**ST.XAVIER’S COLLEGE**

MAITIGHAR, KATHMANDU



Database Management System

Assignment #9

Submitted By:

Binod Paneru

013BSCCSIT014

2nd year/ 4th semester

Submitted to:

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

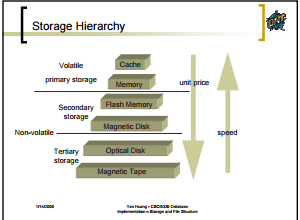
1. **Database Recovery**
   1. **Purpose of Data recovery.**

As a backup administrator, your principal duty is to devise, implement, and manage a backup and recovery strategy. In general, the purpose of a backup and recovery strategy is to protect the database against data loss and reconstruct the database after data loss. Typically, backup administration tasks include the following:

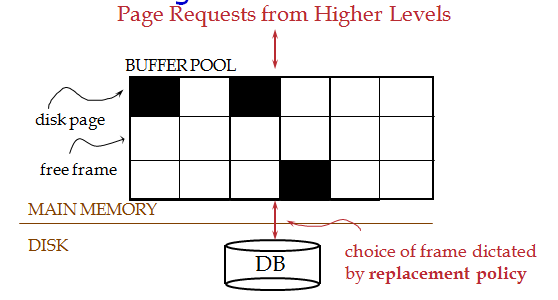
* Planning and testing responses to different kinds of failures
* Configuring the database environment for backup and recovery
* Setting up a backup schedule
* Monitoring the backup and recovery environment
* Troubleshooting backup problems
* Recovering from data loss if the need arises
  1. **Types of failure**
     1. Transaction System
* Caused by errors within the transaction processes.
* Caused by failure of network or operating system or physical threats to the system as a whole.
  + 1. Media
* Failure of hard disk, out of memory errors, out of disk space errors.
  1. **The Storage Hierarchy**

You can doubtless think of many examples of storage hierarchies in ordinary life. For example, people live in neighborhoods, which are in towns, which are in regions, countries, continents, and so on up the line. The relations are generally many-to-one, although there are occasional one-to-one correspondences (e.g., Australia is both a country and a continent), and occasional exceptions (e.g., a person can straddle a city boundary).

Figure shows the storage hierarchy—the physical constructs of a database. The hierarchy of physical objects suggests that—with occasional one-to-one correspondences or exceptions—data rows live in pages, which are in extents, which are in files, table spaces, and databases. There is a reason for each level of grouping. To see what the reason is, we'll go through each of those objects in order, up the line.

****

* 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 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.

* 1. **Data caching**

A cache, in computing, is a data storing technique that provides the ability to access data or files at a higher speed.

A cache works in both hardware and software to provide similar functionality. In its physical or hardware form, it is a small form factor of internal memory that stores instances of the most frequently executed programs in the main memory to enable faster access when they are requested by the CPU.

A very common example of caching is in a Web browser, where a website's HTML, images, CSS, JavaScript, etc is cached locally so that a page will load faster after its first hit.

* 1. **Transaction roll back(Undo)& 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.

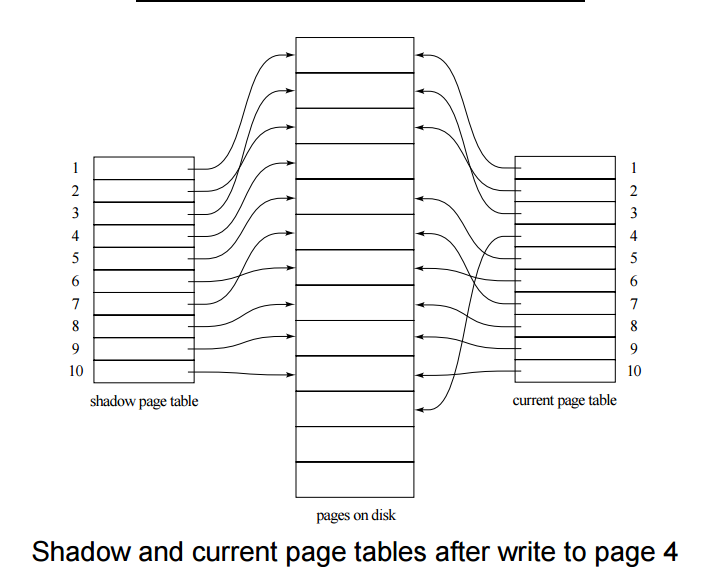
Roll back: The Rollback transaction is a transaction which rolls back the transaction to the beginning of the transaction (Rollback Transaction name). It is possible to use before Commit transaction.

* 1. **Check Pointing, Shadow Paging**

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.

Example of Shadow paging



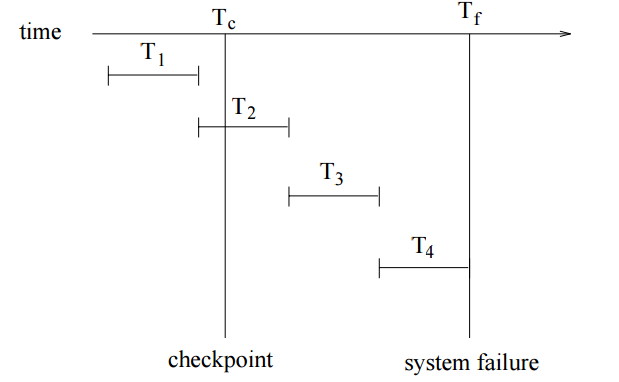
Checkpoint:

* Problems in recovery procedure as discussed earlier :

1. searching the entire log is time-consuming
2. we might unnecessarily redo transactions which have already output their updates to the database.

• Streamline recovery procedure by periodically performing check pointing

* + - 1. Output all log records currently residing in main memory onto stable storage.
      2. Output all modified buffer blocks to the disk.
      3. Write a log record onto stable storage

**Example of checkpoint: **

* T1 can be ignored (updates already output to disk due to checkpoint)
  + T2 and T3 redone
  + T4 undone
  1. **Recovery Schemes (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 (General Concept)**
* 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.
* To recover from disk failure, restore database from most recent
* Dump. Then log is consulted and all transactions that
* Committed since the dump are redone.
* Can be extended to allow transactions to be active during
* Dump; known as fuzzy or online dump.
  1. **Recovery in multi database system.**

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.

**REFERENCE:**

[1]. <http://docs.oracle.com/cd/B28359_01/backup.111/b28270/rcmintro.htm#CHDCBEDG>

[2]. <https://www.dlsweb.rmit.edu.au/toolbox/knowmang/content/backup/failure_types.htm#Types>

[3]. <http://codeidol.com/community/sql/the-storage-hierarchy/3437/>

[4]. <http://www.inf.uni-konstanz.de/dbis/teaching/ss04/architektur-von-dbms/download/local/buffermanager.pdf>

[5]. <http://www.fi.muni.cz/~zezula/lectures/PB154/mod15_1.pdf>

[6]. <https://msdn.microsoft.com/en-us/library/ms190925.aspx>

[7]. <https://www.classle.net/book/recovery-multidatabase-systems>