**ST. XAVIER’S COLLEGE**

**MAITIGHAR, KATHMANDU**

****

**database management system**

**LAb ASSIGNMENT #9**

**Submitted By:**

Abhishek Tamrakar

013BSCIT003

**Submitted To:**

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

Department of Computer Science

St. Xavier’s College, Maitighar

Lecturer

**Database Recovery**

1. **Introduction**

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].

To recover a SQL Server database from a failure, a database administrator has to restore a set of SQL Server backups in a logically correct and meaningful restore sequence. SQL Server restore and recovery supports restoring data from backups of a whole database, a data file, or a data page, as follows:

* The database (a complete database restore)

The whole database is restored and recovered, and the database is offline for the duration of the restore and recovery operations.

* The data file (a file restore)

A data file or a set of files is restored and recovered. During a file restore, the filegroups that contain the files are automatically offline for the duration of the restore. Any attempt to access an offline filegroup causes an error.

* The data page (a page restore)

Under the full recovery model or bulk-logged recovery model, you can restore individual databases. Page restores can be performed on any database, regardless of the number of filegroups [2].

1. **Purpose of data recovery**

* To bring the database into the last consistent state, which existed prior to the failure.
* To preserve transaction properties (Atomicity, Consistency, Isolation and Durability).

Example:

If the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have incorrect value. Thus, the database must be restored to the state before the transaction modified any of the accounts.

1. **Types of failure**

The types of failure in DBMS are:

**3.1 Transaction failures**

When a transaction is failed to execute or it reaches a point after which it cannot be completed successfully it has to abort. This is called transaction failure, where only few transaction or processes are hurt.

* overflow, interrupt, data not available, explicit rollback,
* concurrency enforcement, programming errors
* no memory loss.

Reason for transaction failure could be:

* Logical errors: where a transaction cannot complete because of it has some code error or any internal error condition
* System errors: where the database system itself terminates an active transaction because 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 systems aborts an active transaction.

**3.2 System crashes**

There are problems, which are external to the system, which may cause the system to stop abruptly and cause the system to crash. For example interruption in power supply, failure of underlying hardware or software failure. Examples may include operating system errors.

* due to hardware or software errors
* main memory content is lost

**3.3 Media failures**

* problems with disk head, unreadable media surface
* (parts of ) information on secondary storage may be lost
* lose a disk or a disk controller fails
* a head crash
* file corruption
* the overwriting or deletion of a datafile
* sny of the multiplexed control files are deleted or lost
* anydatafile belonging to the system or the undo tablespace is deleted or lost.
* sn entire redo log group is lost.

**3.4 Natural disasters**

* fire, flood, earthquakes, theft, etc.
  + - * physical loss of all information on all media

**3.5 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 part of disk storage.

**3.6 Statement Failure**

* When a program attempts to enter invalid data into an Oracle table.
* No more room to put the data in.
* Not having the proper privileges to perform a task.

**3.7 User Process Failure**

* A user process may be terminated abruptly (A single database session fails).

**3.9 Network Failure**

* Oracle Net listener.
* The network interface card.
* The network connection has failed.

**3.10 Instance Failure**

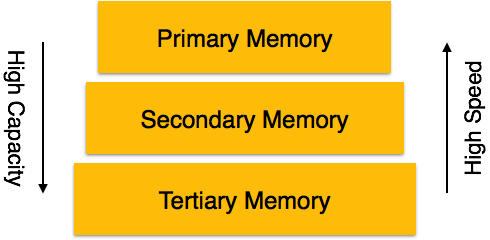
* Hardware failure.
* A power failure.
* An emergency shutdown procedure.
* Oracle background process such as pmon shuts down because of an error condition.

**3.11 User Error**

* Accidentally dropping a table.
* Wrongly modify or delete data from a table [3][4][5].

1. **Storage hierarchy**

Databases are stored in file formats, which contains records. At physical level, actual data is stored in electromagnetic format on some device capable of storing it for a longer amount of time. These storage devices can be broadly categorized in three types:



