
Transaction Management

Database Design Methodology - 1

Transaction Support

Transaction

Action or series of actions, carried out by user or application, which accesses or changes contents of database.

- Logical unit of work on the database.
- Application program is series of transactions with non-database processing in between.
- Transforms database from one consistent state to another, although consistency may be violated during transaction.

Database Design Methodology - 2

Example Transaction

```
delete(Sno = x)
for all Property_for_Rent records, pno
begin
  read(Sno = x, salary)
  salary = salary * 1.1
  write(Sno = x, new_salary)
  read(Pno = pno, sno)
  if (sno = x) then
  begin
    sno = new_sno
    write(Pno = pno, sno)
  end
end
end
```

(a) (b)

Database Design Methodology - 3

Transaction Support

- Can have one of two outcomes:
 - Success - transaction commits and database reaches a new consistent state.
 - Failure - transaction aborts, and database must be restored to consistent state before it started.
 - Such a transaction is rolled back or undone.
- Committed transaction cannot be aborted.
- Aborted transaction that is rolled back can be restarted later.

Database Design Methodology - 4

Properties of Transactions

Four basic (ACID) properties of a transaction are:

Atomicity 'All or nothing' property.

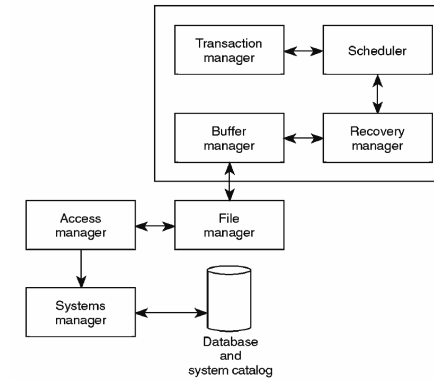
Consistency Must transform database from one consistent state to another.

Isolation Partial effects of incomplete transactions should not be visible to other transactions.

Durability Effects of a committed transaction are permanent and must not be lost because of later failure.

Database Design Methodology - 5

DBMS Transaction Subsystem



Database Design Methodology - 6

Concurrency Control

Process of managing simultaneous operations on the database without having them interfere with one another.

- o Prevents interference when two or more users are accessing database simultaneously and at least one is updating data.
- o Although two transactions may be correct in themselves, interleaving of operations may produce an incorrect result.

Database Design Methodology - 7

Need for Concurrency Control

- o Examples of potential problems caused by concurrency:
 - Lost update problem
 - Uncommitted dependency problem
 - Inconsistent analysis problem.

Database Design Methodology - 8

Lost Update Problem

Time	T ₁	T ₂	bal _x
t ₁		begin_transaction	100
t ₂	begin_transaction	read(bal _x)	100
t ₃	read(bal _x)	bal _x = bal _x + 100	100
t ₄	bal _x = bal _x - 10	write(bal _x)	200
t ₅	write(bal _x)	commit	90
t ₆	commit		90

Loss of T2's update avoided by preventing T1 from reading bal_x until after update.

Database Design Methodology - 9

Uncommitted Dependency Problem

Time	T ₃	T ₄	bal _x
t ₁		begin_transaction	100
t ₂		read(bal _x)	100
t ₃		bal _x = bal _x + 100	100
t ₄	begin_transaction	write(bal _x)	200
t ₅	read(bal _x)	:	200
t ₆	bal _x = bal _x - 10	rollback	100
t ₇	write(bal _x)		190
t ₈	commit		190

Problem avoided by preventing T3 from reading bal_x until after T4 commits or aborts.

Database Design Methodology - 10

Inconsistent Analysis Problem

Time	T ₅	T ₆	bal _x	bal _y	bal _z	sum
t ₁		begin_transaction	100	50	25	
t ₂	begin_transaction	sum = 0	100	50	25	0
t ₃	read(bal _x)	read(bal _x)	100	50	25	0
t ₄	bal _x = bal _x - 10	sum = sum + bal _x	100	50	25	100
t ₅	write(bal _x)	read(bal _y)	90	50	25	100
t ₆	read(bal _z)	sum = sum + bal _y	90	50	25	150
t ₇	bal _z = bal _z + 10		90	50	25	150
t ₈	write(bal _z)		90	50	35	150
t ₉	commit	read(bal _z)	90	50	35	150
t ₁₀		sum = sum + bal _z	90	50	35	185
t ₁₁		commit	90	50	35	185

Problem avoided by preventing T6 from reading bal_x and bal_z until after T5 completed updates.

