Topic 10 – Database Recovery (Chapters 17 and 19 (Silberschatz); and Chapter 8 (Basse)

(Modified for CS 4513)

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:** A database consists of
  - Table PART(pnum, pname, color, city, totqty)
  - Table SUPPLIER (snum, sname, status, city)
  - Table SUPPLIER_PART (snum, pnum, qty)

- **A transaction to add a new shipment (S5, P1, 1000) to the database is written as follows:**

```
•   EXEC SQL whenever SQLERROR GOTO UNDO;
•   EXEC SQL INSERT
•                   INTO SUPPLIER_PART (snum, pnum, qty)
•                   VALUES ('S5', 'P1', 1000)
•   EXEC SQL UPDATE PART
•                   SET totqty = totqty + 1000
•                   WHERE pnum = 'P1';
•   EXEC SQL COMMIT
•                   GOTO FINISH;
•   UNDO:
•     EXEC SQL ROLLBACK;
•   FINISH:         RETURN;
```

TRANSACTION DEFINITION (4)

- In the above example, to avoid database inconsistency, we want both insert and update either to be executed completely or not to be executed at all.

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: Two types of errors that may cause a transaction to fail:
  - Logical error: e.g. bad input, data not found
  - System error: the system has entered an undesirable state (e.g. deadlock)
- System crash: hardware errors or power failure or bugs in DBMS or OS that cause loss of volatile storage and bring transaction processing to a halt, but the content of nonvolatile storage is assumed to remain intact, not corrupted (fail-stop assumption)
- Disk failure: a disk block loses its content as a result of either a head crash or failure during a data transfer operation. Copies on other disks or storage media are used for recovery.

TRANSACTION MODEL

a) Requirements:

- Correctness: each trans must preserve DB consistency. => programmer's job
- Atomicity: each trans must be either executed all or none at all. => DBMS' job.

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 ACID properties:

- **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 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.

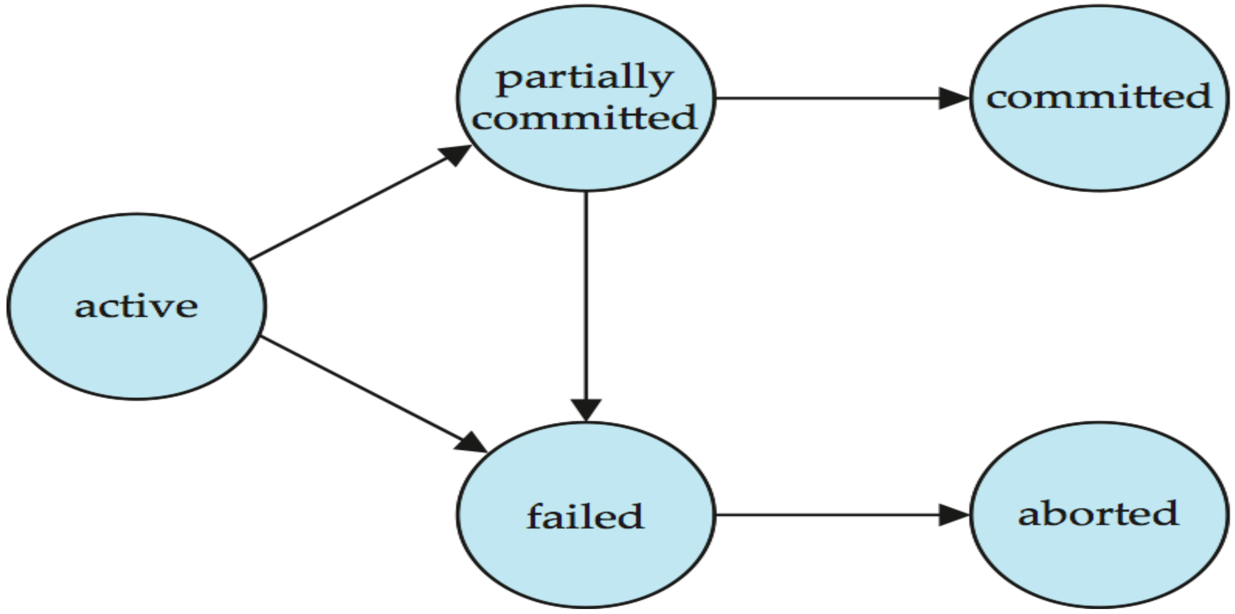
TRANSACTION MODEL (3)

b) Transaction States:

- **Active** – the initial state; the transaction stays in this state while it is executing
- **Partially committed** – after the final statement has been executed
- **Failed -- after the discovery that normal execution can no longer proceed
- **Aborted** (rollback or undo) – after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:
  - restart the transaction

    can be done only if no internal logical error
  - kill the transaction
- **Committed** – after successful completion. Updates done by trans are committed data (clean data) and can now be made permanent.

