**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System**

**Assignment # 9**

**Submitted by:**

Dwarika Shiwakoti  
013BSCCSIT020

**Submitted to:**

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

**Date of Submission:**October 01, 2015

**Database Recovery**

A major responsibility of the database administrator is to prepare for the possibility of hardware, software, network, process, or system failure. If such a failure affects the operation of a database system, you must usually recover the database and return to normal operation as quickly as possible. Recovery should protect the database and associated users from unnecessary problems and avoid or reduce the possibility of having to duplicate work manually.

Recovery processes vary depending on the type of failure that occurred, the structures affected, and the type of recovery that you perform. If no files are lost or damaged, recovery may amount to no more than restarting an instance. If data has been lost, recovery requires additional steps.

1. **Purpose of database recovery**

The purpose of this policy is as follows:

* To provide secure storage for data assets critical to the work flow of official university business
* To prevent loss of data in the case of accidental deletion / corruption of data, system failure, or disaster
* To permit timely restoration of archived data in the event of a disaster or system failure
* This policy applies to all computers, both mobile and desktop, owned by the Library
* Specific locations will be automatically backed up (e.g., My Documents, Desktop,  Bookmarks)
* Any location outside of the automated backup locations will be added on a per request basis

1. **Types of failure**

**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.

Reasons for a 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.

Examples may include operating system errors.

**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.

1. **Storage hierarchy**

**Volatile storage**:

As the name suggests, a volatile storage cannot survive system crashes. Volatile storage devices are placed very close to the CPU; normally they are embedded onto the chipset itself. For example, main memory and cache memory are examples of volatile storage. They are fast but can store only a small amount of information.

**Nonvolatile storage**:

These memories are made to survive system crashes. They are huge in data storage capacity, but slower in accessibility. Examples may include hard-disks, magnetic tapes, flash memory, and non-volatile (battery backed up) RAM.

**Stable storage**:

It is a mythical form of storage that survives all failures.It is approximated by maintaining multiple copies on distinct nonvolatile mediaIt is implemented in the following ways

Maintain multiple copies of each block on separate disks

* copies can be at remote sites to protect against disasters such as fire or flooding.

Failure during data transfer can still result in inconsistent copies: Block transfer can result in

* Successful completion
* Partial failure: destination block has incorrect information
* Total failure: destination block was never updated

Protecting storage media from failure during data transfer (one solution):

* Execute output operation as follows (assuming two copies of each block):
* Write the information onto the first physical block.
* When the first write successfully completes, write the same information onto the second physical block.
* The output is completed only after the second write successfully completes.

1. **Buffer management**

When a Page is Requested ...

Buffer pool information table contains:

<frame#, pageid, pin\_count, dirty>

If requested page is not in pool:

* Choose a frame for replacement.
* Only “un-pinned” pages are candidates!
* If frame is “dirty”, write it to disk
* Read requested page into chosen frame
* Pin the page and return its address.
* If requests can be predicted (e.g., sequential scans), pages can be pre-fetched several pages at a time.

1. **Transaction log**

Every SQL Server database has a transaction log that records all transactions and the database modifications made by each transaction. The transaction log must be truncated on a regular basis to keep it from filling up. However, some factors can delay log truncation, so monitoring log size is important. Some operations can be minimally logged to reduce their impact on transaction log size.

The transaction log is a critical component of the database and, if there is a system failure, the transaction log might be required to bring your database back to a consistent state. The transaction log should never be deleted or moved unless you fully understand the ramifications of doing this.

1. **Data update**

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.

1. **Data caching**

* 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.

1. **Transaction Roll back(undo) and Roll forward**

Roll forward:

The Roll forward is redoing the changes made by a transaction that is after the committed transaction and to over-write the changed value once again to ensure the consistency.

Roll back:

The Rollback transaction is a transaction which rolls back the transaction to the beginning of the transaction (Rollback Transaction name). It is possible to use before Commit transaction.