Database Design Methodology - 11

Serializability

- Objective of a concurrency control protocol is to schedule transactions so as to avoid any interference.
- Could run transactions serially, but this limits degree of concurrency or parallelism in system.
- Serializability identifies those executions of transactions guaranteed to ensure consistency.

Database Design Methodology - 12

Nonserial Schedule

- **Schedule where operations from set of concurrent transactions are interleaved.**
- **Objective of serializability is to find nonserial schedules that allow transactions to execute concurrently without interfering with one another.**
- **In other words, want to find nonserial schedules that are equivalent to some serial schedule. Such a schedule is called serializable.**

Database Design Methodology - 13

Serializability

- **In serializability, ordering of read/writes is important:**
 - If two transactions only read a data item, they do not conflict and order is not important.**
 - If two transactions either read or write completely separate data items, they do not conflict and order is not important.**
 - If one transaction writes a data item and another reads or writes same data item, order of execution is important.**

Database Design Methodology - 14

Example of Serializability

T ₇	T ₈	T ₇	T ₈	T ₇	T ₈
begin_transaction		begin_transaction		begin_transaction	
read(bal _x)		read(bal _x)		read(bal _x)	
write(bal _x)		write(bal _x)		write(bal _x)	
	begin_transaction		begin_transaction		begin_transaction
	read(bal _x)		read(bal _x)		read(bal _x)
	write(bal _x)		write(bal _x)		write(bal _x)
read(bal _y)		read(bal _y)		commit	
write(bal _y)		write(bal _y)			begin_transaction
commit		commit			read(bal _y)
	read(bal _y)		read(bal _y)		write(bal _y)
	write(bal _y)		write(bal _y)		read(bal _x)
	commit		commit		write(bal _x)
					commit

(a)

(b)

(c)

Database Design Methodology - 15

Concurrency Control Techniques

- **Two basic concurrency control techniques:**
 - Locking
 - Timestamping
- **Both are conservative approaches: delay transactions in case they conflict with other transactions.**
- **Optimistic methods assume conflict is rare and only check for conflicts at commit.**

Database Design Methodology - 16

Locking

Transaction uses locks to deny access to other transactions and so prevent incorrect updates.

- Most widely used approach to ensure serializability.
- Generally, a transaction must claim a read (shared) or write (exclusive) lock on a data item before read or write.
- Lock prevents another transaction from modifying item or even reading it, in the case of a write lock.

Database Design Methodology - 17

Locking - Basic Rules

- If transaction has read lock on item, can read but not update item.
- If transaction has write lock on item, can both read and update item.
- Reads cannot conflict, so more than one transaction can hold read locks simultaneously on same item.
- Write lock gives transaction exclusive access to that item.

Database Design Methodology - 18

Example - Incorrect Locking Schedule

For two transactions above, a valid schedule is:

```
S = {write_lock(T9,balx), read(T9,balx),
      write(T9,balx), unlock(T9,balx),
      write_lock(T10,balx), read(T10,balx),
      write(T10,balx), unlock(T10,balx),
      write_lock(T10,baly), read(T10,baly),
      write(T10,baly), unlock(T10,baly), commit(T10),
      write_lock(T9,baly), read(T9,baly), write(T9,baly),
      unlock(T9,baly), commit(T9) }
```

Database Design Methodology - 19

Example - Incorrect Locking Schedule

- If at start, balx = 100, baly = 400, result should be:
 - balx = 220, baly = 330, if T9 executes before T10,
or
 - balx = 210, baly = 340, if T10 executes before T9.
- However, result gives balx = 220 and baly = 340.
- S is not a serializable schedule.

Database Design Methodology - 20

Example - Incorrect Locking Schedule

- o Problem is that transactions release locks too soon, resulting in loss of total isolation and atomicity.
- o To guarantee serializability, need an additional protocol concerning the positioning of lock and unlock operations in every transaction.

Database Design Methodology - 21

Two-Phase Locking (2PL)

Transaction follows 2PL protocol if all locking operations precede first unlock operation in the transaction.

Two phases for transaction:

- o Growing phase - acquires all locks but cannot release any locks.
- o Shrinking phase - releases locks but cannot acquire any new locks.

