Question Bank Answers (Database Management System – IAE2)

> CONTENT WARNING:
> READING THIS DOCUMENT MAY CAUSE SUDDEN BURSTS OF INTELLIGENCE.
> PROCEED WITH CAUTION.

1. What is Deadlock? Explain deadlock handling in concurrency control

A **deadlock** is a situation in a multi-threaded or multi-process system where two or more processes are unable to proceed because each is waiting for the other to release a resource. In simpler terms, it's a standstill where processes are stuck, waiting for resources that the others hold.

Conditions for Deadlock

Deadlock occurs when the following four conditions hold simultaneously:

1. **Mutual Exclusion**: At least one resource is held in a non-shareable mode. Only one process can use the resource at any given time.
2. **Hold and Wait**: A process is holding at least one resource and is waiting to acquire additional resources that are currently being held by other processes.

3. **No Preemption**: Resources cannot be forcibly taken from a process; they must be voluntarily released.
4. **Circular Wait**: A circular chain of processes exists, where each process holds at least one resource needed by the next process in the chain.

Deadlock Handling in Concurrency Control

Deadlock handling can be approached in several ways:

1. **Deadlock Prevention**:
   - Modify the system to prevent one of the four necessary conditions from occurring. For example:
     - **Mutual Exclusion** can be avoided by making resources shareable if feasible.
     - **Hold and Wait** can be prevented by requiring processes to request all required resources at once.
     - **No Preemption** can be managed by allowing resource preemption.
     - **Circular Wait** can be avoided by defining a strict ordering of resource acquisition.
2. **Deadlock Avoidance**:
   - The system can make decisions dynamically to ensure that it will not enter a deadlock state. This can be achieved using algorithms like:
     - **Banker's Algorithm**: This checks whether the resources are allocated safely, ensuring that there’s always enough resources available to complete the processes.
3. **Deadlock Detection and Recovery**:
   - Allow the system to enter a deadlock state but have mechanisms to detect it. Once a deadlock is detected, the system can recover by:
     - Terminating one or more processes to break the deadlock.
     - Preempting resources from some processes to resolve the circular wait condition.
4. **Ignore Deadlocks**:
   - In some systems, particularly in smaller or less critical environments, deadlocks may be ignored. The system relies on the hope that they will be rare and can be manually resolved when they occur.

2. Write short note on following
a. JDBC in Database Systems
b. Need of normalization in Database Design
c.Recursive Queries

a. JDBC in Database Systems

JDBC (Java Database Connectivity) is an API that allows Java applications to interact with databases. It provides methods to connect to a database, execute SQL queries, and retrieve results. JDBC is crucial for Java applications that need to perform database operations.

Key Components:

- Driver Manager: Manages the different database drivers, allowing the application to connect to the database.
- Connection: Represents a connection to the database.
- Statement: Used to execute SQL queries against the database.
- ResultSet: A table of data representing the results of a query.

Importance:

Enables platform-independent access to various databases.

Simplifies database interactions through standard Java methods.

b. Need for Normalization in Database Design

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. It involves dividing a database into tables and defining relationships between them.

Benefits:

- Eliminates Redundancy: Reduces duplicate data by separating information into related tables.
- Improves Data Integrity: Ensures that data is consistent and accurate across the database.
- Enhances Query Performance: Simplifies data retrieval by structuring data logically.
- Facilitates Maintenance: Makes it easier to update data without affecting other parts of the database.

Normalization typically involves several normal forms (1NF, 2NF, 3NF, etc.), each with specific criteria to meet.

c. Recursive Queries

Recursive Queries are SQL queries that reference themselves to retrieve hierarchical or tree-structured data. They are particularly useful for dealing with parent-child relationships, such as organizational charts or folder structures.

Components:

Common Table Expressions (CTEs): Recursive queries often use CTEs, which define a temporary result set that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement.

Example Use Cases:

Retrieving all employees in a hierarchy from a manager down to their subordinates.

Finding all descendants of a particular node in a tree structure.

Benefits:

Simplifies complex queries that would otherwise require multiple joins or nested queries.

Enhances readability and maintainability of SQL code.

3. Define normalization. Explain 1NF and 2NF,3NF,BCNF,4NF,5NF in detail with example

(You can refer your SQL Assignment 2 i.e. normalization too for this answer)
Normalization
is a process in database design that organizes data to reduce redundancy and improve data integrity. It involves dividing a database into smaller, related tables while maintaining relationships between them.

Example Table

A sample table that contains student information, courses, and instructors:

StudentID	Name	Course	Instructor
1	Millie	Math	Dr. Smith
1	Millie	Science	Dr. Jones
2	Hannah	Math	Dr. Smith
3	Eefrah	Science	Dr. Jones
3	Eefrah	Art	Dr. Taylor
4	Henry	Math	Dr. Smith

1. First Normal Form (1NF)

Definition: A table is in 1NF if all attributes contain only atomic (indivisible) values, and each entry in a column is of the same type.

Transformed Table:

StudentID	Name	Course	Instructor
1	Millie	Math	Dr. Smith
1	Millie	Science	Dr. Jones
2	Hannah	Math	Dr. Smith
3	Eefrah	Science	Dr. Jones
3	Eefrah	Art	Dr. Taylor
4	Henry	Math	Dr. Smith

(Already in 1NF as each value is atomic)

2. Second Normal Form (2NF)

Definition: A table is in 2NF if it is in 1NF and all non-key attributes are fully functionally dependent on the primary key.

