**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**Database Management System**

**Theory Assignment #9**

**Submitted by:**

Linus Dhakal  
013BSCCSIT022

**Submitted to:**

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

Date of Submission: Friday, October 02, 2015

# Database Recovery

There are many situations in which a transaction may not reach a commit or abort point.

* 1. An operating system crash can terminate the DBMS processes
  2. The DBMS can crash
  3. The system might lose power
  4. A disk may fail or other hardware may fail.
  5. Human error can result in deletion of critical data.

In any of these situations, data in the database may become inconsistent or lost.

For example, if a transaction has completed 30 out of 40 scheduled writes to the database when the DBMS crashes, then the database may be in an inconsistent state as only part of the transaction’s work was completed.

**Database Recovery** is the process of restoring the database and the data to a consistent state. This may include restoring lost data up to the point of the event (e.g. system crash).

# Purpose of Data Recovery

* To bring the database into the last consistent state, which existed prior to the failure [5].
* To preserve transaction properties:
  + Atomicity
  + Consistency
  + Isolation
  + Durability

# Types of Failure

To see where the problem has occurred, we generalize a failure into various categories, as follows:

**Transaction failure:**

A transaction has to abort when it fails to execute or when it reaches a point from where it can’t go any further. This is called transaction failure where only a few transactions or processes are hurt.

The reasons for transaction failure could be:

* **Logical errors** − Where a transaction cannot complete because it has some code error or any internal error condition.
* **System errors** − Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.

**System Crash:**

There are problems − external to the system − that may cause the system to stop abruptly and cause the system to crash.

For example, interruptions in power supply may cause the failure of underlying hardware or software failure.

**Disk Failure:**

In early days of technology evolution, it was a common problem where hard-disk drives or storage drives used to fail frequently.

Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or a part of disk storage.

# The Storage Hierarchy

Databases are stored in file formats, which contain records. At physical level, the actual data is stored in electromagnetic format on some device. These storage devices can be broadly categorized into three types −



* **Primary Storage** − The memory storage that is directly accessible to the CPU comes under this category. CPU's internal memory (registers), fast memory (cache), and main memory (RAM) are directly accessible to the CPU, as they are all placed on the motherboard or CPU chipset. This storage is typically very small, ultra-fast, and volatile. Primary storage requires continuous power supply in order to maintain its state. In case of a power failure, all its data is lost.
* **Secondary Storage** − Secondary storage devices are used to store data for future use or as backup. Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks, optical disks (DVD, CD, etc.), hard disks, flash drives, and magnetic tapes.
* **Tertiary Storage** − Tertiary storage is used to store huge volumes of data. Since such storage devices are external to the computer system, they are the slowest in speed. These storage devices are mostly used to take the back up of an entire system. Optical disks and magnetic tapes are widely used as tertiary storage.

# Buffer Management

A DBMS must manage a huge amount of data, and in the course of processing the required space for the blocks of data will often be greater than the memory space available. For this there is the need to manage a memory in which to load and unload the blocks. The buffer manager is responsible primarily for managing the operations inherent saving and loading of the blocks.

In fact, the operations that provide the buffer manager are these:

•**FIX:** This command tells the operator of the buffer to load a block from disk and return the pointer to the memory where it is loaded. If the block was already in memory, the buffer manager needs only to return the pointer, otherwise he must load from disk and bring it into memory. If the buffer memory is full but it is possible to have 2 situations: or the possibility of releasing a portion of memory that is occupied by transactions already completed. In this case, before freeing the area the content is written to disk if any block of this area had been changed.

There is the possibility of free memory to be occupied because transitions still ongoing. In this case, the buffer manager can work in 2 ways:

* In the first mode (STEAL), the operator of the free buffer memory occupied by a transition already active, possibly saving your changes to disk.
* In the second mode (NOT STEAL), the transition requested block is made to wait until the free memory.