Database Design Methodology - 22

Preventing Lost Update Problem using 2PL

Time	T ₁	T ₂	bal _x
t ₁		begin_transaction	100
t ₂	begin_transaction	write_lock(bal _x)	100
t ₃	write_lock(bal _x)	read(bal _x)	100
t ₄	WAIT	bal _x = bal _x + 100	100
t ₅	WAIT	write(bal _x)	200
t ₆	WAIT	commit/unlock(bal _x)	200
t ₇	read(bal _x)		200
t ₈	bal _x = bal _x - 10		200
t ₉	write(bal _x)		190
t ₁₀	commit/unlock(bal _x)		190

Database Design Methodology - 23

Preventing Uncommitted Dependency Problem using 2PL

Time	T ₃	T ₄	bal _x
t ₁		begin_transaction	100
t ₂		write_lock(bal _x)	100
t ₃		read(bal _x)	100
t ₄	begin_transaction	bal _x = bal _x + 100	100
t ₅	write_lock(bal _x)	write(bal _x)	200
t ₆	WAIT	rollback/unlock(bal_x)	100
t ₇	read(bal _x)		100
t ₈	bal _x = bal _x - 10		100
t ₉	write(bal _x)		90
t ₁₀	commit/unlock(bal _x)		90

Database Design Methodology - 24

Preventing Inconsistent Analysis Problem using 2PL

Time	T ₅	T ₆	bal _x	bal _y	bal _z	sum
t ₁		begin_transaction	100	50	25	
t ₂	begin_transaction	sum = 0	100	50	25	0
t ₃	write_lock(bal _x)		100	50	25	0
t ₄	read(bal _z)	read_lock(bal _z)	100	50	25	0
t ₅	bal _x = bal _x - 10	WAIT	100	50	25	0
t ₆	write(bal _z)	WAIT	90	50	25	0
t ₇	write_lock(bal _z)	WAIT	90	50	25	0
t ₈	read(bal _z)	WAIT	90	50	25	0
t ₉	bal _z = bal _z + 10	WAIT	90	50	25	0
t ₁₀	write(bal _z)	WAIT	90	50	35	0
t ₁₁	commit/unlock(bal _x , bal _z)	WAIT	90	50	35	0
t ₁₂		read(bal _z)	90	50	35	0
t ₁₃		sum = sum + bal _x	90	50	35	90
t ₁₄		read_lock(bal _x)	90	50	35	90
t ₁₅		read(bal _y)	90	50	35	90
t ₁₆		sum = sum + bal _y	90	50	35	140
t ₁₇		read_lock(bal _z)	90	50	35	140
t ₁₈		read(bal _z)	90	50	35	140
t ₁₉		sum = sum + bal _z	90	50	35	175
t ₂₀		commit/unlock(bal _x , bal _y , bal _z)	90	50	35	175

Database Design Methodology - 25

Cascading Rollback

- If every transaction in a schedule follows 2PL, schedule is serializable.
- However, problems can occur with interpretation of when locks can be released.

Database Design Methodology - 26

Cascading Rollback

Time	T ₁₄	T ₁₅	T ₁₆
t ₁	begin_transaction		
t ₂	write_lock(bal _x)		
t ₃	read(bal _z)		
t ₄	read_lock(bal _y)		
t ₅	read(bal _y)		
t ₆	bal _x = bal _y + bal _x		
t ₇	write(bal _z)		
t ₈	unlock(bal _x)	begin_transaction	
t ₉	:	write_lock(bal _x)	
t ₁₀	:	read(bal _x)	
t ₁₁	:	bal _x = bal _x + 100	
t ₁₂	:	write(bal _z)	
t ₁₃	:	unlock(bal _x)	
t ₁₄	:	:	
t ₁₅	rollback	:	
t ₁₆	:	:	begin transaction
t ₁₇	:	:	read_lock(bal _z)
t ₁₈	:	rollback	:
t ₁₉	:	:	rollback

Database Design Methodology - 27

Cascading Rollback

- Transactions conform to 2PL.
- T₁₄ aborts.
- Since T₁₅ is dependent on T₁₄, T₁₅ must also be rolled back. Since T₁₆ is dependent on T₁₅, it too must be rolled back. Cascading rollback.
- To prevent this with 2PL, leave release of all locks until end of transaction.

Database Design Methodology - 28

Deadlock

An impasse that may result when two (or more) transactions are each waiting for locks held by the other to be released.

