BASIC TRANSACTION CONCEPTS

Basic Definitions

- **A Transaction:** logical unit of database processing that
  - includes one or more access operations (read -retrieval, write – insert or update, delete).
- **A transaction (set of operations)** may be stand-alone specified in a high level language like SQL submitted interactively, or may be embedded within a program.
- **Transaction boundarie**s: Begin and End transaction.
- An **application program** may contain several transactions
  - separated by the Begin and End transaction boundaries.

Simple Model of a Database

(for purposes of discussing transactions):

- **• A database - collection of named data items**
- **• Granularity of data - a field, a record , or a whole disk block**
- **(Concepts are independent of granularity)**
- **Basic operations are read and write**
  - **read_item(X)**: Reads a database item named X into a program_variable. To simplify our notation, we assume that *the program variable is also named X, i.e X=r(x)*
  - **write_item(X)**: Writes the value of program variable X into the database item named X.

Read Operations

- Basic unit of data transfer from the disk to the computer main memory is one block. In general, a data item (what is read or written) will be
  - Field/record/block.
- **read_item(X) command includes the following steps:**
  - 1. Find the address of the disk block that contains item X.
  - 2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
  - 3. Copy item X from the buffer to the program variable named X.

Write Operations

- **write_item(X) command includes the following steps:**
  - 1. Find the address of the disk block that contains item X.
  - 2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
  - 3. Copy item X from the program variable named X into its correct location in the buffer.
  - 4. Store the updated block from the buffer back to disk (either immediately or at some later point in time).

Concurrent Transactions

- What is the motivation: <u>PERFORMANCE</u>
- Potential Problems
  - Lost Update Problem
  - Temporary Update Problem
  - Incorrect Summary Problem

Lost Update Problem

- Initial val X =4
- N =2, M=3
- Final value of X is 7
- X= 2 was lost!

T_1	T_2
read_item(*X*); *X*:=*X*-*N*;	
	read_item(*X*); *X*:=*X*+*M*;
write_item(*X*); read_item(*Y*);	
	write_item(*X*);
Y:=*Y*+*N*; write_item(*Y*);	

Temporary Update Problem

T1 has updated X but has not committed.
T2 has read the updated X.
T1 fails and needs to undo the write to X
Problem: T2 has read this value so it needs to undo as well.
DIRTY READ PROBLEM

T_1	T_2

read_item(*X*);
X:=*X*-*N*;
write_item(*X*);	

	read_item(*X*);
	X:=*X*+*M*;
	write_item(*X*);

read_item(*Y*);	

Incorrect Summary Problem

- Calculate some values before they are updated (Y)
- Calculate some values after they are updated (X)

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*);	

Need for Recovery Techniques

- Causes
  - Computer failure (crash)
  - Transaction/System error (overflow/underflow)
  - Local errors/exception errors
    - E.g. Insufficient funds in a banking transaction
  - Concurrency control enforcement
    - Serializability conflicts (will cover later)
  - Disk failure
  - Physical problems

Additional Operations

- BEGIN_TRANSACTION
- READ OR WRITE
- END_TRANSACTION
- COMMIT_TRANSACTION
  - This is when the transaction is deemed executed and cannot be retracked
- ROLLBACK (or ABORT)
  - Effects of transaction must be undone
- UNDO
  - Rollback to a single operation
- REDO
  - Certain transaction operations must be redone to ensure that all the operations of a committed transaction have been applied successfully to the database

System Log

- Keeps track of all transaction operations that affect values of database items
- Log is kept on disk and periodically backed up to guard against catastrophy.
  - Transaction ID
  - [start, TID]
  - [write_item, TID, X, old_val, new_val]
  - [read_item, TID, X]
  - [commit, TID]
  - [abort, TID]
  - Protocols that use cascading rollbacks need all the above. More on this later.

Recovery Using Log Records

- Chapter 21 for details. Basic idea
  - UNDO: current value of X = 5
    - Suppose you want to undo.
    - Log contains TID, write, X, old_val, new_val(=5)
    - Change X to old_value
    - Basically backtrace through the log file and undo the relevant operations
  - REDO:
    - One can redo the effect of the WRITE operations of a transaction T by tracing forward through the log and setting all items changed by a WRITE operation of T (that did not get done) to their new values.

Commit Point of a Transaction

- Commit Point Criteria
  - All operations have executed successfully
  - All operations have been logged
- Beyond Commit Point, Transaction is considered permanently committed which is logged [commit, TID]
- Roll back
  - Applies to those transactions that have a [start,TID] entry in the log but no [commit,TID]
- Redoing
  - May apply to transactions that have both start and commit entries in the log. When recovering from a crash one may need to redo the effect of a transaction.
    - This presumes only information that has been written back to disk before the crash.

Commiting (contd.)

- Force Writing a Log
  - Many systems require that before a transaction reaches it s commit point any pertion of the log that has not been written tot the disk yet is written to the disk. This process is referred to as force-writing the log file before committing a transaction.

ACID:Desirable TransactionProperties

- Atomicity
  - A transaction is either completely done or not done at all. There is no notion of partial transaction.
- Consistency Preservation
  - A correct execution must take the database from one consistent state to another.
- Isolation: (ideally)
  - A transaction should not make its updates visible to other transactions till commit point. Under strict enforcement this
    - Avoids temporary update problem
    - Avoids cascading rollbacks
- Durability or Permanancy
  - Once committed, transaction effects should never be lost.

Transaction Schedules

- When transactions are excecuting concurrently, the order of operations from various transactions forms a transaction schedule.
  - They can be interleaved
  - Two operations in the same transaction must appear in the schedule in the same order that they appear in the transaction
  - (T1,op1), (T2,op1), (T2,op2), (T1,op2), (T2,op3): **valid**
  - (T1,op1), (T2,op2), (T1,op2), (T2,op1): **invalid**

Recoverable schedule

- One where no transaction needs to be rolled back. Is guaranteed if
  - No transaction T in S commits until all transactions T' that have written an item that T reads has committed

Start T1	Start T1
Start T2	Start T2
R(x) T1	R(x) T1
W(y) T2	R(y) T1
R(y) T1	W(y) T2
Commit T1	Commit T1
R(x) T2	R(x) T2
……	……
(Recoverable?)	(Recoverable?)
NO	YES

If T2 aborts here then T1 would have to be aborted after commit violating Durability of ACID

Cascadeless Schedules

- Those where every transaction reads only the items that are written by committed transactions
- Cascaded Rollback Schedules
  - A schedule in which uncommitteed transactions that read an item from a failed transaction must be rolled back

Start T1	Start T1
Start T2	Start T2
R(x) T1	R(x) T1
W(x) T1	W(x) T1
R(x) T2	R(y) T1
R(y) T1	W(y) T1
W(x) T2	Commit T1
W(y) T1	R(x) T2
……	W(x) T2

If T1 were to abort here then T2 would Have to abort in a cascading fashion. This is a cascaded rollback schedule

Cascadeless Schedule

Strict Schedules

- A transaction can neither read or write an item X until the last transaction that wrote X has committed.

(say x = 9)
Start T1
Start T2
R(y) T2
R(x) T1
W(x) T1 (say x = 5)
R(y) T1
W(y) T1
W(x) T2 (say x = 8)

For this example
Say T1 aborts here
Then the recovery process will restore the value of x to 9
Disregarding (x= 8). Although this is cascadeless it is not
Strict and the problem needs to be resolved: use REDO