to access the database concurrently without interference.
2. **Recovery**:
   - Recovery mechanisms restore the database to a consistent state in the event of system crashes, transaction failures, or hardware issues. This includes rolling back incomplete transactions or redoing committed ones.
   - **Reason**: Recovery mechanisms ensure atomicity and durability, allowing transactions to either complete entirely or leave no trace in case of a failure. This maintains database consistency and helps recover from unexpected issues.

4	A transaction in a Database Management System (DBMS) is a logical unit of work that includes one or more operations on the database (such as reading, writing, updating, and deleting data). It ensures data consistency, reliability, and integrity, especially in multi-user environments where several transactions might occur simultaneously. Here's a detailed explanation of its properties, characteristics, advantages, and disadvantages: **Properties of a Transaction (ACID Properties)** Transactions follow the **ACID** properties, which are essential to maintaining database integrity: 1. **Atomicity**: o **Definition**: A transaction is an atomic unit of operation, meaning it either fully completes or does not happen at all. If any part of the transaction fails, the entire transaction is rolled back. o **Explanation**: If a transaction involves multiple steps (e.g., transferring funds between accounts), either all steps must succeed, or none should take effect. o **Example**: In an e-commerce system, if a customer's payment fails, the entire transaction (including inventory update and order creation) should be reversed. 2. **Consistency**: o **Definition**: A transaction must bring the database from one consistent state to another. All integrity constraints must be satisfied both before and after the transaction. o **Explanation**: Database rules (like foreign keys or constraints) ensure that data remains valid. If any condition fails, the transaction will not complete. o **Example**: A transaction should not result in a negative balance if the system disallows negative account balances. 3. **Isolation**: o **Definition**: Transactions should be executed independently and should not interfere with each other. A transaction's intermediate states should not be visible to other transactions. o **Explanation**: This property prevents issues like dirty reads, unrepeatable reads, and lost updates, maintaining the accuracy of concurrent transactions. o **Example**: In a banking application, two simultaneous withdrawals from the same account should occur in isolation to avoid incorrect balance calculations. 4. **Durability**: o **Definition**: Once a transaction is committed, its changes to the database are permanent, even in the event of a system crash or failure. o **Explanation**: Durability ensures that the database maintains a record of committed transactions, usually through logging and backup mechanisms. o **Example**: If a power failure occurs right after a transaction is committed, the changes should remain when the system restarts. **Characteristics of Transactions** 1. **Consistency Preservation**: Transactions uphold database integrity constraints, ensuring consistency before and after each transaction. 2. **Concurrency Control**: Transactions are designed to operate in environments where multiple users may access the database			

simultaneously. Concurrency control techniques, such as locking or timestamp ordering, are used to maintain isolation.

3. **Recoverability**: A transaction should be able to be rolled back in case of failure, and committed transactions should be persistent. This helps maintain the atomicity and durability of transactions.
4. **Transparency**: The underlying database system handles transactions so that the user only sees the end result, ensuring a seamless user experience regardless of the complexity of the transaction.

Advantages of Transactions

1. **Data Integrity and Consistency**:
   - Transactions help maintain data accuracy and consistency, even in cases of simultaneous updates by multiple users. By enforcing the ACID properties, transactions help protect against data corruption.
2. **Error Handling and Recovery**:
   - Transactions make it easier to handle errors. If any part of a transaction fails, the entire transaction can be rolled back, preventing partial updates that could lead to inconsistencies.
3. **Concurrent Access**:
   - By providing isolation and using concurrency control, transactions allow multiple users to interact with the database simultaneously without conflicting with each other's operations.
4. **Simplifies Complex Operations**:
   - Transactions allow complex sequences of operations to be managed easily, enabling developers to define all or nothing operations that make application development simpler and more reliable.
5. **Guaranteed Durability**:
   - Transactions ensure that once a change is committed, it is permanent. This provides reliability for users, who can trust that completed transactions will not be lost due to system failures.

Disadvantages of Transactions

1. **Performance Overhead**:
   - Implementing transaction management, especially isolation and durability, adds overhead, as mechanisms like locking, logging, and checkpoints are resource-intensive. This can slow down system performance, especially under high concurrency.
2. **Deadlocks**:
   - Concurrency control mechanisms (like locking) can lead to deadlocks, where transactions wait indefinitely for each other to release resources, requiring additional management techniques to resolve.
3. **Increased Complexity**:
   - The algorithms required to manage transactions and enforce the ACID properties are complex, making the DBMS more challenging to implement and maintain, which can increase development time and cost.
4. **Resource Consumption**:
   - Transactions require additional resources, such as memory and storage, to manage logs, backups, and checkpoints. In high-volume transaction environments, this can strain system resources and impact scalability.
5. **Reduced Parallelism**:
   - High levels of isolation (like serializability) restrict parallel

execution of transactions, impacting performance in systems with large numbers of concurrent users.

