Course Code	**:**	**MCS-023**
Course Title	**:**	**Introduction to Database Management Systems**
Assignment Number	**:**	**BCA(III)/023/Assignment/2024-25**
Maximum Marks	**:**	**100**
Weightage	**:**	**25%**
Last Date of Submission	**:**	**31st October,2024(For July Session)** **30th April,2025(For January Session)**

This assignment has eight questions. Answer all questions of total 80 marks. Rest 20 marks are for viva voce. You may use illustrations and diagrams to enhance explanations. Please go through the guidelines regarding assignments given in the Programme Guide for the format of presentation. Answer to each part of the question should be confined to about 300 words.

Q1. **(2 Marks)**

a) What is SQL? Explain its important features.

b) Consider the following schemas:

BOOK (Book_ID, Title, Publisher_ID, Year_of Pub, Price)
AUTHOR (Author_ID, Book_ID, Author Name)
PUBLISHER (Publisher_ID, Book_ID, Address, Name_of Pub, No._of Copies)

Write a query in SQL for the following:

(i) Find the name of authors whose books are published by "ABC Press".
(ii) Find the name of the author and price of the book, whose Book_ID is '100'.
(iii) Find the title of the books which are published by Publisher_ID '20' and are published in year 2011.
(iv) Find the address of the publisher who has published Book_ID "500".

Make suitable assumptions, if any. **(8 Marks)**

Q2.

a) With the help of a suitable example, discuss the insertion, deletion and updation anomalies that can occur in a database. Briefly discuss the mechanism to remove such anomalies. **(6 Marks)**

b) Write SQL commands for each of the following. Also illustrate the usage of each command through suitable example. **(4 Marks)**

(i) Creation of views
(ii) Creation of sequences
(iii) Outer join
(iv) To give access permission to any user

Q3.

a) What are integrity constraints? Discuss the various types of integrity constraints that can be imposed on database. **(3 Marks)**

b) How are database security and database integrity related? Briefly discuss the different levels of security measures which may be considered to protect the database. **(3 Marks)**

c) Consider the relation R (A, B, C, D, E) and the set of functional dependencies :-
F(A $\rightarrow$ D, {A,B} $\rightarrow$ C, D $\rightarrow$ E)

Assume that the decomposition of R into {R1 (A, B, C) and R2 (A, D, E)}.
Is this decomposition lossless? Justify? **(4 Marks)**

Q4.

a) Explain the Log-based recovery scheme with the help of an example. **(5 Marks)**

b) Compute the closure of the following set F of functional dependencies for relation schema
R = (A, B, C, D, E).
$A \rightarrow BC$
$CD \rightarrow E$
$B \rightarrow D$
$E \rightarrow A$

List the candidate keys for R. **(5 Marks)**

Q5.

a) Give the limitations of file based system. How can they be overcome using DBMS? **(5 Marks)**

b) Discuss the importance of file organisation in databases. Mention the different types of file organisations available. Discuss any one of the mentioned file organisations in detail. **(5 Marks)**

Q6.

a) For what reasons is 2-phase locking protocol required? Explain. Discuss the disadvantages of basic 2-phase locking protocol. List the ways and means that can be used to overcome the disadvantages. **(5 Marks)**

b) List and explain the 4 basic properties of a Transaction with the help of appropriate examples. **(5 Marks)**

Q7.

a) What do you mean by fragmentation of a database? What is the need of fragmentation in DDBMS environment? Explain different types of fragmentation with an example of each. **(5 Marks)**

b) Explain the need of Distributed DBMS over Centralized DBMS. Also give the structure of Distributed DBMS. **(5 Marks)**

Q8. An organization needs to provide Medical facilities to its employees and their dependents. Organization is having a list of Doctors, Hospitals and Test centres for the employees facility. An employee may get Medical facility from the list of Doctors, Hospitals and Test centres provided by the organization to them. Employee does not need to pay anything for the facilities availed. The Doctors, Hospitals and Test centres directly raise their bill to the organization.

Identify the entities, relationships, constraints and cardinality and construct an ER diagram for the above mentioned specifications. List your assumptions and clearly indicate the cardinality mappings as well as any role indicators in your ER diagram. **(10 Marks)**

Q1. a) What is SQL? Explain its important features.

b) Consider the following schemas: BOOK

(Book_ID, Title, Publisher_ID, Year_of Pub, Price)

AUTHOR (Author_ID, Book_ID, Author Name)

PUBLISHER (Publisher_ID, Book_ID, Address, Name_of Pub, No._of Copies)

Write a query in SQL for the following:

- **(i) Find the name of authors whose books are published by "ABC Press".**
- **(ii) Find the name of the author and price of the book, whose Book_ID is '100'.**
- **(iii) Find the title of the books which are published by Publisher_ID '20' and are published in year 2011.**
- **(iv) Find the address of the publisher who has published Book_ID "500". Make suitable assumptions, if any.**

Ans

a) What is SQL? Explain its important features.

SQL (Structured Query Language) is a standard programming language specifically designed for managing and manipulating relational databases. SQL is used to query, insert, update, and delete data in database systems like MySQL, Oracle, SQL Server, and others.

