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

Internal Assessment Test III – Nov. 2019

Sub:	**Database Management Systems**					Sub Code:	**17CS53**	Branch:	**ISE**
Date:	18-11-19	Duration:	90 min's	Max Marks:	50	Sem / Sec:	V- A /B (ISE)		OBE

Answer any FIVE FULL Questions

1.Find the minimal cover for the set of functional dependencies

i) R:{B→A, D→A, AB→D}

Given FD

B→A, D→A, AB→D

B^+=BAD	D^+=DA	AB^+=ABD	
(Except B→A) B^+=B (Except D→A) B^+=BAD (Except AB→D) B^+ =BA	(Except B→A) D^+=DA (Except D→A) D^+=A (Except AB→D) D^+ =DA	(Except B→A) AB^+=ABD (Except D→A) AB^+=ABD (Except AB→D) AB^+ =AB	
Mandatory FDs B→A, AB→D	Mandatory FDs D→A	Mandatory FDs AB→D	

Based on the above solution

B→A, AB→D, D→A, AB→D

i.e B→A, AB→D, D→A

Based on transitive rule

Minimal Cover :B→A, AB→A

ii) S:{A→B, AB→C, D→AC, D→E}

Given FD

A→B, AB→C, D→A, D→C, D→E

A^+=ABC	D^+=DABCE	AB^+=ABC	
(Except A→B) A^+=A (Except AB→C) A^+=AB (Except D→A) A^+ =ABC (Except D→C) A^+ =ABC (Except D→E) A^+ =ABC	(Except A→B) D^+=DACE (Except AB→C) D^+=DABCE (Except D→A) D^+ =DCE (Except D→C) D^+ =DABCE (Except D→E) D^+ =DABC	(Except A→B) AB^+=ABC (Except AB→C) AB^+=AB (Except D→A) AB^+ =ABC (Except D→C) AB^+ =ABC (Except D→E) AB^+ =ABC	
Mandatory FDs A→B, AB→C	Mandatory FDs A→B, D→A, D→E	Mandatory FDs AB→C	

Based on the above solution

A→B, AB→C, A→B, D→A, D→E, AB→C

i.e A→B, AB→C, D→A, D→E

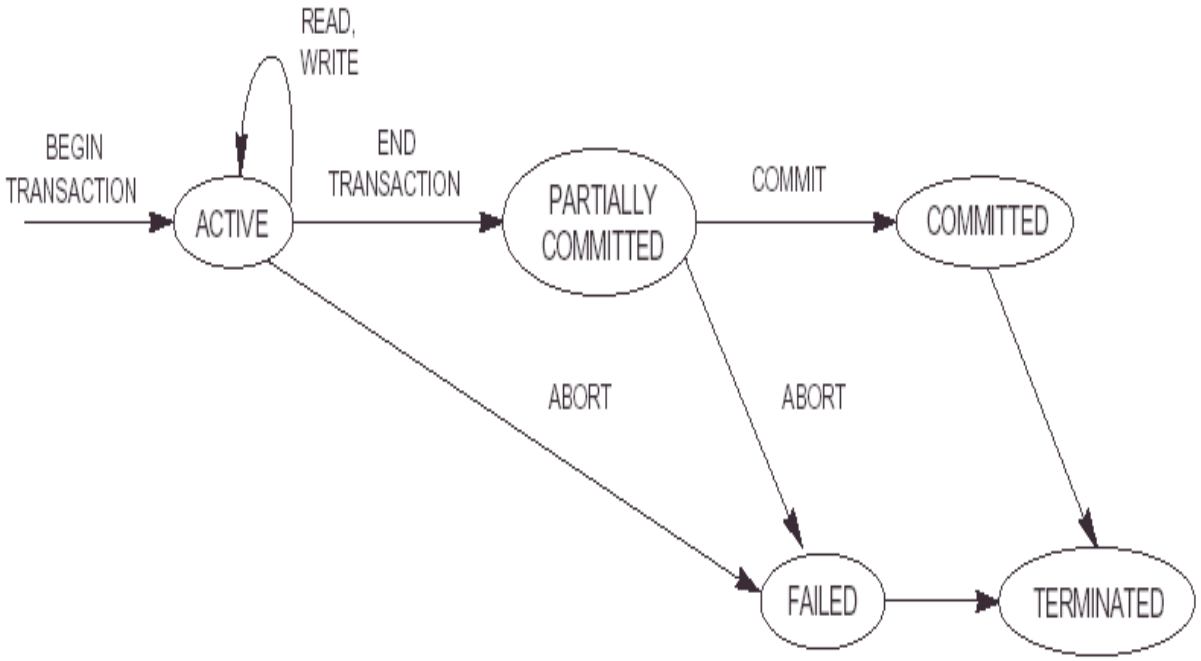
Minimal Cover : A→B, AB→C, D→AE

2 a.What are the ACID Properties? Explain with example

2 b.List the transaction states. Draw and explain state transition diagram of the transaction.

