**ST. XAVIER’S COLLEGE**

**MAITIGHAR, KATHMANDU**

****

**Database Management System**

**Theory Assignment #11**

**Submitted by:**

Anu Kadel

013BSCIT007

**Submitted to:**

|  |  |
| --- | --- |
| **Er. Sanjay Kumar Yadav** |  |

Lecturer

Department of Computer Science

St. Xavier’s College, Maitighar

**Theory Assignment #11**

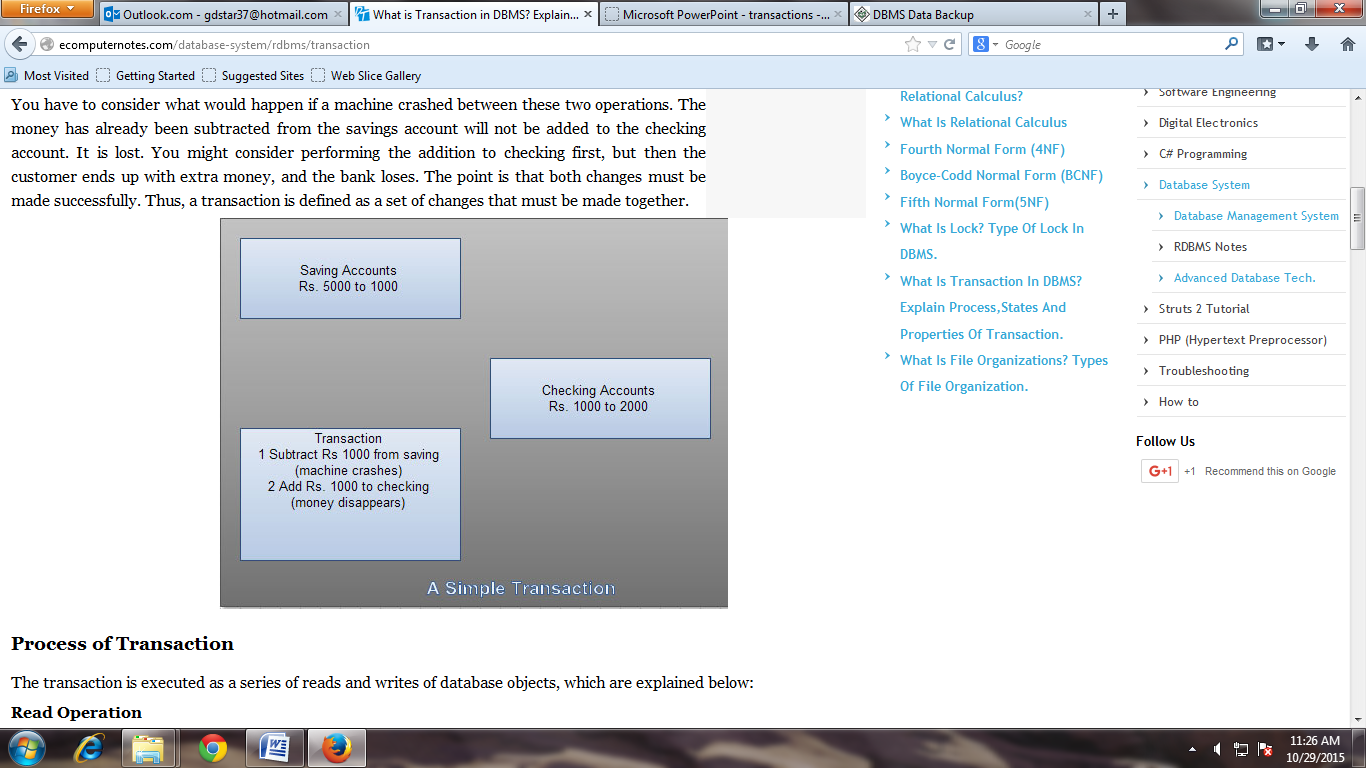
**TRANSACTION MANAGEMENT**

1. **INTRODUCTION**

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

Consider the example, you have to consider what would happen if a machine crashed between these two operations. The money has already been subtracted from the savings account will not be added to the checking account. It is lost. You might consider performing the addition to checking first, but then the customer ends up with extra money, and the bank loses. The point is that both changes must be made successfully. Thus, a transaction is defined as a set of changes that must be made together.



