ST. XAVIER’S COLLEGE

(Affiliated to Tribhuvan University)

Maitighar, Kathmandu



**Database Management System**

**Lab Assignment #8**

**Submitted by:**

Dikita Tuladhar  
013BSCCSIT018

**Submitted to:**

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

Date of Submission: 29th October, 2015

**Transaction Management**

1. **Introduction**

Transaction management is an important part of and RDBMS oriented enterprise applications to ensure data integrity and consistency.

1. **Transactions**

A database transaction is a sequence of actions that are treated as a single unit of work. These actions should either complete entirely or take no effect at all. Transaction management is an important part of and RDBMS oriented enterprise applications to ensure data integrity and consistency. The concept of transactions can be described with following four key properties described as **ACID**:

* **Atomicity:** A transaction should be treated as a single unit of operation which means either the entire sequence of operations is successful or unsuccessful.
* **Consistency:** This represents the consistency of the referential integrity of the database, unique primary keys in tables etc.
* **Isolation:** There may be many transactions processing with the same data set at the same time, each transaction should be isolated from others to prevent data corruption.
* **Durability:** Once a transaction has completed, the results of this transaction have to be made permanent and cannot be erased from the database due to system failure.

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.

1. **System Recovery**
2. **Media Recovery**
3. **Two-Phrase Commit**

In [transaction processing](https://en.wikipedia.org/wiki/Transaction_processing), [databases](https://en.wikipedia.org/wiki/Database), and [computer networking](https://en.wikipedia.org/wiki/Computer_networking), the two-phase commit protocol (2PC) is a type of [atomic commitment protocol](https://en.wikipedia.org/wiki/Atomic_commit) (ACP). It is a [distributed algorithm](https://en.wikipedia.org/wiki/Distributed_algorithm) that coordinates all the processes that participate in a [distributed atomic transaction](https://en.wikipedia.org/wiki/Distributed_transaction) on whether to [commit](https://en.wikipedia.org/wiki/Commit_(data_management)) or abort (roll back) the transaction (it is a specialized type of [consensus](https://en.wikipedia.org/wiki/Consensus_(computer_science)) protocol). The protocol achieves its goal even in many cases of temporary system failure (involving process, network node, communication, etc. failures), and is thus widely utilized. However, it is not resilient to all possible failure configurations, and in rare cases, user (e.g., a system's administrator) intervention is needed to remedy an outcome. To accommodate recovery from failure (automatic in most cases) the protocol's participants use [logging](https://en.wikipedia.org/wiki/Server_log) of the protocol's states. Log records, which are typically slow to generate but survive failures, are used by the protocol's [recovery procedures](https://en.wikipedia.org/wiki/Recovery_procedure). Many protocol variants exist that primarily differ in logging strategies and recovery mechanisms. Though usually intended to be used infrequently, recovery procedures compose a substantial portion of the protocol, due to many possible failure scenarios to be considered and supported by the protocol.

In a "normal execution" of any single [distributed transaction](https://en.wikipedia.org/wiki/Distributed_transaction), i.e., when no failure occurs, which is typically the most frequent situation; the protocol consists of two phases:

1. The commit-request phase (or voting phase), in which a coordinator process attempts to prepare all the transaction's participating processes (named participants, cohorts, or workers) to take the necessary steps for either committing or aborting the transaction and to vote, either "Yes": commit (if the transaction participant's local portion execution has ended properly), or "No": abort (if a problem has been detected with the local portion), and
2. The commit phase, in which, based on voting of the cohorts, the coordinator decides whether to commit (only if all have voted "Yes") or abort the transaction (otherwise), and notifies the result to all the cohorts. The cohorts then follow with the needed actions (commit or abort) with their local transactional resources (also called recoverable resources; e.g., database data) and their respective portions in the transaction's other output (if applicable).
3. **SQL Facilities**

In addition to the advanced facilities noted above, SQL is rich in the type of ease of use capabilities that are necessary to support relational databases from the simple to the complex.

**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
* Delete

**Query Facility:**

With the Query facility, SQL permits you to interactively define queries and have results displayed in a variety of report formats including the following:

* Tabular
* Matrix
* Free format

For those readers who have a System i5 background, you will notice that SQL brings with it its own naming scheme that is significantly different from corresponding native objects. See table 4-1 for specifics