Transaction states

- BEGIN_TRANSACTION: marks start of transaction
- READ or WRITE: two possible operations on the data
- END_TRANSACTION: marks the end of the read or write operations; start checking whether everything went according to plan
- COMIT_TRANSACTION: signals successful end of transaction; changes can be "committed" to DB
- Partially committed
- ROLLBACK (or ABORT): signals unsuccessful end of transaction, changes applied to DB must be undone

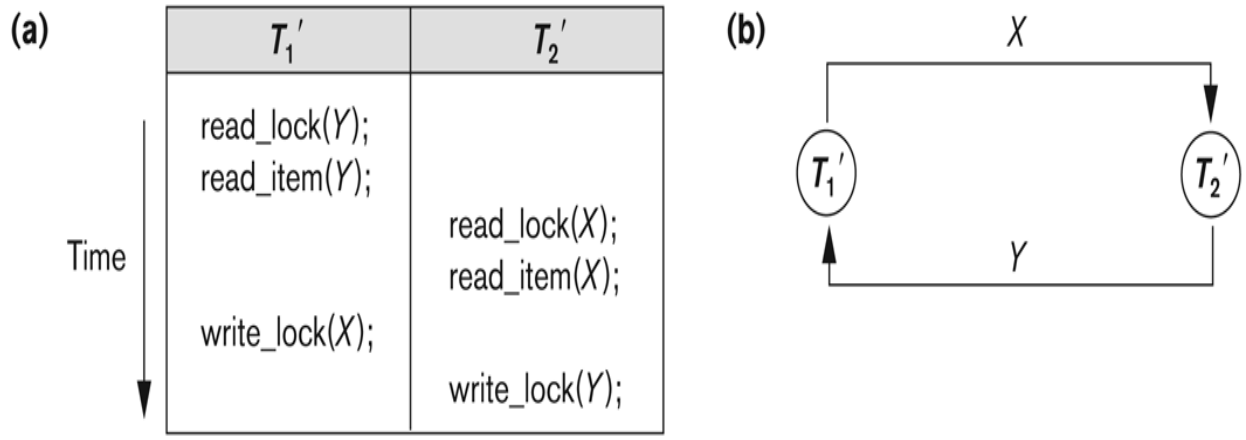


3.Define deadlock and Starvation. Explain the deadlock prevention and detection protocols.

Deadlocks and Starvation in 2PL

- 2PL can produce deadlocks
- Deadlock and starvation in 2PL

Deadlock occurs when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T' in the set.



Illustrating the deadlock problem. (a) A partial schedule of T_1' and T_2' that is in a state of deadlock. (b) A wait-for graph for the partial schedule in (a).

Deadlock prevention protocols
- Conservative 2PL, lock all needed items in advance
- Ordering all items in the database

Possible actions if a transaction is involved in a possible deadlock situation
- Block and wait
- Abort and restart
- Preempt and abort another transaction

Two schemes that prevent deadlock (Timestamp based)

Wait-die

An older transaction is allowed to wait on a younger transaction whereas a younger transaction requesting an item from held by an older transaction is aborted and restarted with the same timestamp

Wound-wait

A younger transaction is allowed to wait on an older transaction whereas an older transaction requesting an item from held by a younger transaction preempts the younger transaction by aborting it.

Starvation

A transaction cannot proceed for an infinite period of time while other transactions in the system continue normally
- Unfair waiting scheme
- Victim selection

4.How to recover the transaction from failure? Explain the list of recovery concepts
- Keeps information about operations made by transactions:
  – *Before-image* (*undo entry*) of updated items
  – *After-image* (*redo entry*) of updated items
- Enables restoring a consistent state after non-catastrophic failure (forward/backward).
- Alternatives:
  – *undo/no-redo*
  – *no-undo/redo*
  – *undo/redo*
  – *no-undo/no-redo.*

Write-Ahead Logging (WAL)

(1) No overwrite of disk data before *undo*-type
log records are forced to disk.
(2) *Both undo- and redo-type* log records
(= before- and after-images) must
be forced to disk before end of commit.

Backup

- Copy of database on archival storage

(off-line, often on tape).