Transformed Tables:

Students Table:

StudentID	Name
1	Millie
2	Hannah
3	Eefrah
4	Henry

Courses Table:

Course	Instructor
Math	Dr. Smith
Science	Dr. Jones
Art	Dr. Taylor

Enrollment Table:

StudentID	Course
1	Math
1	Science
2	Math
3	Science
3	Art
4	Math

3. Third Normal Form (3NF)

Definition: A table is in 3NF if it is in 2NF and all attributes are functionally dependent only on the primary key, with no transitive dependencies.

Transformed Tables:

Students Table (unchanged):

StudentID	Name
1	Millie
2	Hannah
3	Eefrah
4	Henry

Courses Table (unchanged):

Course	Instructor
Math	Dr. Smith
Science	Dr. Jones
Art	Dr. Taylor

4. Boyce-Codd Normal Form (BCNF)

Definition: A table is in BCNF if it is in 3NF and every determinant is a candidate key.

Transformed Tables:

Assuming the following dependency: `Instructor` determines `Room` but `Instructor` is not a candidate key.

Instructors Table:

Instructor	Course	Room
Dr. Smith	Math	Room 101
Dr. Jones	Science	Room 102
Dr. Taylor	Art	Room 201

5. Fourth Normal Form (4NF)

Definition: A table is in 4NF if it is in BCNF and contains no multi-valued dependencies.

Transformed Tables:

Students Table:

StudentID	Course
1	Math
1	Science
2	Math
3	Science
3	Art
4	Math

Hobbies Table (if hobbies were included):

StudentID	Hobby
1	Painting
1	Reading
2	Music
3	Sports
4	Dance

6. Fifth Normal Form (5NF)

Definition: A table is in 5NF if it is in 4NF and every join dependency is implied by the candidate keys.

Transformed Tables:

Projects Table:

Project	Employee
ProjectA	Millie
ProjectA	Hannah
ProjectB	Eefrah
ProjectB	Henry

Roles Table:

Project	Role
ProjectA	Manager
ProjectA	Developer
ProjectB	Developer
ProjectB	Tester

4. Explain ACID properties in details with example.

ACID properties are a set of principles that ensure reliable processing of database transactions. The acronym ACID stands for **Atomicity**, **Consistency**, **Isolation**, and **Durability.**

1. Atomicity

Definition: Atomicity guarantees that a transaction is treated as a single, indivisible unit. This means that either all operations within the transaction are executed, or none are executed at all. If a transaction fails, any changes made during that transaction are rolled back.

Example:
Consider a bank transfer from Account A to Account B. If the transaction fails after deducting from Account A but before adding to Account B, the result would leave Account A with less money and Account B unchanged. Atomicity ensures that both operations are completed successfully or none at all. If any part fails, the entire transaction is rolled back, leaving both accounts unaffected.

2. Consistency

Definition: Consistency ensures that a transaction takes the database from one valid state to another valid state, maintaining the integrity constraints. Any transaction must adhere to the predefined rules, constraints, and cascades in the database.

Example:
If Account A has a balance of $200 and a transaction tries to deduct $300, consistency prevents this action, maintaining valid account states.

3. Isolation

Definition: Isolation ensures that transactions occur independently of one another. Even if multiple transactions are executed concurrently, the results of each transaction should be as if they were executed in isolation, preventing them from interfering with each other.

Example:
If Millie is transferring $100 to Hannah while Bob checks Hannah's balance, isolation ensures that Bob sees either the balance before or after Millie transaction, not an intermediate state.

4. Durability

Definition: Durability guarantees that once a transaction has been committed, it will remain committed even in the event of a system failure, such as a crash or power loss. The changes made by a committed transaction must be permanent and saved to non-volatile storage.

Example:
After Millie’s transfer is completed, if the system crashes, durability guarantees that Hannah’s updated balance remains intact when the system is restored.

5. Discuss procedure, functions and cursors with example.

i. Procedures

Definition: A stored procedure is a precompiled collection of one or more SQL statements that can be executed as a single unit. Procedures can accept parameters and may return multiple results.

Characteristics:

Can perform operations like querying, inserting, updating, or deleting data.

Can have input and output parameters.

Do not return a value directly but can return multiple result sets.

Example:
Here’s an example of a stored procedure that adds a new student record to a database:

```
CREATE PROCEDURE AddStudent(
    IN studentName VARCHAR(100),
    IN studentAge INT,
    OUT studentID INT
)
BEGIN
    INSERT INTO Students (Name, Age) VALUES (studentName, studentAge);
    SET studentID = LAST_INSERT_ID();
END;
```

Usage:

```
CALL AddStudent('John Doe', 20, @newStudentID);
SELECT @newStudentID;  -- Retrieves the ID of the newly added student.
```

ii. Functions

Definition: A function is a stored program that returns a single value. Functions can be called within SQL statements and are often used for computations.

Characteristics:

Must return a value.

Can be used in SELECT statements or other SQL expressions.

Cannot perform transactions like committing or rolling back.

Example:
Here’s an example of a function that calculates the total price including tax:

```
CREATE FUNCTION CalculateTotalPrice(
    price DECIMAL(10, 2),
    taxRate DECIMAL(5, 2)
) RETURNS DECIMAL(10, 2)
```

```
BEGIN
    RETURN price + (price * taxRate);
END;
```

Usage:

```
SELECT CalculateTotalPrice(100.00, 0.07) AS TotalPrice;  -- Returns 107.00
```

iii.. Cursors

Definition: A cursor is a database object used to retrieve, manipulate, and navigate through a result set one row at a time. Cursors are useful when you need to process each row individually.

Characteristics:

Useful for row-by-row processing in complex business logic.

Requires more overhead than set-based operations and can lead to performance issues if not used carefully.

