Sub:	Database Management Systems					Sub Code:	17CS53	
Date:	19/11/19	Duration:	90 mins	Max Marks:	50	Sem /Sec:	V A,B,C	OBE

Answer any FIVE FULL QUESTIONS — CO | RBT

1 a — CO4 — L2

What do you mean by multivalued dependency? Describe 4NF with example.

ANSWER:

MULTIVALUED DEPENDENCY (MVD) AND FOURTH NORMAL FORM

- Multivalued dependencies are a consequence of first normal form (1NF), which disallows an attribute in a tuple to have a set of values, and the accompanying process of converting an unnormalized relation into 1NF.
- If we have two or more multivalued independent attributes in the same relation schema, we get into a problem of having to repeat every value of one of the attributes with every value of the other attribute to keep the relation state consistent and to maintain the independence among the attributes involved. This constraint is specified by a multivalued dependency.

Formal Definition of Multivalued Dependency

A multivalued dependency specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R: If two tuples t1 and t2 exist in r such that t1[X] = t2[X], then two tuples t3 and t4 should also exist in r with the following properties, where we use Z to denote (R – (X ∪ Y)):

t3[X] = t4[X] = t1[X] = t2[X].
t3[Y] = t1[Y] and t4[Y] = t2[Y].
t3[Z] = t2[Z] and t4[Z] = t1[Z].

Whenever $X \rightarrow\rightarrow Y$ holds, we say that X **multidetermines** Y. Because of the symmetry in the definition, whenever $X \rightarrow\rightarrow Y$ holds in R, so does $X \rightarrow\rightarrow Z$. Hence, $X \rightarrow\rightarrow Y$ implies $X \rightarrow\rightarrow Z$ and therefore it is sometimes written as $X \rightarrow\rightarrow Y|Z$.
An MVD $X \rightarrow\rightarrow Y$ in R is called a **trivial MVD** if (a) Y is a subset of X, or (b) $X \cup Y = R$. An MVD that satisfies neither (a) nor (b) is called a **nontrivial MVD**

Definition of 4NF: A relation schema R is in 4NF with respect to a set of dependencies F (that includes functional dependencies and multivalued dependencies) if, for every nontrivial **multivalued dependency** X→→ Y in F+ X is a superkey for R.

In the EMP relation of Figure(A),the values 'X' and 'Y' of Pname are repeated with each value of Dname (or, by symmetry, the values 'John' and 'Anna' of Dname are repeated with each value of Pname).

In EMP relation of figure (C), not every Sname determines various Part_name and not every Sname determines multiple Proj_name. so it is not MVD. Therefore it is in 4NF.

Fourth Normal Form:

(a) The EMP relation with two MVDs Ename->> Pname and Ename->> Dname

(b) Decomposing the EMP relation into two 4NF relations EMP_PROJECTS and EMP_DEPARTMENTS

(A) EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y

EMP_DEPENDENTS

Ename	Dname
Smith	John
Smith	Anna

Decomposing a relation state of EMP that is not in 4NF,(a) EMP relation with additional tuples.(b)Two corresponding 4NF relationsEMP_PROJECTS and EMP_DEPENDENTS.

(B) EMP

ENAME	PNAME	DNAME
SMITH	X	JOHN
SMITH	Y	ANNA
SMITH	X	ANNA
SMITH	Y	JOHN
BROWN	W	JIM
BROWN	X	JIM
BROWN	Y	JIM
BROWN	Z	JIM
BROWN	W	JOAN
BROWN	X	JOAN
BROWN	Y	JOAN
BROWN	Z	JOAN
BROWN	W	BOB
BROWN	X	BOB
BROWN	Y	BOB
BROWN	Z	BOB

