USN | | | | | | | | | |

Internal Assessment Test 3 – March 2024

Sub:	Database Management System				Sub Code:	21CS53	Branch:	CSE
Date:	13 /3/2024	Duration:	90 mins	Max Marks: 50	Sem / Sec:	V/ A, B, C		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	i. Illustrate types of update anomalies with example. ii. A relation R (P, Q, U, S, T) is having two functional dependencies sets R1= {P→Q, PQ→U, S→T} ; R2= {P→QU, S→PT}. Are they equivalent? Justify your answer with proper reasoning.	[4+6]	CO4	L3
2	i. List the properties to be satisfied by a relation in 1NF, BCNF, 4NF and 5NF ii.Write the algorithm for finding a minimal cover for a set of functional dependencies. Consider a relation R (A, B, C, D) having some attributes and FDs as: {B→A, AD→C, C→ABD}. Find the canonical cover for this set of FDs.	[2+8]	CO4	L3
3	Write the algorithm for Dependency-Preserving and Nonadditive (Lossless) Join Decomposition into 3NF Schemas. Consider a relation R (A, B, C, D, E, F, G, H) having FDs as: {A→B, ABCD→E, EF→GH, ACDF→EG}. i. Find candidate key of this relation R. ii.Preserving the dependency, decompose R into 3NF.	[2+8]	CO4	L3
4	What type of problems may arise if concurrency control and recovery are not maintained in DBMS transaction? Justify your answer with proper example.	[10]	CO4	L2
5	Explain 2PL protocol for concurrency control. How does it guarantee serializability? Justify your answer with a suitable example.	[10]	CO4	L2
6a	Explain the basic time stamping protocol for concurrency control.	[4]	CO4	L2
6b	Consider the below mentioned schedule with transactions T1, T2, T3 with read () and write () operations on data items X, Y, Z. Draw the precedence graph for the schedule and state whether the schedule is serializable or not. If the schedule is serializable, write down the serial schedule. **S1:** r3(Y); r3(Z); r1(X); w1(X); w3(Y); w3(Z); r2(Z), r1(Y); w1(Y); r2(Y); w2(Y); r2(X); w2(X).	[6]	CO4	L3

Model Solution

1) i) Illustrate types of update anomalies with example.

***update anomalies**

- insertion anomalies
- deletion anomalies,
- modification anomalies

Insertion Anomalies

□**Insertion anomalies** can be differentiated into **two types**, illustrated by the following examples based on the EMP_DEPT relation:

EMP_DEPT (Redundancy: Dname, Dmgr_ssn)

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555

1. To insert a new employee tuple into EMP_DEPT, we must include either the attribute values for the department that the employee works for, or NULLs

- For example, to insert a new tuple for an employee who works in department number 5, we must enter all the attribute values of department 5 correctly so that they are consistent with the corresponding values for department 5 in other tuples in EMP_DEPT
- In the design of Employee in fig 1, we do not have to worry about this consistency problem because we enter only the department number in the employee tuple; all other attribute values of department 5 are recorded only once in the database, as a single tuple in the DEPARTMENT relation

Deletion Anomalies

□ The problem of deletion anomalies is related to the second insertion anomaly situation just discussed

- If we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database
- This problem does not occur in the database of Figure 2 because DEPARTMENT tuples are stored separately.

Modification Anomalies

□ In EMP_DEPT, if we change the value of one of the attributes of a particular department—say, the manager of department 5—we must update the tuples of all employees who work in that department; otherwise, the database will become inconsistent

□ If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would be wrong

ii) A relation R (P, Q, U, S, T) is having two functional dependencies sets R1= {P→Q, PQ→U, S→T} ; R2= {P→QU, S→PT}. Are they equivalent? Justify your answer with proper reasoning.

Ans:

Case 1) Determining Whether R2 ⊃ R1 or not

Step 1)

- (P)+ = {P, Q, U} // closure of left side of P → QU using set R1.
- (S)+ = {P, Q, U, S, T} // closure of left side of S → PT using set R1.

Step 2)

- $(P)^+ = \{P, Q, U\}$ // closure of left side of P → QU using set R2.
- $(S)^+ = \{P, Q, U, S, T\}$ // closure of left side of S → PT using set R2.