5 The **Select** and **Project** operations are fundamental operations in relational algebra used to manipulate and query relational databases. Here's a detailed explanation, including syntax and examples:

---

1. Select Operation (σ)

- **Purpose**: The select operation retrieves rows (tuples) from a relation (table) that satisfy a specified condition.
- **Symbol**: σ (sigma)
- **Syntax**:

  σcondition(Relation)\sigma_{\text{condition}}(\text{Relation})σcondition(Relation)

  Where `condition` is the criterion used to filter rows from `Relation`.

- **Example**: Suppose we have a relation (table) called `Employee` with the following schema and data:

EmpID	Name	Department	Salary
101	Alice	HR	5000
102	Bob	Sales	6000
103	Charlie	IT	7000
104	David	Sales	5500
105	Eve	IT	7500

- **Query**: Retrieve all employees in the Sales department.
- **Select Operation**:
- σDepartment = 'Sales'(Employee)\sigma_{\text{Department = 'Sales'}}(\text{Employee})σDepartment = 'Sales'(Employee)
- **Result**:

EmpID	Name	Department	Salary
102	Bob	Sales	6000
104	David	Sales	5500

- **Explanation**: The select operation filtered out only those rows where the department is 'Sales'.

---

2. Project Operation (π)

- **Purpose**: The project operation retrieves specific columns (attributes) from a relation, effectively creating a subset of columns in a new relation.
- **Symbol**: π (pi)
- **Syntax**:

  πcolumn1, column2, ...(Relation)\pi_{\text{column1, column2,

...}}(\text{Relation})πcolumn1, column2, ...(Relation)

Where `column1, column2, ...` are the attributes you want to retrieve from `Relation`.

- **Example**: Using the same `Employee` table, let's say we want to retrieve only the names and salaries of all employees.

  Project Operation:

  πName, Salary(Employee)\pi_{\text{Name, Salary}}(\text{Employee})πName, Salary(Employee)

  Result:

Name	Salary
Alice	5000
Bob	6000
Charlie	7000
David	5500
Eve	7500

  Explanation: The project operation selected only the `Name` and `Salary` columns, discarding other attributes.

6 In relational algebra, **Union**, **Intersection**, and **Minus** are set operations used to combine or differentiate between two relations (tables) with the same schema. Here's an explanation of each operation with examples.

---

1. Union Operation (∪)

- **Purpose**: Combines the results of two relations and returns all unique rows present in either or both relations.
- **Symbol**: ∪
- **Syntax**:

  Relation1∪Relation2\text{Relation1} \cup \text{Relation2}Relation1∪Relation2

  Both `Relation1` and `Relation2` must have the same schema (same number of columns with matching data types).

- **Example**: Suppose we have two tables `Employee_A` and `Employee_B`:

  Employee_A:

EmpID	Name
101	Alice
102	Bob
103	Charlie

Employee_B:

EmpID	Name
103	Charlie
104	David
105	Eve

Union Operation:

Employee_A∪Employee_B\text{Employee_A} \cup \text{Employee_B}Employee_A∪Employee_B

Result:

EmpID	Name
101	Alice
102	Bob
103	Charlie
104	David
105	Eve

Explanation: The union operation returns all unique rows from both tables, combining the data without duplicates.

2. Intersection Operation (∩)

- **Purpose**: Returns only the rows that are present in both relations.
- **Symbol**: ∩
- **Syntax**:

  Relation1∩Relation2\text{Relation1} \cap \text{Relation2}Relation1∩Relation2

  As with union, both `Relation1` and `Relation2` must have the same schema.

- **Example**: Using the same tables `Employee_A` and `Employee_B`:

  Intersection Operation:

  Employee_A∩Employee_B\text{Employee_A} \cap \text{Employee_B}Employee_A∩Employee_B

  Result:

EmpID	Name
103	Charlie

Explanation: The intersection operation returns only the rows that appear in both tables.

3. Minus Operation (−)

- **Purpose**: Returns rows that are present in the first relation but not in the second.
- **Symbol**: −
- **Syntax**:

  Relation1−Relation2\text{Relation1} - \text{Relation2}Relation1−Relation2

  As with the other two operations, both `Relation1` and `Relation2` must have the same schema.