(B)EMP_PROJECTS

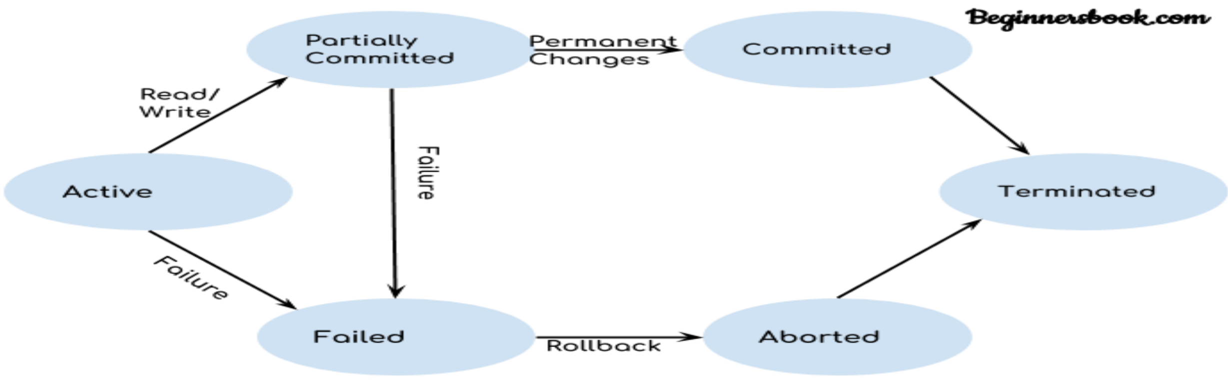
ENAME	PNAME
SMITH	X
SMITH	Y
BROWN	W
BROWN	X
BROWN	Y
BROWN	Z

EMP_DEPENDENTS

ENAME	DNAME
SMITH	ANNA
SMITH	JOHN
BROWN	JIM
BROWN	JOAN
BROWN	BOB

1 b	**Draw state transition diagram of a transaction. Explain different states of a transaction.**	CO4	L1

ANSWER:



Active State
A transaction is a sequence of operations. If a transaction is in execution then it is said to be in active state. It doesn't matter which step is in execution, until unless the transaction is executing, it remains in active state.

Failed State
If a transaction is executing and a failure occurs, either a hardware failure or a software failure then the transaction goes into failed state from the active state.

Partially Committed State
A transaction goes into "partially committed" state from the active state when there are read and write operations present in the transaction.

A transaction contains number of read and write operations. Once the whole transaction is successfully executed, the transaction goes into partially committed state where we have all the read and write operations performed on the main memory (local memory) instead of the actual database.

A transaction can fail during execution so if we are making the changes in the actual database instead of local memory, database may be left in an inconsistent state in case of any failure. This state helps us to rollback the changes made to the database in case of a failure during execution.

Committed State
If a transaction completes the execution successfully then all the changes made in the local memory during **partially committed** state are permanently stored in the database. You can also see in the above diagram that a transaction goes from partially committed state to committed state when everything is successful.

Aborted State
If a transaction fails during execution then the transaction goes into a failed state. The changes made into the local memory (or buffer) are rolled back to the previous consistent state and the transaction goes into aborted state from the failed state.

Terminated state
The terminated state corresponds to the transaction leaving the system. The transaction information that is maintained in system tables while the transaction has been running is removed when the transaction terminates.

2 a	**Define Minimal cover. Write an algorithm for finding a minimal cover G for a set of functional dependencies F.**	CO4	L1

ANSWER:

Minimal Sets of Functional Dependencies:

A **minimal cover** of a set of functional dependencies E is a set of functional dependencies F that satisfies the property that every dependency in E is in the closure F+ of F.

This property is lost if any dependency from the set F is removed; F must have no redundancies in it, and the dependencies in E are in a standard form.

To satisfy these properties, we can formally define a set of functional dependencies F to be minimal if it satisfies the following conditions:

a) Every dependency in F has a single attribute for its right-hand side.

b) We cannot replace any dependency X → A in F with a dependency Y → A, where *Y* is a proper subset of X, and still have a set of dependencies that is equivalent to F.

c) We cannot remove any dependency from F and still have a set of dependencies that is equivalent to F.

A minimal cover of a set of functional dependencies E is a minimal set of dependencies F that is equivalent to E. There can be several minimal covers for a set of functional dependencies.

Algorithm : Finding a Minimal Cover F for a Set of Functional Dependencies E

1. Set F := E.

2. Replace each functional dependency X → {A1, A2, ..., An} in F by the n functional dependencies X → A1, X → A2, ..., X → An.