Step 3)

Comparing the results of Step 1 and Step 2, we find,

- Functional dependencies of set R2 can determine all the attributes which have been determined by the functional dependencies of set R1.
- Thus, we conclude R2 is a subset of R1 i.e. R2 ⊃ R1.

Case 2) Determining Whether R1 ⊃ R2 or not

Step 1)

- $(P)^+ = \{P, Q, U\}$ // closure of left side of P→ Q using set R2.
- $(PQ)^+ = \{P, Q, U\}$ // closure of left side of PQ → U using set R2.
- $(S)^+ = \{P, Q, U, S, T\}$ // closure of left side of S → PU and S → T using set R2.

Step 2)

- $(P)^+ = \{P, Q, U\}$ // closure of left side of P→ Q using set R1.
- $(PQ)^+ = \{P, Q, U\}$ // closure of left side of PQ → U using set R1.
- $(S)^+ = \{P, Q, U, S, T\}$ // closure of left side of S → PU and S → T using set R1.

Step 3)

Comparing the results of Step 1 and Step 2, we find,

- Functional dependencies of set R1 can determine all the attributes which have been determined by the functional dependencies of set R2.
- Thus, we conclude R1 is a subset of R2 i.e. R1 ⊃ R2.

Case 3) Determining Whether Both R1 and R2 satisfy each other or not

- From Step 1, we conclude R2 ⊃ R1.
- From Step 2, we conclude R1 ⊃ R2.

  Thus, we conclude that both R1 and R2 satisfied each other i.e. R1 = R2.

2)i) List the properties to be satisfied by a relation in 1NF, BCNF, 4NF and 5NF

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.
4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

ii) Consider a relation R (A, B, C, D) having some attributes and FDs as: {B→A, AD→C, C→ABD}. Find the canonical cover for this set of FDs.

Step-1 : Decompose the functional dependencies using Decomposition rule(Armstrong's Axiom) i.e. single attribute on right hand side.

FD1 : B A

FD2 : AD C

FD3 : C A

FD4 : C B

FD5 : C D

Step-2 : Remove extraneous attributes from LHS of functional dependencies by calculating the closure of FD's having two or more attributes on LHS.

Here, only one FD has two or more attributes of LHS i.e. AD C.

{A}+ = {A}

{D}+ = {D} In this case, attribute "A" can only determine "A" and "D" can only determine "D". Hence, no extraneous attributes are present and the FD will remain the same and will not be removed.

Step-3 : Remove FD's having transitivity.

FD1 : B A

FD2 : C A

FD3 : C B

FD4 : AD C

FD5 : C D . Above FD1, FD2 and FD3 are forming transitive pair. Hence, using Armstrong's law of transitivity i.e. if X Y, Y X then X Z should be removed. Therefore we will have the following FD's left :

FD1 : B A

FD2 : C B

FD3 : AD C

FD4 : C D

Also, FD2 & FD4 can be clubbed together now. Hence, the canonical cover of the relation R(A,B,C,D) will be:

Mc {R(ABCD)} = {B A , C BD, AD C}

3) Write the algorithm for Dependency-Preserving and Nonadditive (Lossless) Join Decomposition into 3NF Schemas.

Algorithm:

Relational Synthesis into 3NF with Dependency Preservation and Nonadditive Join Property

Input: ***A universal relation R and a set of functional dependencies F on the attributes of R.***

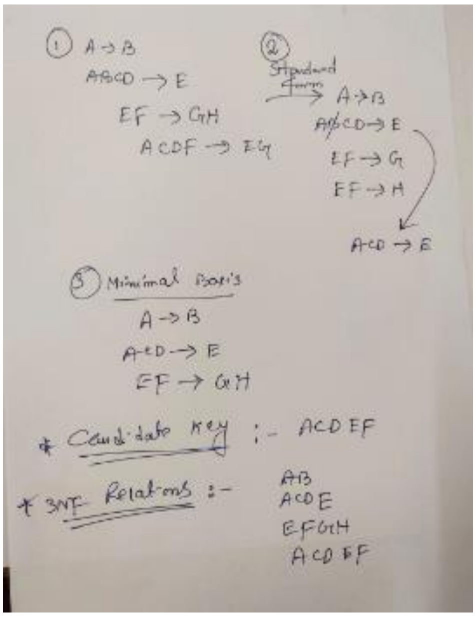
1. Find a minimal cover G for F

2. For each left-hand-side X of a functional dependency that appears in G, create a relation schema in D with attributes {X ∪ {A1} ∪ {A2} ... ∪ {Ak}}, where X→A1, X→A2, ..., X→Ak are the only dependencies in G with X as left-hand-side (X is the key of this relation)

