**ST. XAVIER’S COLLEGE**

**MAITIGHAR, KATHMANDU**

****

**Database Management System**

**Theory Assignment #11**

**Submitted by:**

Biraj Poudyal

013BSCIT015

**Submitted to:**

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

Lecturer

Department of Computer Science

St. Xavier’s College, Maitighar

**Theory Assignment #11**

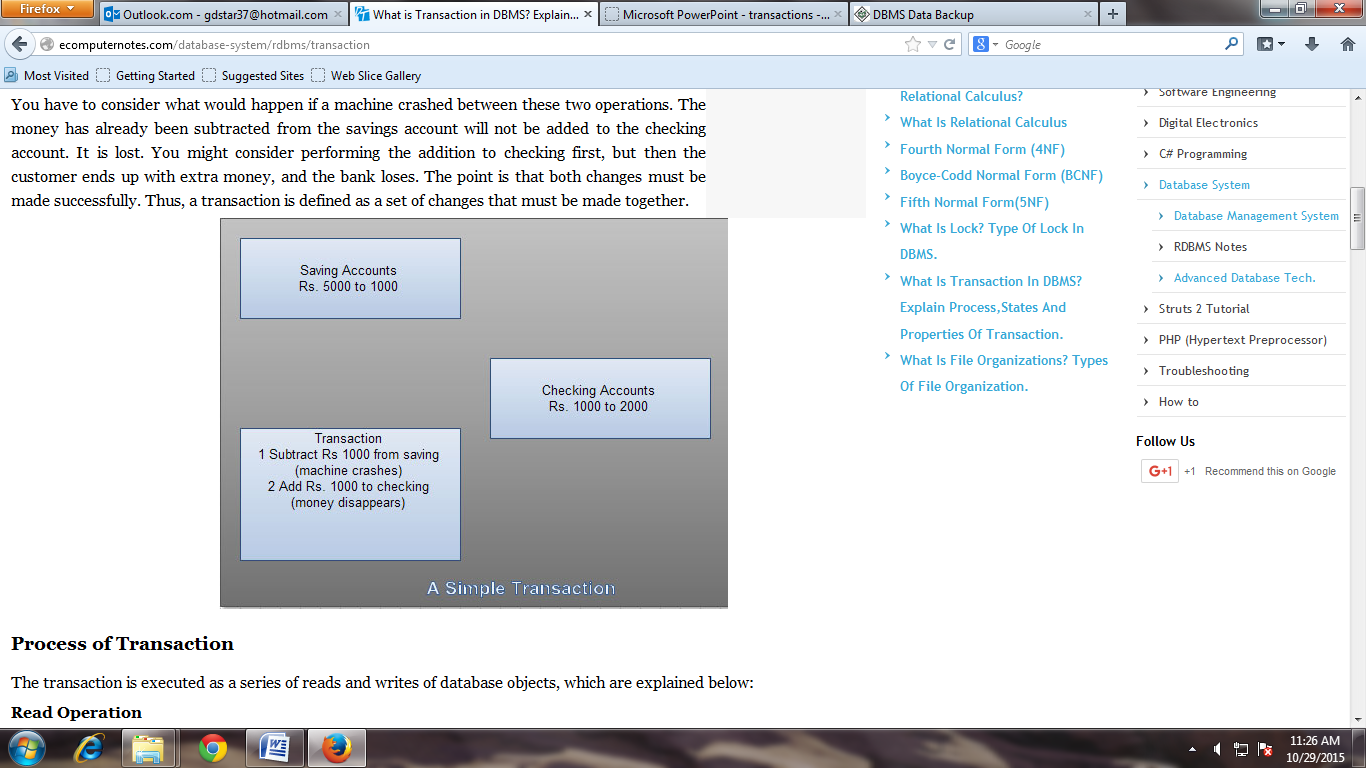
**TRANSACTION MANAGEMENT**

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**

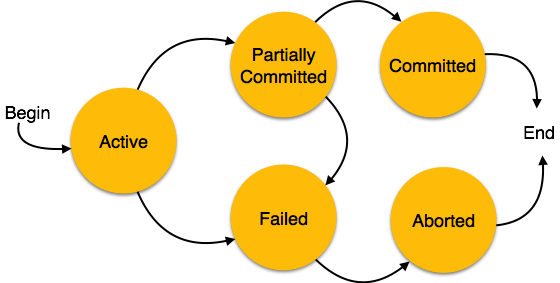
A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation, and **D**urability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

* **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**

Every Microsoft® SQL Server™ 2000 database has a transaction log that records data modifications made in the database. The log records the start and end of every transaction and associates each modification with a transaction. An instance of SQL Server stores enough information in the log to either redo (roll forward) or undo (roll back) the data modifications that make up a transaction. Each record in the log is identified by a unique log sequence number (LSN). All of the log records for a transaction are chained together.

An instance of SQL Server records many different types of information in the transaction log. Instances of SQL Server 2000 primarily log the logical operations performed. The operation is reapplied to roll forward a modification, and the opposite of the logical operation is performed to roll back a modification.

