**ST. XAVIER’S COLLEGE**

**MAITIGHAR, KATHMANDU**

****

**Database Management System**

**Theory Assignment #9**

**Submitted by:**

Anu Kadel

013BSCIT007

**Submitted to:**

|  |  |
| --- | --- |
| **Er. Sanjay Kumar Yadav** |  |

Lecturer

Department of Computer Science

St. Xavier’s College, Maitighar

**THEORY ASSIGNMENT#9**

**DATABASE RECOVERY**

Data recovery is the process of restoring data that has been lost, accidentally deleted, corrupted or made inaccessible for any reason.

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 DATA 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**

Two common kinds of failures

* System failure (e.g. power outage)
  + affects all transactions currently in progress but does not physically damage the data (soft crash)
* Media failures (e.g. Head crash on the disk)
  + damage to the database (hard crash)
  + need backup data

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

**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. **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**

The primary purpose of a SQL Server database is to store and retrieve data, so intensive disk I/O is a core characteristic of the Database Engine. And because disk I/O operations can consume many resources and take a relatively long time to finish, SQL Server focuses on making I/O highly efficient**. Buffer management** is a key component in achieving this efficiency.

The buffer management component consists of two mechanisms: the buffer manager to access and update database pages, and the buffer cache (also called the buffer pool), to reduce database file I/O.

The subsystem responsible for the allocation of buffer space is called the **buffer manager**

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

* **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** –

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.

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.

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

1. **TRANSACTION LOG**

Transactions

* A database is updated by processing *transactions* that result in changes to one or more records.
* A user’s program may carry out many operations on the data retrieved from the database, but the DBMS is only concerned with data read/written from/to the database.
* The DBMS’s abstract view of a user program is a sequence of transactions (reads and writes).

Log File

* Log file may be duplexed or triplexed.
* Log file sometimes split into two separate randomaccess files.
* Potential bottleneck; critical in determining overall performance.

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**

Many applications today are being developed and deployed on multi-tier environments that involve browser-based clients, web application servers and backend databases. These applications need to generate web pages on-demand by talking to backend databases because of their dynamic nature, making middle-tier database caching an effective approach to achieve high scalability and performance.

**Benefits:**

* **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

1. **TRANSACTION ROLL BACK(UNDO) AND ROLL FRWARD**

Database buffers in the buffer cache in the SGA are written to disk only when necessary, using a least-recently-used algorithm. Because of the way that the DBW*n* process uses this algorithm to write database buffers to datafiles, datafiles might contain some data blocks modified by uncommitted transactions and some data blocks missing changes from committed transactions.

Two potential problems can result if an instance failure occurs:

* Data blocks modified by a transaction might not be written to the datafiles at commit time and might only appear in the redo log. Therefore, the redo log contains changes that must be reapplied to the database during recovery.
* After the roll forward phase, the datafiles may contain changes that had not been committed at the time of the failure. These uncommitted changes must be rolled back to ensure transactional consistency. These changes were either saved to the datafiles before the failure, or introduced during the roll forward phase