**4.1 Primary Storage:**

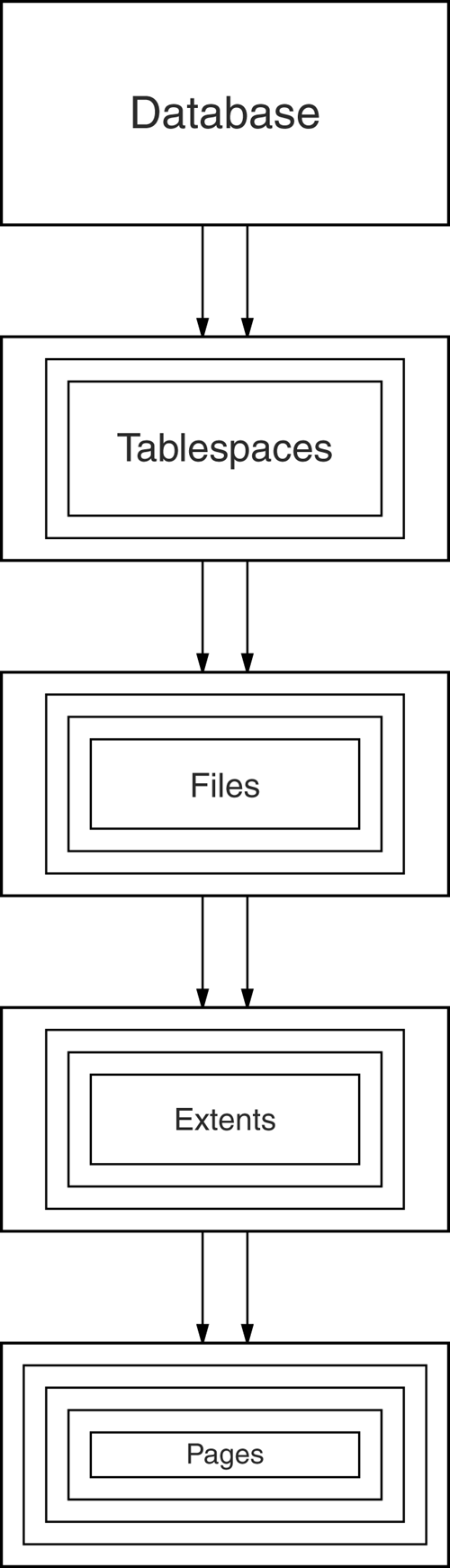
The memory storage, which is directly accessible by the CPU, comes under this category. CPU's internal memory (registers), fast memory (cache) and main memory (RAM) are directly accessible to CPU as they all are placed on the motherboard or CPU chipset. This storage is typically very small, ultra fast and volatile. This storage needs continuous power supply in order to maintain its state, i.e. in case of power failure all data are lost.

**4.2 Secondary Storage:**

The need to store data for longer amount of time and to retain it even after the power supply is interrupted gave birth to secondary data storage. All memory devices, which are not part of CPU chipset or motherboard comes under this category. Broadly, magnetic disks, all optical disks (DVD, CD etc.), flash drives and magnetic tapes are not directly accessible by the CPU. Hard disk drives, which contain the operating system and generally not removed from the computers are, considered secondary storage and all other are called tertiary storage.

**4.3 Tertiary Storage:**

Third level in memory hierarchy is called tertiary storage. This is used to store huge amount of data. Because this storage is external to the computer system, it is the slowest in speed. These storage devices are mostly used to backup the entire system. Optical disk and magnetic tapes are widely used storage devices as tertiary storage [6].

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, tablespaces, 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.

**Pages**

Depending on the DBMS, a page is also called a data block, a block, a blocking unit, a control interval, and a row group.A page is a fixed-size hopper that stores rows of data. Pages have four common characteristics, which are not true by definition but are always true in practice. All pages in a file have the same size. Indeed for some DBMSs, it is true that all pages in all files have the same size, but the usual case is that you have a choice when making a new object.

**Extents**

An extent is a group of contiguous pages. Extents exist to solve the allocation problem. The allocation problem is that, when a file gets full, the DBMS must increase its size. If the file size increases by only one page at a time, waste occurs because:

* The operating system must update the file allocation tables. The amount of updating is about the same whether the addition is one page or eight pages.
* If file A grows, then file B grows, then file A grows again, and so on; the operating system will have to maintain a succession of short (one page) chains: ABABABAB. This is a type of fragmentation. As we've hinted before, fragmentation is bad; we'll discuss it later in this chapter.

**Files**

A file is a group of contiguous extents. Surprisingly, a file is not a physical representation of a table.

**Tablespaces**

A tablespace (also called a dbspace by some DBMSs, e.g., Informix) is a file, or a group of files, that contains data [7].

