Lecture 7

Database Management Systems

Winter 2004

CMPUT 391: Transactions Models

Dr. Osmar R. Zaïane

University of Alberta

Chapter 21 of Textbook

Objectives of Lecture 7

Transactions Models

- Illustrate how single tasks may be broken up into several transactions
- Describe some transaction structuring mechanisms
- Hint on issues related to distributed transactions

Flat Transaction

- Consists of:
  - Computation on local variables
  - Access to DBMS using call or statement level interface
- No internal structure
- Accesses a single DBMS
- Adequate for simple applications

```
begin transaction
    :
  EXEC SQL …..
    :
  EXEC SQL …..
    :
commit
```


Flat Transaction

- Abort causes the execution of a program that restores the variables updated by the transaction to the state they had when the transaction first accessed them.

```
begin transaction
    :
  EXEC SQL …..
    :
  EXEC SQL …..
    :
  if condition then abort
    :
commit
```

Some Limitations of Flat Transactions

- Only total rollback (abort) is possible
  - Partial rollback not possible
- All work lost in case of crash
- Limited to accessing a single DBMS
- Entire transaction takes place at a single point in time

Providing Structure Within a Single Transaction

Savepoints

- **Problem**: Transaction detects condition that requires rollback of *recent* database changes that it has made
- **Solution 1**: Transaction reverses changes itself
- **Solution 2**: Transaction uses the rollback facility within DBMS to undo the changes

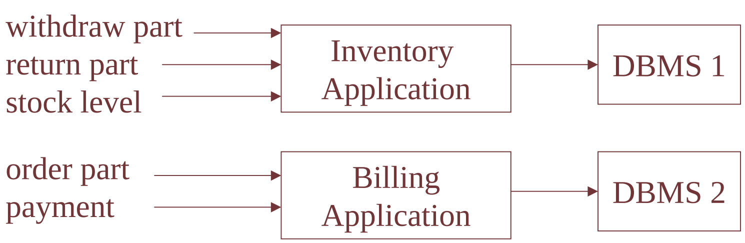
Savepoints

```
begin transaction
   S1;
   sp1 := create_savepoint();
   S2;
   sp2 := create_savepoint();
   S3;
   if (condition) {rollback (sp1); S5};
   S4;
commit
```

- Rollback to sp_i causes database updates subsequent to creation of sp_i to be undone
- Program counter and local variables are *not* rolled back (why not?)
- Savepoint creation does not make prior database changes durable (abort rolls *all* changes back)

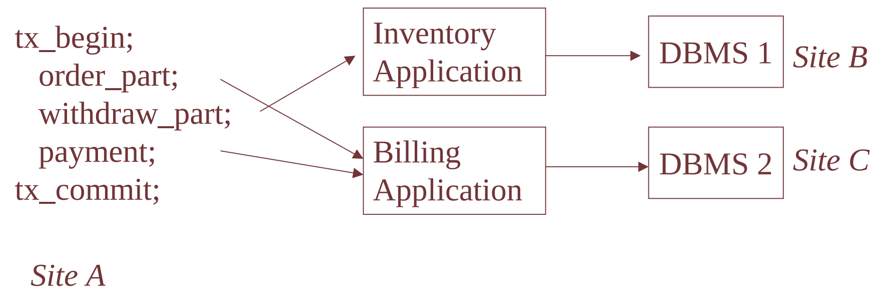
Integration of Applications

- **Problem**: Many enterprises consist of multiple **legacy systems** doing separate tasks. Increasing automation requires that these systems be integrated



Distributed Transactions

- Incorporate (legacy) transactions at multiple servers into a single (distributed) transaction



Distributed Transactions

- **Goal**: distributed transaction should be ACID
  - Each subtransaction is *locally* ACID (*e.g., local* constraints maintained, *locally* serializable)
  - In addition the transaction should be *globally* ACID
    - **A**: Either *all* subtransactions commit or all abort
    - **C**: *Global* integrity constraints are maintained
    - **I**: Concurrently executing distributed transactions are *globally* serializable
    - **D**: Each subtransaction is durable

Banking Example

- **Global atomicity** - funds transfer
  - Either both subtransactions commit or neither does

```
tx_begin;
  withdraw(acct1);
  deposit(acct2);
tx_commit;
```

Banking Example (con't)

- **Global consistency -**
  - Sum of all account balances at bank branches = total assets recorded at main office

Banking Example (con't)

- **Global isolation -** local serializability at each site does not guarantee global serializability
  - *post_interest* subtransaction is serialized after *audit* subtransaction in DBMS at branch 1 and before *audit* in DBMS at branch 2 (local isolation), *but*
  - there is no global order