Example:
Here's an example of how to declare and use a cursor to fetch student names from a table:

```
DECLARE studentCursor CURSOR FOR
SELECT Name FROM Students;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN studentCursor;

SET done = FALSE;
FETCH studentCursor INTO @studentName;

WHILE NOT done DO
    SELECT @studentName;
    FETCH studentCursor INTO @studentName;
END WHILE;

CLOSE studentCursor;
```

6. Write a short note on conflict serializability

→ Conflict Serializability

Definition: A schedule is conflict serializable if it can be transformed into a serial schedule (where transactions are executed one after the other) by swapping non-conflicting operations.

Conflict equivalent:
Two schedules are conflict equivalent if the relative order of any two conflicting
operations is the same in both schedules.
Commonly used definition of schedule equivalence

Two operations are conflicting if:

- They access the same data item X.
- They are from two different transactions.

- At least one is a write operation

Example:

- Read-Write conflict example: r1(X) and w2(X)
- Write-write conflict example: w1(Y) and w2(Y)

Problem-01:

Check whether the given schedule S is conflict serializable or not-

S : $R_1(A)$, $R_2(A)$, $R_1(B)$, $R_2(B)$, $R_3(B)$, $W_1(A)$, $W_2(B)$

T1	T2	T3
R1(A)		
	R2(A)	
R1(B)		
	R2(B)	
		R3(B)
W1(A)	W2(B)	

Solution-

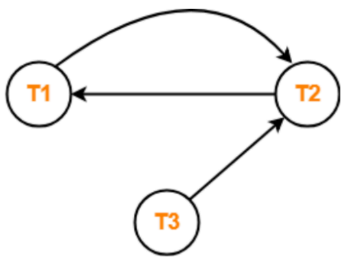
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

$R_2(A)$, $W_1(A)$	($T_2 \rightarrow T_1$)
$R_1(B)$, $W_2(B)$	($T_1 \rightarrow T_2$)
$R_3(B)$, $W_2(B)$	($T_3 \rightarrow T_2$)

Step-02:

Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

7. Explain concept of log-based recovery

Log-Based Recovery

Log is a sequence of records, which maintains the records of actions performed by a transaction. It is important that the logs are written prior to the actual modification and stored on a stable storage media, which is failsafe.

Log-based recovery works as follows –

- The log file is kept on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log about it.

<Tn, Start>

- When the transaction modifies an item X, it write logs as follows –

<Tn, X, V1, V2>

It reads Tn has changed the value of X, from V1 to V2.

- When the transaction finishes, it logs –

<Tn, commit>

The database can be modified using two approaches –

- **Deferred database modification** – All logs are written on to the stable storage and the database is updated when a transaction commits.
- **Immediate database modification** – Each log follows an actual database modification. That is, the database is modified immediately after every operation.

8. Explain Conflict and View Serializability in detail

→ Conflict Serializability

Definition: A schedule is conflict serializable if it can be transformed into a serial schedule (where transactions are executed one after the other) by swapping non-conflicting operations.

Conflict equivalent:
Two schedules are conflict equivalent if the relative order of any two conflicting
operations is the same in both schedules.
Commonly used definition of schedule equivalence

Two operations are conflicting if:

- They access the same data item X.
- They are from two different transactions.
- At least one is a write operation

Example:

- Read-Write conflict example: r1(X) and w2(X)
- Write-write conflict example: w1(Y) and w2(Y)

Problem-01:

Check whether the given schedule S is conflict serializable or not-

S : $R_1(A)$, $R_2(A)$, $R_1(B)$, $R_2(B)$, $R_3(B)$, $W_1(A)$, $W_2(B)$

T1	T2	T3
R1(A)		
	R2(A)	
R1(B)		
	R2(B)	
		R3(B)
W1(A)	W2(B)	

Solution-

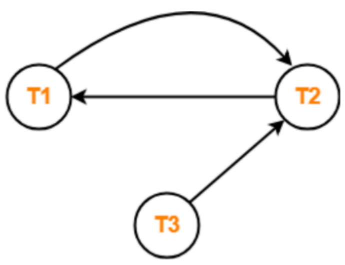
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

• $R_2(A)$, $W_1(A)$	$(T_2 \rightarrow T_1)$
• $R_1(B)$, $W_2(B)$	$(T_1 \rightarrow T_2)$
• $R_3(B)$, $W_2(B)$	$(T_3 \rightarrow T_2)$

Step-02:

Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

→ View Serializability

View Serializability

- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.
- The view serializable which does not conflict serializable contains blind writes.

View Equivalent

Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions:

1. Initial Read

An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.

T1	T2
Read(A)	
	Write(A)

Schedule S1

T1	T2
	Write(A)
Read(A)	

Schedule S2

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. Updated Read

In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.

T1	T2	T3
Write(A)		
	Write(A)	
		Read(A)

Schedule S1

T1	T2	T3
	Write(A)	
Write(A)		
		Read(A)

Schedule S2

Above two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

3. Final Write

A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.

T1	T2	T3
Write(A)		
	Read(A)	
		Write(A)

Schedule S1

T1	T2	T3
	Read(A)	
Write(A)		
		Write(A)

Schedule S2

Above two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

Example:

T1	T2	T3
Read(A)		
	Write(A)	
Write(A)		
		Write(A)

Schedule S

With 3 transactions, the total number of possible schedule

1. = 3! = 6
2. S1 = <T1 T2 T3>
3. S2 = <T1 T3 T2>
4. S3 = <T2 T3 T1>
5. S4 = <T2 T1 T3>
6. S5 = <T3 T1 T2>
7. S6 = <T3 T2 T1>

Taking first schedule S1:

T1	T2	T3
Read(A)		
Write(A)		
	Write(A)	
		Write(A)

Schedule S1

Step 1: final updation on data items

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2: Initial Read

The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3: Final Write

The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.

