CMR
INSTITUTE
OFTECHNOLOGY

SCHEMEOF EVALUATION

InternalAssessmentTest3–March2024

Sub:	**DATABASEMANAGEMENTSYSTEMS**		**Code:**	21CS53
Date: 15/3/2024 Duration: 90mins MaxMarks: 50		**Sem:**	**Branch:**	AIML
		5		

Note: ***Answeranyfivefullquestions.*** ***(5X10= 50)***

Question No.		**Description**	**MarksSplitup**		**Total Marks**
1.	a	✓ Multivalued dependency ✓ Fourth normal form	2 3	5M	10M
	b	✓ Diagram ✓ Explanation of each state	2 3	5M	
2.	a)	✓ Explation of 2 phase protocol with example	10	10M	10M
3.	a)	✓ Functional dependency	5	10M	10M
	b)	Algorithm for minimal cover	5		
4.		Numerical 1 Numerical 2	5 5	10M	10M
5.	a)	Explain each anomalies with example	10 M	10M	10M
6.		✓ Explain each transaction property with example	10M		10M

USN | | | | | | | | | | |

Internal Assessment Test 3 – March 2024

Sub:	Database Management System					Sub Code:	21CS53	Branch:	AIML	
Date:	15/03/24	Duration:	90 minutes	Max Marks:	50	Sem/Sec:	V			OBE

		Answer any FIVE FULL Questions	MARKS	CO	RBT
1	a	What do you mean by multivalued dependency? Describe 4NF with example	[5]	4	L2
	b	Draw state transition diagram of a transaction. Explain different states of a transaction.	[5]	3	L1
2		What is two-phase locking protocol? How does it guarantee serializability?	[10]	3	L2
3	a	Define Functional dependency with example.	[5]	4	L2
	b	Define Minimal cover. Write an algorithm for finding a minimal cover G for a set of functional dependencies F.	[5]	4	L2
4	a	A relation R (A, C, D, E, H) satisfies the following FDs. A→C AC→D E→AD E→H. Find the canonical cover for this set of FDs.	[5]	4	L3
	b	Given below two sets of FDs for a relation R (A, B, C, D, E). Are they equivalent? i) A→B AB→C D→AC D→E ii)A→BC D→AE	[5]	4	L3
5		What are the anomalies that can occur due to concurrent execution of transactions? Explain them with example.	[10]	4	L1
6		Discuss the desirable properties of transaction.	[10]	3	L2

CI **CCI** **HoD**

CI **CCI** **HoD**

AlltheBest

1(a) What do you mean by multivalued dependency? Describe 4NF with example

Ans. What is Multivalued Dependency in DBMS?

Multivalued dependency would occur whenever two separate attributes in a given table happen to be independent of each other. And yet, both of these depend on another third attribute. The multivalued dependency contains at least two of the attributes dependent on the third attribute. This is the reason why it always consists of at least three of the attributes.

Example

Suppose that there is a car manufacturing company that produces two of the colours in the market, i.e., red and yellow of each of their models, every year.

CAR_MODEL	MANUF_MONTH	COLOUR
S2011	JAN	Yellow

S2001	FEB	Red
S3001	MAR	Yellow
S3001	APR	Red
S4006	MAY	Yellow
S4006	JUN	Red

In this case, the columns COLOUR and MANUF_MONTH are dependent on CAR_MODEL, and they are independent of each other. Thus, we can call both of these columns multivalued. These are, as a result, dependent on CAR_MODEL. Here is a representation of the dependencies we discussed above:

CAR_MODEL → → MANUF_MONTH

CAR_MODEL → → COLOUR

We can read this as "CAR_MODEL multidetermined MANUF_MONTH" and "CAR_MODEL multidetermined COLOUR".

Fourth Normal Form (4NF)

Fourth Normal Form (4NF) is a level of database normalization that requires a relation to be in BCNF and have no non-trivial multivalued dependencies other than the candidate key, to eliminate redundant data and maintain data consistency. If a table violates this standard, it needs to be split into two tables to achieve 4NF.

For a relation R to be in 4NF, it must meet two conditions −

1. It should be in Boyce-Codd Normal Form (BCNF).
2. It should not have any non-trivial multivalued dependencies.

Example:

To remove the multivalued dependency (MVD) in the "Students" table example, we can create two new tables, one for "Courses" and another for "Textbooks," and establish a relationship between them using foreign keys.