- **Example**: Using the same tables `Employee_A` and `Employee_B`:

  Minus Operation:

  Employee_A−Employee_B\text{Employee_A} - \text{Employee_B}Employee_A−Employee_B

  Result:

EmpID	**Name**
101	Alice
102	Bob

  Explanation: The minus operation returns only the rows that are in `Employee_A` but not in `Employee_B`.

7.

- **Employees in Research Department**:

πName,address,Salary(σDno=DNum(EMP×DEPT)∧Dname = 'Research')\pi_{Name, address, Salary}(\sigma_{Dno = DNum}(\text{EMP} \times \text{DEPT}) \land \text{Dname = 'Research'})πName,address,Salary(σDno=DNum (EMP×DEPT)∧Dname = 'Research')

- **Employees working on all projects in department 4**:

πName(EMP) where SSN∈(WORKS_ON÷P4)\pi_{Name}(EMP) \text{ where } SSN \in (WORKS_ON \div P_4)πName(EMP) where SSN∈(WORKS_ON÷P4)

- **Employees in department 4 or supervising someone in department 4**:

EDept4∪SDept4E_{Dept4} \cup S_{Dept4}EDept4∪SDept4

8

- **Embedded SQL** is great for static queries in host languages.
- **Dynamic SQL** offers flexibility for runtime query generation.
- **JDBC** provides a standard API for Java applications to interact with databases.
- **Stored Procedures** encapsulate complex logic in the database, improving performance and security.

- **SQLJ** combines the benefits of SQL with Java, offering type safety and integration.

9 • **DBC Driver**: A software component that enables Java applications to interact with a specific database. JDBC drivers can be categorized into four types:

- **Type 1**: JDBC-ODBC Bridge Driver
- **Type 2**: Native-API Driver
- **Type 3**: Network Protocol Driver
- **Type 4**: Thin Driver (pure Java driver)

Type 4 drivers are commonly used because they are platform-independent and don't require any native libraries.

• **Connection Interface**: Establishes a connection to the database. This interface includes methods to create statements, manage transactions, and close the connection.

• **Statement Interface**: Used to execute SQL statements against the database. There are three types of statements:

- **Statement**: Used for executing simple SQL queries without parameters.
- **PreparedStatement**: Used for executing parameterized SQL queries, which helps prevent SQL injection and improves performance.
- **CallableStatement**: Used to execute stored procedures in the database.

• **ResultSet Interface**: Represents the result set of a query. It provides methods to retrieve data from the result set, navigate through the rows, and update data if applicable.

• **Cursors** play an essential role in database programming by enabling fine-grained control over the processing of result sets, making them useful for complex data manipulation tasks. However, their performance overhead necessitates careful consideration when designing database interactions.

```
DECLARE
   CURSOR emp_cursor IS SELECT * FROM employees;
   emp_record emp_cursor%ROWTYPE;
BEGIN
   OPEN emp_cursor;
   LOOP
      FETCH emp_cursor INTO emp_record;
      EXIT WHEN emp_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(emp_record.name || ' - ' || emp_record.salary);
   END LOOP;
   CLOSE emp_cursor;
END;
```

10 Embedded SQL allows the integration of SQL statements within host programming languages (like C, C++, or Java) to facilitate database interactions.

Key Features

- **Static SQL**: SQL statements are defined at compile time.
- **Preprocessing**: A preprocessor translates SQL into calls to a database interface.
- **Type Safety**: Compile-time checks ensure data type consistency.

Example (C)

c
Copy code

```
EXEC SQL SELECT name INTO :emp_name FROM employees WHERE id =
:emp_id;
```

Advantages

- **Performance**: Faster execution due to precompilation.
- **Security**: Helps prevent SQL injection attacks.
- **Error Checking**: Syntax errors are caught at compile time.

Disadvantages

- **Less Flexibility**: Hard to create dynamic queries.
- **Tight Coupling**: Links application logic closely to database schema.

II. SQLJ

Overview

SQLJ is an extension of SQL for Java, allowing SQL statements to be embedded directly in Java code while maintaining strong typing.

Key Features

- **Type Safety**: Compile-time checks for SQL statements and variables.
- **Precompilation**: A precompiler converts SQLJ into standard JDBC code.
- **Java Integration**: Combines Java syntax with SQL statements.

Example (Java)

```
java
Copy code
#sql {
    SELECT name INTO :empName FROM employees WHERE id = 1
};
```

Advantages

- **Type Safety**: Errors caught at compile time.
- **Ease of Use**: SQL statements are readable and integrated with Java.
- **Performance**: Optimized as SQLJ compiles to JDBC.

Disadvantages

- **Precompilation Step**: Requires an additional build step.
- **Learning Curve**: Developers need to learn SQLJ syntax and JDBC.