**Part B**

**21. Why 4NF in Normal Form is more desirable than BCNF?**

BCNF (Boyce code normal form) has all functional dependencies A to B are trivial of discriminator should be superkey. To get relation in BCNF, Splitting the relation schema not necessarily preserve all functional dependency, Loss less decomposition and dependency are main points for the normalization sometime, it is not possible to get a BCNF decomposition that is dependency, preserving. While 4NF has very similar definition asBCNF. A relational Schema is in 4NF, if all multivalued dependencies A to B are trivial and determinate A is superkey of schema. If a relational schema is in 4nf, it is already in BCNF. and 4NF decomposition preserve the all-functional dependency. so 4NF is preferable than to have BCNF.

**22. Write about the importance of functional dependencies in the normalization process. Explain how identifying and understanding functional dependencies helps in achieving higher normal forms.**

Functional dependencies play a crucial role in the normalization process of database design. They represent the relationships between attributes (columns) in a relation (table) and are essential for achieving higher normal forms. Here's how identifying and understanding functional dependencies contribute to achieving higher normal forms:

1. **Ensuring Data Integrity:** Functional dependencies help ensure data integrity by defining the relationships between attributes. By identifying which attributes uniquely determine other attributes, we can prevent data redundancy and eliminate the risk of update anomalies. For example, in a table of employee data, the employee ID uniquely determines the employee's name, department, and other attributes. This functional dependency helps maintain data integrity and prevents inconsistencies.
2. **Normalization Process:** Functional dependencies guide the normalization process by identifying the primary key and candidate keys of a relation. The primary key uniquely identifies each tuple in the relation, and candidate keys are sets of attributes that could serve as the primary key. By identifying functional dependencies, we can determine which attributes should be part of the primary key and how to decompose the relation into smaller, more manageable tables to achieve higher normal forms.
3. **Achieving Higher Normal Forms:** Understanding functional dependencies is essential for decomposing relations into higher normal forms (such as 2NF, 3NF, BCNF, and beyond). Each normal form aims to eliminate specific types of data redundancy and dependency by ensuring that attributes are functionally dependent on the primary key. By identifying and removing partial and transitive dependencies, we can normalize the database schema to higher normal forms, resulting in a more efficient and well-structured database design.
4. **Optimizing Query Performance:** Functional dependencies also help optimize query performance by allowing for more efficient data retrieval. With a well-normalized database schema, queries can be executed more efficiently, as they only need to access the relevant attributes from the appropriate tables. This reduces the need for unnecessary joins and improves overall database performance.
5. **Facilitating Maintenance and Updates:** By understanding functional dependencies, database administrators can better manage and maintain the database schema. Changes to the database structure or business requirements can be easily accommodated without introducing inconsistencies or data redundancy. Functional dependencies provide a clear understanding of how attributes are related, making it easier to update and maintain the database over time.

**23. Explain Two-Phase locking Protocol.**

Definition:

Two-Phase Locking is a concurrency control mechanism that ensures transactions acquire all the locks they need before starting to execute, and then release all the locks they hold only after completing their execution.

Phases:

Growing Phase: During this phase, transactions can acquire locks but cannot release any lock.

Shrinking Phase: After a transaction releases its first lock, it cannot acquire any new lock but can release locks it has already acquired.

Types of locks:

Shared Lock (S-lock): Allows multiple transactions to read a resource concurrently but prohibits any transaction from writing to it.

Exclusive Lock (X-lock): Prevents any other transaction from accessing a resource, neither for reading nor for writing, until the lock is released.

**Part C**

**24. Consider the below scenario:**

**In a banking database system, two transactions, T1 and T2, are concurrently processing transactions involving the same account. T1 transfers 100𝑓𝑟𝑜𝑚 𝐴𝑐𝑐𝑜𝑢𝑛𝑡𝐴 𝑡𝑜 𝐴𝑐𝑐𝑜𝑢𝑛𝑡𝐵, 𝑤ℎ𝑖𝑙𝑒 𝑇2 𝑡𝑟𝑎𝑛𝑠𝑓𝑒𝑟𝑠 100 from Account A to Account B , while T2 transfers 50 from Account B to Account A.**

**Question:**

**Based on the given scenario, explain how the concept of serializability in database management ensures that the final state of the accounts remains consistent despite the concurrent execution of transactions T1 and T2.**

Initial State:

Let's say both Account A and Account B initially have a balance of $500.