Important features of SQL:

1. **Data Definition Language (DDL):** SQL provides commands like CREATE, ALTER, DROP, and TRUNCATE to define and manage the structure of the database, including tables, indexes, and relationships.
2. **Data Manipulation Language (DML):** SQL allows manipulation of the data within the database using commands like SELECT, INSERT, UPDATE, and DELETE.
3. **Data Control Language (DCL):** SQL includes commands such as GRANT and REVOKE that manage access to the data in the database.
4. **Transaction Control Language (TCL):** Commands like COMMIT, ROLLBACK, and SAVEPOINT are used to manage transactions in SQL, ensuring the integrity of the database.
5. **Query Capabilities:** SQL is designed to query the database and retrieve specific data using complex conditions and filters with commands like WHERE, GROUP BY, HAVING, and ORDER BY.
6. **Joins:** SQL can combine rows from two or more tables based on a related column between them using different types of joins like INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN.
7. **Aggregate Functions:** SQL includes functions like COUNT, SUM, AVG, MIN, and MAX to perform calculations on a set of values and return a single value.
8. **Scalability:** SQL is designed to handle large datasets efficiently, making it suitable for use in large-scale enterprise environments.

---

b) SQL Queries for the Given Schemas

Consider the following schemas:

- **BOOK** (Book_ID, Title, Publisher_ID, Year_of_Pub, Price)
- **AUTHOR** (Author_ID, Book_ID, Author_Name)
- **PUBLISHER** (Publisher_ID, Book_ID, Address, Name_of_Pub, No._of_Copies)

i) Find the name of authors whose books are published by "ABC Press".

```sql
SELECT A.Author_Name
FROM AUTHOR A
JOIN PUBLISHER P ON A.Book_ID = P.Book_ID
WHERE P.Name_of_Pub = 'ABC Press';
```

ii) Find the name of the author and price of the book, whose Book_ID is '100'.

```sql
SELECT A.Author_Name, B.Price
FROM AUTHOR A
JOIN BOOK B ON A.Book_ID = B.Book_ID
WHERE B.Book_ID = '100';
```

iii) Find the title of the books which are published by Publisher_ID '20' and are published in the year 2011.

```sql
SELECT B.Title
FROM BOOK B
WHERE B.Publisher_ID = '20' AND B.Year_of_Pub = 2011;
```

iv) Find the address of the publisher who has published Book_ID "500".

```sql
SELECT P.Address
FROM PUBLISHER P
WHERE P.Book_ID = '500';
```

Assumptions:

- The Book_ID in the AUTHOR, BOOK, and PUBLISHER tables refers to the same book.
- The Publisher_ID uniquely identifies each publisher in the PUBLISHER table.
- The year of publication in the BOOK table is stored as a four-digit integer (e.g., 2011).

Q2. a) With the help of a suitable example, discuss the insertion, deletion and updation anomalies that can occur in a database. Briefly discuss the mechanism to remove such anomalies. (6 Marks)

b) Write SQL commands for each of the following. Also illustrate the usage of each command through suitable example. (4 Marks)

(i) Creation of views

(ii) Creation of sequences

(iii) Outer join

(iv) To give access permission to any user

Ans

a) Insertion, Deletion, and Updation Anomalies in a Database (6 Marks)

In a relational database, anomalies can occur during insertion, deletion, and updating of data due to poor database design, especially when the database is not normalized. Let's discuss each anomaly with an example and how normalization can help remove these anomalies.

1. Insertion Anomaly:

- **Example:** Suppose we have a table STUDENT_COURSE with the following structure:

lua

Copy code

| Student_ID | Student_Name | Course_ID | Course_Name | Instructor_Name |

|---|---|---|---|---|

| 1 | John | 101 | Database | Dr. Smith |

If a new course is introduced but no student has enrolled in it yet, we cannot insert the course details without a corresponding Student_ID. This leads to an insertion anomaly because we are forced to insert a NULL value or a dummy Student_ID, which is not desirable.

- **Solution:** Normalization can help by separating the COURSE and STUDENT entities into different tables. For example, having a COURSE table and a STUDENT table with a COURSE_STUDENT mapping table will eliminate the need to insert a dummy Student_ID.

2. Deletion Anomaly:

- **Example:** Consider the same STUDENT_COURSE table. If a student named John (Student_ID=1) drops all his courses, and we delete his record from the table, we might also accidentally delete information about the courses if they are not linked to any other student. This causes a deletion anomaly.
- **Solution:** Again, normalization can help by storing courses in a separate COURSE table and linking them to students via a mapping table. This way, deleting a student record does not affect the course data.

3. Updation Anomaly:

- **Example:** Suppose Dr. Smith, who is the instructor for the Database course, changes her name. In the STUDENT_COURSE table, her name appears multiple times for different students taking the Database course. To update her name, we need to update it in all the rows where she appears. If we miss any row, it leads to inconsistency, which is an updation anomaly.

- **Solution:** By normalizing the database, we can store instructor information in a separate INSTRUCTOR table and reference it in the COURSE table. This way, we only need to update the name in one place.