Here's how we can create the tables:

Table 1: Students

Student ID	Course ID
1	1
1	2
2	3
2	4
2	5

Student ID	Course ID
3	6

Table 2: Courses

Course ID	Course Name
1	Math
2	Science
3	Art
4	History

Table 3: Textbooks

Textbook ID	Textbook Name	Course ID
1	Algebra	1
2	Calculus	2
3	Biology	2
4	Chemistry	2
5	Art History	3
6	American History	4

So, we removed the multivalued dependency by splitting the “Course” and “Textbook” columns into separate tables.

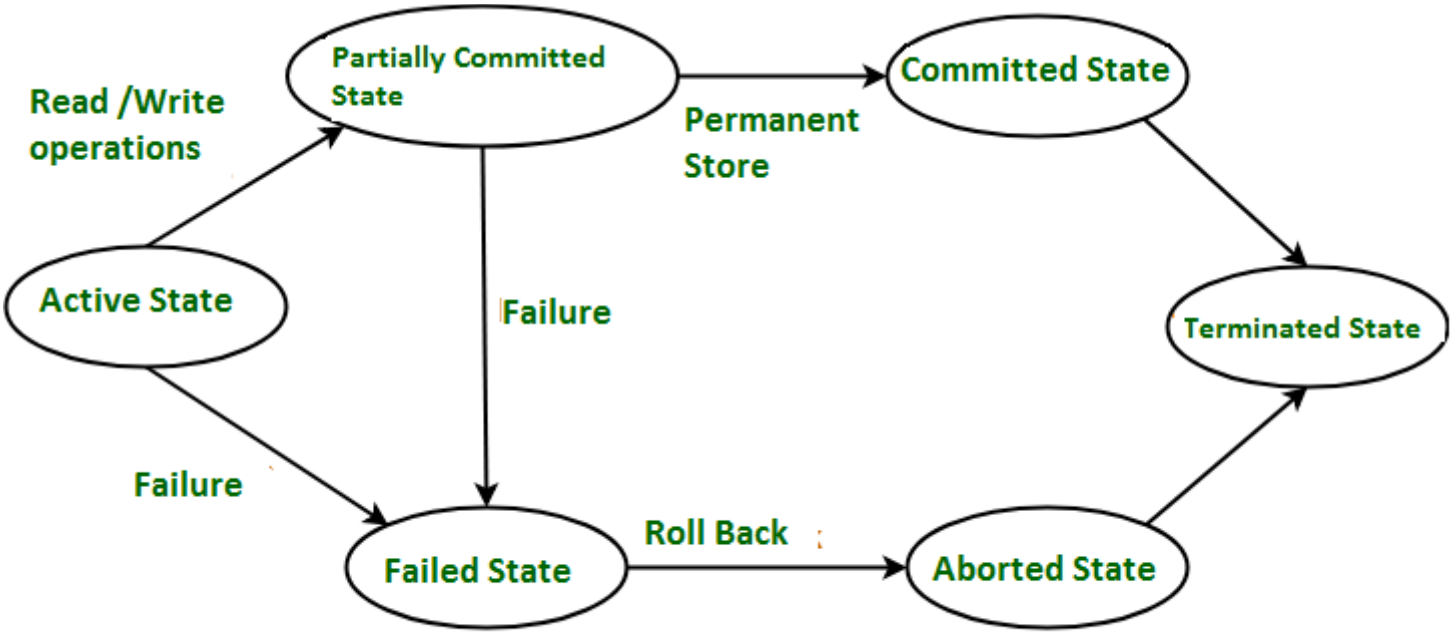
We have also added a new “Course ID” column to the “Students” table. It has a foreign key that references the “Course ID” column in the “Courses” table. Similarly, the “Textbooks” table also has a “Course ID” column that serves as a foreign key referencing the “Course ID” column in the “Courses” table.

Hence, we have achieved the fourth normal form (4NF) for the “Students” table. It has done after by removing the multivalued dependency and creating separate tables. The Resultant schema eliminates data redundancy and improves data integrity, making it easier to manage and query the database.

1(b) Draw state transition diagram of a transaction. Explain different states of a transaction.

Ans. States through which a transaction goes during its lifetime. These are the states which tell about the current state of the Transaction and also tell how we will further do the processing in the transactions. These states govern the rules which decide the fate of the transaction whether it will commit or abort.

They also use **Transaction log.** Transaction log is a file maintain by recovery management component to record all the activities of the transaction. After commit is done transaction log file is removed.



