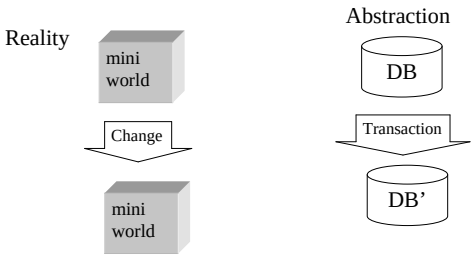
Transaction Processing

Introduction

The state of the fragment of the real world, called *miniworld*, is represented by an abstraction, called the *database,* and the transformation of the miniworld is mirrored by the execution of a program, called a transaction, that transform the database from one consistent state into another consistent one.

Introduction

The basic abstraction of transaction processing system may be represented as follows.



Transaction

A transaction is a collection of operations on the physical and abstract application state.

A transaction is a sequence of database operations (actions) that transforms a database from one consistent state into another one. The actions that can be executed by a transaction include **reads** and **writes** of database objects. In addition to reading and writing, each transaction must specify as its final action either **commit** (i.e. complete successfully) or **abort** (i.e. terminate and undo all the actions carried out so far).

Example

Consider an example of a transaction T that transfers an amount N from the account A to B.

```
begin
      // withdraw an amount N from account A;
      update account
            SET balance = balance - N
            where id_account = A;
      // deposit an amount N on account B;
      update account
            SET balance = balance + N
            where id_account = B;
      commit;
 end
```

Problems with writing database applications

- **Creating inconsistent results**. Our application is transferring money from one account to another. After the first account has had money subtracted from its balance and this change has been recorded on disk, the system crashes due to a power failure. When the machine back up, our application does not remember what logic it was executing – the application has destroyed money with the single account record update it made.

Problems with writing database applications

- **Errors of concurrent execution.** Concurrent access to the same data may lead to database anomalies: inconsistent analysis, dirty updates, etc.
- **Uncertainty as to when changes become permanent**. Popular pages remain in memory (buffer) for an extended period. After crash, we can only remember what has been written to disk – so, all pages remaining in the memory are lost.

Transaction Properties

•**A**(tomicity)**C**(onsistency)**I**(solation)**D**(urability)

A transaction can be considered as a collection of actions with the following properties:

• **Atomicity**. A transaction's changes to the state are atomic: either all happen or none happen. These changes include database actions, messages, and actions on external devices. Particularly, the property guarantees that a set of updates that are part of a transaction is atomic. Thus either all updates of a transaction occur in the database or none of them occurs (it solves the first problem).

Transaction Properties

• **Consistency**. A transaction is a correct transformation of the state. The actions taken as a group do not violate any of integrity constraints associated with the state. This requires that the transaction be a correct program. Correct execution of a transaction transforms a database from one consistent state into another.

• **Isolation.** Transactions are isolated which means that one transaction can only affect another as it would if they were not concurrent. In other words, even though transactions execute concurrently, it appears to each transaction, T, that other transactions are executed either before T or after T, but not both (it solves the second problem).

Transaction Properties

• **Durability.** Once a transaction completes successfully, its changes to the state survive failures. The property guarantees that when the system returns to the program logic after a Commit statement, the transaction is guaranteed to be recoverable In other words, a transaction is resistant to a crash (it solves the third problem).

ACID properties

Example (cont):

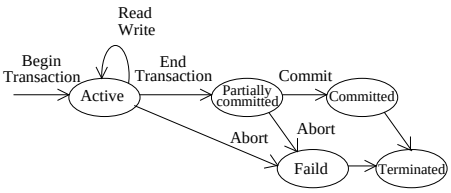
```
begin
   // withdraw an amount N from account A;
   update account
        SET balance = balance - N
        where id_account = A;
   // deposit an amount N in account B;
   update account
        SET balance = balance + N
        where id_account = B;
   commit;
end;
```

Example (cont):

The transaction is:

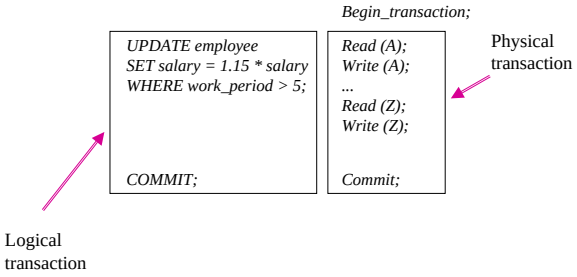
- **Atomic** if we correctly transfer money from A to B;
- **Consistent** if the money withdrew from A is the same as the deposit to B;
- **Isolated** if the transaction can be unaware of other transactions reading or writing accounts A and B concurrently (e.g. your spouse making a concurrent debit);
- **Durable** if, once the transaction is complete, the accounts A and B are sure to reflect the withdrawal and deposit.

Transaction's State Diagram



Begin_transaction: begin of the transaction.
Read, ***Write***: read, write operations on the database
End_transaction: end of the transaction:
Commit: commit of the transaction.
Rollback (abort): roll back of the transaction

Logical vs. Physical Transaction



Problems with concurrent execution of transactions

- **Database inconsistency resulting from a crash of the system**. Due to the failure (crash) of the system only a part of a transaction (part of transaction's operations) was performed.
- **Anomalies resulting from the concurrent execution of transactions.** Interleaving of operations of concurrently executed transactions may cause some anomalies.
- **Lost of data due to the system crash**. Results of committed transactions (i.e. transactions that successfully completed) stored in the main memory have been lost due to the system crash.

Anomalies of concurrent execution of transactions

Lost update

T1	T2
read(X);	
X := X-N;	
	read(X);
	X := X+M;
write(X);	
read(Y);	
	write(X);
	commit;
Y := Y+N;	
write(Y);	
commit;	

Dirty read

T1	T2
read(X);	
X := X-N;	
write(X);	
	read(X);
	X := X+M;
	write(X);
	commit;
read(Y);	
abort;	

Anomalies of concurrent execution of transactions

Incorrect Summary

T1	T3
	sum := 0;
read(X);	
X := X-N;	
write(X);	
	read(X);
	sum := sum+X;
	read(Y);
	sum := sum+Y;
	commit;
read(Y);	
Y := Y+N;	
write(Y);	

Where transactions come from?

•Transactions specify simple failure semantics for a computation.

It is up to the system implementers to provide these properties to the application programmers.

The transaction concept is the computer equivalent of contract law. If nothing ever goes wrong, contracts are just overhead. But if something doesn't quite work, the contract specifies how to clean up the situation.

Where transactions come from?

- Many of the techniques used to achieve ACID properties have direct analogies in human systems.
  - The notions of atomicity and durability are explicit in contract law
  - Christian wedding ceremony is the example of *two-phase commit protocol*
- There are many examples of systems that tried and failed to implement fault-tolerant or distributed computations using ad-hoc techniques rather than a transaction concept. Now, almost all information systems (99%) are transaction processing systems.