- Enables partial recovery from *catastrophic* failures:

– For *committed* transactions:

Load backup and apply redo operations from the
log (if the log survived).

– *Non-committed* transactions must be restarted

(= re-executed).

Cache

- In-memory buffer for database pages.
- A *directory* (page table) keeps track of pages

in cache.

- Page-replacement strategy needed, e.g.
  FIFO (First-In-First-Out), or
  LRU (Least Recently Used)
- *Dirty bit* tells for each page, if it has changed
- *Flushing* means (force-)writing buffer pages to disk.
- *Pin/unpin bit* tells if the page can be written

Rollback

- At failure, apply undo-type log records (before-images) to updated items.
- A recoverable schedule may allow *cascading rollback*.
- Most practical protocols avoid cascading rollbacks. Then no *read-entries* are required in the log (no dirty reads).

5.Explain the timestamp ordering techniques for concurrency control.

Timestamp --- a unique identifier created by the DBMS to identify a transaction

- read-TS(X)
- The largest timestamp among all the timestamps of transactions that have successfully read item X
- write-TS(X)
- The largest timestamp among all the timestamps of transactions that have successfully written item X

Timestamp Ordering (TO) algorithms

- Basic timestamp ordering
- Strict timestamp ordering
- Thomas's write rule

Basic timestamp ordering algorithm

Order transactions based on their timestamps

- T issues a write(X)
- If read-TS(X) > TS(T) or if write-TS(X) > TS(T) the abort and rollback T and reject the operation.
- If condition above does not occur then execute the operation and set write-TS(X) = TS(T)
- T issues a read(X)

- If write-TS(X) > TS(T) then abort and rollback T and reject the operation.
- If write-TS(X) ≤ TS(T) then execute the operation and set read-TS(X) = max (read-TS(X), TS(T))
- The schedules produced by basic TO are guaranteed to be conflict serializable
- No deadlocks but cyclic restart are possible (hence starvation)

Strict Timestamp Ordering (strict TO)

- A transaction T that issues a read-item(*X*) or write-item(*X*) such that TS(T) > write-TS(*X*) has its read or write operation *delayed* until the transaction T' that *wrote* the value of *X* (hence TS(T') = write-TS(*X*)) has committed or aborted.
- No deadlocks, since T waits for T' only if TS(T) > TS(T')
- **Strict TO** ensures that the schedules are both **strict** (for easy recoverability) and (conflict) serializable

Thomas's write rule

- It rejects fewer write operations, by modifying the basic TO checks for the write-item(*X*) operation as follows:

1. If read-TS(*X*) > TS(T), then abort and roll back T and reject the operation.
2. If write-TS(*X*) > TS(T), then do not execute the write operation but continue processing.

[This is because some transaction with timestamp greater than TS(T)—and hence after T in the timestamp ordering—has already written the value of *X*. Hence, we must ignore the write_item(*X*) operation of T because it is already outdated and obsolete. Notice that any conflict arising from this situation would be detected by case (1).]

3. If neither the condition in part (1) nor the condition in part (2) occurs, then execute the write-item(*X*) operation of T and set write-TS(*X*) to TS(T).

Thomas' write rule does not enforce conflict serializability

6.Define lock. Write and explain the algorithm to control the concurrency using Two-Phase Locking Protocol.

Lock

A variable associated with a data item

Describes status of the data item with respect to operations that can be performed on it

Types of locks

Binary locks

Locked/unlocked

Enforces mutual exclusion

Multiple-mode locks:

Each data item can be in one of three lock states

Read lock or shared lock

Write lock or exclusive lock

Unlock

Two-Phase Locking (2PL) Protocol

Transaction is said to follow the *two-phase-locking protocol* if all locking operations precede the *first* unlock operation