Time	T ₁₇	T ₁₈
t ₁	begin_transaction	
t ₂	write_lock(bal _x)	begin_transaction
t ₃	read(bal _x)	write_lock(bal _y)
t ₄	bal _x = bal _x - 10	read(bal _y)
t ₅	write(bal _x)	bal _y = bal _y + 100
t ₆	write_lock(bal _y)	write(bal _y)
t ₇	WAIT	write_lock(bal _x)
t ₈	WAIT	WAIT
t ₉	WAIT	WAIT
t ₁₀	:	WAIT
t ₁₁	:	:

Database Design Methodology - 29

Deadlock

- Only one way to break deadlock: abort one or more of the transactions.
- Deadlock should be transparent to user, so DBMS should restart transaction(s).
- Two general techniques for handling deadlock:
 - Deadlock prevention.
 - Deadlock detection and recovery.

Database Design Methodology - 30

Deadlock Prevention

- DBMS looks ahead to see if transaction would cause deadlock and never allows deadlock to occur.
- Could order transactions using transaction timestamps:
 - Wait-Die - only an older transaction can wait for younger one, otherwise transaction is aborted (dies) and restarted with same timestamp.
 - Wound-Wait - only a younger transaction can wait for an older one. If older transaction requests lock held by younger one, younger one is aborted (wounded).

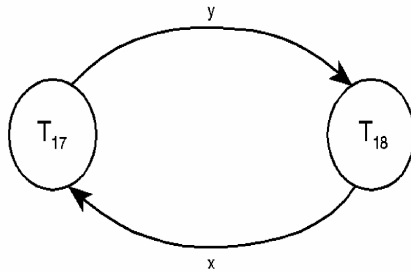
Database Design Methodology - 31

Deadlock Detection and Recovery

- DBMS allows deadlock to occur but recognizes it and breaks it.
- Usually handled by construction of wait-for-graph showing transaction dependencies:
 - Create a node for each transaction.
 - Create edge T_i -> T_j, if T_i waiting to lock item locked by T_j.
- Deadlock exists if and only if WFG contains cycle.
- Wait-for-graph is created at regular intervals.

Database Design Methodology - 32

Example - Wait-For-Graph



Database Design Methodology - 33

Timestamping

- Transactions ordered globally so that older transactions, transactions with smaller timestamps, get priority in the event of conflict.
- Conflict is resolved by rolling back and restarting transaction.
- No locks so no deadlock.

Timestamp

A unique identifier created by DBMS that indicates relative starting time of a transaction.

(from system clock or by incrementing a counter every time a new transaction starts)

Database Design Methodology - 34

Timestamping

- Read/write proceeds only if *last update on that data item* was carried out by an older transaction.
- Otherwise, transaction requesting read/write is restarted and given a new timestamp.
- Also timestamps for data items:
 - read-timestamp - timestamp of last transaction to read item.
 - write-timestamp - timestamp of last transaction to write item.

Database Design Methodology - 35

Timestamping

- Read(x)

Consider a transaction T with timestamp $ts(T)$:

$$ts(T) < \text{write_timestamp}(x)$$

- x already updated by younger (later) transaction.
- Rollback T and restarted with new timestamp.

$$ts(T) \geq \text{write_timestamp}(x)$$

- Read is executed
- Set $\text{read_timestamp}(x) =$

$$\max(\text{read_timestamp}(x), ts(T))$$

Database Design Methodology - 36

Timestamping

o Write(x)

Consider a transaction T with timestamp $ts(T)$:

$ts(T) < read_timestamp(x)$

- x already read by younger transaction.
- Roll back T and restart it using a later timestamp.

$ts(T) < write_timestamp(x)$

- x already written by younger transaction.
- Roll back T and restart it using a later timestamp.

o Otherwise, operation is accepted and executed.