3 If none of the relation schemas in D contains a key of R, then create one more relation schema in D that contains attributes that form a key of R

4 Eliminate redundant relations from the resulting set of relations in the relational database schema. A relation R is considered redundant if R is a projection of another relation S in the schema; alternately, R is subsumed by S

Consider a relation R (A, B, C, D, E, F, G, H) having FDs as: {A→B, ABCD→E, EF→GH, ACDF→EG}. Find candidate key of this relation R. Preserving the dependency, decompose R into 3NF.

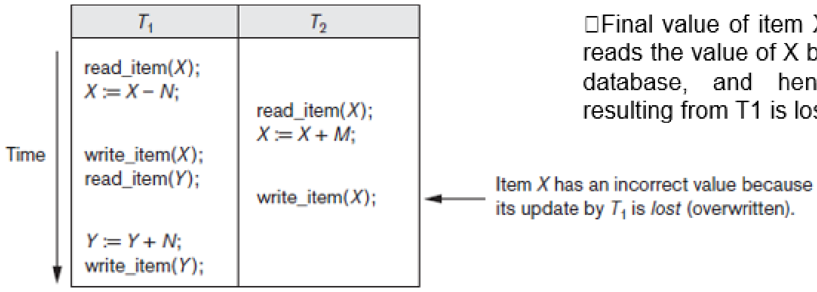


4. What type of problems may arise if concurrency control and recovery are not maintained in DBMS transaction? Justify your answer with proper example

1.The Lost Update Problem [WW conflict]

 Occurs when two transactions that access the same DB items have their operations interleaved in a way that makes the value of some DB item incorrect

 Suppose that transactions T1 and T2 are submitted at approximately the same time, and suppose that their operations are interleaved as shown in Figure below



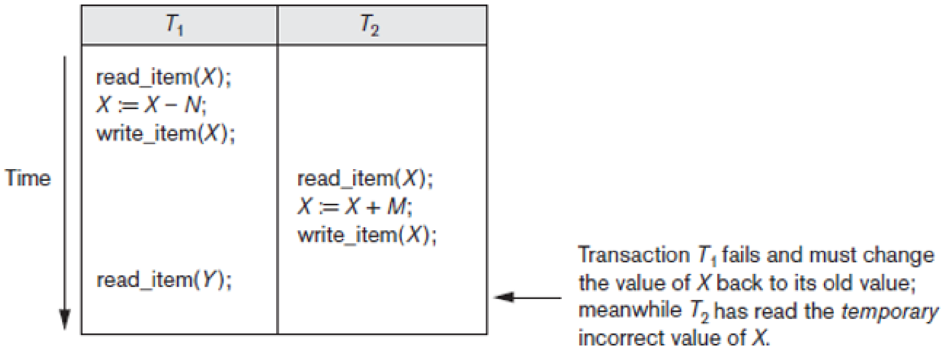
Final value of item X is incorrect because T2 reads the value of X before T1 changes it in the database, and hence the updated value resulting from T1 is lost.

 For example:
X = 80 at the start (there were 80 reservations on the flight)
N = 5 (T1 transfers 5 seat reservations from the flight corresponding to X to the flight corresponding to Y)
M = 4 (T2 reserves 4 seats on X) The final result should be X = 79.
 The interleaving of operations shown in Figure is X = 84 because the update in T1 that removed the five seats from X was lost.

2.The Temporary Update / Dirty Read Problem [WR conflict]

occurs when one transaction updates a database item and then the transaction fails for some reason before doing commit.

Meanwhile the updated item is accessed by another transaction before it is changed back to its original value



3. The Incorrect Summary Problem

•If one transaction is calculating an aggregate summary function on a number of DB items while other transactions are updating some of these items, the aggregate function may calculate some values before they are updated and others after they are updated.

