Transaction Processing Concepts

Transaction

A transaction is a sequence of database operations (actions) that transformes 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

Transfer the amount N from account A to account B:

```
begin
    // subtract amount N from account A;
    UPDATE account
        SET balance = balance - N
        WHERE account_id = A;
    // deposit amount N on account B;
    UPDATE account
        SET balance = balance + N
        WHERE account_id = B;
    COMMIT;
end
```

Problems with writing database applications

- **Creating an inconsistent results**. Our application is transfering 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.
- **Errors of concurrent execution.** Concurrent access to the same data may lead to database anomalies: inconsistent analysis, dirty updates, etc.
- **Uncertainity as to when changes become permanent**. Popular pages remains 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)

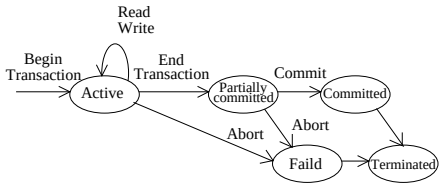
Atomicity. The property guarantees that a set of updates that are part of a transaction is atomic. Thus either all updates of a transaction occur in the database or none of them occurs (it solves the firs problem).

Consistency. Correct execution of a transaction transformes a database from one consistrent state into another.

Isolation. Transactions are isolated which means that one transaction can only affect another as it would if they were not concurrent. This guarantee solves problem 2.

Durability. 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).

State transition diagram for transaction execution



- ***Begin_transaction***: this specify the beginning of transaction execution.
- ***Read***, ***Write***: these specify read and write operations on the database items that are executed as part of a transaction.
- ***End_transaction***: this specifies that read and write operations have ended and marks the end limit of transaction execution. At this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed0 or whether the transaction has to be aborted.

- ***Commit***: this signals a successful end of the transaction so that any updates executed by the transaction can be safely committed to the database and will not be undone.
- ***Rollback***: this signals that the transaction has ended unsuccessfully, so that any chages or effects taht the transaction may have applied to the database must be undone.

```
                                Begin_transaction;
UPDATE employee                 Read (A);
SET salary = 1.15 * salary      Write (A);
WHERE work_period > 5;          ...
                                Read (Z);
                                Write (Z);
COMMIT;                         Commit;
```

Why concurrency control is needed?

- ***Lost update***

T_1	T_2
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***

T_1	T_2
read(X);	
X := X-N;	
write(X);	
	read(X);
	X := X+M;
	write(X);
	commit;
read(Y);	
abort;	

- ***Incorrect Summary***

T_1	T_3
	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);	