**Part B**

**21. Compare and contrast the benefits and drawbacks of denormalization with normalization in the context of database design.**

Normalization:

* Benefits:
  1. Ensures data integrity by minimizing redundancy and eliminating update anomalies.
  2. Provides flexibility for modifications without risking data inconsistencies.
  3. Optimized for transactional systems where data integrity is crucial.
  4. Reduces storage space by eliminating redundant data.
* Drawbacks:
  1. May incur performance overhead due to join operations needed to reconstruct denormalized views.
  2. Increased complexity of schema with interconnected tables and foreign key relationships.
  3. Read-heavy workloads may suffer from the performance impact of join operations.
  4. Not optimal for reporting or analytics where query performance is critical.

Denormalization:

* Benefits:
  1. Improves query performance by reducing the need for join operations.
  2. Simplifies queries by eliminating complex joins across multiple tables.
  3. Well-suited for reporting and analytics, optimizing query performance.
  4. Reduces join overhead, improving scalability and read-heavy workload performance.
* Drawbacks:
  1. Introduces data redundancy, increasing the risk of data inconsistency.
  2. Maintaining data integrity can be challenging, especially with updates to redundant data.
  3. Increases storage requirements due to redundant data storage.

Maintenance can be difficult, requiring careful management of redundant data updates.

**22. “Stable storage helps in the protecting data during failures”. State the validity of the sentences with justification.**

**Data Durability**

**Write Durability**

**Transaction Atomicity- ACID properties**

**Data Recovery**

**23. What is deferred DB Modification? Give an example?**

Deferred database modification refers to a strategy in database management where changes to the database are postponed or deferred until a specific point in time, typically after a transaction completes successfully. This approach is often used to improve transaction performance and reduce contention for database resources.

Example with explanation

**Part C**

**24. You are the database administrator for a banking system that handles millions of transactions daily. Recently, your team has implemented a new feature that allows customers to transfer funds between accounts. However, during stress testing, it was discovered that some transactions were not executing in a serializable manner, leading to incorrect account balances and potential financial discrepancies.**

**Outline the steps you would take to address the issue, ensuring that all transactions maintain serializability and the system can recover from failures effectively. Consider the following aspects:**

**a) Define the concept of serializability and explain why it is essential for maintaining data consistency in a multi-user database environment.**

**b) Describe how you would test for serializability in the banking system, including any tools or techniques you might utilize.**

**c) Propose strategies to ensure that all transactions execute in a serializable manner, preventing conflicts and ensuring data integrity.**

**d) Discuss the importance of system recovery mechanisms in the event of a database failure, and outline the steps you would take to implement robust recovery procedures.**

**a) Serializability in Database Transactions:**

Serializability is a property of a schedule of transactions in a database system, ensuring that the execution of transactions produces the same result as if they were executed serially, one after the other. In other words, it ensures that the database remains in a consistent state despite concurrent execution of transactions.

Importance for Data Consistency: In a multi-user database environment like a banking system, serializability is crucial for maintaining data consistency and integrity. It prevents anomalies such as lost updates, inconsistent reads, and dirty reads by ensuring that transactions do not interfere with each other's operations.

b) Testing for Serializability:

Concurrency Control Testing: Use concurrency control testing techniques to simulate concurrent execution of transactions and analyze the resulting schedules for serializability violations.

Schedule Validation: Employ tools like the Conflict Serializability Checker (CSC) or Transaction Schedule Analyzer (TSA) to analyze transaction schedules and identify non-serializable executions.

Transaction Isolation Levels: Test different transaction isolation levels (e.g., Read Committed, Serializable) to evaluate their impact on serializability and data consistency.

c) Strategies for Ensuring Serializability:

Concurrency Control Mechanisms

Transaction Design

Conflict Resolution

d) System Recovery Mechanisms:

Backup and Restore Procedures

Transaction Logging

Fault Tolerance: Deploy fault-tolerant hardware and software solutions to reduce the risk of system failures and improve system reliability.

**25. Case Study: A hospital maintains a database to manage patient records, including patient ID, name, diagnosis, and treatment information. The current database design lacks normalization, resulting in data redundancy and inconsistency. Normalize the database schema to eliminate these issues up to the third normal form (3NF). Explain each step of the normalization process and discuss how the resulting design improves data integrity and efficiency.**

STEP 1:1 NF

In 1NF, we ensure that each column contains atomic values, meaning no multi-valued attributes or repeating groups.