1. **Buffer management**

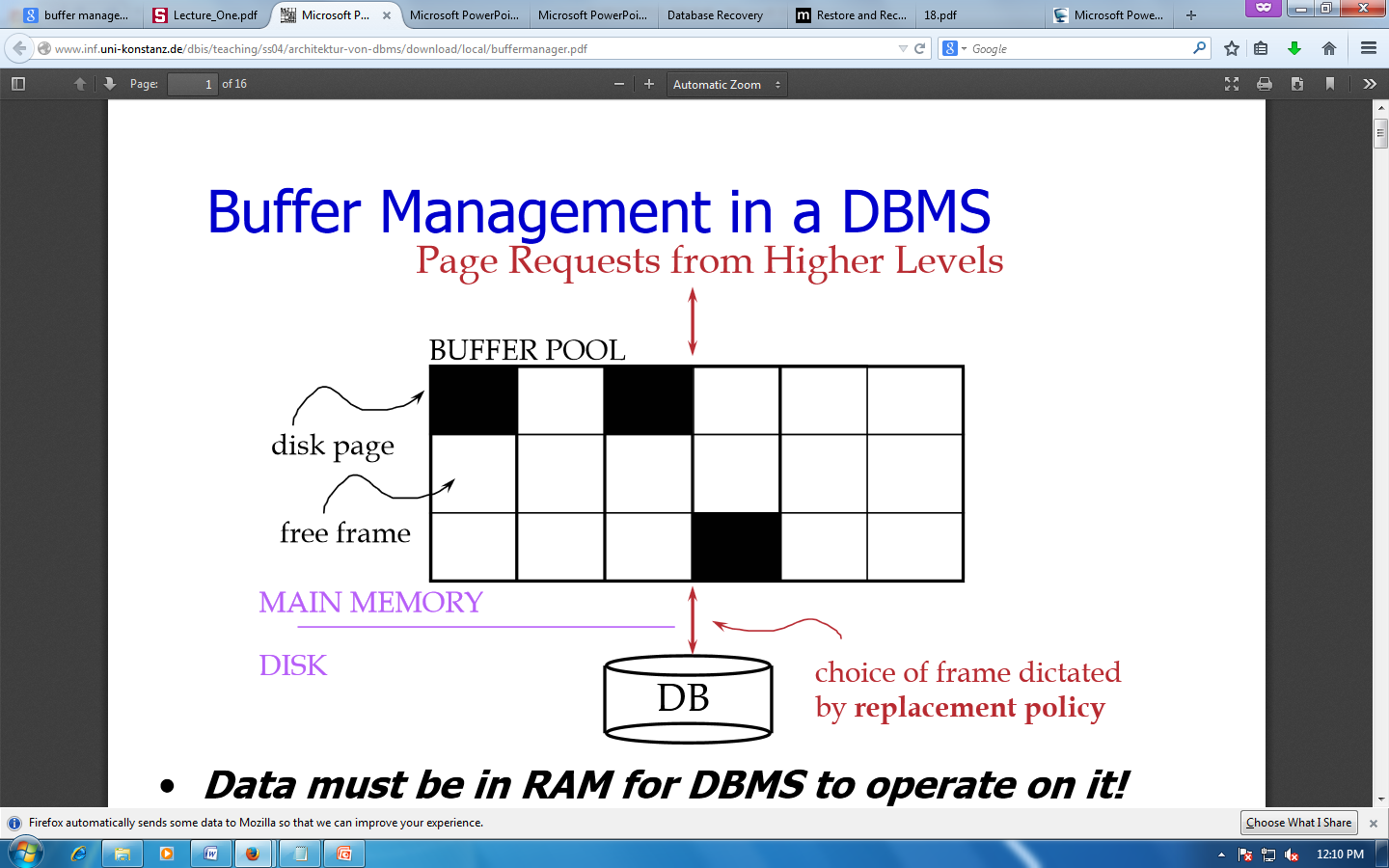


Figure Buffer Management in DBMS

Data must be in RAM for DBMS to operate on it. Buffer Mgr hides the fact that not all data is in RAM

* 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.
* Once we choose a page to remove
  + A page is dirty, if its contents have been changed after writing
  + Buffer Manager keeps a dirty bit
  + Say we choose to evict, if P is dirty, we write it to disk [8]

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 recoveryrequirement 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 thetransaction 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 automaticallyupdates 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 [9].

1. **Data Updates**

The UPDATE statement is used to change or modify the existing records in a database table. It is typically used in conjugation with the WHERE clause to apply the changes to only those records that matches specific criteria.

