ST. XAVIER’S COLLEGE

**Maitighar,Kathmandu**

**(Affiliated to Tribhuvan University)**



**Database Management System**

**Lab Assignment #9**

**Submitted By**

Aashish Raj Shrestha

B.Sc. CSIT

Year II/IV Semester

013BSCIT002

**Submitted To**

Er. Sanjay Kumar Yadav

Lecturer,

Department of Computer Science

St. Xavier’s College

Maitighar, Kathmandu

**Submitted On**

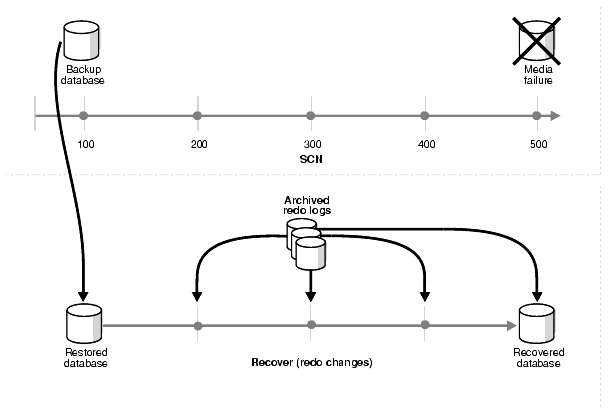
October 4, 2015

**Database Recovery**

Reconstructing the contents of all or part of a database from a backup typically involves two phases: retrieving a copy of the data file from a backup, and reapplying changes to the file since the backup from the archived and online redo logs, to bring the database to a desired SCN since the backup (usually, the present).

To **restore** a data file or control file from backup is to retrieve the file onto disk from a backup location on tape, disk or other media, and make it available to the database server.

To **recover** a data file (also called **performing recovery** on a data file), is to take a restored copy of the data file and apply to it changes recorded in the database's redo logs. To recover a whole database is to perform recovery on each of its data files [1].



*Figure: Restoring and Recovering a Database*

1. **Purpose of Data Recovery**

Understanding the concepts of database recovery requires a clear comprehension of two factors:

* 1. The type of failure the database has to cope with
  2. The notion of consistency that is assumed as a criterion for describing the state to be re-established [2]

A major responsibility of the database administrator is to prepare for the possibility of hardware, software, network, process, or system failure. If such a failure affects the operation of a database system, you must usually recover the database and return to normal operation as quickly as possible. Recovery should protect the database and associated users from unnecessary problems and avoid or reduce the possibility of having to duplicate work manually.

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 [3].

Its Purpose:

1. To bring the database into the last consistent state, this existed prior to the failure.
2. 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**

Several problems can halt the normal operation of an Oracle database or affect database I/O to disk. The following sections describe the most common types. For some of these problems, recovery is automatic and requires little or no action on the part of the database user or database administrator.

**User Error**

A database administrator can do little to prevent user errors (for example, accidentally dropping a table). Usually, user error can be reduced by increased training on database and application principles. Furthermore, by planning an effective recovery scheme ahead of time, the administrator can ease the work necessary to recover from many types of user errors.

**Statement Failure**

Statement failure occurs when there is a logical failure in the handling of a statement in an Oracle program. For example, assume all extents of a table (in other words, the number of extents specified in the MAXEXTENTS parameter of the CREATE TABLE statement) are allocated, and are completely filled with data; the table is absolutely full. A valid INSERT statement cannot insert a row because there is no space available. Therefore, if issued, the statement fails.

If a statement failure occurs, the Oracle software or operating system returns an error code or message. A statement failure usually requires no action or recovery steps; Oracle automatically corrects for statement failure by rolling back the effects (if any) of the statement and returning control to the application. The user can simply re-execute the statement after correcting the problem indicated by the error message.

**Process Failure**

A process failure is a failure in a user, server, or background process of a database instance (for example, an abnormal disconnect or process termination). When a process failure occurs, the failed subordinate process cannot continue work, although the other processes of the database instance can continue.