The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.

Hence, view equivalent serial schedule is:

1. T1 → T2 → T3

9. What is conflict serializability? Explain the conditions to check whether a transaction is conflict serializable or not.

→ **Conflict Serializability**

Definition: A schedule is conflict serializable if it can be transformed into a serial schedule (where transactions are executed one after the other) by swapping non-conflicting operations.

Conflict equivalent:
Two schedules are conflict equivalent if the relative order of any two conflicting
operations is the same in both schedules.
Commonly used definition of schedule equivalence

Two operations are conflicting if:

- They access the same data item X.
- They are from two different transactions.
- At least one is a write operation

Example:

- Read-Write conflict example: r1(X) and w2(X)
- Write-write conflict example: w1(Y) and w2(Y)

Problem-01:

Check whether the given schedule S is conflict serializable or not-

S : $R_1(A)$, $R_2(A)$, $R_1(B)$, $R_2(B)$, $R_3(B)$, $W_1(A)$, $W_2(B)$

T1	T2	T3
R1(A)		
	R2(A)	
R1(B)		
	R2(B)	
		R3(B)
W1(A)	W2(B)	

Solution-

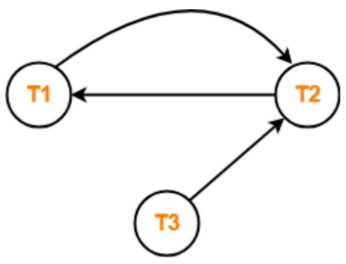
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

• $R_2(A)$, $W_1(A)$	$(T_2 \rightarrow T_1)$
• $R_1(B)$, $W_2(B)$	$(T_1 \rightarrow T_2)$
• $R_3(B)$, $W_2(B)$	$(T_3 \rightarrow T_2)$

Step-02:

Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

10. What is view serializability? Explain the conditions to check whether a transaction is view serializable or not.

View Serializability

- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.
- The view serializable which does not conflict serializable contains blind writes.

View Equivalent

Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions:

1. Initial Read

An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.

T1	T2
Read(A)	
	Write(A)

Schedule S1

T1	T2
	Write(A)
Read(A)	

Schedule S2

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. Updated Read

In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.

T1	T2	T3
Write(A)		
	Write(A)	
		Read(A)

Schedule S1

T1	T2	T3
	Write(A)	
Write(A)		
		Read(A)

Schedule S2

Above two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

3. Final Write

A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.

T1	T2	T3
Write(A)		
	Read(A)	
		Write(A)

Schedule S1

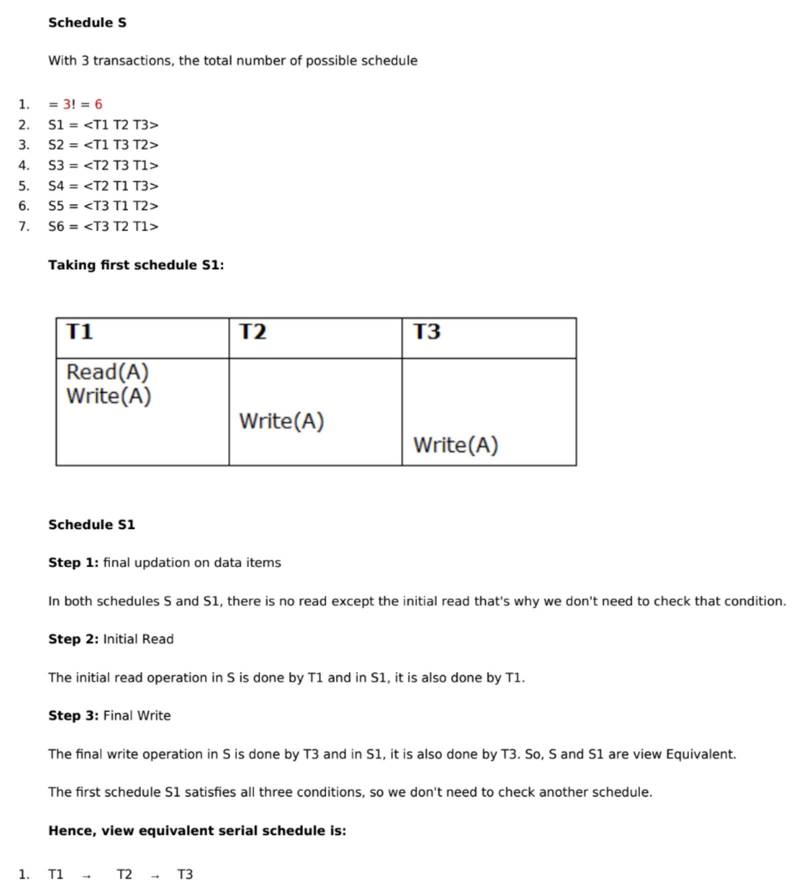
T1	T2	T3
	Read(A)	
Write(A)		
		Write(A)

Schedule S2

Above two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

Example:

T1	T2	T3
Read(A)		
	Write(A)	
Write(A)		
		Write(A)


Schedule S

With 3 transactions, the total number of possible schedule

1. = 3! = 6
2. S1 = <T1 T2 T3>
3. S2 = <T1 T3 T2>
4. S3 = <T2 T3 T1>
5. S4 = <T2 T1 T3>
6. S5 = <T3 T1 T2>
7. S6 = <T3 T2 T1>

Taking first schedule S1:

T1	T2	T3

Read(A)
Write(A)
	Write(A)	Write(A)

Schedule S1

Step 1: final updation on data items

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2: Initial Read

The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3: Final Write

The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.

