Database Recovery
(Ref: Chapters 17 and 19, Silberschatz' Text))

TRANSACTION DEFINITION

- TRANSACTION: A logical unit of work: a sequence of several database operations that perform a single logical function in a DB application.

TRANSACTION DEFINITION (2)

- **Example:**

TRANSACTION DEFINITION (4)

- In the above example, to avoid database inconsistency, WHAT DO WE WANT???
- Answer:

TRANSACTION DEFINITION (5)

- General requirement for all transactions:
  - "NONE OR ALL": trans either executes in its entirety or is totally cancelled. This means each transaction must be a unit of ATOMICITY.

- Major issue: how to preserve atomicity despite failures?

TYPES OF FAILURES

- Transaction failure:

- System crash:

- Disk failure:

TRANSACTION MODEL

a) Requirements:

- Correctness:

- Atomicity:

TRANSACTION MODEL (2)

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 the ACID properties:

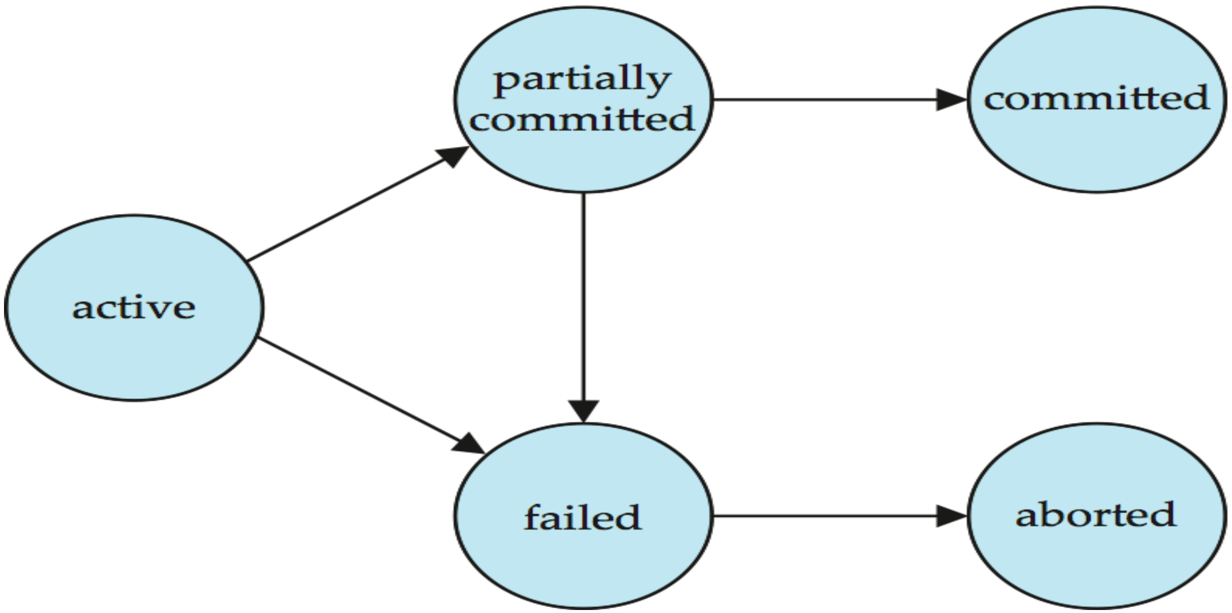
- **Atomicity:**
- **Consistency:**
- **Isolation:**
- **Durability:**

TRANSACTION MODEL (3)

b) Transaction States:

- **Active:**
- **Partially committed:**
- **Failed:**
- **Aborted** (rollback or undo):
- **Committed:**

TRANSACTION MODEL (5): Transaction State (Cont.)



FAILURE RECOVERY ALGORITHM

- Has two parts:
  - Actions taken during normal transaction processing:
    - Purpose?
  - Actions taken after a failure:
    - Purpose?