The Oracle background process PMON detects aborted Oracle processes. If the aborted process is a user or server process, PMON resolves the failure by rolling back the current transaction of the aborted process and releasing any resources that this process was using. Recovery of the failed user or server process is automatic. If the aborted process is a background process, the instance usually cannot continue to function correctly. Therefore, you must shut down and restart the instance.

**Network Failure**

When your system uses networks (for example, local area networks, phone lines, and so on) to connect client workstations to database servers, or to connect several database servers to form a distributed database system, network failures (such as aborted phone connections or network communication software failures) can interrupt the normal operation of a database system. For example:

* A network failure might interrupt normal execution of a client application and cause a process failure to occur. In this case, the Oracle background process PMON detects and resolves the aborted server process for the disconnected user process, as described in the previous section.
* A network failure might interrupt the two-phase commit of a distributed transaction. Once the network problem is corrected, the Oracle background process RECO of each involved database server automatically resolves any distributed transactions not yet resolved at all nodes of the distributed database system.

**Database Instance Failure**

Database instance failure occurs when a problem arises that prevents an Oracle database instance (SGA and background processes) from continuing to work. An instance failure can result from a hardware problem, such as a power outage, or a software problem, such as an operating system crash. Instance failure also results when you issue a SHUTDOWN ABORT or STARTUP FORCE command.

**Recovery from Instance Failure**

Crash or instance recovery recovers a database to its transaction-consistent state just before instance failure. *Crash recovery* recovers a database in a single-instance configuration and *instance recovery* recovers a database in an Oracle Parallel Server configuration.

Recovery from instance failure is automatic. For example, when using the Oracle Parallel Server, another instance performs instance recovery for the failed instance. In single-instance configurations, Oracle perform crash recovery for a database when the database is restarted (mounted and opened to a new instance). The transition from a mounted state to an open state automatically triggers crash recovery, if necessary.

Crash or instance recovery consists of the following steps:

1. Rolling forward to recover data that has not been recorded in the data files yet has been recorded in the online redo log, including the contents of rollback segments. This is called *cache recovery*.
2. Opening the database. Instead of waiting for all transactions to be rolled back before making the database available, Oracle allows the database to be opened as soon as cache recovery is complete. Any data that is not locked by unrecovered transactions is immediately available.
3. Marking all transactions system-wide that were active at the time of failure as DEAD and marking the rollback segments containing these transactions as PARTLY AVAILABLE.
4. Rolling back dead transactions as part of SMON recovery. This is called *transaction recovery*.
5. Resolving any pending distributed transactions undergoing a two-phase commit at the time of the instance failure.
6. As new transactions encounter rows locked by dead transactions, they can automatically roll back the dead transaction to release the locks. If you are using Fast-Start Recovery, just the data block is immediately rolled back, as opposed to the entire transaction.

**Media (Disk) Failure**

An error can arise when trying to write or read a file that is required to operate an Oracle database. This occurrence is called *media failure* because there is a physical problem reading or writing to files on the storage medium.