Mechanism to Remove Anomalies:

- **Normalization:** The process of organizing data in a database into tables and columns to minimize redundancy and dependency. The most common normal forms are:
    - **1NF (First Normal Form):** Ensures that the table has only atomic (indivisible) values and each column contains unique values.
    - **2NF (Second Normal Form):** Ensures that all non-key attributes are fully functional dependent on the primary key.
    - **3NF (Third Normal Form):** Ensures that all the attributes are dependent only on the primary key, removing transitive dependencies.
- **Using Referential Integrity:** Enforcing foreign keys to maintain relationships between tables, ensuring that deletion or update operations do not result in loss of crucial data.

---

b) SQL Commands and Examples (4 Marks)

i) Creation of Views:

A view is a virtual table that is based on the result set of an SQL query. Views can be used to simplify complex queries, provide data security, and present data in a specific format.

SQL Command:

```
CREATE VIEW Student_View AS
SELECT Student_ID, Student_Name, Course_Name
FROM STUDENT_COURSE
WHERE Course_ID = 101;
```

Explanation: This command creates a view named Student_View that shows the Student_ID, Student_Name, and Course_Name for students enrolled in the course with Course_ID 101.

ii) Creation of Sequences:

A sequence is a database object that generates a sequence of unique numbers. Sequences are often used to generate primary key values.

SQL Command:

```
CREATE SEQUENCE student_seq
START WITH 1
INCREMENT BY 1
NOCACHE;
```

Explanation: This command creates a sequence named student_seq that starts with 1 and increments by 1. It will generate unique numbers, which can be used as primary keys for the Student_ID column.

iii) Outer Join:

An outer join returns all rows from one table and the matched rows from another. If there is no match, NULL values are returned.

SQL Command:

```
SELECT S.Student_Name, C.Course_Name
FROM STUDENT S
LEFT JOIN COURSE C ON S.Course_ID = C.Course_ID;
```

Explanation: This command performs a left outer join, returning all students and their corresponding courses. If a student is not enrolled in any course, the Course_Name will show as NULL.

iv) To Give Access Permission to Any User:

SQL allows you to grant specific permissions to users for various operations like SELECT, INSERT, UPDATE, etc.

SQL Command:

```
GRANT SELECT, INSERT ON STUDENT TO user_name;
```

Explanation: This command grants SELECT and INSERT permissions on the STUDENT table to the user user_name. The user will be able to view and insert data into the STUDENT table but cannot update or delete records.

Q3. a) What are integrity constraints? Discuss the various types of integrity constraints that can be imposed on database. (3 Marks)

b) How are database security and database integrity related? Briefly discuss the different levels of security measures which may be considered to protect the database. (3 Marks)

c) Consider the relation R (A, B, C, D, E) and the set of functional dependencies :- F(A → D, {A,B} → C, D → E) Assume that the decomposition of R into {R1 (A, B, C) and R2 (A, D, E)}. Is this decomposition lossless? Justify? (4 Marks)

Ans

a) Integrity Constraints

Integrity constraints are rules that ensure the accuracy and consistency of data within a database. These constraints enforce certain conditions on the data to maintain the database's integrity. The primary types of integrity constraints include:

1. **Domain Constraints**: Ensure that the data entered into a database field matches the defined data type, format, and range. For example, an age field might be constrained to accept only positive integers.
2. **Entity Integrity Constraints**: Ensure that each table has a primary key and that the key uniquely identifies each row in the table. This prevents duplicate records and ensures that the key fields are never null.

3. **Referential Integrity Constraints**: Ensure that relationships between tables remain consistent. If a foreign key is used to link two tables, referential integrity constraints ensure that the foreign key value always points to a valid, existing record in the referenced table.
4. **Unique Constraints**: Ensure that all values in a column (or a set of columns) are unique across the table, preventing duplicate entries.
5. **Not Null Constraints**: Ensure that a column cannot have a null value, meaning that the field must always have data.

b) Database Security and Database Integrity

Database security and database integrity are closely related, but they address different aspects of database management.

- **Database Security**: Involves protecting the database from unauthorized access, misuse, or malicious attacks. It focuses on ensuring that only authorized users have access to the database and that they can only perform actions within their permission levels.
- **Database Integrity**: Ensures that the data remains accurate, consistent, and reliable over time, regardless of the operations performed on it. It is maintained by enforcing integrity constraints and other validation rules.

Security Measures:

1. **Physical Security**: Protects the physical hardware and network infrastructure that hosts the database. This can include locked server rooms, surveillance, and environmental controls.
2. **Authentication and Authorization**: Controls who can access the database and what they can do. Authentication verifies user identity, while authorization determines their level of access.
3. **Encryption**: Protects sensitive data by converting it into a secure format. Data can be encrypted both at rest (in storage) and in transit (during transmission).
4. **Auditing and Monitoring**: Tracks and records database activities to detect and respond to unauthorized access or changes. Regular audits ensure compliance with security policies.
5. **Access Control**: Defines what actions users can perform based on their roles, such as read, write, update, or delete permissions.

c) Lossless Decomposition of Relation R

Given the relation R(A,B,C,D,E) and the set of functional dependencies F={A→D,{A,B}→C,D→E}, and the decomposition R into R1(A,B,C) and R2(A,D,E), we need to determine if this decomposition is lossless.