3. For each functional dependency X → A in F
for each attribute B that is an element of X
if {{F - {X → A} } U {(X - {B}) → A} } is equivalent to F,
then replace X → A with (X - {B}) → A in F.

4. For each
remaining functional dependency
X → A in F if { F - {X → A} } is equivalent to
F,then remove X → A from F.

2 b **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.**

3 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

2 b	2(b). R(A, C, D, E, H) F = { A→C, AC→D, E→AD, E→H } Find the canonical cover for this set of FDs. **Step 1** Right reduction: G: {A→C AC→D E→A E→D E→H} **Step 2** Left reduction. AC→D $(AC)^+ = ACD$ $A^+ = ACD$ $C^+ = C$, A is not extraneous. C is extraneous. AC→D can be replaced by A→D. G: { A→C A→D E→A E→D E→H } **Step 3** Redundant FD removal. (A→C) $A^+ = AD$ A→C is not redundant (A→D) $A^+ = AC$ A→D is not redundant (E→A) $E^+ = EDH$ E→A is not redundant (E→D) ✗ $E^+ = EAHCD$ E→D is redundant — can be removed.	CO4	L3

(E → H)

$E^+ = EACD$

E → H is not redundant

Minimal Cover

$G: \{A \to C,\ A \to D,\ E \to A,\ E \to H\}$

Canonical cover

$\{A \to CD,\ E \to AH\}$.

(3b). CO4 L3

R(ABCDE)

$X = \{A \to B\quad AB \to C\quad D \to AC\quad D \to E$

$Y = \{A \to BC\quad D \to AE$

are they equivalent?

Step1

X Covers Y. or not?

(A → BC)

A^+ from X

$A^+ = ABC$ A → BC is included.

(D → AE)

D^+ from X

$D^+ = DACE$ D → AE is included.

X Covers Y means $X \supseteq Y$.

Step2

Y covers X or not?

5

$A \rightarrow B$

A^+ from Y

$A^+ = ABC$ $A \rightarrow B$ is included.

$AB \rightarrow C$

$(AB)^+$ from Y

$(AB)^+ = ABC$ $AB \rightarrow C$ ✓

$D \rightarrow AC$

D^+ from Y

$D^+ = DAEBC$ $D \rightarrow AC$ ✓

$D \rightarrow E$

D^+ from Y

$D^+ = DAEBC$ $D \rightarrow E$.

Y covers X means $Y \supseteq X$

$X \supseteq Y$

$Y \supseteq X$

$\boxed{X \equiv Y}$

Both are equivalent.

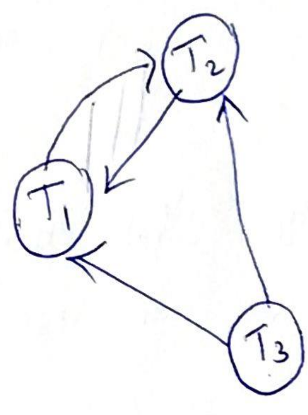
	(5) A schedule S of n transactions (T_1, T_2, ... T_n) is Serializable if it is equivalent to Some serial Schedule of the Same n transactions. To ensure serializability we have to check Conflict serializability. A schedule is Conflict Serializable if it is Conflict equivalent to Some Serial schedule. 2 Shedules are conflict equivalent if the relative Order of any two Conflicting operations is Same in both schedules. If the given schedule is not Conflict Serializable Check View Serializability. A Schedule is View Serializable if it is View equivalent to Some Serial schedule. 2 Schedules are View equivalent if it holds all the 3 following Conditions. ① Initial read ② Read write sequence ③ Final write.	CO4	L3

(B) $r_2(A); r_1(C); r_3(A); r_2(C); r_3(B); W_2(A); W_1(C);$
$W_3(B); W_1(B); r_2(B);$

S

T_1	T_2	T_3
	$r_2(A)$	
$r_1(C)$		
		$r_3(A)$
	$r_2(C)$	
		$r_3(B)$
	$W_2(A)$	
$W_1(C)$		
		$W_3(B)$
$W_1(B)$		
	$r_2(B)$	

Precedence graph.



loop exist. So not Conflict serializable.

