ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**Database Management System**

**TheoryAssignment #11**

**SUBMITTED BY:**

**Apil Neupane**

**013BSCCSIT008**

**SUBMITTED TO:**

|  |  |
| --- | --- |
| **Er. Sanjay Kr. Yadav**  **Lecturer** |  |
| **Department of Computer Science** | |

Submission Date: 29th October 2015

**TRANSATION MANAGEMENT**

1. **INTRODUCTION:**

A transaction is the basic logical unit of execution in an information system. A transaction is a sequence of operations that must be executed as a whole, taking a consistent (& correct) database state into another consistent (& correct) database state;

* A collection of actions that make consistent transformations of system states while preserving system consistency
* Is an indivisible unit of processing

1. **TRANSACTIONS:**

* A logical unit of work on a database
  + An entire program
  + A portion of a program
  + A single command
* The entire series of steps necessary to accomplish a logical unit of work
* Successful transactions change the database from one CONSISTENT STATE to another

(One where all data integrity constraints are satisfied)

**Example of Transition Management in SQL:**

When a transaction sequence is initiated it must continue through all succeeding SQL statements until:

* + A Commit Statement is Reached
  + A Rollback Statement is Reached
  + The End of the Program is Reached (Commit)
  + The Program is Abnormally Terminated (Rollback)

1. **TRANSITION RECOVERY:**

**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 which were lost
* The entries kept in the system log are consulted during recovery.
* No need to use the complete archival copy of the database.

**Transaction as a Recovery Unit**

* 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 Transaction Atomicity

1. **SYSTEM 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.

1. **MEDIA RECOVERY:**

The failures in this category are caused because the media that hold the data, such as the hard disk fails. This affects the database, the media, and the transition that were being processed at the time of media failure. Media recovery, thus, is an attempt to recover from those problems.

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:**

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 > ;

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

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

* The option commalist specifies an access point, an isolation level, or both
* 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