A decomposition is **lossless** if the natural join of the decomposed relations results in the original relation without any loss of information. This is true if at least one of the following conditions holds:

1. R1∩R2 is a superkey for R1, or
2. R1∩R2 is a superkey for R2.

In this case:

- R1(A,B,C)
- R2(A,D,E)

- $R1 \cap R2 = \{A\}$
- Now, let's check if A is a superkey in either R1 or R2.
- In R2, $A \rightarrow D$ and $D \rightarrow E$, so A can determine D and E. Therefore, A is a superkey for R2(A,D,E)

Since A is a superkey for R2, the decomposition is lossless.

Q4. a) Explain the Log-based recovery scheme with the help of an example. (5 Marks)

b) Compute the closure of the following set F of functional dependencies for relation schema R = (A, B, C, D, E).

$A \rightarrow BC$

$CD \rightarrow E$

$B \rightarrow D$

$E \rightarrow A$

List the candidate keys for R.

Ans

a) Log-based Recovery Scheme

Log-based recovery is a technique used in database management systems to ensure that the database can be restored to a consistent state after a failure. The basic idea is to maintain a log (or journal) of all the transactions and the changes they make to the database. In the event of a crash or failure, the log can be used to either **redo** or **undo** transactions to bring the database back to a consistent state.

Types of Log Entries:

1. **[Start, T]**: Indicates the start of transaction T.
2. **[T, X, old_value, new_value]**: Records the operation of transaction T on data item X, where the value of X is changed from old_value to new_value.
3. **[Commit, T]**: Indicates that transaction T has completed successfully.
4. **[Abort, T]**: Indicates that transaction T has been aborted.

Example:

Consider a scenario where we have a database with a single table containing a balance field. We have two transactions:

- **T1**: Transfers $50 from Account A to Account B.
- **T2**: Adds $100 to Account A.

Initially:

- Account A balance = $1000
- Account B balance = $1500

Transaction T1:

- Start transaction T1: [Start, T1]
- Read A's balance (which is $1000), subtract $50, and write the new balance of $950 to A: [T1, A, 1000, 950]
- Read B's balance (which is $1500), add $50, and write the new balance of $1550 to B: [T1, B, 1500, 1550]
- Commit transaction T1: [Commit, T1]

Transaction T2:

- Start transaction T2: [Start, T2]
- Read A's balance (which is $950), add $100, and write the new balance of $1050 to A: [T2, A, 950, 1050]
- Commit transaction T2: [Commit, T2]

Now, suppose a system crash occurs after T2 has updated A's balance but before it commits. The log will have the following entries:

csharp

Copy code

[Start, T1]

[T1, A, 1000, 950]

[T1, B, 1500, 1550]

[Commit, T1]

[Start, T2]

[T2, A, 950, 1050]

Recovery Process:

1. **Redo Phase**: The system checks the log and re-applies all the committed transactions to ensure all their effects are present in the database. Here, T1 is committed, so we redo the updates made by T1. T2's updates are also redone since it was in progress and its effects are in the log.
2. **Undo Phase**: The system checks for uncommitted transactions and rolls them back. Since T2 was not committed before the crash, we need to undo the update made by T2 to revert A's balance back to $950.

After recovery, the database will reflect the committed transaction T1, and the effects of the incomplete T2 will be undone.

b) Closure of the Functional Dependencies

Given the relation schema R=(A,B,C,D,E)) and the set of functional dependencies $F=\{A\rightarrow BC, CD\rightarrow E, B\rightarrow D, E\rightarrow A\}$, we can compute the closure of F, denoted as F^+.

Step 1: Start with the given FDs:

- $A\rightarrow BC$

- CD→E
- B→D
- E→A

Step 2: Compute the closure F^+:

1. **A→BC**: From A, we can derive B and C.
2. **B→D**: From B, we can derive D.
3. **CD→E**: From C and D, we can derive E.
4. **→A**: From E, we can derive A.

Let's apply these dependencies:

- **A^+={A,B,C,D,E}**: Using A→BC,B→D,CD→E, and E→A, all attributes can be derived from A. Hence, AAA is a superkey.
- **B^+={B,D,E,A,C}**: From B, we derive D, from D and C, we derive E, and from E, we derive A and C. Hence, BBB is also a superkey.
- **C^+={C}**: C alone doesn't derive any other attribute.
- **D^+={D}**: D alone doesn't derive any other attribute.
- **E^+={E,A,B,C,D}**: Using E→A, we can derive all attributes. Hence, E is also a superkey.

Candidate Keys:

- From the closure, we see that the attributes A, B, and E can each determine all the attributes in R. Therefore, the candidate keys for R are A, B, and E.