Set write-timestamp(x) =

$\max(\text{write-timestamp}(x), ts(T))$

Database Design Methodology - 37

Example

Time	Op	T ₁₉	T ₂₀	T ₂₁
t ₁		begin_transaction		
t ₂	read(bal _x)	read(bal _x)		
t ₃	bal _x = bal _x + 10	bal _x = bal _x + 10		
t ₄	write(bal _x)	write(bal _x)		
t ₅	read(bal _y)		begin_transaction	
t ₆	bal _y = bal _y + 20		bal _y = bal _y + 20	begin_transaction
t ₇	read(bal _y)			read(bal _y)
t ₈	write(bal _y)		write(bal _y) [*]	
t ₉	bal _y = bal _y + 30			bal _y = bal _y + 30
t ₁₀	write(bal _y)			write(bal _y)
t ₁₁	bal _z = 100			bal _z = 100
t ₁₂	write(bal _z)			write(bal _z)
t ₁₃	bal _z = 50	bal _z = 50		commit
t ₁₄	write(bal _z)	write(bal _z) [‡]	begin_transaction	
t ₁₅	read(bal _y)	commit	read(bal _y)	
t ₁₆	bal _y = bal _y + 20		bal _y = bal _y + 20	
t ₁₇	write(bal _y)		write(bal _y)	
t ₁₈			commit	

Database Design Methodology - 38

Optimistic Techniques

- o Based on assumption that conflict is rare and more efficient to let transactions proceed without delays to ensure serializability.
- o At commit, check is made to determine whether conflict has occurred.
- o If there is a conflict, transaction must be rolled back and restarted.
- o Potentially allows greater concurrency than traditional protocols.

Database Design Methodology - 39

Optimistic Techniques

- o Three phases:
 - Read
 - Validation
 - Write.

Database Design Methodology - 40

Optimistic Techniques - Read Phase

- Extends from start until immediately before commit.
- Transaction reads values from database and stores them in local variables. Updates are applied to a local copy of the data.

Database Design Methodology - 41

Optimistic Techniques - Validation Phase

- Follows the read phase.
 - For read-only transaction, checks that data read are still current values. If no interference, transaction is committed, else aborted and restarted.
 - For update transaction, checks transaction leaves database in a consistent state, with serializability maintained.

Database Design Methodology - 42

Optimistic Techniques - Write Phase

- Follows successful validation phase for update transactions.
- Updates made to local copy are applied to the database.

Database Design Methodology - 43

3 Timestamps

- Start(T) – start of execution
- Validation(T) – start of validation phase
- Finish(T) – write phase complete

ts(T) = Validation(T) rather than start(T) for better response (provided conflict rates among transactions are low).

Database Design Methodology - 44

Validation Test

- For validation of T_j requires that for all $ts(T_i) < ts(T_j)$

1. $Finish(T_i) < Start(T_j)$

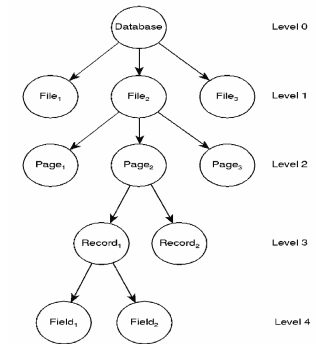
2. The set of data items written by T_i does not intersect with the set of data items read by T_j and T_i completes its write phase before T_j starts its validation phase

$$Start(T_j) < Finish(T_i) < Validation(T_j)$$

Database Design Methodology - 45

Granularity of Data Items

- Size of data items chosen as unit of protection by concurrency control protocol.



Database Design Methodology - 46

Database Recovery

Process of restoring database to a correct state in the event of a failure.

- Need for Recovery Control
 - Two types of storage: volatile (main memory) and nonvolatile.
 - Volatile storage does not survive system crashes.
 - Stable storage represents information that has been replicated in several nonvolatile storage media with independent failure modes.

Database Design Methodology - 47

Types of failures

- System crashes, resulting in loss of main memory.
- Media failures, resulting in loss of parts of secondary storage.
- Application software errors.
- Natural physical disasters.
- Carelessness or unintentional destruction of data or facilities.
- Sabotage.

Database Design Methodology - 48

Transactions and Recovery

- Transactions represent basic unit of recovery.
- Recovery manager responsible for atomicity and durability.
- If failure occurs between commit and database buffers being flushed to secondary storage then, to ensure durability, recovery manager has to redo (rollforward) transaction's updates.

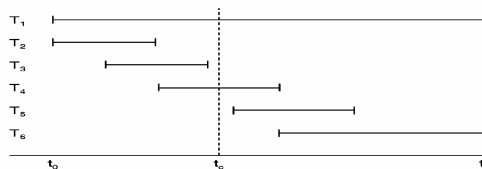
Database Design Methodology - 49

Transactions and Recovery

- If transaction had not committed at failure time, recovery manager has to undo (rollback) any effects of that transaction for atomicity.
- Partial undo - only one transaction has to be undone.
- Global undo - all transactions have to be undone.

