**ST.XAVIER’S COLLEGE**

MAITIGHAR, KATHMANDU



Database Management System

Assignment #9

Submitted By:

Aabhash Dhakal

013BSCCSIT001

2nd year/ 4th semester

Submitted to:

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

# Database Recovery

* 1. **Purpose of Data Recovery**

Recovery techniques can be used to restore data in a system to a usable state. Such techniques are widely used in filing systems and database systems in order to cope with failures. A failure is an event at which the system does not perform according to specifications. Some failures are caused by hardware faults (e.g., a power failure or disk failure), software faults {e.g., bugs in programs or invalid data), or human errors (e.g., the operator mounts a wrong tape on a drive, or a user does something unintentional). A failure occurs when an erroneous state of the system is processed by algorithms of the system. The term error is, in this context, used for that part of the state which is "incorrect." An error is thus a piece of information which can cause a failure.

In order to cope with failures, additional components and abnormal algorithms can be added to a system. These components and algorithms attempt to ensure that occurrences of erroneous states do not result in later system failures; ideally, they remove these errors and restore them to "correct" states from which normal processing can continue.

The purpose of database recovery is to bring the database into the last consistent state, which existed prior to the failure as well as to preserve transaction properties (Atomicity, Consistency, Isolation and Durability).

* 1. **Types of Failure**

Failures may be

|  |  |
| --- | --- |
| Transaction | Caused by errors within the transaction processes. |
| System | Caused by failure of network or operating system or physical threats to the system as a whole. |
| Media | Failure of hard disk, out of memory errors, out of disk space errors. |

Transaction errors, system errors, system crashes, concurrency problems and local errors or exceptions are the more common causes of system failure.  The system must be able to recover from such failures without loss of data.

* 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**

We need to use disk storage for the database, and to transfer blocks of data between MM and disk. We also want to minimize the number of such transfers, as they are time-consuming. One way is to keep as many blocks as possible in MM.

Usually, we cannot keep all blocks in MM, so we need to manage the allocation of available MM space. The **buffer** is the part of MM available for storage of **copies** of disk blocks. The subsystem responsible for the allocation of buffer space is called the **buffer manager**. The buffer manager handles all requests for blocks of the database.

If the block is already in MM, the address in MM is given to the requestor. If not, the buffer manager must read the block in from disk (possibly displacing some other block if the buffer is full) and then pass the address in MM to the requestor.

The buffer manager must use some sophisticated techniques in order to provide good service:

**Replacement Strategy** - When there is no room left in the buffer, some block must be removed to make way for the new one. Typical operating system memory management schemes use a ``least recently used'' (**LRU**) method. (Simply remove the block least recently referenced.) This can be improved upon for database applications.

**Pinned Blocks** - For the database to be able to recover from crashes, we need to restrict times when a block maybe written back to disk. A block not allowed to be written is said to be **pinned**. Many operating systems do not provide support for pinned blocks, and such a feature is essential if a database is to be ``crash resistant''.

**Forced Output of Blocks** - Sometimes it is necessary to write a block back to disk even though its buffer space is not needed. (Called the **forced output** of a block.) This is due to the fact that MM contents (and thus the buffer) are lost in a crash, while disk data usually survives.

**Replacement Strategy:** Goal is minimization of accesses to disk. Generally it is hard to predict which blocks will be referenced. So operating systems use the history of past references as a guide to prediction.

**General Assumption:** Blocks referenced recently are likely to be used again.

**Therefore:** if we need space, throw out the least recently referenced block. (LRU replacement scheme)

LRU is acceptable in **operating systems**, however, a database system is able to predict future references more accurately.

* 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 (updates, 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.
  1. **Data Updates**

There are various kinds of updates that can be performed in database system, some of which are:

* 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**

For the efficiency of recovery purpose, the caching of disk pages is handled by the DBMS instead of the OS. Typically, a collection of in-memory buffers, called DBMS cache kept under the control of the DBMS. A directory for the cache is used to keep track of which DB items are in the buffers.

– A table of <disk page address, buffer location> entries.

The DBMS cache holds the database disk blocks including

• Data blocks

• Index blocks

• Log blocks

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 Rollback and Roll-forward**

To maintain atomicity, a transaction’s operations are redone or undone.