Q5. a) Give the limitations of file based system. How can they be overcome using DBMS? (5 Marks)

b) Discuss the importance of file organisation in databases. Mention the different types of file organisations available. Discuss any one of the mentioned file organisations in detail. (5 Marks)

Ans

a) Limitations of File-Based Systems and How DBMS Overcomes Them

Limitations of File-Based Systems:

1. **Data Redundancy and Inconsistency**: In a file-based system, the same data might be duplicated in multiple files, leading to redundancy. This can cause data inconsistency if one file is updated but others are not.
2. **Lack of Data Integration**: File-based systems often store data in isolated files that are not connected, making it difficult to retrieve related data from different files.
3. **Limited Data Sharing and Security**: Access to data is restricted, and there is no centralized control over who can view or modify the data, leading to potential security issues.

4. **Data Dependence**: Applications are closely tied to the data files they use. Changes in file formats or structures can require significant modifications to the application code.
5. **Difficulty in Data Access**: Retrieving data often requires custom programming, making it difficult and time-consuming to query data in a flexible and efficient manner.

How DBMS Overcomes These Limitations:

1. **Reduced Data Redundancy and Improved Consistency**: DBMS uses normalization techniques to minimize redundancy by organizing data into related tables, ensuring consistency across the database.
2. **Data Integration**: DBMS integrates data into a single database where related data can be easily linked and retrieved through relationships (e.g., foreign keys), making data retrieval more efficient.
3. **Improved Data Sharing and Security**: DBMS provides centralized control over data access, allowing administrators to define roles and permissions to control who can view or modify data.
4. **Data Independence**: DBMS abstracts the data from the applications that use it. Changes to the data structure (e.g., adding a new column) do not typically require changes to application code.
5. **Enhanced Data Access and Querying**: DBMS supports powerful query languages like SQL, allowing users to retrieve data flexibly and efficiently without needing to write custom programs.

b) Importance of File Organization in Databases

Importance of File Organization:

File organization refers to the way data is stored in files within a database system. The efficiency of data retrieval, storage, and management largely depends on how files are organized. Proper file organization ensures that:

1. **Efficient Data Retrieval**: Well-organized files allow for quicker data access and retrieval, reducing the time taken to execute queries.
2. **Optimized Storage Utilization**: File organization techniques help in utilizing storage space effectively, reducing wasted space and optimizing disk usage.
3. **Data Integrity and Security**: Proper organization helps maintain data integrity and provides mechanisms for securing data, ensuring that data remains accurate and safe from unauthorized access.
4. **Scalability and Performance**: As the volume of data grows, a well-organized file system can scale efficiently without significant degradation in performance.
5. **Ease of Maintenance**: Organized files make database maintenance tasks such as backups, updates, and indexing easier to perform.

Types of File Organizations:

1. **Heap (Unordered) File Organization**
2. **Sequential File Organization**
3. **Hashed File Organization**
4. **Clustered File Organization**

5. **Indexed File Organization**

Detailed Discussion: Sequential File Organization

Sequential File Organization: In sequential file organization, records are stored in a specific order, usually based on the value of a key field. For example, if records are stored in sequential order by a customer ID, then all records are arranged in ascending or descending order of the customer ID.

Characteristics:

- **Simple Structure**: Sequential file organization is straightforward, where records follow one after the other in a sequence.
- **Efficient for Batch Processing**: It is particularly efficient for batch processing systems where a large number of records are processed together, and the access pattern is predictable.

Advantages:

- **Fast Access for Sorted Data**: If the file is sorted based on the search key, sequential access is very fast.
- **Efficient Use of Storage**: It uses storage efficiently when dealing with sorted data.
- **Good for Range Queries**: Sequential file organization is effective for range queries, where records within a certain range need to be retrieved.

Disadvantages:

- **Inflexibility for Insertions and Deletions**: Insertions and deletions can be time-consuming, as they may require shifting records to maintain the order.
- **Poor Performance for Random Access**: Random access to records is slow because the system might need to search through the entire file to find a specific record.

Example: Consider a file storing customer records, sorted by Customer ID. If a new customer needs to be added, the system must find the correct position based on the ID, which may require shifting several records to maintain the order. This is efficient when reading data sequentially but becomes cumbersome when the file frequently undergoes updates.

Q6. a) For what reasons is 2-phase locking protocol required? Explain. Discuss the disadvantages of basic 2-phase locking protocol. List the ways and means that can be used to overcome the disadvantages. (5 Marks)

b) List and explain the 4 basic properties of a Transaction with the help of appropriate examples. (5 Marks)

Ans

a) Two-Phase Locking (2PL) Protocol

Reasons for 2-Phase Locking Protocol:

The Two-Phase Locking (2PL) protocol is essential in database management to ensure **serializability**—a key property of transaction schedules that ensures transactions are executed in a way that the final result is equivalent to some serial execution of those transactions. Without proper locking, transactions could interleave in a way that leads to **inconsistent** data or **violations** of integrity constraints.

The 2PL protocol is divided into two phases:

1. **Growing Phase**: A transaction can acquire locks (read or write) on data items, but it cannot release any locks during this phase.
2. **Shrinking Phase**: After acquiring all the necessary locks, the transaction releases locks and cannot acquire any new ones.

The protocol ensures that once a transaction starts releasing locks, it cannot acquire any more, thereby preventing scenarios where a transaction might release a lock and then try to re-acquire it, which could lead to deadlocks or inconsistencies.