Each instance of SQL Server controls when modifications are written from its data buffers to disk. An instance of SQL Server may cache modifications in buffers for a period of time to optimize disk writes. A buffer page that contains modifications that have not yet written to disk is known as a [dirty page](https://technet.microsoft.com/en-us/library/aa224747(v=sql.80).aspx#sql:dirty_pages). Writing a dirty buffer page to disk is called flushing the page. When modifications are cached, care must be taken to ensure that no data modification is flushed before the corresponding log image is written to the log file. This could create a modification that could not be rolled back if necessary. To ensure that they can recover all modifications, instances of SQL Server use a write-ahead log, which means that all log images are written to disk before the corresponding data modification.

A commit operation forces all log records for a transaction to the log file so that the transaction is fully recoverable even if the server is shut down. A commit operation does not have to force all the modified data pages to disk as long as all the log records are flushed to disk. A system recovery can roll the transaction forward or backward using only the log records.

Periodically, each instance of SQL Server ensures that all dirty log and data pages are flushed. This is called a checkpoint. Checkpoints reduce the time and resources needed to recover when an instance of SQL Server is restarted. For more information on checkpoint processing, see [Checkpoints and the Active Portion of the Log](https://technet.microsoft.com/en-us/library/aa174542(v=sql.80).aspx).

##### **Rolling Back an Individual Transaction**

If any errors occur during a transaction, the instance of SQL Server uses the information in the log file to roll back the transaction. This rollback does not affect the work of any other users working in the database at the same time. Usually, the error is returned to the application, and if the error indicates a possible problem with the transaction, the application issues a ROLLBACK statement. Some errors, such as a 1205 deadlock error, roll back a transaction automatically. If anything stops the communication between the client and an instance of SQL Server while a transaction is active, the instance rolls back the transaction automatically when notified of the stoppage by the network or operating system. This could happen if the client application terminates, if the client computer is shut down or restarted, or if the client network connection is broken. In all of these error conditions, any outstanding transaction is rolled back to protect the integrity of the database.

##### **Recovery of All Outstanding Transactions at Start-up**

It is possible for an instance of SQL Server to sometimes stop processing (for example, if an operator restarts the server while users are connected and working in databases). This can create two problems:

* There may be an unknown number of SQL Server transactions partially completed at the time the instance stopped. These incomplete transactions need to be rolled back.
* There may be an unknown number of data modifications recorded in the SQL Server database log files, but the corresponding modified data pages were not flushed to the data files before the server stopped. Any committed modifications must be rolled forward.

When an instance of SQL Server is started, it must find out if either of these conditions exist and address them. The following steps are taken in each SQL Server database that is in the instance:

* The LSN of the last checkpoint is read from the database boot block along with the Minimum Recovery LSN.
* The transaction log is scanned from the Minimum Recovery LSN to the end of the log. All committed dirty pages are rolled forward by redoing the logical operation recorded in the log record.
* The instance of SQL Server then scans backward through the log file rolling back all uncompleted transactions by applying the opposite of the logical operation recorded in the log records.

The RESTORE statement also uses this type of recovery, unless a user specifies the NORECOVERY option. When restoring a sequence of database, differential, or log backups to recover a database to a point of failure, you specify NORECOVERY on all RESTORE statements except when restoring the last log backup. When the last backup in the sequence is restored, the RESTORE statement also has to ensure that all uncompleted transactions are rolled back. You specify the RECOVERY option on this RESTORE statement, in which case it uses the same logic as the startup recovery process to roll back all transactions that are still marked incomplete at the end of the last log.

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**

A commit operation is, by definition, an all-or-nothing affair. If a series of operations bound as a transaction cannot be completed, the rollback must restore the system (or cooperating systems) to the pre-transaction state.

In order to ensure that a transaction can be rolled back, a software system typically logs each operation, including the commit operation itself. A transaction/recovery manager uses the log records to undo (and possibly redo) a partially completed transaction.

When a transaction involves multiple distributed resources, for example, a database server on each of two different network hosts, the commit process is somewhat complex because the transaction includes operations that span two distinct software systems, each with its own resource manager, log records, and so on. (In this case, the distributed resources are the database servers.)

Two-phase commit is a transaction protocol designed for the complications that arise with distributed resource managers. With a two-phase commit protocol, the distributed transaction manager employs a coordinator to manage the individual resource managers.

The commit process proceeds as follows:

**Phase 1**

* Each participating resource manager coordinates local operations and forces all log records out:
* If successful, respond "OK"
* If unsuccessful, either allow a time-out or respond "OOPS"

**Phase** **2**

* If all participants respond "OK":
  + Coordinator instructs participating resource managers to "COMMIT"
  + Participants complete operation writing the log record for the commit
* Otherwise:
  + Coordinator instructs participating resource managers to "ROLLBACK"
  + Participants complete their respective local undos

1. **SQL FACILITIES**

**Table Facility**

First and foremost, SQL provides a table facility that enables a prompted, intuitive interface for the following functions:

* Defining databases
* Populating databases with rows
* Manipulating databases

**Table Editor**

SQL also provides a table editor that makes it easy for you to perform the following functions against rows in table data that is structured in row and column format:

* Access
* Insert
* Update

**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>