ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System**

**Theory Assignment#9**

**Submitted by:**

Arjun Neupane

013BSCCSIT009

**Submitted to:**

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

Date of submission: October 1, 2015

**Database Recovery:**

DBMS is a highly complex system with hundreds of transactions being executed every second. The durability and robustness of a DBMS depends on its complex architecture and its underlying hardware and system software. If it fails or crashes amid transactions, it is expected that the system would follow some sort of algorithm or techniques to recover lost data.

**1.Purpose of Data Recovery:**

In enterprise information technology (IT), data recovery typically refers to the restoration of data to a desktop, laptop, server, or external storage system from a backup. The data recovery process may vary, depending on the circumstances of the data loss, the data recovery software used to create the backup, and the backup target media. For example, many desktop and laptop backup software platforms allow end users to restore lost files themselves, while restoration of a corrupted database from a tape backup is a more complicated process that requires IT intervention. Data recovery can also be provided as service. Such services are typically used to retrieve important files that were not backed up and accidentally deleted from a computer's file system but still remain on disk in fragments.

An organization's disaster recovery plan should make known who in the organization is responsible for recovering data, provide a strategy for how data will be recovered and document acceptable recovery point and recovery time objectives.

**2. Types of failure:**

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

**2.1. 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.

**2.2. 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.

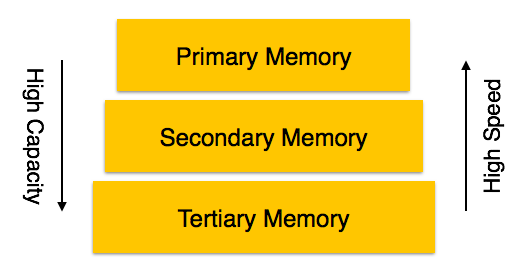
**2.3. Disk Failure:**

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.

**3. 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 –



**3.1. 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.

**3.2. 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.

**3.3. 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.

**4. 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.

**5. Transaction Log:**

A transaction is a sequential group of database manipulation operations, which is performed as if it were one single work unit. In other words, a transaction will never be complete unless each individual operation within the group is successful. If any operation within the transaction fails, the entire transaction will fail.

**Properties:**

Transactions have the following four standard properties, usually referred to by the acronym ACID:

**Atomicity:** ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure and previous operations are rolled back to their former state.

**Consistency:** ensures that the database properly changes states upon a successfully committed transaction.

**Isolation:** enables transactions to operate independently on and transparent to each other.

**Durability:** ensures that the result or effect of a committed transaction persists in case of a system failure.

There are following commands used to control transactions:

**COMMIT:** to save the changes.

**ROLLBACK:** to rollback the changes.

**SAVEPOINT:** creates points within groups of transactions in which to ROLLBACK

**SET TRANSACTION:** Places a name on a transaction.

**6. Data Updates:**

**Immediate Update:** As soon as a data item is modified in cache, the disk copy is updated.

**Deferred Update:** All modified data items in the cache is written either after a transaction ends its execution or after a fixed number of transactions have completed their execution.

**Shadow update:** The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.

**In-place update:** The disk version of the data item is overwritten by the cache version.

**7. Data Coaching:**

The database cache is a memory buffer which stores copies of portions of the database that the DBMS is currently using. Reading from memory is much faster than reading from the disk. The DBMS therefore returns a record more quickly if it is already stored in cache. As long as the required data is stored in cache, the data is immediately available. When the required data is not stored in cache, it must be copied from the disk and then stored in cache. Database Caching is the overall process of caching the ongoing data on the database cache so that it can be accessed faster.

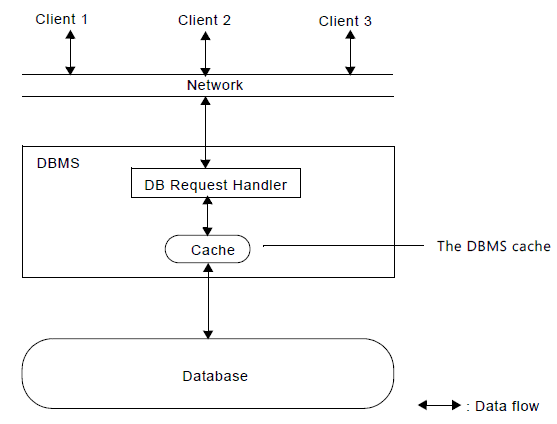
The DBMS cache is transparent to the user. For example, when a user requests data, the data is automatically copied into the cache and stored there. If the data is modified, it is automatically copied back to the physical disk. These data transfers take place automatically. The user does not need to know about the cache.

For example, three users send requests to the DBMS. When user 2 sends a request to read data from the database, the request handler determines whether the desired data can be fetched directly from the cache or whether it must be fetched from a disk.

At the same time, another user can modify a record in a table in the database. The modified data will be written to the DBMS cache, and not to the disk. When this user completes the write transaction (that is, commits the changes), the data in the cache that was modified during the transaction is written to the disk. The cache is then said to be flushed.