View serializable

To test initial read and Final write preserv

~~of Serial schedule~~

	A	B	C
Initial read	T_2, T_3	T_3	T_1, T_2
Write	T_2	T_3, T_1	T_1
Final write	T_2	T_1	T_1

By analysing A.

$T_3 - T_2 \rightarrow$ (1)

By analyzing B

$T_3 - T_1 \rightarrow$ (2)

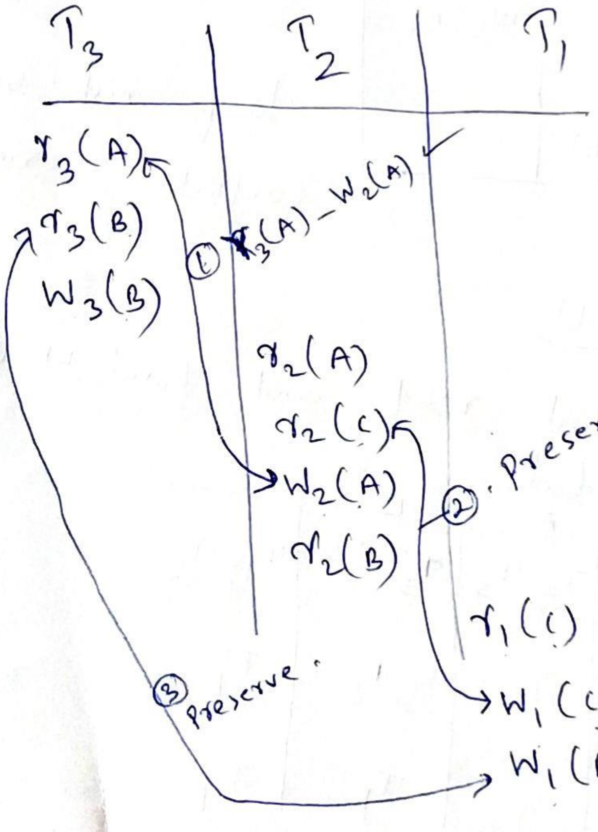
By analyzing C

$T_2 - T_1 \rightarrow$ (3)

By combining (1) & (3)

$T_3 - T_2 - T_1 \rightarrow$

To Test that this sequence preserve Read-write sequence or not?



6	6. Conditions to be satisfied for a non-additive join decomposition. 1. $R_1 \cup R_2 \cup R_3 = R$ 2. $R_1 \cap R_2 \neq \phi$ 3. The Common attribute must be a Candidate key of R_1 or Candidate key of R_2 or R itself. R(ABCDE) $F = \{BC \to D, AC \to BE, B \to E\}$ <u>Key of R</u> $(AC)^+ = ACBED$ AC can determines all the attributes of R. so AC is the Candidate key. As AC is the only candidate key, <u>AC</u> is the only choice for primary key. so Key of R is <u>AC</u> Prime attributes = {A, C} Non prime attributes = {B, D, E}		L3

Step2

Left reduction or Extraneous attribute removal

Consider

$BC \to D$

$(BC)^+ = BCDE$

To test C is extraneous or not

$B^+ = BE$; $B^+ \neq (BC)^+$ so C is not extraneous

To test B is extraneous or not

$C^+ = C$; $C^+ \neq (BC)^+$ so B is also not extraneous.

$BC \to D$ is not reducible.

Consider

$AC \to B$.

$(AC)^+ = ACBED$

To test C is extraneous or not

$A^+ = A$

$C^+ = C$

$AC \to B$ not reducible.

Consider

$AC \to E$

not reducible.

$G: \{ BC \to D, AC \to B, AC \to E, B \to E \}$.

Step 3

$BC \to D$

$(BC)^+ = BCE$ $BC \to D$ not redundant

$AC \to B$

$AC^+ = ACE$ $AC \to B$ not redundant

$AC \to E$

$(AC)^+ = ACBED$ $AC \to E$ is redundant & can be removed.

$B \to E$

$B^+ = B$ $B \to E$ is not redundant.