T_1	T_3

	sum := 0;
	read_item(*A*);
	sum := *sum* + *A*;
	⋮

read_item(*X*);
X := *X* − *N*;
write_item(*X*);	

	read_item(*X*);
	sum := *sum* + *X*;
	read_item(*Y*);
	sum := *sum* + *Y*;

read_item(*Y*);
Y := *Y* + *N*;
write_item(*Y*);	

← T_3 reads *X* after *N* is subtracted and reads *Y* before *N* is added; a wrong summary is the result (off by *N*).

4. The Unrepeatable Read Problem [RW conflict]

□ **Transaction T reads the same item twice and gets different values on each read, since the item was modified by another transaction T` between the two reads.**

□ for example, if during an airline reservation transaction, a customer inquires about seat availability on several flights

□ When the customer decides on a particular flight, the transaction then reads the number of seats on that flight a second time before completing the reservation, and it may end up reading a different value for the item.

Why Recovery Is Needed

Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either:

- All the operations in the transaction are completed successfully and their effect is recorded permanently in the database ***(The transaction is committed)*** **or**
- The transaction does not have any effect on the database or any other transactions

□In the first case, the transaction is said to be **committed,** whereas in the second case, the transaction is **aborted**

□If a transaction fails after executing some of its operations but before executing all of them, the operations already executed **must be undone and have no lasting effect.**

5. Explain 2PL protocol for concurrency control. How does it guarantee serializability? Justify your answer with a suitable example.

The concept of locking data items is one of the main techniques used for controlling the concurrent execution of transactions. **A lock is a variable associated with a data item in the database.** Generally there is a lock for each data item in the database.

A lock describes the status of the data item with respect to possible operations that can be applied to that item. **It is used for synchronizing the access by concurrent transactions to the database items.**

□ A transaction locks an object before using it

□ When an object is locked by another transaction, the requesting transaction must wait

This **Two-phase locking (2PL)** protocol divides the execution phase of a transaction into three parts.

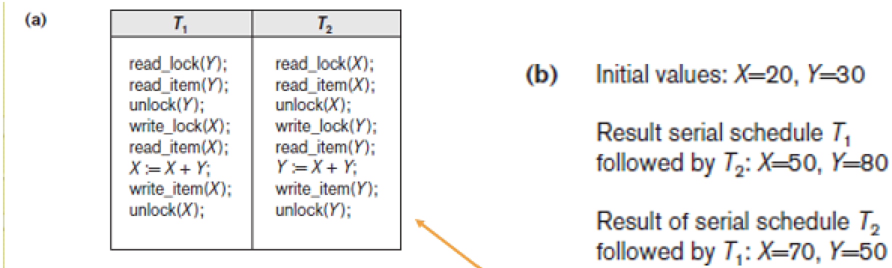
<u>In the first part,</u> when the transaction starts executing, it seeks permission for the locks it requires.
It is a concurrency control method that guarantees serializability

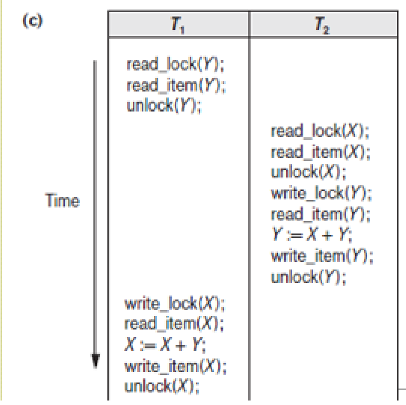
<u>The second part</u> is where the transaction acquires all the locks.

<u>As soon as the transaction releases its first lock, the third phase starts</u>.
- In this phase, the transaction cannot demand any new locks.
- It only releases the acquired locks.
- The protocol utilizes locks, applied by a transaction to data, which may block other transactions from accessing the same data during the transaction's life.

Guaranteeing Serializability by Two-Phase Locking

- A transaction is said to follow the two-phase locking protocol if all locking operations (read_lock, write_lock) precede the first unlock operation in the transaction
- Such a transaction can be divided into two phases:
- **Expanding or growing (first) phase**, during which new locks on items can be acquired but none can be released
- **Shrinking (second) phase**, during which existing locks can be released but no new locks can be acquired
- If lock conversion is allowed, then upgrading of locks (from read-locked to write-locked) must be done during the expanding phase, and downgrading of locks (from write-locked to read-locked) must be done in the shrinking phase.