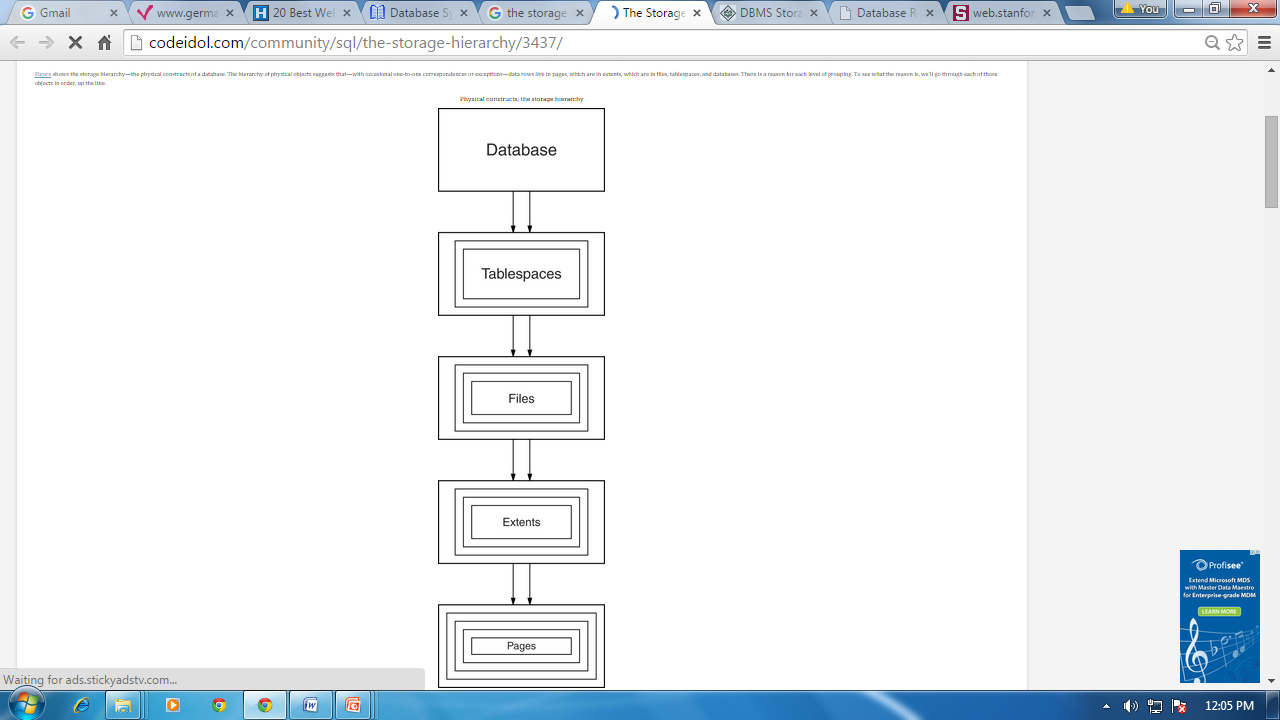
A common example of media failure is a disk head crash, which causes the loss of all files on a disk drive. All files associated with a database are vulnerable to a disk crash, including data files, online redo log files, and control files.

The appropriate recovery from a media failure depends on the files affected [3].

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**

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.

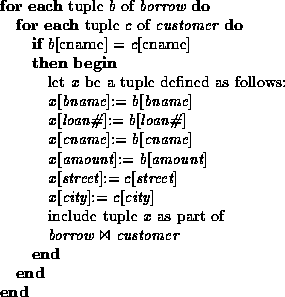
**Therefore:** if we need space, throw out the least recently referenced block. (LRU replacement scheme)

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

Consider processing of the relational algebra expression

http://www.cs.sfu.ca/CourseCentral/354/zaiane/material/notes/Chap7/_6746_displaymath453.gif

Further, assume the strategy to process this request is given by the following pseudo-code:



Assume that the two relations in this example are stored in different files.

Once a tuple of *borrow* has been processed it is not needed again.

Therefore, once processing of an entire block of tuples is finished, that block is not needed in MM.

Note that this block has been used **very** recently.

Buffer manager should free the space occupied by a borrow block as soon as it is processed.

This strategy is called **toss-immediate**.

Consider blocks containing *customer* tuples.

Every block of *customer* tuples must be examined once for every tuple of the *borrow* relation.

When processing of a *customer* block is completed, it will not be used again until all other *customer* blocks have been processed.

This means the most recently used (MRU) block will be the last block to be re-referenced, and the least recently used will be referenced next.

This is the opposite of LRU assumptions.

So for inner block, use MRU strategy - if a customer block must be removed from the buffer, choose MRU block.

For MRU strategy, the system must **pin** the *customer* block currently being processed until the last tuple has been processed.

Then it may be unpinned, becoming the most recently used block.

The buffer manager may also use statistical information regarding the probability that a request will reference a particular relation.

The data dictionary is the most frequently-used part of the database.

It should, therefore, not be removed from MM unless necessary.