Transaction States in DBMS

These are different types of Transaction States :

1. **Active State –**
   When the instructions of the transaction are running then the transaction is in active state. If all the 'read and write' operations are performed without any error then it goes to the "partially committed state"; if any instruction fails, it goes to the "failed state".

2. **Partially Committed –**
   After completion of all the read and write operation the changes are made in main memory or local buffer. If the changes are made permanent on the DataBase then the state will change to "committed state" and in case of failure it will go to the "failed state".

3. **Failed State –**
   When any instruction of the transaction fails, it goes to the "failed state" or if failure occurs in making a permanent change of data on Data Base.

4. **Aborted State –**
   After having any type of failure the transaction goes from "failed state" to "aborted state" and since in previous states, the changes are only made to local buffer or main memory and hence these changes are deleted or rolled-back.

5. **Committed State –**
   It is the state when the changes are made permanent on the Data Base and the transaction is complete and therefore terminated in the "terminated state".

6. **Terminated State –**
   If there isn't any roll-back or the transaction comes from the "committed state", then the system is consistent and ready

for new transaction and the old transaction is terminated.

2 What is two-phase locking protocol? How does it guarantee serializability?

Ans. Two Phase Locking
A transaction is said to follow the Two-Phase Locking protocol if Locking and Unlocking can be done in two phases.

- **Growing Phase:** New locks on data items may be acquired but none can be released.
- **Shrinking Phase:** Existing locks may be released but no new locks can be acquired.

Note: If lock conversion is allowed, then upgrading of lock(from S(a) to X(a)) is allowed in the Growing Phase, and downgrading of lock (from X(a) to S(a)) must be done in the shrinking phase.

Let's see a transaction implementing 2-PL.

	T1	**T2**
1	lock-S(A)	
2		lock-S(A)
3	lock-X(B)	
4	…….	…….
5	Unlock(A)	
6		Lock-X(C)
7	Unlock(B)	
8		Unlock(A)
9		Unlock(C)
10	…….	…….

This is just a skeleton transaction that shows how unlocking and locking work with 2-PL. Note for:

Transaction T1

- The growing Phase is from steps 1-3
- The shrinking Phase is from steps 5-7
- Lock Point at 3

Transaction T2

- The growing Phase is from steps 2-6

- The shrinking Phase is from steps 8-9
- Lock Point at 6

Lock Point

The Point at which the growing phase ends, i.e., when a transaction takes the final lock it needs to carry on its work. Now look at the schedule, you'll surely understand. I have said that 2-PL ensures serializability, but there are still some drawbacks of 2-PL. Let's glance at the drawbacks.

- Cascading Rollback is possible under 2-PL.
- Deadlocks and Starvation are possible.

Cascading Rollbacks in 2-PL

Let's see the following Schedule

Cascading Roll-Back

Take a moment to analyze the schedule. Yes, you're correct, because of Dirty Read in T2 and T3 in lines 8 and 12 respectively, when T1 failed we have to roll back others also. Hence, Cascading Rollbacks are possible in 2-PL. I have taken skeleton schedules as examples because it's easy to understand when it's kept simple. When explained with real-time transaction problems with many variables, it becomes very complex.

Deadlock in 2-PL

Consider this simple example, it will be easy to understand. Say we have two transactions T1 and T2.

Schedule: Lock-X1(A) Lock-X2(B) Lock-X1(B) Lock-X2(A)

Drawing the precedence graph, you may detect the loop. So Deadlock is also possible in 2-PL.

Two-phase locking may also limit the amount of concurrency that occurs in a schedule because a Transaction may not be able to release an item after it has used it. This may be because of the protocols and other restrictions we may put on the schedule to ensure serializability, deadlock freedom, and other factors. This is the price we have to pay to ensure serializability and other factors, hence it can be considered as a bargain between concurrency and maintaining the ACID properties.

The above-mentioned type of 2-PL is called **Basic 2PL**. To sum it up it ensures Conflict Serializability but **does not** prevent Cascading Rollback and Deadlock. Further, we will study three other types of 2PL, **Strict 2PL**, **Conservative 2PL**, and **Rigorous 2PL**.

Problem with Two-Phase Locking

- It does not insure recoverability which can be solved by strict two-phase locking and rigorous two-phase locking.
- It does not ensure a cascade-less schedule which can be solved by strict two-phase locking and rigorous two-phase locking.
- It may suffer from deadlock which can be solved by conservative two-phase locking.