Database Design Methodology - 50

Example



- DBMS starts at time t_0 , but fails at time t_1 . Assume data for transactions T_2 and T_3 have been written to secondary storage.
- T_1 and T_6 have to be undone. In absence of any other information, recovery manager has to redo T_2 , T_3 , T_4 , and T_5 .

Database Design Methodology - 51

Recovery Facilities

- DBMS should provide following facilities to assist with recovery:
 - Backup mechanism, which makes periodic backup copies of database.
 - Logging facilities, which keep track of current state of transactions and database changes.

Database Design Methodology - 52

Recovery Facilities

- o Checkpoint facility, which enables updates to database in progress to be made permanent.
- o Recovery manager, which allows DBMS to restore the database to a consistent state following a failure.

Database Design Methodology - 53

Log File

- o Contains information about all updates to database:
 - Transaction records.
 - Checkpoint records.
- o Often used for other purposes (for example, auditing).

Database Design Methodology - 54

Log File

- o Transaction records contain:
 - Transaction identifier.
 - Type of log record, (transaction start, insert, update, delete, abort, commit).
 - Identifier of data item affected by database action (insert, delete, and update operations).
 - Before-image of data item.
 - After-image of data item.
 - Log management information.

Database Design Methodology - 55

Sample Log File

Tid	Time	Operation	Object	Before image	After image	PPtr	NPtr
T1	10:12	START				0	2
T1	10:13	UPDATE	STAFF SL21	(old value)	(new value)	1	8
T2	10:14	START				0	4
T2	10:16	INSERT	STAFF SG37		(new value)	3	5
T2	10:17	DELETE	STAFF SA9	(old value)		4	6
T2	10:17	UPDATE	PROPERTY PG16	(old value)	(new value)	5	9
T3	10:18	START				0	11
T1	10:18	COMMIT				2	0
	10:19	CHECKPOINT	T2, T3				
T2	10:19	COMMIT				6	0
T3	10:20	INSERT	PROPERTY PG4		(new value)	7	12
T3	10:21	COMMIT				11	0

Database Design Methodology - 56

Checkpointing

Checkpoint

Point of synchronization between database and log file. All buffers are force-written to secondary storage.

- Checkpoint record is created containing identifiers of all active transactions.
- When failure occurs, redo all transactions that committed since the checkpoint and undo all transactions active at time of crash.

Database Design Methodology - 57

Checkpointing

- In previous example, with checkpoint at time t_c , changes made by T2 and T3 have been written to secondary storage.
- Thus:
 - only redo T4 and T5,
 - undo transactions T1 and T6.

Database Design Methodology - 58

Recovery Techniques

- If database has been damaged:
 - Need to restore last backup copy of database and reapply updates of committed transactions using log file.
- If database is only inconsistent:
 - Need to undo changes that caused inconsistency. May also need to redo some transactions to ensure updates reach secondary storage.
 - Do not need backup, but can restore database using before- and after-images in the log file.

Database Design Methodology - 59

Main Recovery Techniques

Three main recovery techniques:

- Deferred Update
- Immediate Update
- Shadow Paging.

Database Design Methodology - 60

Deferred Update

- Updates are not written to the database until after a transaction has reached its commit point.
- If transaction fails before commit, it will not have modified database and so no undoing of changes required.
- May be necessary to redo updates of committed transactions as their effect may not have reached database.

Database Design Methodology - 61

Immediate Update

- Updates are applied to database as they occur.
- Need to redo updates of committed transactions following a failure.
- May need to undo effects of transactions that had not committed at time of failure.
- Essential that log records are written before write to database. Write-ahead log protocol.

Database Design Methodology - 62

Immediate Update

- If no "transaction commit" record in log, then that transaction was active at failure and must be undone.
- Undo operations are performed in reverse order in which they were written to log.

Database Design Methodology - 63

Shadow Paging

- Maintain two page tables during life of a transaction: current page and shadow page table.
- When transaction starts, two pages are the same.
- Shadow page table is never changed thereafter and is used to restore database in event of failure.
- During transaction, current page table records all updates to database.
- When transaction completes, current page table becomes shadow page table.

Database Design Methodology - 64