**ACID Properties**

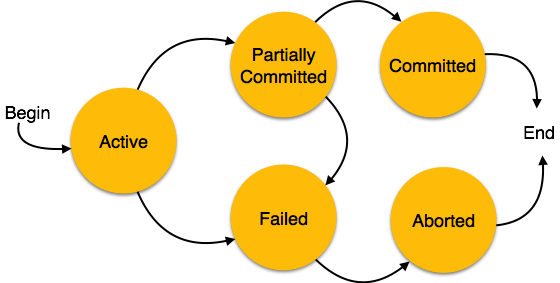
A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation, and **D**urability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

* **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.

1. **TRANSACTIONS**

**States of Transactions**

A transaction in a database can be in one of the following states –



* **Active** − In this state, the transaction is being executed. This is the initial state of every transaction.
* **Partially Committed** − When a transaction executes its final operation, it is said to be in a partially committed state.
* **Failed** − A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.
* **Aborted** − If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts −
  + Re-start the transaction
  + Kill the transaction
* **Committed** − If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

1. **TRANSACTION RECOVERY**

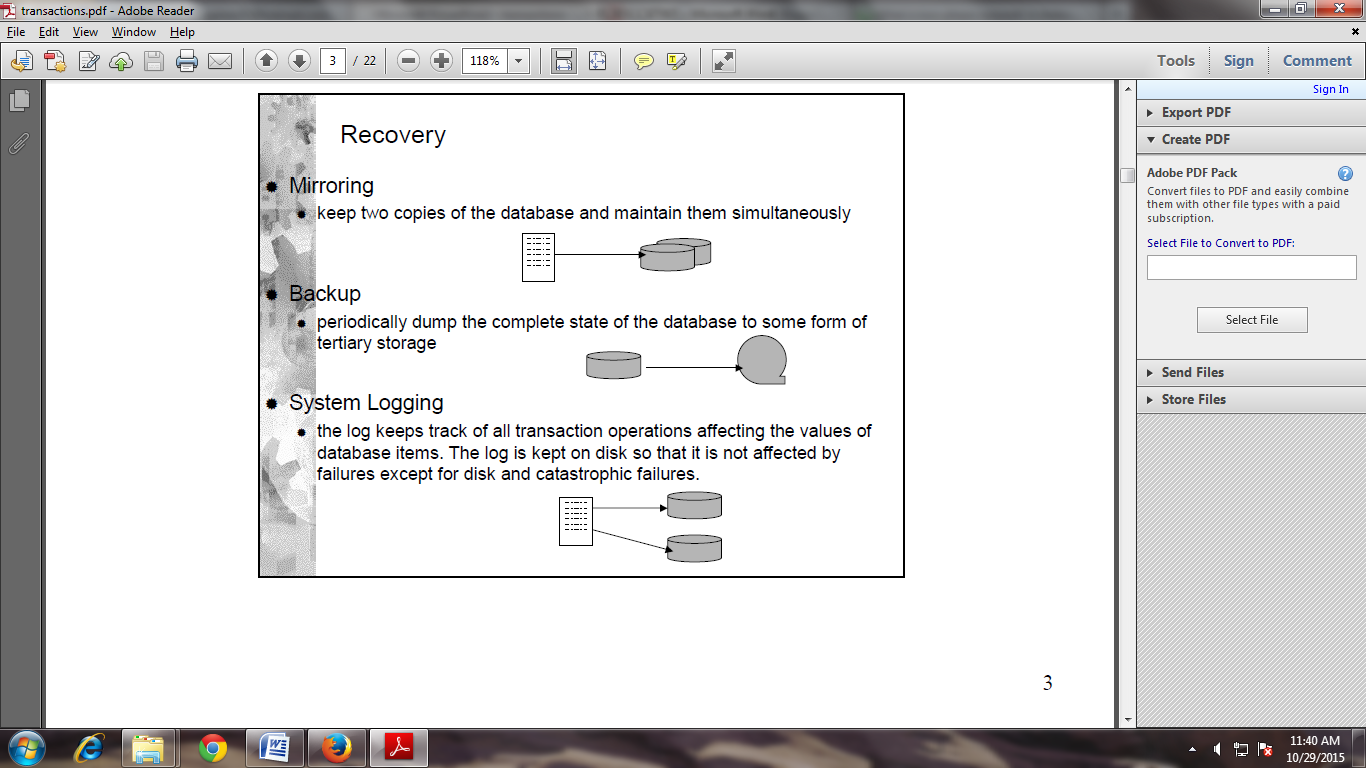
If an error or hardware/software crash occurs between the begin and end, the database will be inconsistent