3(a) Define Functional dependency with example.

Ans. The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

1. $X \rightarrow Y$

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

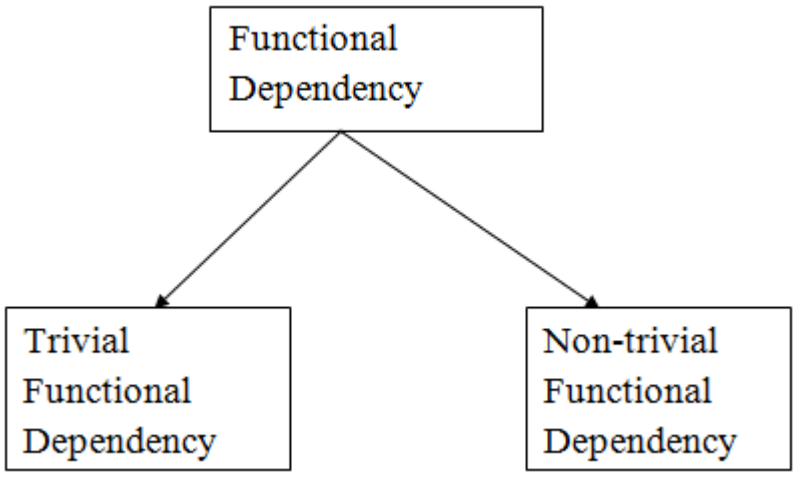
Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

1. Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

Types of Functional dependency



1. Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B

Example:

1. Consider a table with two columns Employee_Id and Employee_Name.
2. {Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as
3. Employee_Id is a subset of {Employee_Id, Employee_Name}.
4. Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial dependencies too.

2. Non-trivial functional dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then A → B is called as complete non-trivial.

Example:

1. ID → Name,
2. Name → DOB

3(b) Define Minimal cover. Write an algorithm for finding a minimal cover G for a set of functional dependencies F.

Ans. If we have a set of functional dependencies, we get the simplest and irreducible form of functional dependencies after reducing these functional dependencies. This is called the **Minimal Cover** or **Irreducible Set** (as we can't reduce the set further). It is also called a **Canonical Cover**.

Let us understand the procedure to find the minimal cover by this example:

The Given Functional Dependencies are – A->B, B ->C, D->ABC, AC-> D

We can find the minimal cover by following the 3 simple steps.

Step: 1 First split the all left-hand attributes of all FDs (functional dependencies).

A->B, B->C, D->A, D->B, D->C, AC->D

[Note: We can't split AC->D as A->D, C->D]

Step: 2 Now remove all redundant FDs.

[Redundant FD is if we derive one FD from another FD]

Let, 's test the redundance of A->B

A+ = A (A is only closure contains to A,

simply we can derive A from A)

So, A->B is not redundant.

Similarly, B->C is not redundant.

But, D->B and D->C is redundant

because D+= A and A+=B, So D+=B can be

derived which means D->B is redundant.

So, We remove D->B from

the FDs set.

Now, check for D->C, it is not redundant.

because we can't D+=B and B+=C as we remove D->B from the list.

At last, we check for AC->D. This is also not redundant.

AC+=AC

So, the final FDs are: A->B, B->C, D->A,D->C, AC->D

Step: 3 Find the Extraneous attribute and remove it.

In this case, we should only check ->D. Simply

we can say the right-hand attributes are pointed

by only one attribute at one time.

AC->D, either A or C, or none can be extraneous.

If A=+ C then C is extraneous and it can be removed.

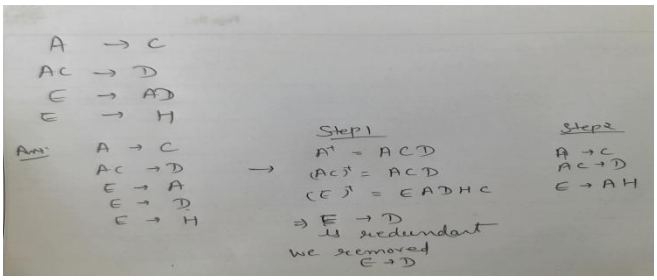
If C+=A then A is extraneous and it can be removed.

So, the final FDs are: A->B, B->C, D->A,D->C, AC->D

Hence, we can write it as A->B, B->C, D->AC, AC->D this is the minimum cover.

4 (a) A relation R (A, C, D, E, H) satisfies the following FDs. A→C AC→D E→AD E→H. Find the canonical cover for this set of FDs.