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



FAILURE RECOVERY ALGORITHM

- Has two parts:
  - Actions taken during normal transaction processing to ensure that enough information exists to allow recovery from failure
  - Actions taken after a failure to recover the database contents to a state that ensures database consistency, transaction atomicity, and durability.

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

- Transaction T0 transfers $50 from account A to account B:
- T0: read (A, a1)
- a1 = a1 - 50
- write (A, a1)
- read (B, b1)
- b1 = b1 + 50
- write (B, b1)
- Transaction T1 withdraws $100 from account C
- T1: read (C, c1)
- c1 = c1 - 100
- write (C, c1)
- A schedule of transaction execution:
- T0
- T1
- Assume that before the execution:
- A = 1000, B = 2000, C = 700

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

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

Trans Operation	Log Record	DB status
		A = 1000
		B = 2000
		C = 700
read(A, a1)	<T0, starts>	
a1 = a1-50		
write(A, a1)	<T0, A, 950>	
read(B, b1)		
b1 = b1 + 50		
write(B, b1)	<T0, B, 2050>	
partial commit of T0	<T0, commits>	
		A = 950
		B = 2050
read(C, c1)	<T1, starts>	
c1 = c1 -100		
write(C,c1)	<T1, C, 600>	
partial commit of T1	<T1, commits>	
		C = 600

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:

  <T0, starts>

  .

  .

  <T0, commits>

  <T1, starts>

  <T1, C, 600>

  ------------------->> System crashes here

- Recovery from the above system crash: **the system will perform operation REDO(T0)**

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 (cont.)

Trans Operation	Log Record	DB status
		A = 1000
		B = 2000
		C = 700
read(A, a1)	<T0, starts>	
a1 = a1-50		
write(A, a1)	<T0, A,1000, 950>	
		A = 950
read(B, b1)		
b1 = b1 + 50		
write(B, b1)	<T0, B, 2000, 2050>	
		B = 2050
partial commit of T0	<T0, commits>	
read(C, c1)	<T1, starts>	
c1 = c1 -100		
write(C,c1)	<T1, C, 700, 600>	
		C = 600
partial commit of T1	<T1, commits>	

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

- Example: a system crash occurs right after write(C, c1) is executed in the above execution schedule:

  => the system will perform REDO(T0) and UNDO(T1)

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
  - Then need recovery operations for transactions T67 through T100 only.

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:**

  <T1, starts>

  .

  .

  <T2, starts>

  .

  .

  <T2, commits>

  .

  .

  <T3, starts>

  .

  .

  <T4, starts>

  <checkpoint 1, (T1, T3, T4)>

  .

  .

  <T3, commits>

  .

  .

  <T5, starts>

  .

  .

  <T1, commits>

  -------------------->> System crashes here

CHECKPOINTS (6)

- **REDO-list: T1, T3**
- **UNDO-list: T4, T5**

- **The system will perform the recovery operations:**
  - **UNDO(T4, T5)**
  - **REDO(T1, T3)**

FAILURES AND ERRORS IN COMPUTER SYSTEMS

Chapter 8 (Basse)

EXAMPLES OF COMPUTER SYSTEM FAILURES AND ERRORS

- Headlines describing real incidents:
  - “Navigation System Directs Car Into River”
  - “Data Entry Typo Mutes Millions of U.S. Pagers”
  - “Flaws Found in Software That Tracks Nuclear Materials”
  - “Software Glitch Makes Scooter Wheels Suddenly Reverse Direction”
  - “IRS Computer Sends Bill for $68 Billion in Penalties”
  - “Robots Kill Worker”
  - “California Junks $100 Million Child Support System”
  - “Man Arrested Five Times Due to Faulty FBI Computer Data”

