**ST. XAVIER’S COLLEGE**

**MAITIGHAR, KATHMANDU**

****

**Database Management System**

**Theory Assignment #11**

**Submitted by:**

Hemanchal Joshi

013BSCIT021

**Submitted to:**

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

Submission Date: 29’th September, 2015

1. **INTRODUCTION**

A transaction is an event which occurs on the database. Generally a transaction reads a value from the database or writes a value to the database. If you have any concept of Operating Systems, then we can say that a transaction is analogous to processes.

Although a transaction can both read and write on the database, there are some fundamental differences between these two classes of operations. A read operation does not change the image of the database in any way. But a write operation, whether performed with the intention of inserting, updating or deleting data from the database, changes the image of the database. That is, we may say that these transactions bring the database from an image which existed before the transaction occurred (called the Before Image or BFIM) to an image which exists after the transaction occurred (called the After Image or AFIM)

Let’s take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

A’s Account

Open\_Account(A)

Old\_Balance = A.balance

New\_Balance = Old\_Balance - 500

A.balance = New\_Balance

Close\_Account(A)

B’s Account

Open\_Account(B)

Old\_Balance = B.balance

New\_Balance = Old\_Balance + 500

B.balance = New\_Balance

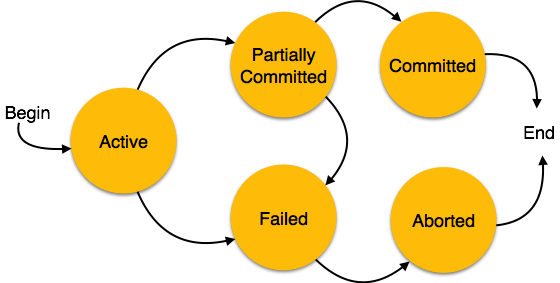
Close\_Account(B)

1. **TRANSACTIONS**

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

**States of Transactions**

A transaction in a database can be in one of the following states −

 There are the following six states in which a transaction may exist:

**Active**: The initial state when the transaction has just started execution.

**Partially Committed:** At any given point of time if the transaction is executing properly, then it is going towards it COMMIT POINT. The values generated during the execution are all stored in volatile storage.

**Failed:** If the transaction fails for some reason. The temporary values are no longer required, and the transaction is set to ROLLBACK. It means that any change made to the database by this transaction up to the point of the failure must be undone. If the failed transaction has withdrawn Rs. 100/- from account A, then the ROLLBACK operation should add Rs 100/- to account A.

**Aborted:** When the ROLLBACK operation is over, the database reaches the BFIM. The transaction is now said to have been aborted.

**Committed**: If no failure occurs then the transaction reaches the COMMIT POINT. All the temporary values are written to the stable storage and the transaction is said to have been committed.

**Terminated:** Either committed or aborted, the transaction finally reaches this state.

1. **TRANSACTION RECOVERY**

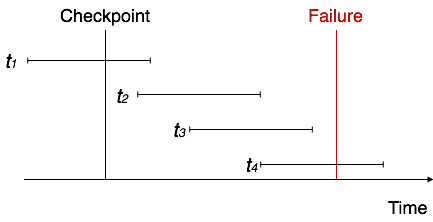
When more than one transaction are being executed in parallel, the logs are interleaved. At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering. To ease this situation, most modern DBMS use the concept of 'checkpoints'.

**Checkpoint**

Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all. Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

1. **SYSTEM RECOVERY**

When a system with concurrent transactions crashes and recovers, it behaves in the following manner −



* The recovery system reads the logs backwards from the end to the last checkpoint.
* It maintains two lists, an undo-list and a redo-list.
* If the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>, it puts the transaction in the redo-list.
* If the recovery system sees a log with <Tn, Start> but no commit or abort log found, it puts the transaction in undo-list.

1. **MEDIA RECOVERY**

Typically, the term "media recovery" refers to recovery of datafiles. Block media recovery is a more specialized operation that you can only perform with RMAN.

**Datafile Media Recovery**

Datafile media recovery is used to recover from a lost or damaged current datafile or control file. It is also used to recover changes that were lost when a tablespace went offline without the OFFLINE NORMAL option. Datafile media recovery and instance recovery have in common the requirement to repair database integrity. However, these types of recovery differ with respect to their additional features. Media recovery has the following characteristics:

* Applies needed changes using restored backups of damaged datafiles.
* Can use archived logs as well as the online logs.
* Requires explicit invocation by a user.
* Does not detect media failure (that is, the need to restore a backup) automatically. After a backup has been restored, however, detection of the need to recover it through media recovery is automatic.
* Has a recovery time governed solely by user policy (for example, frequency of backups, parallel recovery parameters) rather than by Oracle internal mechanisms.

The database cannot be opened if any of the online datafiles needs media recovery, nor can a datafile that needs media recovery be brought online until media recovery has been executed. The following scenarios necessitate media recovery:

* You restore a backup of a datafile.
* You restore a backup control file (even if all datafiles are current).
* A datafile is taken offline (either by you or automatically by Oracle) without the OFFLINE NORMAL option.
* Unless the database is not open by any instance, datafile media recovery can only operate on offline datafiles. You can initiate datafile media recovery before opening a database even when crash recovery would have sufficed. If so, crash recovery still runs automatically at database open.

Note that when a file requires media recovery, you must perform media recovery even if all necessary changes are contained in the online logs. In other words, you must still run recovery even though the archived logs are not needed. Media recovery may find nothing to do -- and signal the "no recovery required" error -- if invoked for files that do not need recovery.

**Block Media Recovery**

Block media recovery is a technique for restoring and recovering individual data blocks while all database files remain online and available. If corruption is limited to only a few blocks among a subset of database files, then block media recovery may be preferable to datafile recovery.

The interface to block media recovery is provided by RMAN. If you do not already use RMAN as your principal backup and recovery solution, then you can still perform block media recovery by cataloging into the RMAN repository the necessary user-managed datafile and archived redo log backups.

1. **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.

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 > ;

* The option commalist specifies an access point, an isolation level, or both
* Similarly, syntax for COMMIT and ROLLBACK is;

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

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

* 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