The basic syntax of the UPDATE statement can be given with:

UPDATE table\_name SET column1=value, column2=value2,... WHERE column\_name=some\_value

Let's make a SQL query using the UPDATE statement and WHERE clause, after that we will execute this SQL query through passing it to the mysqli\_query() function to update the tables records. Consider the following "persons" table inside the "demo" database:

+-----------+------------+-----------+----------------------+

| person\_id | first\_name | last\_name | email\_address |

+-----------+------------+-----------+----------------------+

| 1 | Peter | Parker | peterparker@mail.com |

| 2 | John | Rambo | johnrambo@mail.com |

| 3 | Clark | Kent | clarkkent@mail.com |

| 4 | John | Carter | johncarter@mail.com |

| 5 | Harry | Potter | harrypotter@mail.com |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

1. **Data Caching:**

• Cache the disk pages (blocks)–containing DB items to be updated- into main memory buffers.

• Then update the main memory buffers before being written back to disk.

• 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

• When DBMS requests action on some item

– First checks the cache directory to determine if the corresponding disk page is in the cache.

– If no, the item must be located on disk and the appropriate disk pages are copied into the cache.

– It may be necessary to replace (flush) some of the cache buffers to make space available for the new item.

• FIFO or LRU can be used as replacement strategies.

• Dirty bit.

– Associate with each buffer in the cache a dirty bit.

• The dirty bit can be included in the directory entry.

– It indicates whether or not the buffer has been modified.

• Set dirty bit=0 when the page is first read from disk to the buffer cache.

• Set dirty bit=1 as soon as the corresponding buffer is modified.

– When the buffer content is replaced –flushed- from the cache, write it back to the corresponding disk page only if dirty bit=1

1. **Transaction rollback and roll forward:**

• If a transaction fails, roll back this transaction.

• Any data item values changed by this transaction and written to the DB must be restored to their previous values –BFIM-

• 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 cascadless schedules

– When a crash occurs, transaction T3 has not terminated and must be rolled back.

• The WRITE operations of T3, marked by a single \*, are the ones to be undone during transaction rollback

– We check now for cascading rollback.

•T2 reads the value of item B that was written by transaction T3

–This can also be determined by examining the log.

• T3 rolled back so T2 rolled back too.

• The write operations of T2 to be rolled back are the ones marked by \*\* in the log.

– Rollback actions

– The DB item D is restored to its old value 25.

– The DB item B is first restored to its old value 12 and finally to 15.

– Insert the following log records into the log, [T2, abort] and [T3, abort].

1. **Checkpoint :**

• In case of failure, the recovery manager requires that the entire log be examined to process recovery→time consuming

•A quick way to limit the amount of log to scan on recovery can be achieved using checkpoints.

•A [checkpoint] record is written into the log periodically at that point when the system writes out to the database on disk all

DBMS buffers that have been modified.

• Or the intervals measured in the number of committed transactions since the last checkpoint, say t transactions.

– m& t are system parameter

•Take a checkpoint consists of the following

1. Suspend execution of transactions temporarily.

2. Force-write all main memory buffers that have been modified to disk.

3. Write a [checkpoint] record to the log, and force-write the log to disk.

4. Resume executing transactions.

•The time needed to force-write all modified memory buffers may delay transaction processing

• Because of step 1.

• Use fuzzy checkpointing to reduce this delay.

– The system can resume transaction processing after [checkpoint] record written to the log without having to wait for step 2 to finish.

– However, until step 2 is completed, the previous [checkpoint] record should remain to be valid.

• The system maintains a pointer to the valid checkpoint which points to the previous [checkpoint] recording the log

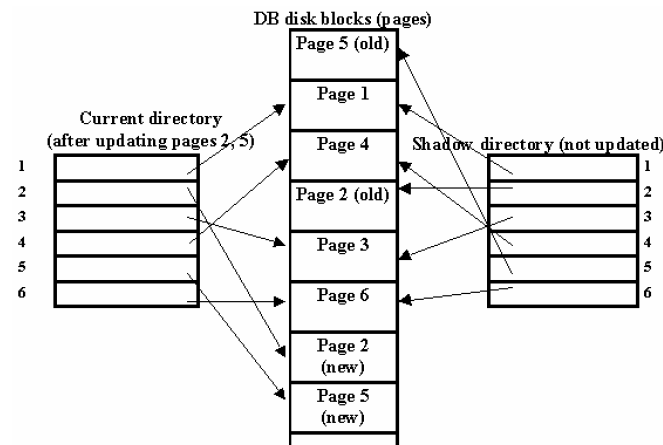
– Once step2 is conducted, that pointer is changed to point to the new checkpoint in the log.