Expanding (growing) phase

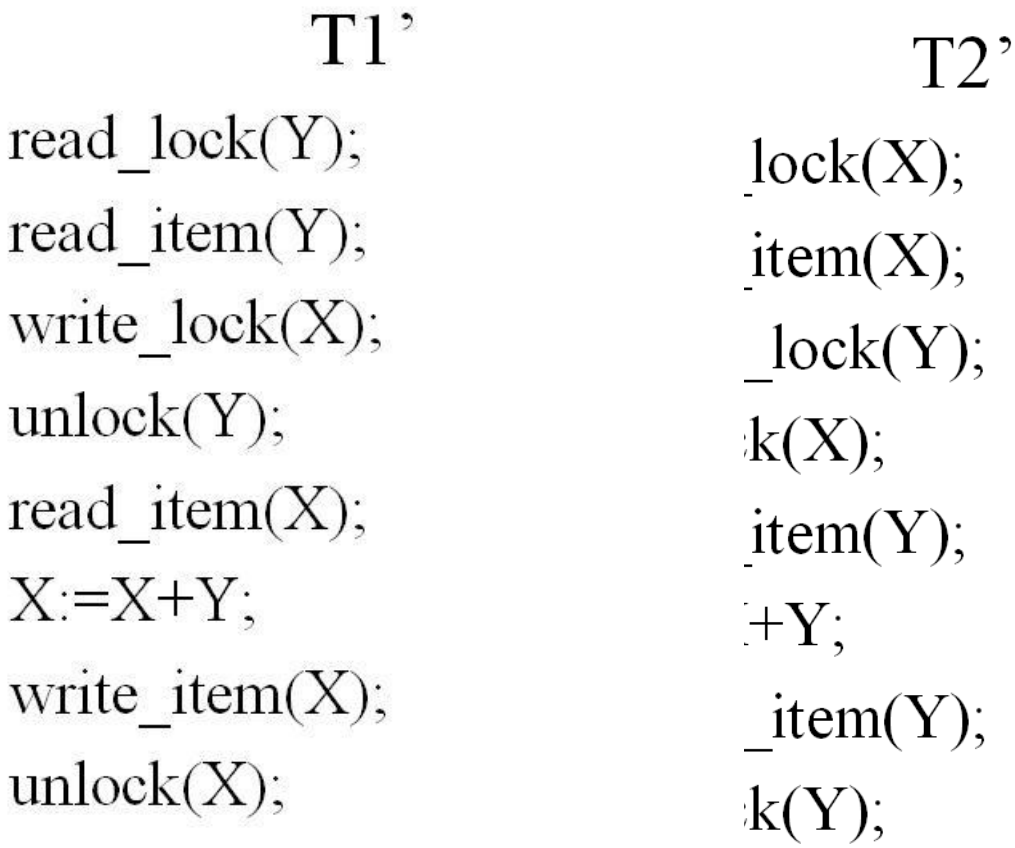
Shrinking phase

During the shrinking phase no new locks can be acquired

Downgrading ok

Upgrading is not

2PL Example



Both T1' and T2' follow the 2PL protocol
Any schedule including T1' and T2' is guaranteed to be serializable
Limits the amount of concurrency
Two-phase locking protocol (2PL)
All lock operations precede the first unlock operation
Expanding phase and shrinking phase
Upgrading of locks must be done in expanding phase and downgrading of locks must be done in shrinking phase
If every transaction in a schedule follows 2PL protocol then the schedules is guaranteed to be serializable.
Variants of 2PL
Basic, conservative, strict, and rigorous

7. Let R={Ssn, Ename, Pnumber, Pname, Plocation, Hours} and D={R1, R2, R3} where R1=EMP{Ssn, Ename}, R2=PROJ{Pnumber, Pname, Plocation}, R3=WORKS_ON{Ssn, Pnumber,Hour} following functional dependencies on Relation R. F={ Ssn→ Ename, Pnumber → {Pname, Plocation}, {Ssn, Pnumber}→Hour}. Determine whether the each decomposition has the lossless join propery with respect to F