More Recent Examples of Computer Failures and Errors

- **RBS computer failure to cost bank £100m**
  - Royal Bank of Scotland expected to announce compensation fund and may increase PPI mis-selling provision
  - guardian.co.uk, Thursday 2 August 2012 09.19 EDT
  - Chief executive Stephen Hester is expected to again apologise to the up to 13 million customers of RBS, NatWest and Ulster Bank who could not be certain how much money they had in their accounts in the biggest computer problem ever incurred by a UK bank, which was first discovered on June 12, 2012
- **United Airlines' Flights Are Delayed by a Computer Failure**
  - By THE ASSOCIATED PRESS
  - Published: November 15, 2012
  - A computer failure at United Airlines delayed thousands of travelers on Thursday and embarrassed the airline at a time when it has been trying to win back customers after problems earlier this year.
  - The two-hour failure held up 250 of the 5,679 United flights scheduled for Thursday, the airline said.
- **The U.S. Federal Government Healthcare Website failure:**
  - Healthcare.gov was Launched October 1, 2013;
  - Few people could enroll due to the Website failure as of now.

SOME FACTORS IN COMPUTER SYSTEM FAILURES AND ERRORS

- Design and development
  - Inadequate attention to potential safety risks
  - Interaction with physical devices that do not work as expected
  - Incompatibility of software and hardware or of application software and the operating system
  - Not planning and designing for unexpected inputs or circumstances
  - Insufficient testing
  - Reuse of software from another system without adequate checking
  - Overconfidence in software
  - Carelessness.

SOME FACTORS IN COMPUTER SYSTEM FAILURES AND ERRORS (2)

- Management and use
  - Data entry errors
  - Inadequate training of users
  - Errors in interpreting results or output
  - Failure to keep information in databases up to date (questions for database professionals: what design features in databases can encourage updating or reduce problems resulting from out-of-date information?)
  - Overconfidence in software by users
  - Insufficient planning for failures, not backup systems and procedures.

SOME FACTORS IN COMPUTER SYSTEM FAILURES AND ERRORS (3)

- Misrepresentation, hiding problems, and inadequate response to reported problems
- Insufficient market or legal incentives to do a better job.

PROFESSIONAL TECHNIQUES

- Software engineering and professional responsibility
  - Follow good software engineering techniques in all stages of development:
    - Specifications, design, implementation, documentation, and testing
  - Study and use the techniques and tools that are available and follow the procedures and guidelines established in relevant codes of ethics and professional practices (e.g. ACM Code of Ethics and Professional Conduct)

PROFESSIONAL TECHNIQUES (2)

- User interfaces and human factors
  - Well-designed user interfaces can help avoid many computer-related problems
  - System designers and programmers need to learn from psychologists and human-factor experts
- Redundancy and self-checking
- Testing
  - adequate and well-planned
  - not arbitrary
  - even small changes need thorough testing

LAW, REGULATION, AND MARKETS

- Criminal and civil penalties
  - for faulty systems
- Warranties for consumer software
- Taking responsibility: market mechanisms:
  - Business pressure: in some cases of computer errors, businesses pay customers for problems or damages
  - Availability requirement:
    - e.g. some businesses with high availability requirement may pay a higher rate for uninterrupted services

IMPORTANT DIFFERENCES BETWEEN COMPUTERS AND OTHER TECHNOLOGIES

- Computers make decisions; electricity does not
- The power and flexibility of computers encourages us to build more complex systems where failures have more serious consequences
- The pace of change in computer technology is much faster than that in other technologies
- Software is not built from standard, trusted parts as is the case in many engineering fields
- These differences affect the kind and scope of the risks we face
- They need our attention as computer professionals and as members of the public

SUMMARY

- The potential for serious disruption of normal activities and danger to people's lives and health because of flaws in computer systems should always remind the computer professional of the importance of doing his/her job responsibly
- Computer professionals and other professionals responsible for planning and choosing systems must
  - assess risks carefully and honestly
  - include safety protections
  - make appropriate plans
    - for shutdown of a system when it fails
    - for backup systems where appropriate
    - for recovery

END OF TOPIC 10