$G: \{BC \to D, AC \to B, B \to E\}$. ← Minimal Cover.

decomposition to 3NF

$R(ABCDE)$

$BC \to D$, $AC \to B$, $B \to E$

R is in 2NF. ⟸ So. { $BC \to D$ is a Full fd. $AC \to B$ is Full fD. $B \to E$ is a Full fd. }

To check R is in 3NF or not

~~$BC \to D$~~ D is not a prime attribute or BC is not CK

~~$BC \to D$ violates 3NF~~

~~$B \to E$ also violates~~ 3NF.

R is not in 3NF

Decomposition to 3NF.

$R_1(\underline{BC}D)$ $BC \to D$

$R_2(\underline{AC}B)$ $AC \to B$

$R_3(\underline{B}E)$ $B \to E$

1. Decomposition preserves dependency preservation property.

2. $R_1 \cup R_2 \cup R_3 = R$. So attribute preservation property also satisfied.

3. $(R_1 \cup R_3) \cup R_2$. There exists a common attribute for performing natural join operation, in the order $R_1 \cup R_3$ followed by $(R_1 \cup R_3) \cup R_2$.
And common attribute is candidate key of R_3 in case of $(R_1 \cup R_3)$ and candidate key of R_2 in case of $(R_1 \cup R_3) \cup R_2$.

3 A	**Discuss the properties of a Transaction.** **ANSWER:** Consider a transaction to transfer $50 from account A to account B: 1. **read**(*A*) 2. $A := A - 50$ 3. **write**(*A*) 4. **read**(*B*) 5. $B := B + 50$ 6. **write**(*B*) **Atomicity requirement** If the transaction fails after step 3 and before step 6, money will be "lost" leading to an inconsistent database state ▸ Failure could be due to software or hardware The system should ensure that updates of a partially executed transaction are not reflected in the database **Durability requirement** — once the user has been notified that the transaction has completed (i.e., the transfer of the $50 has taken place), the updates to the database by the transaction must persist even if there are software or hardware failures. **Consistency requirement** in above example: The sum of A and B is unchanged by the execution of the transaction In general, consistency requirements include ▸ Explicitly specified integrity constraints such as primary keys and foreign keys ▸ Implicit integrity constraints e.g., sum of balances of all accounts, minus sum of loan amounts must equal value of cash-in-hand A transaction, when starting to execute, must see a consistent database. During transaction execution the database may be temporarily inconsistent. When the transaction completes successfully the database must be consistent Erroneous transaction logic can lead to inconsistency A **transaction** is a unit of program execution that accesses and possibly updates various data items. To preserve the integrity of data the database system must ensure **Atomicity.** Either all operations of the transaction are properly reflected in the database or none are. **Consistency.** Execution of a transaction in isolation preserves the consistency of the database. **Isolation.** Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions. That is, for every pair of transactions T_i and T_j, it appears to T_i that either T_j, finished	CO4	L2

execution before T_i started, or T_j started execution after T_i finished.

Durability. After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

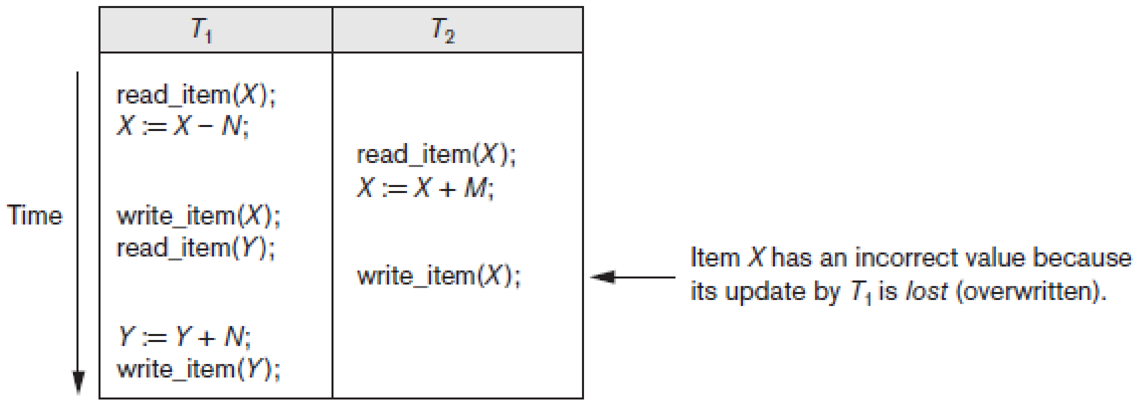
4 **What are the anomalies that can occur due to concurrent execution of transactions? Explain them with example.** CO4 L1

Database inconsistencies can occur when more than one transaction is working concurrently on the same objects. In the space of time between when objects are read and then written, the same objects can be read from the database and even manipulated by other transactions. This leads to concurrency anomalies. Depending on the kind of operations and the order in which they are executed, various kinds of concurrency anomalies can occur. The following lists some typical examples.