Disadvantages of Basic 2-Phase Locking:

1. **Deadlocks**: Since transactions might wait indefinitely for locks held by other transactions, deadlocks can occur. A deadlock happens when two or more transactions wait for each other to release locks, leading to a standstill.
2. **Cascading Rollbacks**: If a transaction fails and releases its locks, other transactions that depended on the failed transaction's locks might also need to be rolled back, leading to a chain reaction of rollbacks.
3. **Reduced Concurrency**: The protocol can significantly reduce the level of concurrency because transactions must hold all locks until they enter the shrinking phase. This leads to increased waiting times, especially in high-transaction environments.

Ways to Overcome Disadvantages:

1. **Deadlock Prevention and Detection**:
    - **Prevention**: Techniques like "wait-die" and "wound-wait" are used to prevent deadlocks. These techniques decide whether a transaction should wait or abort based on a predefined order.
    - **Detection**: Deadlock detection mechanisms involve periodically checking for cycles in the wait-for graph and aborting transactions to break the cycle.
2. **Strict Two-Phase Locking (Strict 2PL)**: In this variant, transactions hold all write locks until they commit or abort, thereby preventing cascading rollbacks. This ensures that no transaction reads uncommitted data, maintaining data integrity.
3. **Timeouts**: Implementing timeouts can help detect and resolve deadlocks by aborting transactions that have been waiting too long for a lock.
4. **Optimistic Concurrency Control**: Instead of using locks, this technique allows transactions to execute without restrictions but validates them at commit time. If conflicts are detected, the transaction is rolled back and restarted.

b) ACID Properties of Transactions

ACID stands for **Atomicity, Consistency, Isolation,** and **Durability**, which are the four fundamental properties that ensure reliable processing of database transactions.

1. **Atomicity**:

- **Definition**: Ensures that a transaction is treated as a single unit of operation, which either fully completes or fully fails. If any part of the transaction fails, the entire transaction is rolled back, leaving the database in its original state.
- **Example**: Suppose a bank transaction involves transferring $100 from Account A to Account B. If the debit from Account A succeeds but the credit to Account B fails, the entire transaction is rolled back, so no money is transferred.

2. **Consistency**:
   - **Definition**: Ensures that a transaction brings the database from one valid state to another, maintaining all predefined rules, constraints, and data integrity.
   - **Example**: If a transaction involves transferring funds, the sum of balances in all accounts before and after the transaction should remain the same. For instance, in the transfer of $100 from Account A to Account B, if Account A's balance decreases by $100, Account B's balance must increase by $100.
3. **Isolation**:
   - **Definition**: Ensures that the operations of a transaction are isolated from those of other transactions. Intermediate states of a transaction should not be visible to other transactions.
   - **Example**: Consider two transactions: one updating the inventory and the other processing a sale. Isolation ensures that the sale transaction cannot see the inventory being updated until the inventory update transaction has completed.
4. **Durability**:
   - **Definition**: Ensures that once a transaction has been committed, its changes are permanent and will persist even in the event of a system failure.
   - **Example**: After a successful transfer of funds between two accounts, the updated account balances are stored permanently in the database. Even if the system crashes immediately after the transaction, the transfer will not be lost.

These ACID properties collectively ensure that database transactions are processed reliably and ensure the integrity of the data within the database system.

Q7. a) What do you mean by fragmentation of a database? What is the need of fragmentation in DDBMS environment? Explain different types of fragmentation with an example of each. (5 Marks) b) Explain the need of Distributed DBMS over Centralized DBMS. Also give the structure of Distributed DBMS. (5 Marks).

a) Fragmentation of a Database in Distributed DBMS

Fragmentation of a Database: Fragmentation in a Distributed Database Management System (DDBMS) refers to dividing a database into smaller, more manageable pieces called "fragments." Each fragment contains a subset of the database's data and can be stored at different sites in a distributed system. Fragmentation is critical in DDBMS because it improves performance, enables parallel processing, and enhances data localization.

Need for Fragmentation in DDBMS:

1. **Improved Performance**: By storing fragments close to where they are most frequently accessed, data retrieval becomes faster, reducing response times.
2. **Parallel Processing**: Different fragments can be processed simultaneously at different sites, leading to faster query execution and load balancing.
3. **Data Localization**: Data that is frequently accessed together can be stored together, reducing the need for data transfers across the network.
4. **Enhanced Security**: Sensitive data can be fragmented and stored in secure locations, reducing the risk of unauthorized access.
5. **Scalability**: Fragmentation allows for better scalability by distributing the workload across multiple sites.