Advantages:

* **Scalability**: distribute query workload from backend to multiple cheap front-end systems.
* **Flexibility**: achieve QoS, where each cache hosts different parts of the backend data, e.g., the data of Platinum customers are cached while that of ordinary customers are not.
* **Availability**: by continued service for applications that depend only on cached tables even if the backend server is unavailable.
* **Performance**: by potentially responding fast because of locality of data and smoothing out load peaks by avoiding round-trips between middle-tier and data-tier



**8. Transaction Roll back (Undo):**

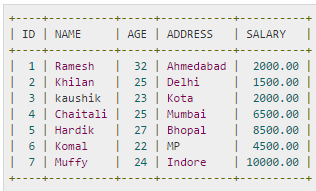
The **ROLLBACK** command is the transactional command used to undo transactions that have not already been saved to the database.

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

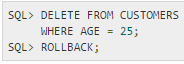
The syntax for ROLLBACK command is as follows:

****

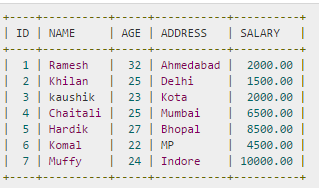
Example: consider a table CUSTOMER having following records



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



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

****

**9. Transaction Roll forward:**

RollForward occurs when the database restarts after an abnormal shutdown. It’s a process of going to the log files and applying changes from the log files to the underlying database. In the case where the underlying tables have been restored from an old backup this can involve millions of updates and take several hours.

**10. Check Pointing, Shadow Paging:**

**10.1. Check pointing:**

It refers to a validation point that compares the current value for specified properties or current state of an object with the expected value which can be inserted at any point of time in the script. 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.

**10.2. Shadow paging:**

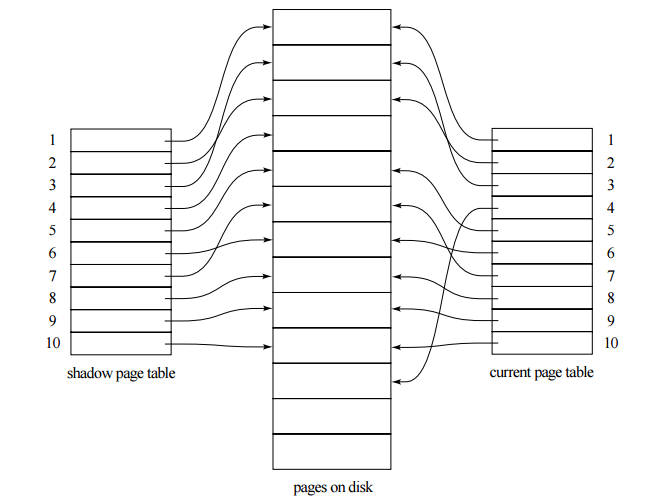
• Alternative to log-based recovery

• Idea: maintain two page tables during the lifetime of a transaction – the current page table, and the shadow page table

• Store the shadow page table in nonvolatile storage, such that state of the database prior to transaction execution may be recovered. Shadow page table is never modified during execution

• To start with, both the page tables are identical. Only current page table is used for data item accesses during execution of the transaction.

• Whenever any page is about to be written for the first time, a copy of this page is made onto an unused page. The current page table is then made to point to the copy, and the update is performed on the copy.



Advantages:

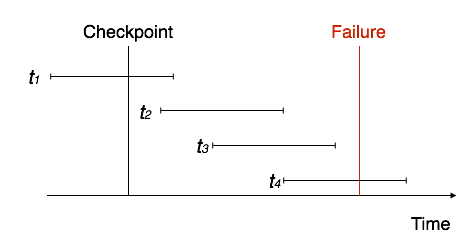
* No Overhead for writing log records.
* No Undo / No Redo algorithm.
* Recovery is faster.

Disadvantages:

* Data gets fragmented or scattered.
* After every transaction completion database pages containing old version of modified data need to be garbage collected.
* Hard to extend algorithm to allow transaction to run concurrently.

**11. Recovery Schemes (WAL: Write Ahead Logging Protocol):**

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.

All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

**12. 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.

**13. Recovery in Multi database System:**

Multi database transaction requires access to multiple databases. The DBs may even be stored on different types of DBMS.

To maintain the atomicity of a multi database 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 multi database 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.

The net effect of the two-phase commit protocol is that either all participating databases commit the effect of the transaction or none of them do. In case any of the participants—or the coordinator—fails, it is always possible to recover to a state where either the transaction is committed or it is rolled back. Failure during or before Phase 1 usually requires the transaction to be rolled back, whereas a failure during Phase 2 means that a successful transaction can recover and commit.

**Reference:**

<http://www.tutorialspoint.com/dbms/dbms_data_recovery.htm>

<http://programmers.stackexchange.com/questions/101350/are-rollback-and-roll-forward-transactions-equivalent-for-fault-tolerance>