1.Lost Update Problem

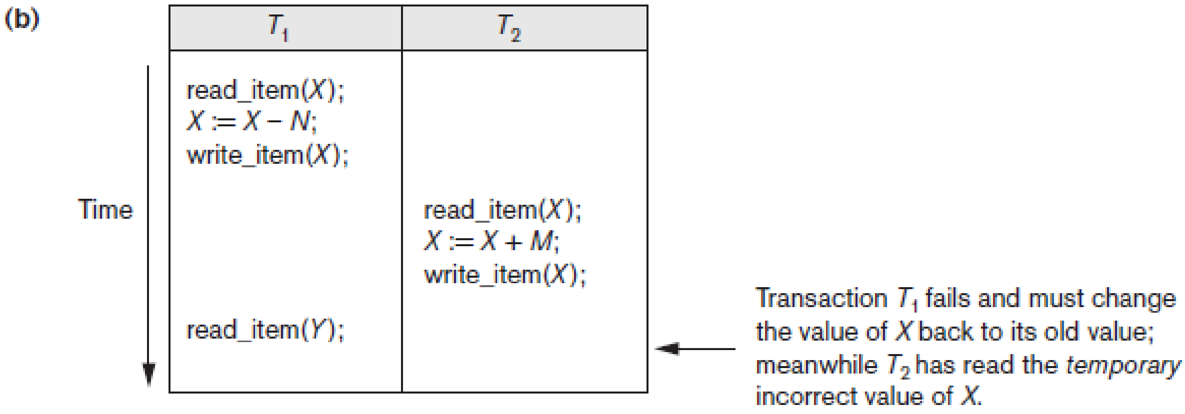
This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database items incorrect. Suppose that transactions $T1$ and $T2$ are submitted at approximately the same time, and suppose that their operations are interleaved as shown in below figure.Then the 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, if $X = 80$ at the start (originally 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), and $M = 4$ ($T2$ reserves 4 seats on X), the final result should be $X = 79$. However, in the interleaving of operations shown in the example, it is $X = 84$ because the update in $T1$ that removed the five seats from X was *lost*.

(a)



2. The Temporary Update (or Dirty Read) Problem.

This problem occurs when one transaction updates a database item and then the transaction fails for some reason . Meanwhile, the updated item is accessed (read) by another transaction before it is changed back (or rolled back) to its original value. Below given example shows where $T1$ updates item X and then fails before completion, so the system must roll back X to its original value. Before it can do so, however, transaction $T2$ reads the *temporary* value of X, which will not be recorded permanently in the database because of the failure of $T1$. The value of item X that is read by $T2$ is called *dirty data* because it has been created by a transaction that has not completed and committed yet; hence, this problem is also known as the *dirty read problem*.

