**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System**

**Lab Assignment #**

**Submitted by:**

Binod Marikhu  
013BSCCSIT013

**Submitted to:**

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

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

**Transactions**

The term transaction refers to a collection of operations that form a single logical unit  
of work. For instance, transfer of money from one account to another is a transaction  
consisting of two updates, one to each account. A **transaction** is a **unit** of program execution that accesses and possibly updates various data items. Following properties of the transactions ensure the integrity of the database:

* **Atomicity**. Either all operations of the transaction are reflected properly in the  
  database, or none are.  
  *•* **Consistency**. Execution of a transaction in isolation (that is, with no other  
  transaction executing concurrently) preserves the consistency of the database.  
  *•* **Isolation**. Even though multiple transactions may execute concurrently, the  
  system guarantees that, for every pair of transactions *Ti* and *Tj*, it appears  
  to *Ti* that either *Tj* finished execution before *Ti* started, or *Tj* started execution after *Ti* finished. Thus, each transaction is unaware of other transactions  
  executing concurrently in the system.  
  *•* **Durability**. After a transaction completes successfully, the changes it has made  
  to the database persist, even if there are system failures.

These properties are often called ACID properties. The acronym is derived from the first letter of each of the four properties.

**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.

**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

**Media Recovery**

There are three basic media recovery commands, which differ only in the way the set of files being recovered is determined. They all use the same criteria for determining if files can be recovered. Media recovery signals an error if it cannot get the lock for a file it is attempting to recover. This prevents two recovery sessions from recovering the same file. It also prevents media recovery of a file that is in use.

**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.

**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 become nonserializable with respect to other transactions.