• **SET DIRTY:** invoking this command, you mark a block of memory as amended. Before introducing the last 2 commands you need to anticipate that the DMBS can operate in 2 modes: Force and NOT FORCE. When working in FORCE mode, the rescue disk is in synchronous mode with the commit of a transaction. When working mode is NOT FORCE the rescue is carried out from time to time in asynchronous manner. Typically, commercial database operating mode NOT FORCE because this allows an increase in performance: the block may undergo multiple changes in memory before being saved, then you can choose to make the saves when the system is unloading.

•**Force:** This command will cause the operator of the buffer to make the writing in synchronously with the completion (commit) the transaction

•**FLUSH:** This command will cause the operator of the buffer to perform the rescue,when in how NOT FORCE.

# Transaction Log

The transaction log is a serial record of all the transactions that have been performed against the database since the transaction log was last backed up. With transaction log backups, you can recover the database to a specific point in time (for example, prior to entering unwanted data), or to the point of failure.

If a log backup is missing or damaged, you must create a database or differential database backup and start backing up the transaction logs again. Retain the previous transaction logs backups if you want to restore the database to a point in time within those backups. The only time database or differential database backups must be synchronized with transaction log backups is when starting a sequence of transaction log backups. Every sequence of transaction log backups must be started by a database or differential database backup.

# Data Caching

Caching is just the practice of storing data in and retrieving data from a high-performance store (usually memory) either explicitly or implicitly.

* + Data items to be modified are first stored into database cache by the Cache Manager (CM) and after modification they are flushed (written) to the disk.
  + The flushing is controlled by **Modified** and **Pin-Unpin** bits.
    - **Pin-Unpin**: Instructs the operating system not to flush the data item.
    - **Modified**: Indicates the AFIM of the data item.

# Transaction Rollback and Roll Forward

A rollback is the undoing of partly completed database changes when a database transaction is determined to have failed. The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database.

The ROLLBACK command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for ROLLBACK command is as follows:

ROLLBACK;

## Example:

Consider the CUSTOMERS table having the following records:

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

Following is the example, which would delete records from the table having age = 25 and then ROLLBACK the changes in the database.

SQL> DELETE FROM CUSTOMERS

WHERE AGE = 25;

SQL> ROLLBACK;

As a result, delete operation would not impact the table and SELECT statement would produce the following result:

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

Recovering a database by applying different transactions that recorded in the database log files. It is nothing but re-doing the changes made by a transaction i.e. after the committed transaction and to over write the changed value again to ensure consistency.

# Check Pointing

Time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery. The following steps defines a checkpoint operation:

* + - Suspend execution of transactions temporarily.
    - Force write modified buffer data to disk.
    - Write a [checkpoint] record to the log, save the log to disk.
    - Resume normal transaction execution.

## Shadow Paging

Shadow paging is an alternative to log-based recovery techniques, which has both advantages and disadvantages. It may require fewer disk accesses, but it is hard to extend paging to allow multiple concurrent transactions. The paging is very similar to paging schemes used by the operating system for memory management.  
The idea is to maintain two page tables during the life of a transaction: the current page table and the shadow page table. When the transaction starts, both tables are identical. The shadow page is never changed during the life of the transaction. The current page is updated with each **write** operation. Each table entry points to a page on the disk. When the transaction is committed, the shadow page entry becomes a copy of the current page table entry and the disk block with the old data is released. If the shadow is stored in nonvolatile memory and a system crash occurs, then the shadow page table is copied to the current page table.

There are drawbacks to the shadow-page technique:

1. **Commit overhead.** The commit of a single transaction using shadow paging requires multiple blocks to be output -- the current page table, the actual data and the disk address of the current page table. Log-based schemes need to output only the log records.
2. **Data fragmentation.** Shadow paging causes database pages to change locations (therefore, no longer contiguous.
3. **Garbage collection.** Each time that a transaction commits, the database pages containing the old version of data changed by the transactions must become inaccessible. Such pages are considered to be garbage since they are not part of the free space and do not contain any usable information. Periodically it is necessary to find all of the garbage pages and add them to the list of free pages. This process is called garbage collection and imposes additional overhead and complexity on the system.

