ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System Assignment #11**

**Submitted by:**

Bikash Paneru  
013BSCCSIT012

**Submitted to:**

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

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

**Transaction Management**

**Introduction**

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:

1. 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.
2. To provide isolation between programs accessing a database concurrently. If this isolation is not provided, the programs' outcomes are possibly erroneous.

**Transactions**

A transaction is a logical unit of work that contains one or more SQL statements. A transaction is an atomic unit. The effects of all the SQL statements in a transaction can be either all committed (applied to the database) or all rolled back (undone from the database).

A transaction begins with the first executable SQL statement. A transaction ends when it is committed or rolled back, either explicitly with a COMMIT or ROLLBACK statement or implicitly when a DDL statement is issued.

To illustrate the concept of a transaction, consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction can consist of three separate operations:

1. Decrement the savings account
2. Increment the checking account
3. Record the transaction in the transaction journal

**Transaction Recovery**

When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items. Transactions are made of various operations, which are atomic in nature. But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.

When a DBMS recovers from a crash, it should maintain the following −

* It should check the states of all the transactions, which were being executed.
* A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
* It should check whether the transaction can be completed now or it needs to be rolled back.
* No transactions would be allowed to leave the DBMS in an inconsistent state.

There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction −

* Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
* Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

**System Recovery**

* **Possible Levels of Recovery**:
  1. Recovery to the correct state.
  2. Recovery to a checkpointed (past) correct state.
  3. Recovery to a possible previous state.
  4. Recovery to a valid state.
  5. Recovery to a consistent state.
  6. Crash resistance (prevention).

The bigger the damage, the cruder the recovery technique used.

* **Recovery Techniques**:
  1. **Salvation program:** Run after a crash to attempt to restore the system to a valid state. No recovery data used. Used when all other techniques fail or were not used. Good for cases where buffers were lost in a crash and one wants to reconstruct what was lost...(4,5)
  2. **Incremental dumping:** Modified files copied to archive after job completed or at intervals. (3,4)
  3. **Audit trail:** Sequences of actions on files are recorded. Optimal for "backing out" of transactions. (Ideal if trail is written out before changes). (1,2,3)
  4. **Differential files:** Separate file is maintained to keep track of changes, periodically merged with the main file. (2,3)
  5. **Backup/current version:** Present files form the current version of the database. Files containing previous values form a consistent backup version. (2,3)
  6. **Multiple copies:** Multiple active copies of each file are maintained during normal operation of the database. In cases of failure, comparison between the versions can be used to find a consistent version. (6)
  7. **Careful replacement:** Nothing is updated in place, with the original only being deleted after operation is complete. (2,6)

**Media Recovery**

A DBMS uses these basic media recovery SQL statements, which differ only in the way the system determines the set of files to be recovered:

* RECOVER DATABASE
* RECOVER TABLESPACE
* RECOVER DATAFILE

Each statement uses the same criteria to determine whether files are recoverable. Oracle prevents two recovery sessions from recovering the same file and prevents media recovery of a file that is in use.

You can also use the SQL statement ALTER DATABASE RECOVER, although Oracle strongly recommends you use the SQL\*Plus RECOVER statement instead so that Oracle will prompt you for the names of the archived redo logs.

**Two-phase Commit**

The database ensures the integrity of data in a distributed transaction using the **two-phase commit mechanism**. In the **prepare phase**, the initiating node in the transaction asks the other participating nodes to promise to commit or roll back the transaction. During the **commit phase**, the initiating node asks all participating nodes to commit the transaction. If this outcome is not possible, then all nodes are asked to roll back.

All participating nodes in a distributed transaction should perform the same action: they should either all commit or all perform a rollback of the transaction. The database automatically controls and monitors the commit or rollback of a distributed transaction and maintains the integrity of the **global database** (the collection of databases participating in the transaction) using the two-phase commit mechanism. This mechanism is completely transparent, requiring no programming on the part of the user or application developer.

The commit mechanism has the following distinct phases, which the database performs automatically whenever a user commits a distributed transaction:

| **Phase** | **Description** | |
| --- | --- | --- |
| Prepare phase | | The initiating node, called the **global coordinator**, asks participating nodes other than the commit point site to promise to commit or roll back the transaction, even if there is a failure. If any node cannot prepare, the transaction is rolled back. |
| Commit phase | | If all participants respond to the coordinator that they are prepared, then the coordinator asks the commit point site to commit. After it commits, the coordinator asks all other nodes to commit the transaction. |
| Forget phase | | The global coordinator forgets about the transaction. |

**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

**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

**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.

**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. When implemented, they provide major advantages for client server and intelligent Web processing. The major benefit is that the application performs better since the server code is precompiled and because there is minimal back and forth action over the network between the client and the server. Additionally, because the code is on the server, one set of code can be reused for as many clients as necessary.