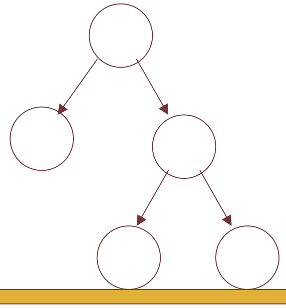
time ↓	*post_interest*	*audit*
		sum balances at branch 1;
	post interest at branch 1;	
	post interest at branch 2;	
		sum balances at branch 2;

Multidatabase

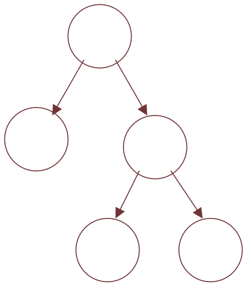
- Set of databases accessed by a distributed transaction is referred to as a **multidatabase** (or federated database)
  - Each local database retains its local autonomy and might execute local (non-distributed) transactions
- Multidatabase might have global integrity constraints
  - *e.g.*, Sum of balances of individual bank accounts at all branch offices = total assets stored at main office

Transaction Hierarchy

- A distributed transaction invokes subtransactions.
- General model: hierarchy of subtransactions.



Models of Distributed Transactions



- Can siblings execute concurrently?
- Can parent execute concurrently with children? If yes, can parent communicate with child?
- Who initiates commit?

Hierarchical Model: No concurrency (hence no communication between subtransactions), root initiates commit

Peer Model: Concurrency among siblings and between parent and children, concurrent subtransactions can communicate, any subtransaction can initiate commit

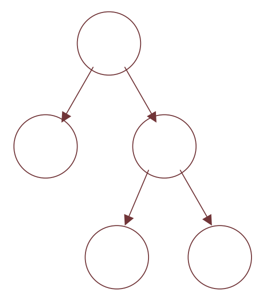
Distributed Transactions

- Transaction designer has little control over the structure. Decomposition fixed by distribution of data and/or exported interfaces
- Essentially a *bottom-up* design

Nested Transactions

- **Problem**: Lack of mechanisms that allow:
  - a *top-down*, functional decomposition of a transactional application into subtransactions
  - individual subtransactions to abort without aborting the entire transaction
- Although a nested transaction looks similar to a distributed transaction, it is *not* conceived of as a tool for accessing a multidatabase

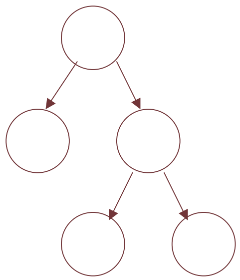
Characteristics of Nested Transactions



- (1) Parent can create a set of children that execute concurrently; parent waits until all children complete (no communication between parent and children).

- (2) Each subtransaction (together with its descendants) is isolated with respect to each sibling (and its descendants). Hence, siblings are serializable, but order is not determined and nested transaction is *non-deterministic*.
- (3) Concurrent nested transactions are serializable.

Characteristics of Nested Transactions



- (4) A subtransaction is atomic. It can abort or commit independently of other subtransactions. Commitment is *conditional* on commitment of parent. Abort causes abort of all its children.

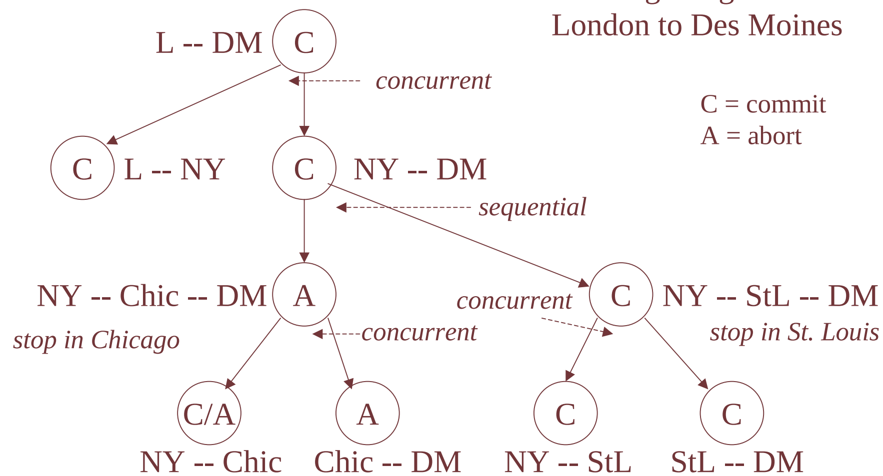
- (5) Nested transaction commits when root commits. At that point updates of committed subtransactions are made durable.

(6) Individual subtransactions are not necessarily consistent, but nested transaction as a whole is consistent

Nested Transaction - Example

Booking a flight from London to Des Moines

C = commit
A = abort



Nested Transactions