Types of Fragmentation:

1. **Horizontal Fragmentation**:
    - **Definition**: Horizontal fragmentation divides a table into subsets of rows (tuples) based on a condition. Each fragment contains a subset of the rows in the table.
    - **Example**: Consider a table Employee(EmpID, Name, Dept, Salary):
        - Fragment 1: Employees from the "Sales" department.
        - Fragment 2: Employees from the "HR" department.
        - SQL Fragmentation:
2.
```
Fragment1 = SELECT * FROM Employee WHERE Dept = 'Sales';
Fragment2 = SELECT * FROM Employee WHERE Dept = 'HR';
```
Vertical Fragmentation:
    - **Definition**: Vertical fragmentation divides a table into subsets of columns (attributes). Each fragment contains a subset of the columns in the table, along with a primary key to maintain linkage.
    - **Example**: Consider the same Employee table:
        - Fragment 1: EmpID, Name
        - Fragment 2: EmpID, Dept, Salary
        - SQL Fragmentation:
3.
```
Fragment1 = SELECT EmpID, Name FROM Employee;
Fragment2 = SELECT EmpID, Dept, Salary FROM Employee;
```
Hybrid (Mixed) Fragmentation:
    - **Definition**: Hybrid fragmentation is a combination of both horizontal and vertical fragmentation. A table is first horizontally fragmented, and then each horizontal fragment is further vertically fragmented (or vice versa).

- **Example**: Consider the Employee table first horizontally fragmented by department and then vertically fragmented by separating personal information (Name) from job-related information (Dept, Salary):
    - Fragment 1: EmpID, Name for the "Sales" department.
    - Fragment 2: EmpID, Dept, Salary for the "Sales" department.
    - Fragment 3: EmpID, Name for the "HR" department.
    - Fragment 4: EmpID, Dept, Salary for the "HR" department.

b) Need for Distributed DBMS over Centralized DBMS and Structure of Distributed DBMS

Need for Distributed DBMS (DDBMS) over Centralized DBMS:

1. **Data Locality**: In a centralized DBMS, all data is stored in a single location, leading to high communication costs and delays for geographically dispersed users. DDBMS allows data to be stored closer to where it is needed, improving access times and reducing latency.
2. **Reliability and Availability**: Centralized systems have a single point of failure. If the central server fails, the entire system becomes unavailable. DDBMS distributes data across multiple sites, so even if one site fails, others can continue to operate, enhancing reliability and availability.
3. **Scalability**: Centralized DBMS can become a bottleneck as the system grows. DDBMS scales more effectively by distributing the workload across multiple sites, handling increasing amounts of data and users more efficiently.
4. **Performance**: DDBMS can improve performance by executing queries in parallel across multiple sites and reducing the need for large data transfers across networks.
5. **Autonomy and Flexibility**: Different sites in a DDBMS can operate independently, allowing for local control and autonomy. This is particularly useful in organizations where different departments or regions manage their own data.

Structure of Distributed DBMS:

A Distributed DBMS typically has the following components:

1. **Database**:
    - The actual data that is distributed across multiple sites. This includes fragmented, replicated, or partitioned data.
2. **Data Fragmentation and Allocation Manager**:
    - Responsible for fragmenting the database and allocating the fragments to various sites. It ensures that the data is correctly partitioned and stored according to the chosen fragmentation strategy (horizontal, vertical, hybrid).
3. **Distributed Query Processor**:
    - This component is responsible for processing queries in a distributed environment. It optimizes query execution by determining the best way to access and join data across multiple sites.
4. **Distributed Transaction Manager**:

- Manages transactions across multiple sites. It ensures that transactions are executed in a way that maintains ACID properties, even when data is spread across different locations. It handles issues like concurrency control and recovery in a distributed setting.

5. **Distributed Directory (Catalog) Manager**:
   - Maintains metadata about where data is stored, how it is fragmented, and how to access it. It acts as a reference guide for locating data within the distributed system.
6. **Communications Manager**:
   - Handles the communication between different sites. It manages data transmission, message passing, and coordination between sites to ensure that distributed transactions and queries are executed correctly.

Structure Diagram:

Q8. An organization needs to provide Medical facilities to its employees and their dependents. Organization is having a list of Doctors, Hospitals and Test centres for the employees facility. An employee may get Medical facility from the list of Doctors, Hospitals and Test centres provided by the organization to them. Employee does not need to pay anything for the facilities availed. The Doctors, Hospitals and Test centres directly raise their bill to the organization.

Identify the entities, relationships, constraints and cardinality and construct an ER diagram for the above mentioned specifications. List your assumptions and clearly indicate the cardinality mappings as well as any role indicators in your ER diagram. (10 Marks)

Ans

To design an ER diagram for the given scenario, we first need to identify the entities, relationships, constraints, and cardinalities.

Entities:

1. **Employee**: Represents the employees of the organization.
2. **Dependent**: Represents the dependents of the employees.
3. **Doctor**: Represents the doctors available to provide medical facilities.
4. **Hospital**: Represents the hospitals available to provide medical facilities.
5. **Test Center**: Represents the test centers available for medical tests.
6. **Medical Facility**: Represents the medical services availed by the employees or their dependents.
7. **Bill**: Represents the bill raised by the Doctor, Hospital, or Test Center to the organization.