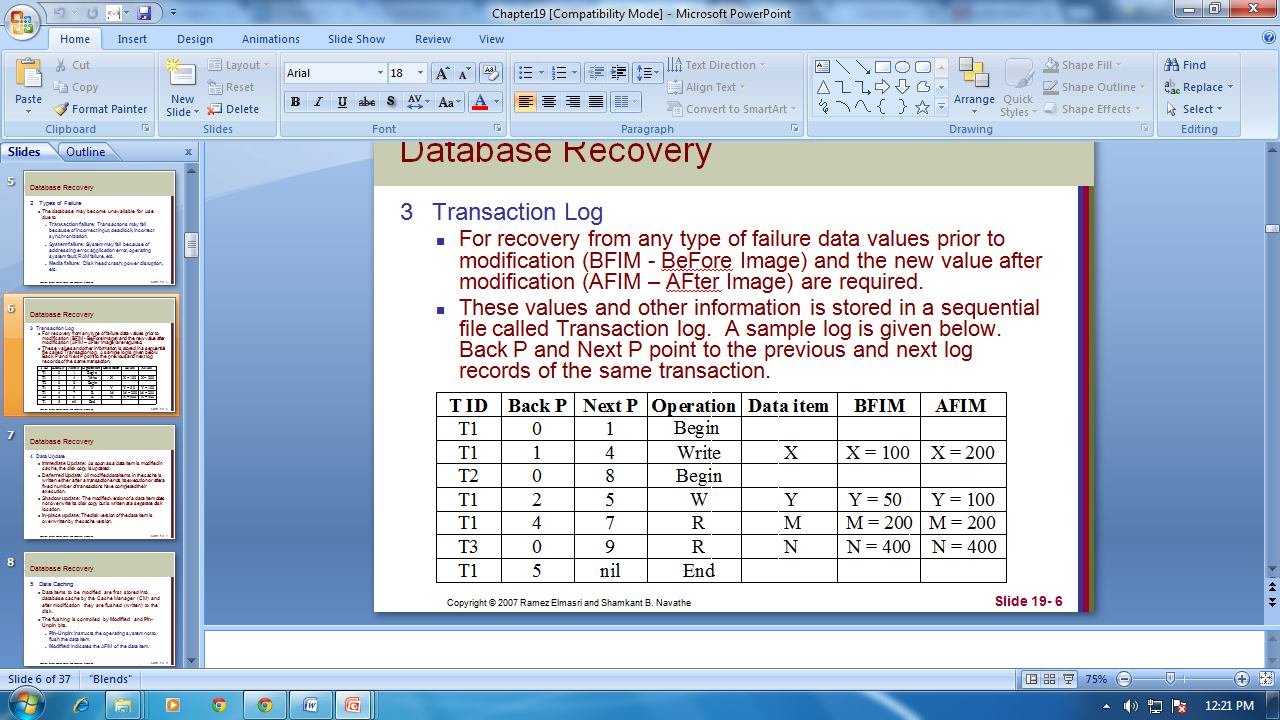
File indices (Chapter 8) are also frequently used, and should generally be in MM. No single strategy is known that handles all possible scenarios well. Many database systems use LRU, with all its faults.

Other factors enter with concurrent users, where requests may be delayed to ensure integrity of the database.

1. **Transaction Log**

For recovery from any type of failure data values prior to modification, (BFIM - BeFore Image) and the new value after modification (AFIM – AFter Image) are required.

These values and other information is stored in a sequential file called Transaction log. A sample log is given below. Back P and Next P point to the previous and next log records of the same transaction.



1. **Data Updates**
   1. **Immediate Update**: As soon as a data item is modified in cache, the disk copy is updated.
   2. **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.
   3. **Shadow update**: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.
   4. **In-place update**: The disk version of the data item is overwritten by the cache version.
2. **Data Caching**
   1. 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.
   2. The flushing is controlled by **Modified** and **Pin-Unpin** bits.
      1. **Pin-Unpin**: Instructs the operating system not to flush the data item.
      2. **Modified**: Indicates the AFIM of the data item.
3. **Transaction Roll-back (Undo) and Roll-forward (Redo)**
   1. To maintain atomicity, a transaction’s operations are redone or undone.
      1. **Undo**: Restore all BFIMs on to disk (Remove all AFIMs).
      2. **Redo**: Restore all AFIMs on to disk.
   2. 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.
4. **Check Pointing, Shadow Paging**