1. **Shadow paging:**

This recovery scheme does not require the use of a log in a single-user environment. In a multiuser environment, a log may be needed for the concurrency control method. Shadow paging considers the database to be made up of a number of fixed-size disk pages (or disk blocks)—say, n—for recovery purposes. A directory with n entries is constructed, where the ith entry points to the ith database page on disk. The directory is kept in main memory if it is not too large, and all references—reads or writes—to database pages on disk go through it. When a transaction begins executing, the current directory—whose entries point to the most recent or current database pages on disk—is copied into a shadow directory. The shadow directory is then saved on disk while the current directory is used by the transaction.

During transaction execution, the shadow directory is never modified. When a write\_item operation is performed, a new copy of the modified database page is created, but the old copy of that page is not overwritten. Instead, the new page is written elsewhere—on some previously unused disk block. The current directory entry is modified to point to the new disk block, whereas the shadow directory is not modified and continues to point to the old unmodified disk block. For pages updated by the transaction, two versions are kept. The old version is referenced by the shadow directory, and the new version by the current directory.

To recover from a failure during transaction execution, it is sufficient to free the modified database pages and to discard the current directory. The state of the database before transaction execution is available through the shadow directory, and that state is recovered by reinstating the shadow directory. The database thus is returned to its state prior to the transaction that was executing when the crash occurred, and any modified pages are discarded. Committing a transaction corresponds to discarding the previous shadow directory. Since recovery involves neither undoing nor redoing data items, this technique can be categorized as a NO-UNDO/NO-REDO technique for recovery.



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

•Two types of log entry –log record- information for awrite 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.

– Suppose that that the BFIM of a data item on disk has been overwritten by the AFIM on disk and a crash occurs.

• Without ensuring that this BFIM is recorded in the appropriate log entry and the log is flushed to disk before the BFIM is overwritten with the AFIM in the DB on disk, the recovery will not be possible.

– Suppose a transaction made a change and committed with some of its changes not yet written to disk?

• Without a record of these changes written to disk, there would be no way to ensure that the changes of the committed transaction survive crashes

• The log is a simple sequential (append-only) file

– When a log record is written, it is stored in the current log in the DBMScache and after written to disk as soon as is feasible.

– With Write-Ahead Logging, the log blocks containing the associated log records for a particular data block update must first be written to disk before the data block itself can be written back to disk.

• WAL protocol for a recovery algorithm that requires both

UNDO and REDO.

1. 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.

2. 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.

• Commercial DBMSs and WAL

– IBM DB2, Informix, Microsoft SQL Server, Oracle 8, and Sybase ASE all use a WAL scheme for recovery.

• To facilitate the recovery process, the DBMS recovery subsystem may need to maintain a number of lists.

• List of active transactions: transactions started but not committed yet

• List of committed transactions since last checkpoint.

• List of aborted transactions since last checkpoint.

1. **Failure with loss of non-volatile storage (general concepts** ):

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

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

b.      No transaction may be active during the dump procedure; a procedure similar to checkpointing must take place

                                                              i.      Output all log records currently residing in main memory onto stable storage.

                                                            ii.      Output all buffer blocks onto the disk.

\                                                          iii.      Copy the contents of the database to stable storage.

                                                         iv.      Output a record <**dump**> to log on stable storage.

1. **Recovery in multi database system:**

• A multidatabase transaction require access to multipledatabases.

– 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 managerseparate from those of the other DBMSs.

• Use a two-level recovery mechanism to maintain the atomicity of a multi database transaction.

– A global recovery manager, or coordinator.

– The local recovery managers.

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 localeffect is done by using the log at each participating database

**Reference**

1. “Database Recovery”

<http://www.csee.umbc.edu/portal/help/oracle8/server.815/a67781/c28recov.htm>

1. “Restore and recovery overview”

<http://msdn.microsoft.com/en-us/library/ms191253.aspx>

1. “DBMS System”

<http://webdocs.cs.ualberta.ca/~zaiane/courses/cmput391/slides/L9-391-04.pdf>

1. “Types of database failures”

<http://www.saraie.net/index.php?option=com_content&view=article&id=45:types-of-database-failures-&catid=5:recovery-manager-rman&Itemid=19>

1. “Database data recovery”

<http://www.tutorialspoint.com/dbms/dbms_data_recovery.htm>

1. “DBMS storage system”

<http://www.tutorialspoint.com/dbms/dbms_storage_system.htm>

1. “The storage hierarchy”

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

1. “Buffer management in a database”

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