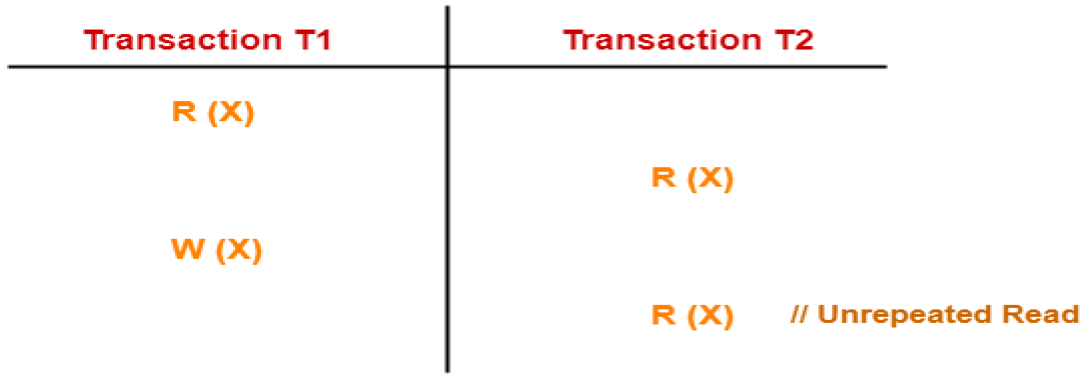


Non-repeatable Read

A non-repeatable read occurs when a object is read twice within a transaction; and between the reads, it is modified by another transaction, therefore, the second read returns different values as compared to the first; i.e., the read operation is non-repeatable. Open Access ORM uses in-memory copies of database objects. Once an object is loaded into memory, there is no need to fetch it from the database each time a member is accessed. A new read operation, and therefore a non-repeatable read, could only occur when an application explicitly refreshes an object. This depends on the backend and/or its configuration (e.g., if it is a "versioning" database that maintains multiple versions of a row for concurrently running transactions)

This problem occurs when a transaction gets to read unrepeated i.e. different values of the same variable in its different read operations even when it has not updated its value.

Example-



Here,

- T1 reads the value of X (= 10 say).
- T2 reads the value of X (= 10).
- T1 updates the value of X (from 10 to 15 say) in the buffer.
- T2 again reads the value of X (but = 15).

In this example,

- T2 gets to read a different value of X in its second reading.
- T2 wonders how the value of X got changed because according to it, it is running in isolation.

Inconsistent Retrievals Problem

- Inconsistent Retrievals Problem is also known as unrepeatable read. When a transaction calculates some summary function over a set of data while the other transactions are updating the data, then the Inconsistent Retrievals Problem occurs.
- A transaction T1 reads a record and then does some other processing during which the

transaction T2 updates the record. Now when the transaction T1 reads the record, then the new value will be inconsistent with the previous value.

Example:

Suppose two transactions operate on three accounts.

Transaction-X	Time	Transaction-Y
——	t1	——
Read Balance of Acc-1 sum <-- 200 Read Balance of Acc-2	t2	——
Sum <-- Sum + 250 = 450	t3	——
——	t4	Read Balance of Acc-3
——	t5	Update Balance of Acc-3 150 --> 150 - 50 --> 100
——	t6	Read Balance of Acc-1
——	t7	Update Balance of Acc-1 200 --> 200 + 50 --> 250
Read Balance of Acc-3	t8	COMMIT
Sum <-- Sum + 250 = 550	t9	——

Transaction-X is doing the sum of all balance while transaction-Y is transferring an amount 50 from Account-1 to Account-3.

- Here, transaction-X produces the result of 550 which is incorrect. If we write this produced result in the database, the database will become an inconsistent state because the actual sum is 600.
- Here, transaction-X has seen an inconsistent state of the database.

Phantom Read

Phantom reads are of a totally different nature than the anomalies introduced previously. They can occur when a transaction defines a subset of data items that the transaction wants to work with; e.g., by performing a query and obtaining a query result. At this point, it is possible that data items are concurrently changed by another transaction so that they no longer qualify for inclusion in the query result, or vice versa. The same applies to objects that are inserted or deleted. This problem occurs when a transaction reads some variable from the buffer and when it reads the same variable later, it finds that the variable does not exist.

Example-

Transaction T1	Transaction T2
R (X)	
	R (X)
Delete (X)	
	Read (X)

Here,

1. T1 reads X.
2. T2 reads X.
3. T1 deletes X.
4. T2 tries reading X but does not find it.

In this example,

- T2 finds that there does not exist any variable X when it tries reading X again.
- T2 wonders who deleted the variable X because according to it, it is running in isolation.

The Incorrect Summary Problem.

If one transaction is calculating an aggregate summary function on a number of database 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. For example, suppose that a transaction *T*3 is calculating the total number of reservations on all the flights; meanwhile, transaction *T*1 is executing. If the interleaving of operations shown in the result of *T*3 will be off by an amount *N* because *T*3 reads the value of *X* *after* *N* seats have been subtracted from it but reads the value of *Y* *before* those *N* seats have been added to it.

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*).