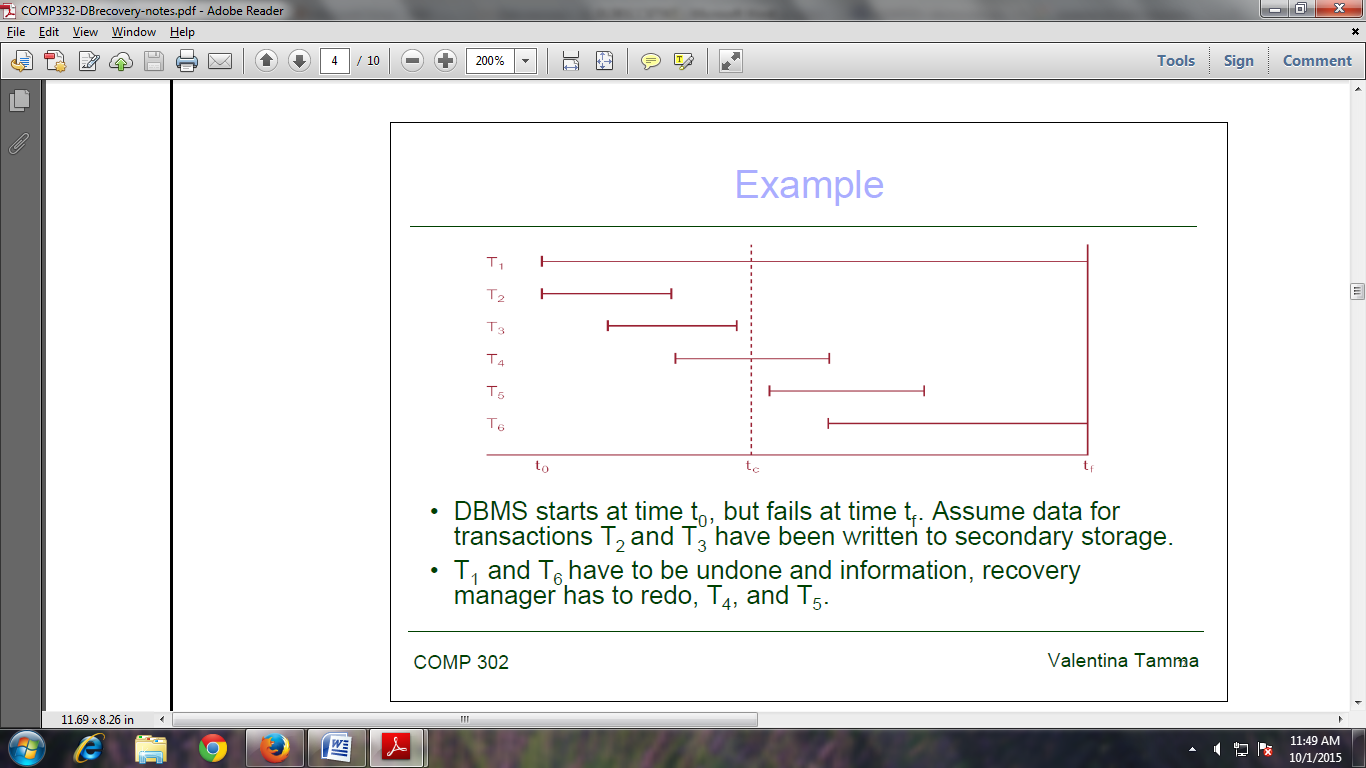
1. **CHECK POINTING, SHADOW PAGING**

**CHECKPOINT:** Point of synchronization between database and log file. All buffers are force-written to secondary storage.

* Checkpoint record is created containing identifiers of all active transactions.
* When failure occurs, redo all transactions that committed since the checkpoint and undo all transactions active at time of crash.

Example, with checkpoint at time tc, changes made by T2 and T3 have been written to secondary storage. Thus:

* only redo T4 and T5,
* undo transactions T1 and T6.



**SHADOW PAGING**

* Maintain two page tables during life of a transaction: current page and shadow page table.
* When transaction starts, two pages are the same.
* Shadow page table is never changed thereafter and is used to restore database in event of failure.
* During transaction, current page table records all updates to database.
* When transaction completes, current page table becomes shadow page table.

1. **RECOVERY SCHEMES(WAL: WRITE AHEAD LOGGING PROTOCOLS)**

Recovery scheme responsible for handling failures and restoring database to consistent state

Recovery algorithm has two parts

* Actions taken during normal operation to ensure system can recover from failure (e.g., backup, log file)
* Actions taken after a failure to restore database to consistent state

**Write-Ahead Logging (WAL)**

Two types of log entry –log record- information for a write command.

The information needed for UNDO.

* A UNDO-type log entries including the old values (BFIM).
  + Since this is needed to undo the effect of the operations from the log.

The information needed for REDO.

* A REDO-type log entries including the new values (AFIM).
  + Since it is needed to redo the effect of the operations from the log
  + In UNDO/REDO algorithms, both types of log entries are combined.
  + The log includes read commands only when cascading rollback is possible

Write-Ahead Logging (WAL) is the fundamental rule that ensures that a record –entry- of every change to the DB is available while attempting to recover from a crash.

WAL protocol for a recovery algorithm that requires both UNDO and REDO.

* The before image of an item cannot be overwritten by its after image in the database on disk until all UNDO-type log records for the updating transaction –up to this point in time- have been force-written to disk.
  + Ensures atomicity.
* The commit operation of a transaction cannot be completed until all the REDO-type and UNDO-type log records for that transaction have been force-written to disk.
  + Ensures durability.

1. **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.

1. **RECOVERY IN MULTIDATABASE SYSTEM**

A multidatabase transaction require access to multiple databases.The DBs may even be stored on different types of DBMS.

To maintain the atomicity of a multidatabase 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 multidatabase 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. A 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.