The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.

Hence, view equivalent serial schedule is:

1. T1 → T2 → T3

11. CONSIDER RELATION Employee (Eid, Ename, Salary, Experience, Dept_name, Location) Write relational algebra queries for. Assume data wherever required.

a. Find names of employees whose location is 'Mumbai'

$$\pi Ename(\sigma Location = {'}Mumbai{'}(Employee))$$

b. Find min salary of employee from employee table

$$\gamma min(Salary)(Employee)$$

c. Find names of the employees whose Eid is greater than 3

$$\pi Ename(\sigma Eid > 3(Employee))$$

d. Display the name of employee working in department no 30

$$\pi Ename(\sigma Deptname = 30(Employee))$$

e. Rename column Eid to Eno

$$\rho Eno/Eid(Employee)$$

f. Find the total number of employees working in dept no 10

$$\gamma count(Eid)(\sigma Deptname = 10(Employee))$$

g. Find maximum salary of employee having experience more than 10 years

$$\gamma max(Salary)(\sigma Experience > 10(Employee))$$

12. Explain stored procedure, functions with example.

i. Procedures

Definition: A stored procedure is a precompiled collection of one or more SQL statements that can be executed as a single unit. Procedures can accept parameters and may return multiple results.

Characteristics:

Can perform operations like querying, inserting, updating, or deleting data.

Can have input and output parameters.

Do not return a value directly but can return multiple result sets.

Example:
Here's an example of a stored procedure that adds a new student record to a database:

```
CREATE PROCEDURE AddStudent(
    IN studentName VARCHAR(100),
    IN studentAge INT,
    OUT studentID INT
)
BEGIN
    INSERT INTO Students (Name, Age) VALUES (studentName, studentAge);
    SET studentID = LAST_INSERT_ID();
END;
```

Usage:

```
CALL AddStudent('John Doe', 20, @newStudentID);
SELECT @newStudentID;  -- Retrieves the ID of the newly added student.
```

ii. Functions

Definition: A function is a stored program that returns a single value. Functions can be called within SQL statements and are often used for computations.

Characteristics:

Must return a value.

Can be used in SELECT statements or other SQL expressions.

Cannot perform transactions like committing or rolling back.

Example:
Here's an example of a function that calculates the total price including tax:

```
CREATE FUNCTION CalculateTotalPrice(
    price DECIMAL(10, 2),
    taxRate DECIMAL(5, 2)
) RETURNS DECIMAL(10, 2)
```

```
BEGIN
    RETURN price + (price * taxRate);
END;
```

Usage:

```
SELECT CalculateTotalPrice(100.00, 0.07) AS TotalPrice;  -- Returns 107.00
```

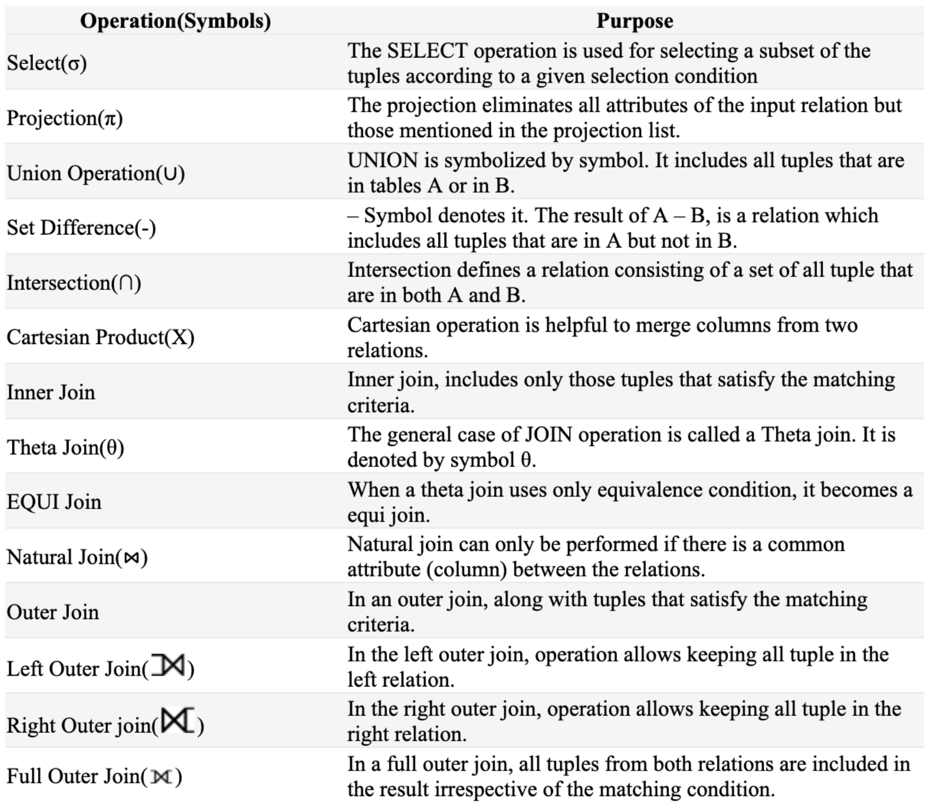
13. Define serializability. List various types of serializability .

Serializability is a concept in database systems that ensures that the outcome of executing a set of concurrent transactions is the same as if those transactions were executed in some sequential order. In other words, it guarantees that the end state of the database remains consistent and correct, even when transactions are executed concurrently.