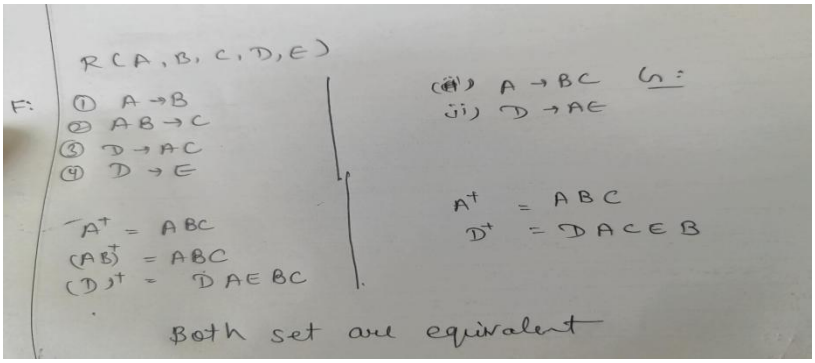
Ans.



4(b) Given below two sets of FDs for a relation R (A, B, C, D, E). Are they equivalent?

i) A→B AB→C D→AC D→E ii)A→BC D→AE

Ans.



5 What are the anomalies that can occur due to concurrent execution of transactions? Explain them with example.

Ans. Concurrency Problem

Concurrency problems are common problems faced while maintaining any database management systems. Concurrency problems occur when multiple transactions are being executed on the same database in unrestricted problems.

Types of concurrency problems in DBMS are as follows:

- Lost Update Problem(write-write conflict)
- Temporary Update Problem(dirty read problem)
- Incorrect Summary Problem
- Unrepeatable Read Problem
- Phantom Read Problem

Lost Update Problem:

- A lost update problem occurs due to the update of the same record by two different transactions at the same time.
- In simple words, when two transactions are updating the same record at the same time in a DBMS then a lost update problem occurs. The first transaction updates a record and the second transaction updates the same record again, which nullifies the update of the first transaction. As the update by the first transaction is lost this concurrency problem is known as the lost update problem.

		X	
Transaction A	**Transaction B**	**A**	**B**
Read (X)		X = 100	
X = X+15		115	
	Read (X)		X = 100
	x = x-25		X = 75
	WRITE (X)		X = 75
WRITE (X)		X = 115	

What Are the Different Ways to Prevent Lost Updates?

As we can see from the above example, the lost update problem occurs in DBMS due to concurrent transactions being performed on a DB and a lack of control over these transactions.

But, a lost update problem can prove fatal in the case of critical databases like banking databases. And we need to control and prevent lost update concurrency problems. In this section, we will see numerous ways to prevent lost update problem and ensure that update from any transaction is not lost.

Increase Transaction Isolation Level

Increasing the isolation level of transactions on a database is one of the ways to prevent lost update problems. Isolation is **I** in ACID properties. This method ensures that the isolation level of transactions on a database is increased to 'Repeatable Read' so that database can perform efficient checks in conjunction. The method to set the isolation level to 'Repeatable Read' is different in multiple database systems.

Optimistic Locking

Optimistic locking is the most common method used to prevent lost update problems. This method allows any update of a record to happen only when the value of that record has not changed after its last read. Optimistic Locking checks if the current value of the record is the same as it was when previously read then the update is not allowed, and the read-modify-write cycle has to be repeated. Optimistic Locking is also called Conditional Update or Compare-And-Update.

Pessimistic Locking

Pessimistic locking is also a way to prevent lost update problems in DBMS. In this approach, the DB objects that are going to be updated are explicitly locked using the 'PESSIMISTICWRITE' mode. After locking the object read-modify-write operations are performed on the objects and then the object is released. During these operations, if another transaction tries to read the same object it has to wait until the read-modify-write cycle of the first transaction is completed.

Atomic write operations

One of the many ways to prevent lost update problems is Atomic write operations. Atomic write operations read records at a memory location and write a new value into it simultaneously. Atomic write operations remove the need to implement read-modify-write cycles repeatedly. In this method a record is read and written at the same time, hence the lost update problem is avoided.

Dirty Read Problem

The dirty read problem in DBMS occurs when *a transaction reads the data that has been updated by another transaction that is still uncommitted.* It arises due to multiple uncommitted transactions executing simultaneously.

Example: Consider two transactions A and B performing read/write operations on a data DT in the database DB. The current value of DT is 1000: The following table shows the read/write operations in A and B transactions.