Transactions T1 and T2 in Figure 22.3(a) do not follow the two-phase locking protocol because the write_lock(X) operation follows the unlock(Y) operation in T1, and similarly the write_lock(Y) operation follows the unlock(X) operation in T2. so not following the rule of growing phase (only can acquire lock, cant release) and shrinking phase (only can release lock, cant acquire).

Figure 21.3 Transactions that do not obey two-phase locking (a) Two transactions T1 and T2 (b) Results of possible serial schedules of T1 and T2 (c) A nonserializable schedule S that uses locks

□ If we enforce two-phase locking, the transactions can be rewritten as T1' and T2' as shown in Figure 22.4.

□ Now, the schedule shown in Figure 22.3(c) is not permitted for T1_ and T2_ (with their modified order of locking and unlocking operations) under the rules of locking because T1_ will issue its write_lock(X) before it unlocks item Y; consequently, when T2_ issues its read_lock(X), it is forced to wait until T1_ releases the lock by issuing an unlock (X) in the schedule.

Here during growing phase only acquiring locks, not releasing any and during shrinking phase only releasing locks, not acquiring any lock, so satisfies 2PL property.

T_1'	T_2'

read_lock(Y);
read_item(Y);
write_lock(X);
unlock(Y)
read_item(X);
$X := X + Y$;
write_item(X);
unlock(X);
	read_lock(X);
	read_item(X);
	write_lock(Y);
	unlock(X)
	read_item(Y);
	$Y := X + Y$;
	write_item(Y);
	unlock(Y);

Figure 22.4
Transactions T_1' and T_2', which are the same as T_1 and T_2 in Figure 22.3, but follow the two-phase locking protocol. Note that they can produce a deadlock.

□ **If every transaction in a schedule follows the two-phase locking protocol, schedule guaranteed to be serializable**

6a) Explain the basic time stamping protocol for concurrency control

The concurrency control algorithm must check whether conflicting operations violate the timestamp ordering in the following two cases:

1.Whenever a transaction T issues a write_item(X) operation, the following is checked:
a.If read_TS(X) > TS(T) or if write_TS(X) > TS(T), then abort and roll back T and reject the operation. This should be done because some younger transaction with a timestamp greater than TS(T)—and hence after T in the timestamp ordering—has already read or written the value of item X before T had a chance to write X, thus violating the timestamp ordering.
b.If the condition in part (a) does not occur, then execute the write_item(X) operation of T and set write_TS(X) to TS(T).

2.Whenever a transaction T issues a read_item(X) operation, the following is checked:
a.If write_TS(X) > TS(T), then abort and roll back T and reject the operation. This should be done because some younger transaction with timestamp greater than TS(T)—and hence after T in the timestamp ordering—has already written the value of item X before T had a chance to read X.
b.If write_TS(X) ≤ TS(T), then execute the read_item(X) operation of T and set read_TS(X) to the larger of TS(T) and the current read_TS(X).

Whenever the basic TO algorithm detects two conflicting operations that occur in the incorrect order, it rejects the later of the two operations by aborting the transaction that issued it. The schedules produced by basic TO are hence guaranteed to be conflict serializable.

6b) Consider the below mentioned schedule with transactions T1, T2, T3 with read () and write () operations on data items X, Y, Z. Draw the precedence graph for the schedule and state whether the schedule is serializable or not. If the schedule is serializable, write down the serial schedule.

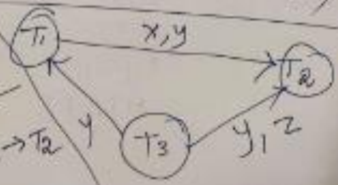
S1: r3(Y); r3(Z); r1(X); w1(X); w3(Y); w3(Z); r2(Z), r1(Y); w1(Y); r2(Y); w2(Y); r2(X); w2(X).

T1	T2	T3
		read (y)
		read (z)
read(x)		
write (x)		
		write (y)
		write (z)
	read (z)	
read (y)		
write (y)		
	read (y)	
	write (y)	
	read (x)	
	write (x)	

Time ↓

equivalent serial schedule

T3 → T1 → T2



No cycle, no conflict, so the schedule is serializable.