Relationships:

1. **Employee-Dependent**: An employee can have one or more dependents, but a dependent is associated with only one employee.
    - **Cardinality**: 1 (One Employee to Many Dependents)
2. **Employee-Medical Facility**: An employee can avail of multiple medical facilities, and each medical facility can be availed by one or more employees.
    - **Cardinality**: M (Many Employees to Many Medical Facilities)
3. **Dependent-Medical Facility**: A dependent can avail of multiple medical facilities, and each medical facility can be availed by one or more dependents.
    - **Cardinality**: M (Many Dependents to Many Medical Facilities)
4. **Doctor-Medical Facility**: A doctor can provide multiple medical facilities, and each medical facility can be provided by one or more doctors.
    - **Cardinality**: M (Many Doctors to Many Medical Facilities)
5. **Hospital-Medical Facility**: A hospital can provide multiple medical facilities, and each medical facility can be provided by one or more hospitals.
    - **Cardinality**: M (Many Hospitals to Many Medical Facilities)

6. **Test Center-Medical Facility**: A test center can provide multiple medical facilities, and each medical facility can be provided by one or more test centers.
    - **Cardinality**: M (Many Test Centers to Many Medical Facilities)
7. **Medical Facility-Bill**: A bill is associated with one or more medical facilities.
    - **Cardinality**: 1 (One Medical Facility to Many Bills)

Constraints:

- **Unique Identifier for Employee**: Each employee should have a unique identifier (e.g., Employee ID).
- **Unique Identifier for Dependent**: Each dependent should have a unique identifier and be associated with only one employee.
- **Unique Identifier for Doctor, Hospital, and Test Center**: Each of these entities should have unique identifiers (e.g., Doctor ID, Hospital ID, Test Center ID).
- **Mandatory Participation**: Employees and their dependents must be associated with at least one medical facility if they use any services.

Assumptions:

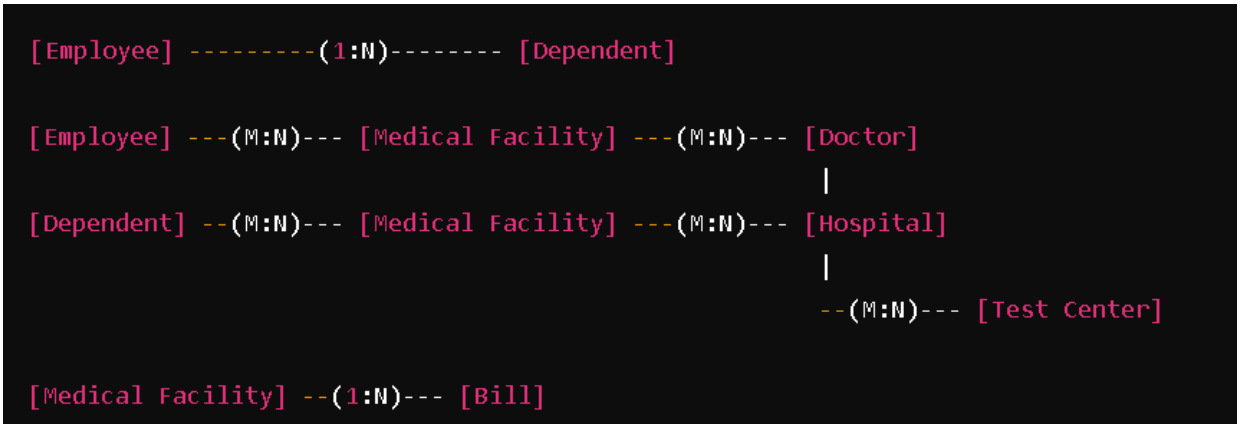
1. Each medical facility availed can be linked to either a doctor, a hospital, or a test center.
2. Each bill is generated for a specific medical facility provided to an employee or dependent.
3. The organization handles all billing directly with the doctors, hospitals, and test centers.

ER Diagram Construction:

Here's a textual description of the ER diagram that you can visualize:

1. **Employee** entity has a one-to-many relationship with **Dependent**. (1)
2. **Employee** entity has a many-to-many relationship with **Medical Facility**. (M)
3. **Dependent** entity has a many-to-many relationship with **Medical Facility**. (M)
4. **Medical Facility** has a many-to-many relationship with **Doctor**, **Hospital**, and **Test Center**. (M for each)
5. **Medical Facility** entity has a one-to-many relationship with **Bill**. (1)

ER Diagram Visualization:



- **[Employee]** and **[Dependent]** are entities.
- **(M)** denotes a many-to-many relationship.
- **(1)** denotes a one-to-many relationship.
- **[Medical Facility]** is a central entity connected to **Employee**, **Dependent**, **Doctor**, **Hospital**, **Test Center**, and **Bill**.
- The relationship between **Medical Facility** and **Bill** is one-to-many, as multiple bills can be associated with a single medical facility.

Cardinality Mappings:

- **Employee to Dependent**: One employee can have many dependents.
- **Employee to Medical Facility**: One employee can avail multiple medical facilities, and one medical facility can be availed by multiple employees.
- **Medical Facility to Doctor/Hospital/Test Center**: One medical facility can be linked to multiple doctors, hospitals, or test centers, and each can provide multiple medical facilities.
- **Medical Facility to Bill**: One medical facility can have multiple bills associated with it.

This diagram and explanation should help you understand how to model the scenario effectively in an ER diagram.