**Check Pointing**

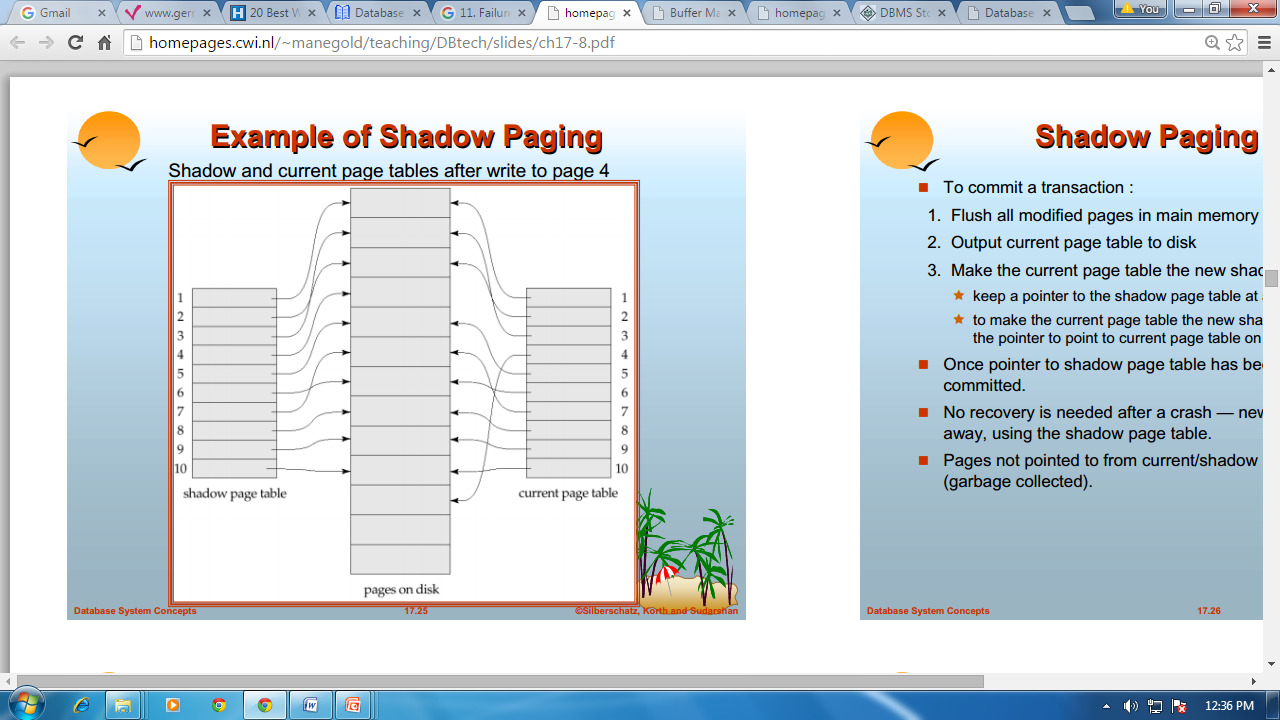
Time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery. The following steps define a checkpoint operation:

* 1. Suspend execution of transactions temporarily.
  2. Force writes modified buffer data to disk.
  3. Write a [checkpoint] record to the log, save the log to disk.
  4. Resume normal transaction execution.