Execution of Transactions:

Transaction T1 transfers $100 from Account A to Account B.

Transaction T2 transfers $50 from Account B to Account A.

Possible Schedule:

One possible interleaved execution of these transactions could be:

T1 reads the balance of Account A ($500).

T2 reads the balance of Account B ($500).

T1 deducts $100 from Account A.

T2 deducts $50 from Account B.

T1 deposits $100 into Account B.

T2 deposits $50 into Account A.

Final State:

After the execution of both transactions, the final balances should be:

Account A: $450 (initial $500 - $50 + $50)

Account B: $550 (initial $500 + $100 - $100)

Serializability:

Despite the concurrent execution of T1 and T2, the final state of the accounts is consistent with a serial execution of the transactions.

If we serialize the transactions, T1 followed by T2 or T2 followed by T1, the final state would be the same as the one achieved through concurrent execution.

Consistency:

Serializability ensures that the database remains in a consistent state throughout the execution of transactions.

**25. 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.

**26. You are the database administrator for a large e-commerce platform experiencing rapid growth in both user traffic and product inventory. With the increase in concurrent transactions, you've noticed occasional instances of data inconsistency and deadlock situations within the database system.**

**Outline your approach to implementing effective concurrency control mechanisms to address these issues and ensure data integrity. Consider the following aspects:**

**1. Identify and explain the types of concurrency control techniques available in a DBMS, highlighting their advantages and limitations.**

**2. Analyze the specific challenges posed by concurrent transactions in the e-commerce platform, such as inventory updates, order processing, and user authentication.**

**3. Propose a concurrency control strategy tailored to the e-commerce platform's requirements, considering factors like transaction isolation levels, locking mechanisms, and deadlock detection/prevention.**

**4. Discuss how you would monitor and fine-tune the concurrency control mechanisms over time to accommodate the platform's evolving workload and transaction patterns.**

**Provide detailed explanations and examples to support your approach, demonstrating your understanding of concurrency control principles and their application in a real-world scenario.**

**Types of Concurrency Control Techniques:**

**a. Lock-Based Concurrency Control:**

Advantages: Provides strong consistency by ensuring exclusive access to data. Helps prevent data corruption and maintain data integrity.

Limitations: May lead to contention and decreased performance, especially in high-concurrency environments. Can result in deadlocks if not managed properly.

**b. Optimistic Concurrency Control (OCC):**

Advantages: Allows multiple transactions to proceed concurrently without blocking, reducing contention and increasing throughput. Suitable for scenarios with low update conflicts.

Limitations: Relies on detecting conflicts during transaction validation, which may lead to rollbacks and retries. Not ideal for high-contention environments or scenarios with frequent conflicts.

**c. Multi-Version Concurrency Control (MVCC):**

Advantages: Provides high concurrency by allowing readers to access consistent snapshots of data while writers operate independently. Reduces contention and improves scalability.

Limitations: Increases storage overhead due to maintaining multiple versions of data. May lead to increased complexity in managing transaction visibility and cleanup of obsolete versions.

**Challenges in the E-commerce Platform:**

Inventory Updates

Order Processing

User Authentication

**Concurrency Control Strategy:**

Transaction Isolation Levels

Locking Mechanisms

Deadlock Detection/Prevention

**Monitoring and Fine-Tuning:**

Performance Monitoring: Continuously monitor database performance metrics such as throughput, response time, and resource utilization to identify bottlenecks and areas for improvement.

Workload Analysis: Analyze transaction patterns and workload characteristics to understand concurrency requirements and optimize concurrency control mechanisms accordingly.

Feedback Loop: Solicit feedback from users and stakeholders to identify pain points and address them through iterative improvements to concurrency control strategies.