# Recovery Schemes

Write-Ahead Logging is a family of techniques for providing atomicity and durability (two of the ACID properties) in database systems.

1. In a system using WAL, all modifications are written to a log before they are applied. Usually both redo and undo information is stored in the log.
2. The purpose of this can be illustrated by an example. Imagine a program that is in the middle of performing some operation when the machine it is running on loses power. Upon restart, that program might well need to know whether the operation it was performing succeeded, half-succeeded, or failed. If a write-ahead log is used, the program can check this log and compare what it was supposed to be doing when it unexpectedly lost power to what was actually done. On the basis of this comparison, the program could decide to undo what it had started, complete what it had started, or keep things as they are.
3. WAL allows updates of a database to be done in-place. Another way to implement atomic updates is with shadow paging, which is not in-place. The main advantage of doing updates in-place is that it reduces the need to modify indexes and block lists.
4. ARIES is a popular algorithm in the WAL family.
5. File systems typically use a variant of WAL for at least file system metadata called journaling.
6. WAL protects all permanent tables and all system tables but is not used to protect the Transient Journal (TJ), since TJ records are stored in the WAL log. WAL also is not used to protect spool or volatile tables.
7. The WAL log is maintained as a separate logical file system from the normal table area. Whole cylinders are allocated to the WAL log, and it has its own index structure.
8. The WAL log data is a sequence of WAL log records and includes the following:

**•**Redo records, used for updating disk blocks and insuring file system consistency during restarts.

**•**TJ records used for transaction rollback.

**Failure with Loss of Non-volatile storage [General Concepts]**

A volatile storage like RAM stores all the active logs, disk buffers, and related data. In addition, it stores all the transactions that are being currently executed. What happens if such a volatile storage crashes abruptly? It would obviously take away all the logs and active copies of the database. It makes recovery almost impossible, as everything that is required to recover the data is lost.

Following techniques may be adopted in case of loss of volatile storage −

* We can have **checkpoints** at multiple stages so as to save the contents of the database periodically.
* A state of active database in the volatile memory can be periodically**dumped** onto a stable storage, which may also contain logs and active transactions and buffer blocks.
* <dump> can be marked on a log file, whenever the database contents are dumped from a non-volatile memory to a stable one.

**Recovery in Multi database System**

To maintain the atomicity of a multidatabase transaction, it is necessary to have a two-level recovery mechanism. A global recovery manager, or coordinator, is needed to maintain information needed for recovery, in addition to the local recovery managers and the information they maintain (log, tables).

The coordinator usually follows a protocol called the two-phase commit protocol, whose two phases can be stated as follows:

• Phase 1: When all participating databases signal the coordinator that the part of the multidatabase transaction involving each has concluded, the coordinator sends a message "prepare for commit" to each participant to get ready for committing the transaction. Each participating database receiving that message will force-write all log records and needed information for local recovery to disk and then send a "ready to commit" or "OK" signal to the coordinator. If the force-writing to disk fails or the local transaction cannot commit for some reason, the participating database sends a "cannot commit" or "not OK" signal to the coordinator. If the coordinator does not receive a reply from a database within a certain time out interval, it assumes a "not OK" response.

• Phase 2: If *all*participating databases reply "OK," and the coordinator’s vote is also "OK," the transaction is successful, and the coordinator sends a "commit" signal for the transaction to the participating databases. Because all the local effects of the transaction and information needed for local recovery have been recorded in the logs of the participating databases, recovery from failure is now possible. Each participating database completes transaction commit by writing a [commit] entry for the transaction in the log and permanently updating the database if needed. On the other hand, if one or more of the participating databases or the coordinator have a "not OK" response, the transaction has failed, and the coordinator sends a message to "roll back" or UNDO the local effect of the transaction to each participating database. This is done by undoing the transaction operations, using the log.