* Computer Failure (system crash)
* A transaction or system error
* Local errors or exception conditions detected by the transaction
* Concurrency control enforcement
* Disk failure
* Physical problems and catastrophes
* The database is restored to some state from the past so that a correct state—close to the time of failure—can be reconstructed from the past state.
* A DBMS ensures that if a transaction executes some updates and then a failure occurs before the transaction reaches normal termination, then those updates are undone.
* The statements COMMIT and ROLLBACK (or their equivalent) ensure

**Recovery**

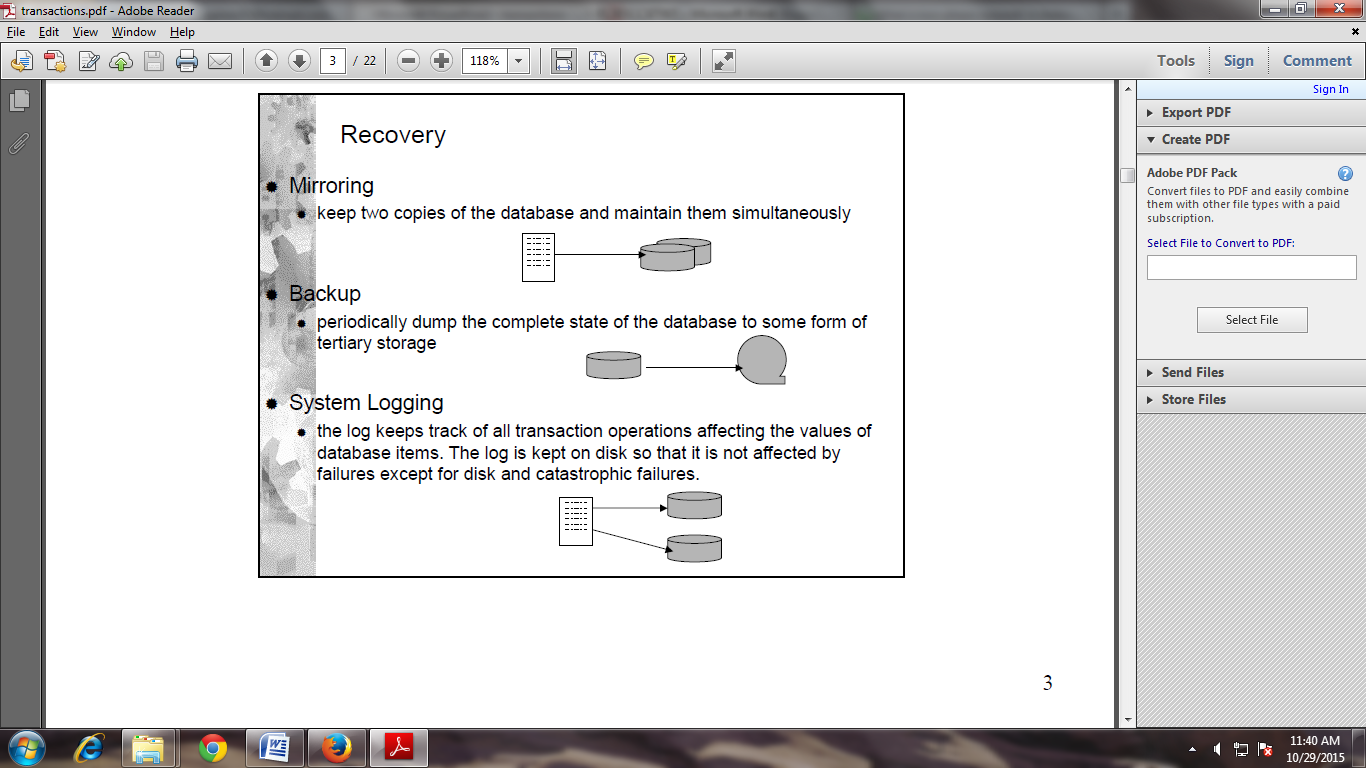
Mirroring

* keep two copies of the database and maintain them simultaneously



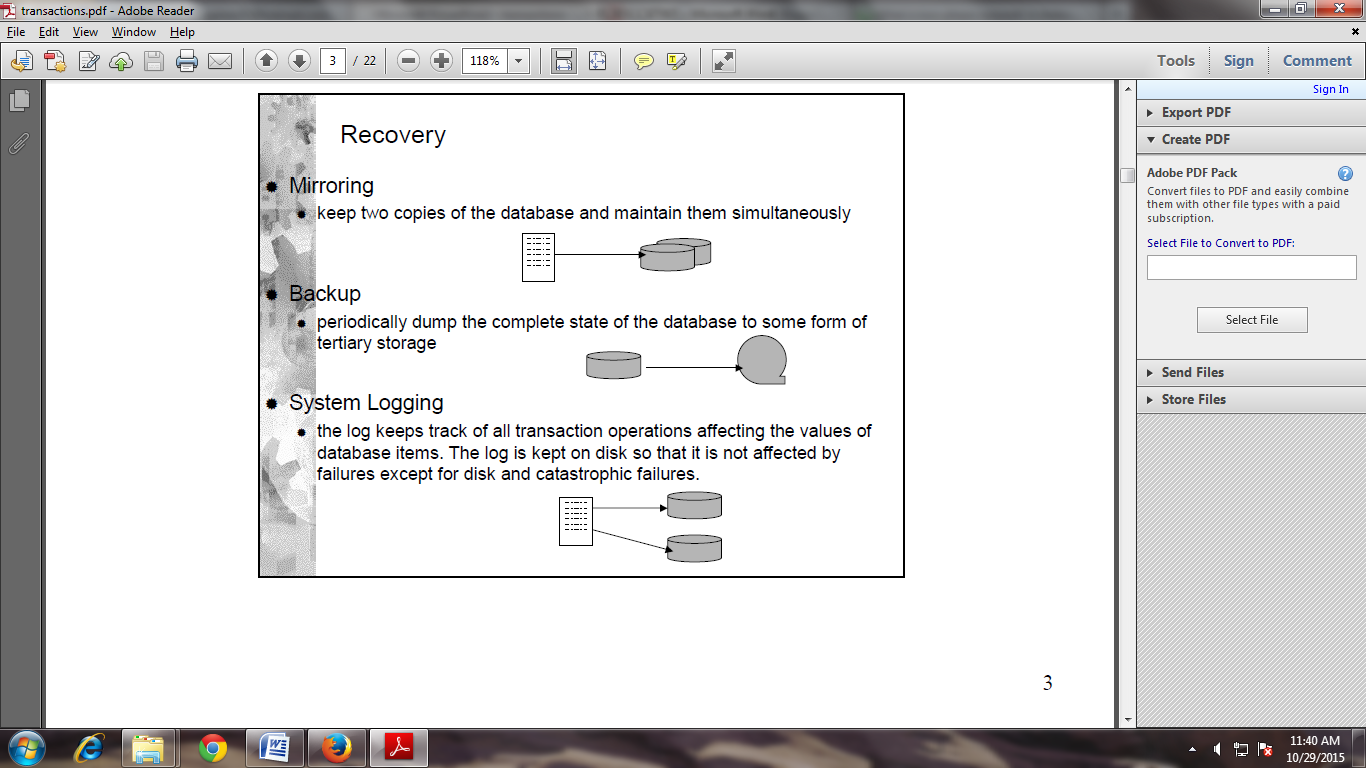
Backup

* Periodically dump the complete state of the database to some form of tertiary storage



**System Logging**

* The log keeps track of all transaction operations affecting the values of database items. The log is kept on disk so that it is not affected by failures except for disk and catastrophic failures.



Catastrophic failure

* Restore a previous copy of the database from archival backup
* Apply transaction log to copy to reconstruct more current state by redoing committed transaction operations up to failure point
* Incremental dump + log each transaction

Non-catastrophic failure

* Reverse the changes that caused the inconsistency by *undoing* the operations and possibly *redoing* legitimate changes whichwere lost
* The entries kept in the system log are consulted during recovery.
* No need to use the complete archival copy of the database.