1. **Check pointing and shadow paging**

**Check pointing:**

Check pointing stores content of a main memory database to backup data files on a regular basis. The purpose of check pointing is to minimize the length of time required for database recovery from a system failure.

ALTIBASE HDB uses fuzzy and ping-pong check pointing methods to safely back up and manage databases.

**Shadow paging:**

It is inconvenient to maintain logs of all transactions from the purposes of recovery. An alternative is to use a system of shadow paging. This is where the database is divided into pages that may be stored in any order on the disk. In order to identify the location of any given page, we use something called a page table.

During the life of a transaction two page tables are maintained, one called a shadow page table and current page table. When a transaction begins both of these page tables point to the same locations (are identical). During the lifetime of a transaction the shadow page table doesn't change at all. However during the lifetime of a transaction changes may be made update values etc. So whenever we update a page in the database we always write the updated page to a new location. This means that when we then update our current page table to reflect the changes that have been made.

Looking at diagram  we see how these tables appear during a transaction. As we can see the shadow page table shows the state of the database just prior to a transaction, and the current page table shows the state of the database during or after a transaction has been completed.

We now have a system whereby if we ever want to undo the actions of a transaction all we have to do is to do is recover the shadow page table to be the current page table. As such this makes the shadow page table particularly important, and so it must always be stored on stable storage. On disk we store a single pointer location that points to the address of the shadow page table. This means that to swapt the shadow table for the current page table (commiting the data) we just need to update this single pointer (very unlikley to fail during this very short fast operation).

1. **Recovery schemes(WALL: Write Ahead Logging Protocol)**

Deferred Update (No Undo/Redo)

* The data update goes as follows:
* A set of transactions records their updates in the log.
* At commit point under WAL scheme these updates are saved on database disk.
* After reboot from a failure the log is used to redo all the transactions affected by this failure. No undo is required because no AFIM is flushed to the disk before a transaction commits.

(WAL: Write Ahead Logging Protocol)

When in-place update (immediate or deferred) is used then log is necessary for recovery and it must be available to recovery manager. This is achieved by Write-Ahead Logging (WAL) protocol. WAL states that

* For Undo: Before a data item’s AFIM is flushed to the database disk (overwriting the BFIM) its BFIM must be written to the log and the log must be saved on a stable store (log disk).
* For Redo: Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

1. **Failure with loss of non volatile storage**

Technique similar to checkpointing used to deal with loss of non-volatile storage

Periodically **dump** the entire content of the database to stable storage

No transaction may be active during the dump procedure; a procedure similar to check pointing must take place

* Output all log records currently residing in main memory onto stable storage.
* Output all buffer blocks onto the disk.
* Copy the contents of the database to stable storage.
* Output a record <**dump**> to log on stable storage.

1. **Recovery in multidatabse system**

A multi database transaction requires access to multiple databases.The DBs may even be stored on different types of DBMS.Some DBMS may be relational, whereas others are object oriented, etc.Each DBMS involved in the multi database transaction may have its own recovery technique and transaction manager separate from those of the other DBMSs.Use a two-level recovery mechanism to maintain the atomicity of a multi database transaction. The coordinator usually follows a two-phase commit protocol.

Phase 1

When all participating databases signal the coordinator that the part of the multi database transaction has concluded, the coordinator sends a message «prepare to 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 or «cannot commit» -or not OK- if it fails for some other reasons. If the coordinator does not receive a reply from a database

Phase 2

If all the participants DB reply «OK» and also the coordinator, the transaction is successful and the coordinator sends a «commit» signal for the transaction to the participant databases. Each participant database completes transaction commit by writing a [commit] entry for the transaction in the log and permanently updating the database if needed. If one or more participating DBs or the coordinator sends «not OK» message, the transaction fails and the coordinator sends a message to «rollback» -or UNDO- the local effect of the transaction to each participating database. The UNDO of the local effect is done by using the log at each participating database