Types of Serializability

1. Conflict Serializability
2. View Serializability

14. List and explain various operators in relational algebra. ***



Operation(Symbols)	Purpose
Select(σ)	The SELECT operation is used for selecting a subset of the tuples according to a given selection condition
Projection(π)	The projection eliminates all attributes of the input relation but those mentioned in the projection list.
Union Operation(∪)	UNION is symbolized by symbol. It includes all tuples that are in tables A or in B.
Set Difference(-)	– Symbol denotes it. The result of A – B, is a relation which includes all tuples that are in A but not in B.
Intersection(∩)	Intersection defines a relation consisting of a set of all tuple that are in both A and B.
Cartesian Product(X)	Cartesian operation is helpful to merge columns from two relations.
Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Theta Join(θ)	The general case of JOIN operation is called a Theta join. It is denoted by symbol θ.
EQUI Join	When a theta join uses only equivalence condition, it becomes a equi join.
Natural Join(⋈)	Natural join can only be performed if there is a common attribute (column) between the relations.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria.
Left Outer Join(⟕)	In the left outer join, operation allows keeping all tuple in the left relation.
Right Outer join(⟖)	In the right outer join, operation allows keeping all tuple in the right relation.
Full Outer Join(⟗)	In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition.

15. Explain trigger in SQL with one example.

A **Trigger** in SQL is a special kind of stored procedure that automatically runs (or "fires") in response to specific events on a particular table or view. Triggers can be used to enforce business rules, validate input data, maintain audit trails, and ensure data integrity.

Types of Triggers

1. **BEFORE Trigger:** Executes before an insert, update, or delete operation.
2. **AFTER Trigger:** Executes after an insert, update, or delete operation.
3. **INSTEAD OF Trigger:** Executes in place of an insert, update, or delete operation (often used with views).

Example of a Trigger

Let's say we have an `Employee` table, and we want to maintain an audit trail in a separate `EmployeeAudit` table every time an employee's salary is updated.

Now, we can create a trigger that logs the old and new salary whenever there is an update on the `Employee` table.

```
CREATE TRIGGER trg_AfterSalaryUpdate
AFTER UPDATE ON Employee
FOR EACH ROW
BEGIN
    IF OLD.Salary <> NEW.Salary THEN
        INSERT INTO EmployeeAudit (Eid, OldSalary, NewSalary)
        VALUES (OLD.Eid, OLD.Salary, NEW.Salary);
    END IF;
END;
```

Usage

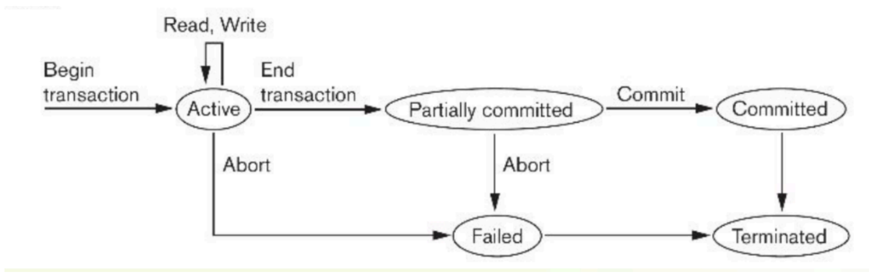
Whenever an employee's salary is updated, the trigger automatically logs the change:

```
UPDATE Employee
SET Salary = 60000
WHERE Eid = 1;
```

16. What is a transaction? Draw and explain the transaction state diagram.

A **transaction** in a database context is a sequence of one or more operations (like reads, writes, updates, or deletions) that are executed as a single logical unit of work. Transactions are fundamental to database systems because they ensure data integrity and consistency, even in the presence of failures or concurrent access by multiple users.

Transaction State Diagram



The transaction state diagram typically includes the following states:

Active: The initial state when the transaction is being executed. Operations can be performed, and it can be rolled back if needed.

Partially Committed: The state after the final operation has been executed but before the transaction has been committed. This state represents a transaction that has completed all its operations.

Failed: The state reached if the transaction cannot complete successfully due to an error or failure. At this point, the transaction may need to be rolled back.

Committed: The state indicating that the transaction has been successfully completed, and all changes have been made permanent.

Aborted: This state represents a transaction that has been rolled back due to failure or a decision by the user.

17. Explain timestamp-based protocol.

Timestamp Based Protocol:
The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.

Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one.

In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last 'read and write' operation was performed on the data item.

Timestamp Ordering Protocol

The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and write operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

- The timestamp of transaction Ti is denoted as TS(Ti).
- Read time-stamp of data-item X is denoted by R-timestamp(X).
- Write time-stamp of data-item X is denoted by W-timestamp(X).

Timestamp ordering protocol works as follows –

- **If a transaction Ti issues a read(X) operation –**
  - If TS(Ti) < W-timestamp(X) if T is old and write oprn done recently then reject R(x) by Ti
    - Operation rejected.
  - If TS(Ti) >= W-timestamp(X) if T is new and write oprn done earlier then accept R(x) by Ti
    - Operation executed.
  - All data-item timestamps updated.
- **If a transaction Ti issues a write(X) operation –**
  - If TS(Ti) < R-timestamp(X) if T is old and read oprn done recently then reject W(x) by Ti
    - Operation rejected.
  - If TS(Ti) < W-timestamp(X) if T is old and write oprn done earlier then reject W(x) by Ti
    - Operation rejected and Ti rolled back.

- Otherwise, operation executed.