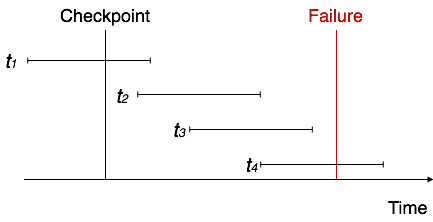
When more than one transaction are being executed in parallel, the logs are interleaved. At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering. To ease this situation, most modern DBMS use the concept of 'checkpoints'.

### Checkpoint

### Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all. Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

### Recovery

When a system with concurrent transactions crashes and recovers, it behaves in the following manner −



* The recovery system reads the logs backwards from the end to the last checkpoint.
* It maintains two lists, an undo-list and a redo-list.
* If the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>, it puts the transaction in the redo-list.
* If the recovery system sees a log with <Tn, Start> but no commit or abort log found, it puts the transaction in undo-list.

All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

* A database may become inconsistent because of a
  + transaction failure (abort)
  + database system failure (possibly caused by OS crash)
  + media crash (disk-resident data is corrupted)
* The recovery system ensures the database contains exactly those updates produced by committed transactions
  + I.e. atomicity and durability, despite failures

1. **SYSTEM RECOVERY**

* The system takes checkpoints automatically
* Upon system restart after a crash, transactions that finished successfully prior to the crash are redone, and those that were not complete prior to the crash are undone
* REDO and UNDO logs
* ARIES: Algorithms for Recovery and Isolation Exploiting Semantics – recovery by repeating history – REDO first, then UNDO

1. **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

1. **TWO-PHASE COMMIT**

The intention of all locking and transaction protocols is to produce an atomic update to distributed shared data, or to replicas of a shared data item. In the case of two phase commit algorithms for transactions on distributed databases the intention is to prevent an update being carried out on only one of several replicas, since this would make the replicas inconsistent with each other. Therefore, the operation is either "successful on all replicas" or "aborted".

The only way that a two phase commit can be implemented safely is for each of the replicas to have some knowledge of where all the participating entities are in the process of updating the shared information. If each of the entities keeps a diary of what it has been doing then they can crash and re-start without becoming confused, or allowing inconsistent data states to develop. Again, two phases are used,

* Start Protocol, write commencement information into log, send transaction to other participants asking them to "commit" that transaction and requesting a response.
* Collect and log responses from other participants, if everyone is ready write "commit" to the log, then send a "commit" message to all other participants. The participants write commit to their logs, commit the transaction locally, and send a "finished" message to the originator.

When collecting responses from the other participants the originator of a transaction cannot wait forever for all the other databases to respond, this will require a timeout to be implemented in the section of the protocol that initiates a commit request.

A feature of [transaction processing](http://www.webopedia.com/TERM/T/transaction_processing.html) systems that enables [databases](http://www.webopedia.com/TERM/D/database.html) to be returned to the pre-transaction state if some error condition occurs. A single transaction can update many different databases. The two-phase commit strategy is designed to ensure that either all the databases are updated or none of them, so that the databases remain synchronized.

1. **SQL FACILITIES**

SQL’s support for transactions, and hence for transaction based recovery, follows the general pattern described in foregoing section.

* First of all, most executable SQL statement are guaranteed to be atomic(CALL and RETURN are exception)
* Second, SQL provides direct analog of BEGIN TRANSACTION, COMMIT, and ROLLBACK called START TRANSACTION, COMMIT WORK and ROLBACK WORK respectively.
* SQL syntax:
* START TRANSACTION < option commalist > ;
* The option commalist specifies an access point, an isolation level, or both
* Similarly, syntax for COMMIT and ROLLBACK is;

COMMIT[WORK] [AND [NO] CHAIN];

ROLLBACK[WORK] [AND [NO] CHAIN];

* Access mode can be READ ONLY or READ WRITE
* Isolation level sets isolation from other transactions
* SAVEPOINT establishes a point within a transaction to which you can ROLLBACK

**REFERENCES**

[1] <http://www.eazynotes.com/pages/database-management-system/transaction-management.html>

[2] <http://ecomputernotes.com/database-system/rdbms/transaction>

[3] <http://www.georeference.org/doc/sql_server_spatial_dbms_facilities.htm>