* Undo: Restore all BFIMs on to disk (Remove all AFIMs).
* Redo: Restore all AFIMs on to disk.

Database recovery is achieved either by performing only Undos or only Redos or by a combination of the two. These operations are recorded in the log as they happen.

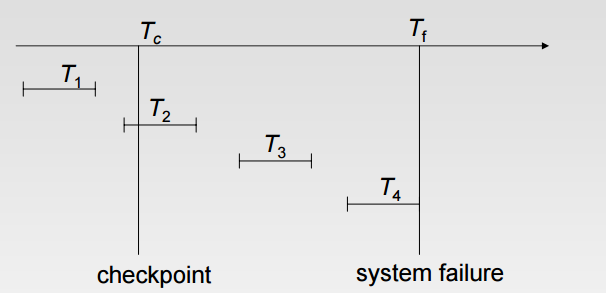
If the recovery protocols ensure recoverable schedules without to ensure strict or cascade-less schedules, cascading rollback can occur. If a transaction T is rolled back, roll back every transaction T’ that read a value of a data item written by T, and so on. Read-item operations are recorded in the log only to determine whether cascading rollback of additional transactions is necessary. Cascading rollback is complex and time-consuming. – Almost all recovery mechanisms are designed such that cascading rollback is never required –they guarantee strict or cascade-less schedules.

* 1. **Check Pointing, shadow paging**

We can streamline the recovery procedure by periodically performing check pointing. All log records correctly residing in main memory are output onto stable storage. Similarly, we output all modified buffer blocks to the disk. Finally, we write a log record <checkpoint> onto stable storage.

During recovery, we need to consider only the most recent transaction Ti that started before the checkpoint and transactions that started after Ti.

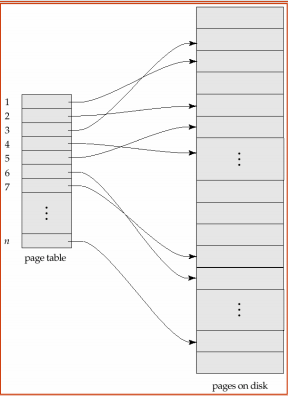
* Scan backwards from end of log to find the most recent <checkpoint> record.
* Continue scanning backwards till a record <Ti start> is found.
* Need only consider the part of log following above start record. Earlier part of log can be ignored during recovery, and can be erased whenever desired.
* For all transactions (starting from Ti or later) with no <Ti commit>, execute undo (Ti).
* Scanning forward in the log, for all transactions starting from Ti or later with a < Ti commit>, execute redo (Ti).

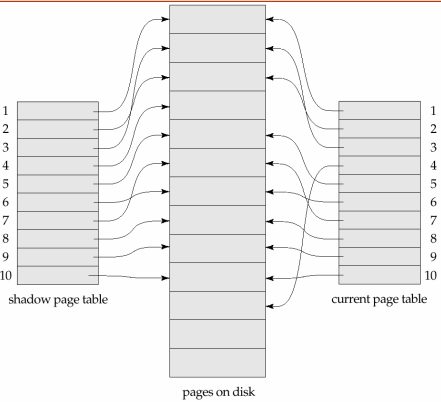


* T1 can be ignored.
* T2 and T3 re-done.
* T4. Undone.

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. z 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
* The update is performed on the copy





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

Write Ahead Logical Protol guarantees that no data modifications are written to disk before the associated log record is written to disk. This maintains the ACID properties for a transaction.

At the time a modification is made to a page in the buffer, a log record is built in the log cache that records the modification. This log record must be written to disk before the associated dirty page is flushed from the buffer cache to disk. If the dirty page is flushed before the log record is written, the dirty page creates a modification on the disk that cannot be rolled back if the server fails before the log record is written to disk. SQL Server has logic that prevents a dirty page from being flushed before the associated log record is written. Log records are written to disk when the transactions are committed.

* 1. **Failure with Loss of Non-Volatile Storage (General Concepts)**

Technique similar to check pointing 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 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**

A multi-database transaction requires access to multiple databases. The Databases 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. Thus, we use a two-level recovery mechanism to maintain the atomicity of a multi-database transaction:

– A global recovery manager or coordinator.

– The local recovery manager.

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 within a certain time out interval, it assumes a «not OK» response.

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 participating 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