18. Explain backup and recovery in transaction management.

Backup and recovery are critical components of transaction management in database systems. They ensure that data remains safe and can be restored after failures, whether those failures are due to hardware malfunctions, software issues, human error, or other catastrophic events. Here's an overview of both concepts:

Backup

Backup refers to the process of creating copies of database data to prevent data loss. Backups can be categorized into different types:

1. **Full Backup:**
   - A complete copy of the entire database is created. This type of backup is comprehensive but can be time-consuming and require substantial storage space.
   - Example: A nightly backup of the entire database.
2. **Incremental Backup:**
   - Only the data that has changed since the last backup (whether full or incremental) is copied. This is faster and uses less storage but requires more complex restoration processes.
   - Example: Backing up only the changes made to the database since the last backup.
3. **Differential Backup:**
   - This backup type captures all changes made since the last full backup. It requires more storage than incremental backups but is easier to restore.
   - Example: Backing up all changes since the last full backup.
4. **Continuous Data Protection (CDP):**
   - A more advanced method that captures changes in real-time, allowing for very recent data recovery.

Recovery

Recovery refers to the processes and techniques used to restore a database to a consistent state after a failure. This includes restoring data from backups and applying transaction logs. Recovery can be categorized into:

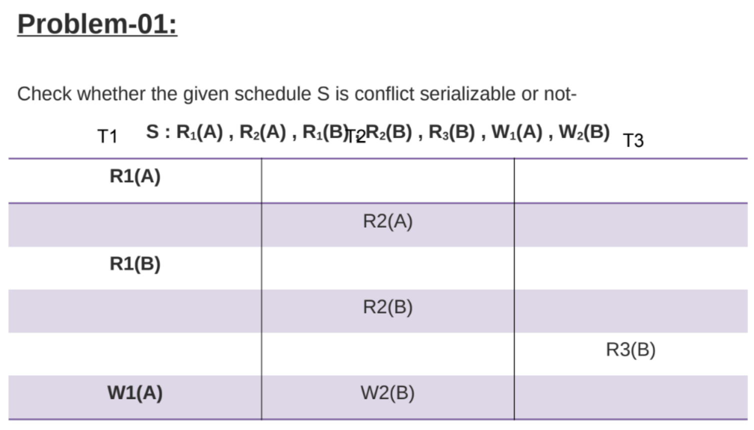
1. **Crash Recovery:**
   - This process is initiated when a system crashes unexpectedly. The database uses transaction logs to determine which transactions were completed and which were not.
   - Techniques include:
     - **Redo:** Re-applies changes from committed transactions.
     - **Undo:** Reverts changes made by uncommitted transactions.
2. **Media Recovery:**
   - This recovery method is used when the entire database or part of it is lost (e.g., due to hardware failure).
   - It often involves restoring the last full backup and applying transaction logs to bring the database to the most recent state.
3. **Point-in-Time Recovery:**
   - This allows the database to be restored to a specific moment before an incident, enabling recovery from human errors or application bugs.

Backup and Recovery Process

1. **Taking Backups:**
   - Regular backups are scheduled to minimize data loss. Backup strategies depend on business requirements for data availability and consistency.
2. **Using Transaction Logs:**
   - Transaction logs record all changes made to the database. They are essential for recovery, especially for ensuring atomicity and durability (part of the ACID properties).
3. **Restoring from Backups:**
   - In the event of a failure, the most recent backup is restored. If the backup is not recent enough, transaction logs are applied to recover changes made since the last backup.
4. **Testing Recovery Procedures:**
   - Regular testing of backup and recovery procedures is crucial to ensure that they work as expected when needed.

19. Solve the problems on serializabilty.

→ **Conflict Serializibility:**

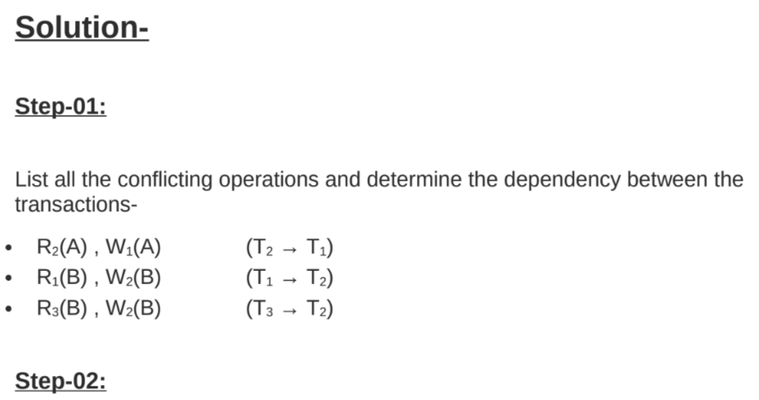


Problem-01:

Check whether the given schedule S is conflict serializable or not-

S : $R_1(A)$, $R_2(A)$, $R_1(B)$, $R_2(B)$, $R_3(B)$, $W_1(A)$, $W_2(B)$

T1	T2	T3
R1(A)		
	R2(A)	
R1(B)		
	R2(B)	
		R3(B)
W1(A)	W2(B)	



Solution-

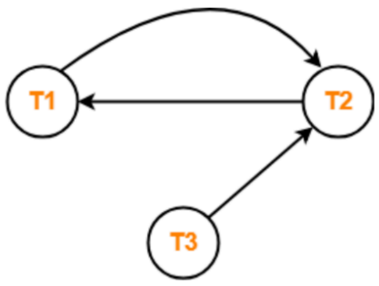
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

• $R_2(A)$, $W_1(A)$	$(T_2 \rightarrow T_1)$
• $R_1(B)$, $W_2(B)$	$(T_1 \rightarrow T_2)$
• $R_3(B)$, $W_2(B)$	$(T_3 \rightarrow T_2)$

Step-02:

