**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System**

**Lab Assignment #**

**Submitted by:**

Arun Sanjel  
013BSCCSIT010

**Submitted to:**

|  |  |
| --- | --- |
| **Er. Sanjay Kumar Yadav**  Lecturer  St. Xavier’s College |  |

**Date of Submission:** October 29, 2015

# Transaction Management

## Introduction

A transaction is an event which occurs on the database. Generally a transaction reads a value from the database or writes a value to the database. If you have any concept of Operating Systems, then we can say that a transaction is analogous to processes.

Although a transaction can both read and write on the database, there are some fundamental differences between these two classes of operations. A read operation does not change the image of the database in any way. But a write operation, whether performed with the intention of inserting, updating or deleting data from the database, changes the image of the database. That is, we may say that these transactions bring the database from an image which existed before the transaction occurred (called the**Before Image** or **BFIM**) to an image which exists after the transaction occurred (called the **After Image** or **AFIM**).

# Transactions

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

Let’s take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

A’s Account

Open\_Account(A)

Old\_Balance = A.balance

New\_Balance = Old\_Balance - 500

A.balance = New\_Balance

Close\_Account(A)

B’s Account

Open\_Account(B)

Old\_Balance = B.balance

New\_Balance = Old\_Balance + 500

B.balance = New\_Balance

Close\_Account(B)

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

* Atomicity − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.
* Consistency − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
* Durability − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
* Isolation − In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

**Transaction Recovery**:

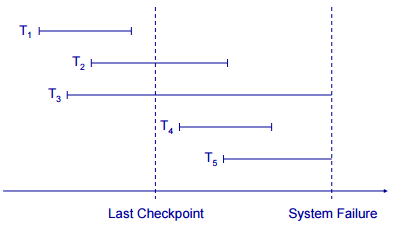
When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items. Transactions are made of various operations, which are atomic in nature.

When a DBMS recovers from a crash, it should maintain the following −

* It should check the states of all the transactions, which were being executed.
* A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
* It should check whether the transaction can be completed now or it needs to be rolled back.

No transactions would be allowed to leave the DBMS in an inconsistent state

**System Recovery:**

****

* Any transaction that was running at the time of failure needs to be undone and restarted .
* Any transactions that committed since the last checkpoint need to be redone .
* Transactions of type T1 need no recovery .
* Transactions of type T3 or T5 need to be undone and restarted Transactions of type T2 or T4 need to be redone

**Media Recovery**:

* Disk failure can corrupt the persistent database
* The database must be restored from backup
* The transaction logs can be used to roll forward from the backup point, to recover as much of the recent transaction history as possible
* Restore the database from the last backup.
* Use the transaction log to redo any changes made since the last backup
* Store the log on a separate physical device to the database
* The risk of losing both is then reduced

**Two-phase commit:**

* Required for distributed or heterogeneous environments, so that correctness is maintained in case of failure during a multi-part COMMIT
* Prepare phase has all local resource managers force logs to a persistent log, local managers reply ok or not
* Commit phase – if all replies are ok, the coordinator commits, and orders the local managers to complete the process; otherwise all are ordered to ROLLBACK

## SQL facilities

A data-manipulation language must include a construct for specifying the set of actions that constitute a transaction.  
The SQL standard specifies that a transaction begins implicitly. Transactions are ended by one of these SQL statements:

*•* **Commit work** commits the current transaction and begins a new one.

*•* **Rollback work** causes the current transaction to abort.

The keyword **work** is optional in both the statements. If a program terminates without either of these commands, the updates are either committed or rolled back which of the two happens is not specified by the standard and depends on the implementation. The standard also specifies that the system must ensure both serializability and freedom from cascading rollback. The definition of serializability used by the standard is that a schedule must have the *same effect* as would some serial schedule. Thus, conflict and view serializability are both acceptable.  
The SQL-92 standard also allows a transaction to specify that it may be executed in  
a manner that causes it to bec ome nonserializable with respect to other transactions.