ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



DATABASE MANAGEMENT SYSTEM

THEORY ASSIGNMENT #9

**Submitted by:**

Lokendra Puri  
013BSCCSIT023

**Submitted to:**

|  |  |
| --- | --- |
| Er. Sanjay Kumar Yadav  Lecturer |  |

Department of Computer Science

Date of Submission: October 1, 2015

**DATA RECOVERY**

* All database reads/writes are within a transaction
* Transactions have the “ACID” properties
  + Atomicity - all or nothing
  + Consistency - preserves database integrity
  + Isolation - execute as if they were run alone
  + Durability - results aren’t lost by a failure
* Recovery subsystem guarantees A & D
* Concurrency control guarantees I
* Application program guarantees C

1. **PURPOSE OF DATA RECOVERY**
2. **TYPES OF FAILURE**

Transaction failure:

Logical errors: transaction cannot complete due to some internal error condition

System errors: the database system must terminate an active transaction due to an error condition (e.g., deadlock)

System crash:

a power failure or other hardware or software failure causes the system to crash.

Fail-stop assumption: non-volatile storage contents are assumed to not be corrupted by system crash

Database systems have numerous integrity checks to prevent corruption of disk data

Disk failure:

a head crash or similar disk failure destroys all or part of disk storage

Destruction is assumed to be detectable: disk drives use checksums to detect failures

1. **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.

1. **BUFFER MANAGEMENT**

Database maintains an in-memory buffer of data blocks

When a new block is needed, if buffer is full an existing block needs to be removed from buffer

If the block chosen for removal has been updated, it must be output to disk

As a result of the write-ahead logging rule, if a block with uncommitted updates is output to disk, log records with undo information for the updates are output to the log on stable storage first.

No updates should be in progress on a block when it is output to disk. Can be ensured as follows.

Before writing a data item, transaction acquires exclusive lock on block containing the data item

Lock can be released once the write is completed.

* + - Such locks held for short duration are called **latches**.

Before a block is output to disk, the system acquires an exclusive latch on the block

* + - Ensures no update can be in progress on the block

Database buffer can be implemented either

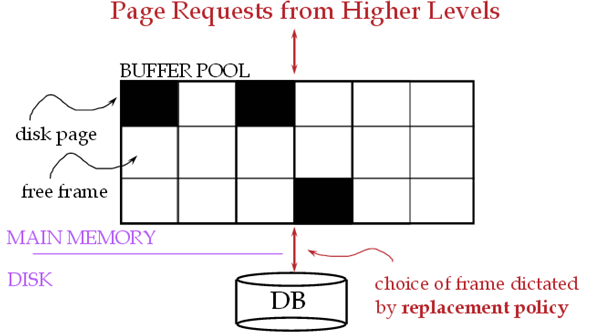
in an area of real main-memory reserved for the database, or

in virtual memory

Implementing buffer in reserved main-memory has drawbacks:

Memory is partitioned before-hand between database buffer and applications, limiting flexibility.

Needs may change, and although operating system knows best how memory should be divided up at any time, it cannot change the partitioning of memory.

****

1. **TRANSACTION LOG**

A DBMS uses a **transaction log** to keep track of all transactions that update the database. The information stored in this log is used by the DBMS for a recovery requirement triggered by a ROLLBACK statement, a program’s abnormal termination, or a system failure such as a network discrepancy or a disk crash. Some RDBMSs use the transaction log to recover a database *forward* to a currently consistent state. After a server failure, for example, Oracle automatically rolls back uncommitted transactions and rolls forward transactions that were committed but not yet written to the physical database.

While the DBMS executes transactions that modify the database, it also automatically updates the transaction log.

The transaction log stores:

 A record for the beginning of the transaction.

 For each transaction component (SQL statement):

 The type of operation being performed (update, delete, insert).

 The names of the objects affected by the transaction (the name of the table).

 The “before” and “after” values for the fields being updated.

 Pointers to the previous and next transaction log entries for the same transaction.

 The ending (COMMIT) of the transaction.

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.