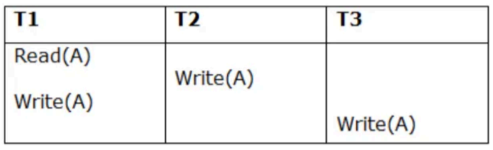
Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

→ View Serializibility

Example:



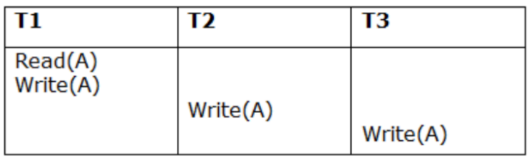
T1	T2	T3
Read(A)		
	Write(A)	
Write(A)		
		Write(A)

Schedule S

With 3 transactions, the total number of possible schedule

1. = 3! = 6
2. S1 = <T1 T2 T3>
3. S2 = <T1 T3 T2>
4. S3 = <T2 T3 T1>
5. S4 = <T2 T1 T3>
6. S5 = <T3 T1 T2>
7. S6 = <T3 T2 T1>

Taking first schedule S1:



T1	T2	T3
Read(A)		
Write(A)		
	Write(A)	
		Write(A)

Schedule S1

Step 1: final updation on data items

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2: Initial Read

The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3: Final Write

The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.

The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.

Hence, view equivalent serial schedule is:

1. T1 → T2 → T3

20. What do you mean by Schedule? Give one example.

In the context of database systems, a **schedule** refers to a sequence of operations (such as reads and writes) from one or more transactions that are executed in a specific order. Schedules are crucial for managing concurrency and ensuring that transactions are executed in a way that maintains the integrity of the database.

Example of a Schedule (any one):

- Schedule A: r1(X); w1(X); r1(Y); w1(Y); r2(X); w2(x);
- Schedule B: r2(X); w2(X); r1(X); w1(X); r1(Y); w1(Y);
- Schedule C: r1(X); r2(X); w1(X); r1(Y); w2(X); w1(Y);
- Schedule D: r1(X); w1(X); r2(X); w2(X); r1(Y); w1(Y);

21. Consider following relation & give relational algebra query for: Student (ssn,name,subject,DOB) Course (course_id,name,dept) Enrol (ssn,course_id,semester,grade)

i. Get the details of all students who registered for the course having id as 10.

$$\pi ssn, name, subject, DOB(\sigma course_id = 10(Enrol) \bowtie Student)$$

ii. Get the details of all students having grade as 'A'.

$$\pi ssn, name, subject, DOB(\sigma grade = \prime A \prime(Enrol) \bowtie Student)$$

iii. Get the details of all courses of semester 3.

$$\pi course_id, name, dept(\sigma semester = 3(Enrol) \bowtie Course)$$

22. Define and explain concurrency control Protocols.

Concurrency control protocols are essential mechanisms in database management systems (DBMS) that manage the simultaneous execution of transactions to ensure data integrity and consistency. They prevent issues that can arise when multiple transactions access and manipulate the same data concurrently, such as lost updates, dirty reads, and uncommitted data.

Types of Concurrency Control Protocols

Lock-based Protocols:

- Use locks to control access to data items.
- **Types of Locks:**
    - **Shared Lock (S-lock):** Allows multiple transactions to read a data item but not modify it.
    - **Exclusive Lock (X-lock):** Allows a transaction to read and modify a data item. Only one transaction can hold an exclusive lock on a data item at a time.
- **Two-Phase Locking (2PL):** A widely used protocol that consists of two phases:
    - **Growing Phase:** A transaction can obtain locks but cannot release any.
    - **Shrinking Phase:** A transaction can release locks but cannot acquire any new locks.
- **Advantages:** Simple to understand and implement.

- **Disadvantages:** Can lead to deadlocks, where two or more transactions wait indefinitely for each other to release locks.

Timestamp-based Protocols:

- Each transaction is assigned a unique timestamp, which determines the order of execution.
- Transactions are executed based on their timestamps to ensure serializability:
  - If a transaction with a later timestamp tries to access a data item that was modified by an earlier transaction, it is aborted.
- **Advantages:** No deadlocks, as transactions are ordered by their timestamps.
- **Disadvantages:** High rollback rates may occur if many transactions are aborted due to conflicts.

23. Explain cursor with its syntax and an example.

A **cursor** is a database object that allows for the retrieval and manipulation of rows returned by a SQL query one at a time. It is particularly useful when processing data row by row in stored procedures or complex operations.

Syntax

1. **Declare the Cursor:**

   ```
   DECLARE cursor_name CURSOR FOR
   SELECT column1, column2 FROM table_name WHERE condition;
   ```

2. **Open the Cursor:**

   ```
   OPEN cursor_name;
   ```

3. **Fetch Data:**

   ```
   FETCH NEXT FROM cursor_name INTO variable1, variable2;
   ```

4. **Close the Cursor:**

   ```
   CLOSE cursor_name;
   ```

5. **Deallocate the Cursor:**

   ```
   DEALLOCATE cursor_name;
   ```

Example

Scenario: Retrieving employee names and salaries from the `Employees` table.

```
DECLARE EmployeeCursor CURSOR FOR
SELECT Name, Salary FROM Employees;

DECLARE @Name VARCHAR(50);
DECLARE @Salary DECIMAL(10, 2);

OPEN EmployeeCursor;
```

```
FETCH NEXT FROM EmployeeCursor INTO @Name, @Salary;

WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT 'Employee Name: ' + @Name + ', Salary: ' + CAST(@Salary AS VARCHAR(10));
    FETCH NEXT FROM EmployeeCursor INTO @Name, @Salary;
END;

CLOSE EmployeeCursor;
DEALLOCATE EmployeeCursor;
```

Mohammed Anas Nathani