Original schema:

* Patient ID(primary key)
* Name
* Diagnosis
* Treatment

To achieve 1NF, we need to ensure that there are no repeating groups. We can do this by separating the diagnosis and treatment information into separate tables, linked by the patient ID.

**Patient Table:**

Patient id(primary key)

Name

**Diagnosis Table:**

Diagnosis ID(primary key)

Patient ID (Foreign Key)

Diagnosis

**Treatment Table:**

Treatment id(primary key)

Patient id (foreign key)

Treatment

Now, each table has atomic values, and there are no repeating groups.

Step 2: 2NF

In 2NF, we ensure that there are no partial dependencies, meaning each non-prime attribute is fully functionally dependent on the primary key.

In our case, both Diagnosis and Treatment tables already satisfy 2NF since all non-prime attributes are fully functionally dependent on the primary key (Patient ID) in each table.

Step 3: 3 NF

In 3NF, we ensure that there are no transitive dependencies, meaning no non-prime attribute is transitively dependent on the primary key through another non-prime attribute.

Original schema :

* Patient ID(primary key)
* Name
* Diagnosis
* Treatment

In the original schema, the Name attribute is transitively dependent on the Patient ID through the Diagnosis or Treatment. To achieve 3NF, we need to remove this transitive dependency.

**Patient Table:**

Patient ID (primary key)

Name

**Diagnosis Table:**

**Diagnosis Table:**

Diagnosis ID(primary key)

Patient ID (Foreign Key)

Diagnosis

**Treatment Table:**

Treatment id(primary key)

Patient id (foreign key)

Treatment

Now, the database schema is in the third normal form (3NF), ensuring that there are no data redundancies or inconsistencies, and each table serves a single purpose. This design improves data integrity by minimizing update anomalies and improves efficiency by reducing redundant data storage and facilitating easier data manipulation and querying.

**26. Case Study: A university manages a database to track student enrollment, courses, and grades. Design relational algebra queries to perform the following tasks:**

**a. Retrieve the names of students who have enrolled in a specific course.**

**b. Calculate the average grade for each course.**

**c. Find the courses with the highest enrollment.**

**d. Determine the students who have failed a course.**

**a. Retrieve the names of students who have enrolled in a specific course:**

Let's assume:

* **Students**: StudentID, StudentName
* **Courses**: CourseID, CourseName
* **Enrollments**: StudentID, CourseID
* The query would be:

πStudentName​(σCourseName=′specific\_course\_name′​(Students⋈Enrollments⋈Courses))

* This query joins the Students, Enrollments, and Courses tables on their respective IDs and selects only the rows where the CourseName matches the specific course name. Then it projects the StudentName from the resulting table.
  1. **Calculate the average grade for each course.**

Let's assume:

* **Grades**: StudentID, CourseID, Grade

The query would be:

πCourseID,AVG(Grade)​(γCourseID,AVG(Grade)​(Grades))

This query groups the Grades table by CourseID and calculates the average grade for each group. Then it projects both the CourseID and the average grade.

* 1. **Find the courses with the highest enrollment.**

Let's assume:

* **Enrollments**: CourseID, StudentID

The query would be:

πCourseID​(γCourseID,COUNT(StudentID)​(Enrollments))÷πCOUNT(StudentID)​(γCourseID,COUNT(StudentID)​(Enrollments))

This query first groups the Enrollments table by CourseID and counts the number of students enrolled in each course. Then it finds the maximum count. Finally, it selects the courses with enrollment counts equal to the maximum count.

* 1. **Determine the students who have failed a course.**

Let's assume:

* **Grades**: StudentID, CourseID, Grade
* **PassingGrade**: PassingGradeValue (assuming passing grade is defined)

The query would be:

πStudentID​(σGrade<PassingGradeValue​(Grades⋈PassingGrade))

This query joins the Grades and PassingGrade tables on the common attribute (assuming there is a table that stores the passing grade for each course). Then it selects only the rows where the grade is less than the passing grade value and projects the StudentID from the resulting table.

These are basic examples of how you could approach these tasks using relational algebra. Depending on your specific database schema and requirements, you might need to adjust these queries accordingly.