ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**Database Management System**

**Assignment # 11**

**SUBMITTED BY:**

Anlisha Maharjan

013BSCCSIT006

**SUBMITTED TO:**

|  |  |
| --- | --- |
| **Er. Sanjay Kr. Yadav**  **Lecturer** |  |

Submission Date: 29thOctober 2015

**TRANSACTION MANAGEMENT**

**INTRODUCTION**

**TRANSACTION**

A transaction symbolizes a unit of work performed within a database management system (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions. A transaction generally represents any change in database. Transactions in a database environment have two main purposes:

* To provide reliable units of work that allow correct recovery from failures and keep a database consistent even in cases of system failure, when execution stops (completely or partially) and many operations upon a database remain uncompleted, with unclear status.
* To provide isolation between programs accessing a database concurrently. If this isolation is not provided, the programs' outcomes are possibly erroneous.

**TRANSACTION RECOVERY**

**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**

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**

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.

**RECOVER DATABASE**

RECOVER DATABASE performs media recovery on all online data files that require redo to be applied. If all instances were cleanly shutdown, and no backups were restored, RECOVER DATABASE indicates a no recovery required error. It also fails if any instances have the database open (since they have the data file locks). To perform media recovery on an entire database (all table spaces), the database must be mounted EXCLUSIVE and closed.

**RECOVER TABLESPACE**

RECOVER TABLESPACE performs media recovery on all data files in the table spaces listed. To translate the table space names into data file names, the database must be mounted and open. The table spaces must be offline to perform the recovery. An error is indicated if none of the files require recovery.

**RECOVER DATAFILE**

RECOVER DATAFILE lists the data files to be recovered. The database can be open or closed, provided the media recovery locks can be acquired. If the database is open in any instance, then data file recovery can only recover off-line files.

**TWO-PHASE COMMIT**

In transaction processing, databases, and computer networking, the two-phase commit protocol (2PC) is a type of atomic commitment protocol (ACP). It is a distributed algorithm that coordinates all the processes that participate in a distributed atomic transaction on whether to commit or abort (roll back) the transaction (it is a specialized type of consensus protocol). The protocol achieves its goal even in many cases of temporary system failure (involving either process, network node, communication, etc. failures), and is thus widely utilized.[1][2][3] 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 of the protocol's states. Log records, which are typically slow to generate but survive failures, are used by the protocol's recovery procedures. 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).

**SQL FACILITIES**

**Declarative Referential Integrity**

This integrity constraint provides the ability for developers to define integrity relationships to be enforced at the database level, rather than the program level. The DB2 UDB for System i5 implemented via SQL provides support for the following actions when the defined integrity rules are attempted to be broken:

* NO ACTION
* RESTRICT
* CASCADE
* SET NULL
* SET DEFAULT

**Triggers**

Triggers are also implemented via the SQL language and the DB2 UDB database. When you have defined that certain actions need to occur when and if certain database values change, you implement that function with “Triggers.” Without triggers, you code these actions into all the programs that touch a database. And, as you know, in most shops, there are many programs that cause updates to the same database. That’s where DB Triggers help out. When a certain action occurs on the database as a result of an insert, or update, or a delete, a trigger fires and a program that you write gets control. In this program you can code whatever has to be done to protect your business at that time.

**Stored Procedures**

Stored procedures consist of compiled code residing on an intelligent database server such as the DB2 UDB for System i5. The major purpose of stored procedures is to reduce the processing burden on the client side of client server as well as to reduce the communication interactions time. These precompiled SQL routines (and other languages such as RPG and COBOL on System i5) are stored on the System i5.

**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