Time	A	B
T1	READ(DT)	------
T2	DT=DT+500	------
T3	WRITE(DT)	------
T4	------	READ(DT)
T5	------	COMMIT
T6	ROLLBACK	------

Transaction A reads the value of data DT as 1000 and modifies it to 1500 which gets stored in the temporary buffer. The transaction B reads the data DT as 1500 and commits it and the value of DT permanently gets changed to 1500 in the database DB. Then some server errors occur in transaction A and it wants to get rollback to its initial value, i.e., 1000 and then the dirty read problem occurs.

Unrepeatable Read Problem

The unrepeatable read problem occurs when two or more different values of the same data are read during the read operations in the same transaction.

Example: Consider two transactions A and B performing read/write operations on a data DT in the database DB. The current value of DT is 1000: The following table shows the read/write operations in A and B transactions.

Time	A	B
T1	READ(DT)	------
T2	------	READ(DT)
T3	DT=DT+500	------
T4	WRITE(DT)	------
T5	------	READ(DT)

Transaction A and B initially read the value of DT as 1000. Transaction A modifies the value of DT from 1000 to 1500 and then again transaction B reads the value and finds it to be 1500. Transaction B finds two different values of DT in its two different read operations.

Phantom Read Problem

In the phantom read problem, data is read through two different read operations in the same transaction. In the first read operation, a value of the data is obtained but in the second operation, an error is obtained saying the data does not exist.

Example: Consider two transactions A and B performing read/write operations on a data DT in the database DB. The current value of DT is 1000: The following table shows the read/write operations in A and B transactions.

Time	A	B
T1	READ(DT)	------
T2	------	READ(DT)
T3	DELETE(DT)	------
T4	------	READ(DT)

Transaction B initially reads the value of DT as 1000. Transaction A deletes the data DT from the database DB and then again transaction B reads the value and finds an error saying the data DT does not exist in the database DB.

Incorrect Summary Problem

The Incorrect summary problem occurs when there is an incorrect sum of the two data. This happens when a transaction tries to sum two data using an aggregate function and the value of any one of the data get changed by another transaction.

Example: Consider two transactions A and B performing read/write operations on two data DT1 and DT2 in the database DB. The current value of DT1 is 1000 and DT2 is 2000: The following table shows the read/write operations in A and B transactions.

Time	A	B
T1	READ(DT1)	------
T2	add=0	------
T3	add=add+DT1	------
T4	------	READ(DT2)
T5	------	DT2=DT2+500
T6	READ(DT2)	------
T7	add=add+DT2	------

Transaction A reads the value of DT1 as 1000. It uses an aggregate function SUM which calculates the sum of two data DT1 and DT2 in variable add but in between the value of DT2 get changed from 2000 to 2500 by transaction B. Variable add uses the modified value of DT2 and gives the resultant sum as 3500 instead of 3000.

6 Discuss the desirable properties of transaction.

Ans. What are the ACID properties of a transaction?

ACID is an acronym that stands for atomicity, consistency, isolation, and durability.

Together, these ACID properties ensure that a set of database operations (grouped together in a transaction) leave the database in a valid state even in the event of unexpected errors.

Atomicity

Atomicity guarantees that all of the commands that make up a transaction are treated as a single unit and either succeed or fail together. This is important as in the case of an unwanted event, like a crash or power outage, we can be sure of the state of the database. The transaction would have either completed successfully or been rollbacked if any part of the transaction failed.

If we continue with the above example, money is deducted from the source and if any anomaly occurs, the changes are discarded and the transaction fails.

Consistency

Consistency guarantees that changes made within a transaction are consistent with database constraints. This includes all rules, constraints, and triggers. If the data gets into an illegal state, the whole transaction fails.

Going back to the money transfer example, let's say there is a constraint that the balance should be a positive integer. If we try to overdraw money, then the balance won't meet the constraint. Because of that, the consistency of the ACID transaction will be violated and the transaction will fail.

Isolation

Isolation ensures that all transactions run in an isolated environment. That enables running transactions concurrently because transactions don't interfere with each other.

For example, let's say that our account balance is $200. Two transactions for a $100 withdrawal start at the same time. The transactions run in isolation which guarantees that when they both complete, we'll have a balance of $0 instead of $100.

Durability

Durability guarantees that once the transaction completes and changes are written to the database, they are persisted. This ensures that data within the system will persist even in the case of system failures like crashes or power outages.