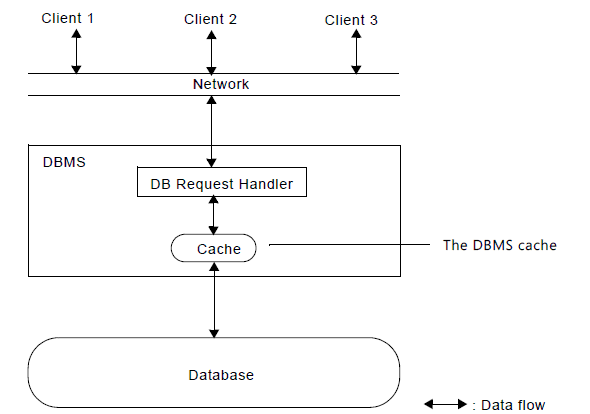
1. **DATA CACHING**

The Database Management System (DBMS) 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.

**DBMS Cache Transparency**

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.

The DBMS cache always contains the most recently used data. The cache is continually updated with the relevant data from the database.

The size of the cache greatly affects performance. When you set the size of the cache, you must remember two simple rules:

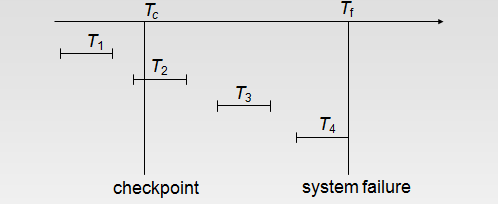
* The more memory you assign to the cache, the more efficient it becomes. (Of course, there is no reason to assign more memory to the cache than the total size of your database.)
* The size of the cache must not exceed the amount of physical memory available on your system. This is because the operating system may swap the cache memory in and out of the disk. This will considerably slow down overall performance.

1. **TRANSACTION ROLL BACK(UNDO) & ROLL FORWARD**

* As with the manual recovery approach we also make periodic backups of the database (time consuming operation).
* In the Automated Recovery approach, we introduce a **Log** file – this is a file separate from the data that records all of the changes made to the database by transactions.
* This *transaction log* Includes information helpful to the recovery process such as: A transaction identifier, the date and time, the user running the transaction, *before images* and *after images*
* **Before Image**: A copy of the table record (or data item)  before it was changed by the transaction.
* **After Image**: A copy of the table record (or data item)  after it was changed by the transaction.
* **Rollback**: Undo any partially completed transactions (ones in progress when the crash occurred) by applying the *before images* to the database.
* **Rollforward**: Redo the transactions by applying the *after images* to the database. This is done for transactions that were committed before the crash.
* The Automated Recovery process uses both rollback and rollforward to restore the database.
* In the worst case, we would need to rollback to the last database backup point and then rollforward to the point just before the crash.
* **Checkpoints** can also be taken (less time consuming) in between database saves.
  + The DBMS flushes all pending transactions and writes all data to disk and transaction log.
  + Database can be recovered from the last checkpoint in much less time.

1. **CHECK POINTING**

* Problem - Prevent Restart from scanning back to the start of the log
* A checkpoint is a procedure to limit the amount of work for Restart
* Commit-consistent checkpointing
  + Stop accepting new update, commit, and abort operations
  + make list of [active transaction, pointer to last log record]
  + flush all dirty pages
  + append a checkpoint record to log, which includes the list
  + resume normal processing
* Database and log are now mutually consistent

****

***T*1 can be ignored (updates already output to disk due to checkpoint)**

***T*2 and *T*3 redone**

***T*4 undone**

**SHADOW PAGING**

Shadow paging is an alternative to log-based recovery; this scheme is useful if transactions execute serially

**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. [to be a new current page]

The current page table is then made to point to the copy

The update is performed on the copy

1. **RECOVERY SCHEMES (WAL: WRITE AHEAD LOGGING PROTOCOLS)**
2. **FAILURE WITH LOSS OF NON VOLATILE STORAGE(GENERAL CONCEPTS)**

So far we assumed no 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 checkpointing 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.
    - Consult the log and redo all transactions that committed after the dump

Can be extended to allow transactions to be active during dump;   
known as **fuzzy dump** or **online dump**

1. **RECOVERY IN MULTI-DATABASE SYSTEM**