LOG-BASED RECOVERY

- To achieve atomicity:
  - Output updates to stable storage (called DB log disk)
  - Use the updates to recover DB in case of transaction/system failures (non-catastrophic failures)
- Two main techniques for failure recovery:
  - Deferred Update
  - Immediate Update

LOG-BASED RECOVERY (2): Deferred Update

a) Deferred Update (or Deferred DB Modification)

- Record DB updates in a log file
- Do not update DB until trans commits
- When trans commits, use the log info to update DB
- Log info is ignored if a failure occurs before the trans commits => NO UNDO operation is needed

LOG-BASED RECOVERY (3): Deferred Update (Cont.)

- When executing a WRITE (X, xi) operation, create a log record for X:

  <Transaction ID, Address of X, New Value of X>

- When trans starts, create a log record:

  <Transaction ID, Starts>

- When trans partially commits, create a log record:

  <Transaction ID, Commits>

LOG-BASED RECOVERY (4): Deferred Update (Cont.): Example

LOG-BASED RECOVERY (5): Deferred Update: Example (Cont.)

- **Show the log and DB status during the execution:**

LOG-BASED RECOVERY (6): Deferred Update (Cont.)

- To recover from failures: apply REDO operation on committed transaction Ti which has both log records <Ti, starts> and <Ti, commits>
- Example: in the log:

- What recovery operation will be performed from the above system crash?

Answer:

LOG-BASED RECOVERY (7)

b) Immediate Update (or Immediate DB Modification)

– DB may be updated by some operations of a trans before the trans commits. These updates are called UNCOMMITTED DATA or DIRTY DATA

– Record both old values and new values of updated items in the log: when executing write(X, x1), create a log record for X:

<Transaction ID, Address of X, Old value of X, New value of X>

LOG-BASED RECOVERY (8): Immediate Update (cont.)

- When trans starts, create a log record:

  <Transaction ID, Starts>

- When trans partially commits, create a log record:

  <Transaction ID, Commits>

- Recovery from a failure:
  - Perform operation REDO on committed transactions
  - Perform operation UNDO on uncommitted transactions.

LOG-BASED RECOVERY (9): Immediate Update Example (cont.)

- **Trans Operation** **Log Record** **DB status**

LOG-BASED RECOVERY (10): Immediate Update (cont.)

- Example: assuming that a system crash occurs right after write(C, c1) is executed in the above execution schedule, which recovery operations will the system perform?
- Answer:

LOG-BASED RECOVERY (11)

- General rule for UNDO and REDO operations: they must be IDEMPOTENT: yield the same results no matter how many times they have been executed.

CHECKPOINTS

- Purpose: to reduce the number of log records that need to be examined to determine transactions for UNDO and REDO operations
- When a checkpoint is performed, the system will perform the following tasks:
  - Output all log records in main memory to log disk
  - Output all modified buffer blocks from main memory to disk
  - Output a log record <Checkpoint ID> to log disk.

CHECKPOINTS (2)

- **Assume no concurrent transaction execution**, after a failure occurs, the system needs to perform recovery operations (REDO and/or UNDO) for:
  - Last transaction Ti that started before the last checkpoint took place
  - All transactions Tj that started after Ti.

CHECKPOINTS (3)

- Example: transactions executed in the order:

  T0, T1, T2, ..., T66, T67,.., T100

  – Assume that the last checkpoint was recorded during the execution of T67

  – Question: for which transactions will the system need to perform recovery operation?

  – Answer:

CHECKPOINTS (4)

- **Assume concurrent transaction execution:**
- Checkpoint log record is <checkpoint id, L>
- L: list of transactions active at the time of checkpoint.
- Recovery from failure:
  - Scan the log backward until a <checkpoint id, L> is found.
  - For each record found of form <Ti, commits>, add Ti to REDO-list
  - For each record found of form <Ti, starts> but not in REDO-list, add Ti to UNDO-list
  - Check list L: for each Ti in L, if Ti is not in REDO-list, then add Ti to UNDO-list
  - Re-scan the log from the most recent record backward and perform operation UNDO(Ti) for each Ti in UNDO-list
  - Scan the log forward and perform operation REDO(Ti) for each Ti in REDO-list

CHECKPOINTS (5)

- **Example: in the log:**

CHECKPOINTS (6)

- **Question: which recovery operations will the system perform for the above example?**
- Answer:

END OF TOPIC “DATABASE RECOVERY”