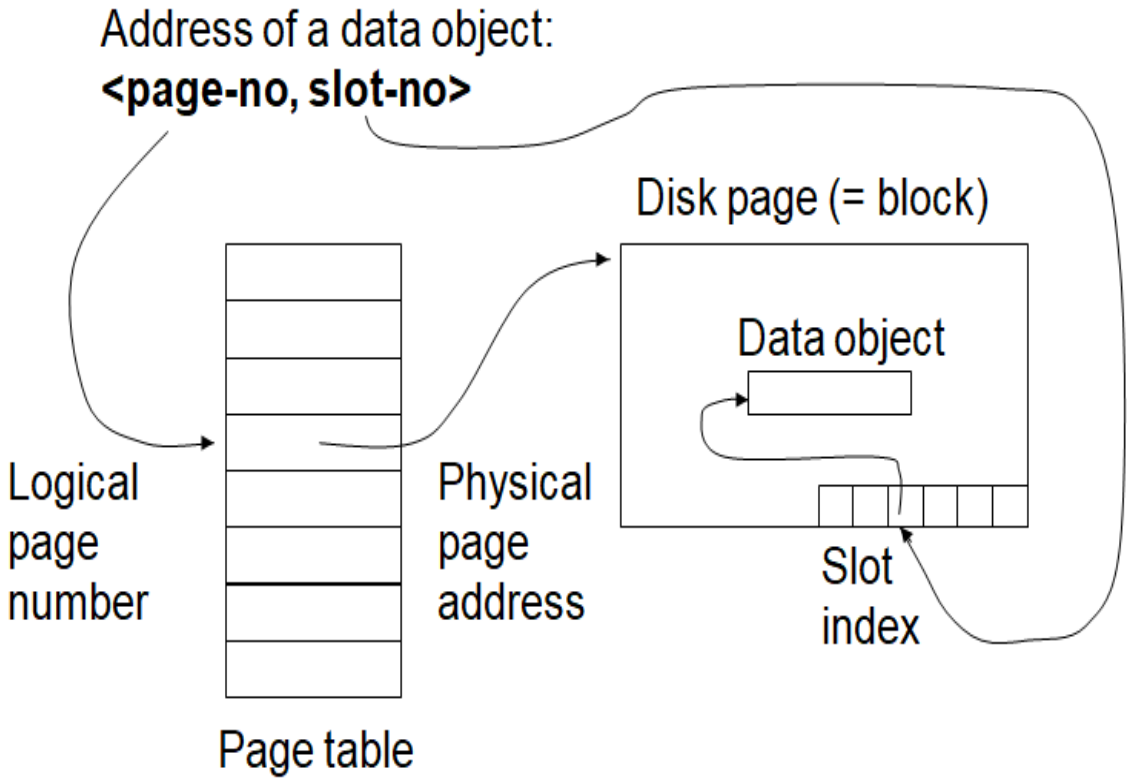
R1=EMP{Ssn, Ename}
R2=PROJ{Pnumber, Pname, Plocation},
R3=WORKS_ON{Ssn, Pnumber,Hour}

8. Write the short note on i) Shadow Paging

Shadow paging

Assumes an *indirect addressing* scheme:
References to database objects consist of
<*page-no*, *slot-no*> pairs, where *page-no* is
the index to a *page table (*'directory'), which contains the true physical addresses.

- Updated pages are written to *different* locations
- *Two* page tables are kept:
  - *Shadow* (pointing to original pages on disk)
  - *Current* (pointing to updated pages)



Advantages:

- Simple recovery, no log in single-user systems (in multi-user systems, logs and checkpoints needed for concurrency control).

Disadvantages:

- Fragmentation of storage (clustering lost).
- Writing the shadow table to disk takes time.
- Garbage collection of pages needed.

ii) Domain Key Normal form

Domain Key Normal form

- **Defintion:**A relation schema is said to be in **DKNF** if all constraints and dependencies that should hold on the valid relation states can be enforced simply by enforcing the domain constraints and key constraints on the relation.

- The **idea** is to specify (theoretically, at least) the "*ultimate normal form*" that takes into account all possible types of dependencies and constraints. .
- For a relation in DKNF, it becomes very straightforward to enforce all database constraints by simply checking that each attribute value in a tuple is of the appropriate domain and that every key constraint is enforced.

The practical utility of DKNF is limited

iii) Template Dependency

Template Dependency

- Template dependencies provide a technique for representing constraints in relations that typically have no easy and formal definitions.
- The idea is to specify a template—or example—that defines each constraint or dependency.
- There are two types of templates: tuple-generating templates and constraint-generating templates.
- A template consists of a number of **hypothesis tuples** that are meant to show an example of the tuples that may appear in one or more relations. The other part of the template is the **template conclusion.**

Templates for some common types of dependencies.
(a) Template for functional dependency *X* –> *Y*.
(b) Template for the multivalued dependency *X* —>> *Y* .
(c) Template for the inclusion dependency *R.X* < *S.Y*.

(a)

$R = \{ A , B , C , D \}$

hypothesis

$a_1 \quad b_1 \quad c_1$

$a_1 \quad b_1 \quad c_2$

$X = \{A , B\}$

$Y = \{C , D\}$

conclusion

$c_1 = c_2$ and $d_1 = d_2$

(b)

$R = \{ A , B , C , D \}$

hypothesis

$a_1 \quad b_1 \quad 1 \quad d_1$

$a_1 \quad b_1 \quad 2 \quad d_2$

$X = \{A , B\}$

$Y = \{C\}$

conclusion

$a_1 \quad b_1 \quad 2 \quad d_1$

$a_1 \quad b_1 \quad 1 \quad d_2$

(c)

$R = \{ A , B , C , D \}$ $S = \{ E , F , G \}$

hypothesis

$a_1 \quad b_1 \quad c_1 \quad d_1$

$X = \{C , D\}$

$Y = \{E , F\}$

conclusion

$c_1 \quad d_1 \quad g$