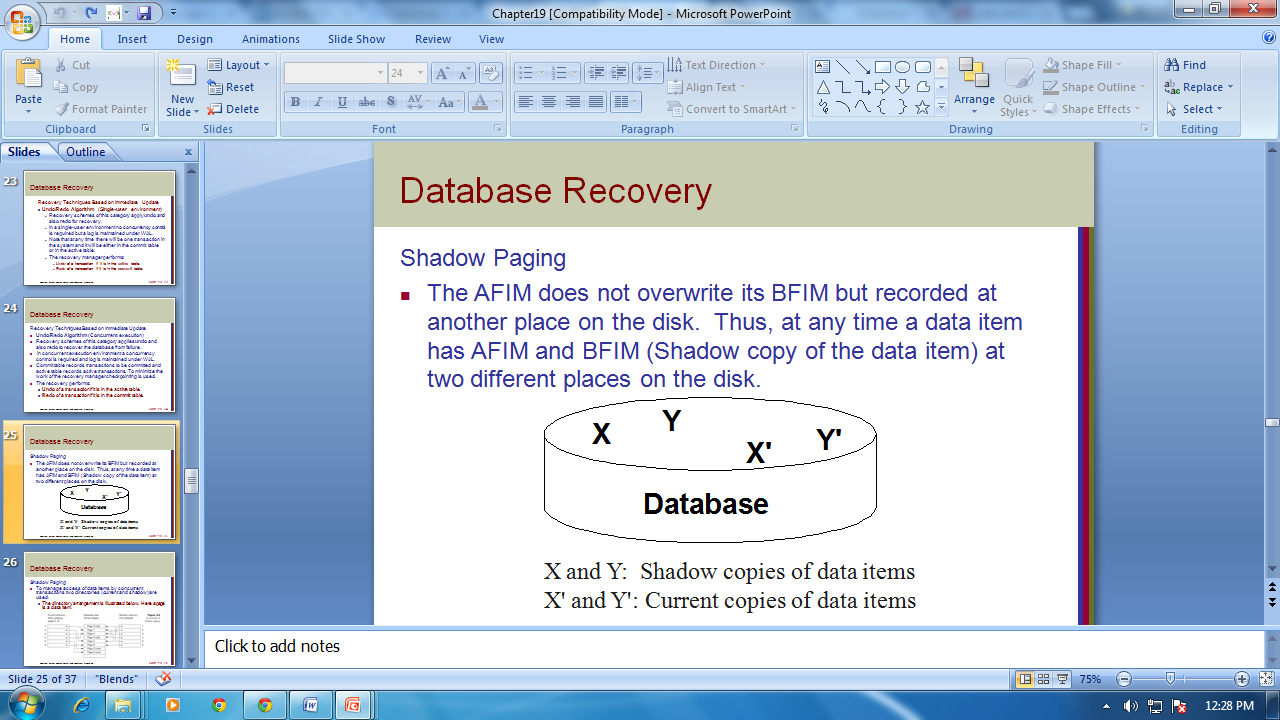
During recovery redo or undo is required to transactions appearing after [checkpoint] record.

**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. . The current page table is then made to point to the copy. The update is performed on the copy.



The AFIM does not overwrite its BFIM but recorded at another place on the disk. Thus, at any time a data item has AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.



To commit a transaction:

1. Flush all modified pages in main memory to disk

2. Output current page table to disk

3. Make the current page table the new shadow page table, as follows:

Keep a pointer to the shadow page table at a fixed (known) location on disk. To make the current page table the new shadow page table, simply update the pointer to point to current page table on disk. Once pointer to shadow page table has been written, transaction is committed. . No recovery is needed after a crash — new transactions can start right away, using the shadow page table. Pages not pointed to from current/shadow page table should be freed (garbage collected).

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

**Write-Ahead Logging**

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

* 1. **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).
  2. **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 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 (MDBS)**

An MDBS integrates a set of autonomous and heterogeneous local DBSs. In turn, each local DBS consists of a local DBMS and a database. Users can access information from multiple sources through global transactions. Operations belonging to global transactions are executed by local DBMSs. Besides global transactions, there exist local transactions in a multi-database environment. Local transactions result from the execution of local applications. Such applications are typically pre-existing with regard to the integration realized by MDBSs.

A computer system is subject to failures. Such failures may provoke loss of information. Hence, MDBSs should be able to react in failure situations in order to restore the multi-database to a consistent state, without human intervention, that is, automatically. However, ensuring reliability in MDBSs is a very complex task. First of all, more types of failures may occur in MDBSs (e.g., a communication failure which can isolate a local DBS from the MDBS) than in centralized DBMSs.

Second, in MDBSs, there is a tradeoff between preserving local autonomy and providing an efficient global recovery mechanism. Since several existing DBMSs do not supports the two-phase commit (2PC) protocol, we have to assume that such a protocol cannot be used when designing a recovery mechanism for MDBSs.

In order to make transaction processing in MDBSs resilient to failures, two types of protocols are required. One type of protocol should enforce that, when a given global transaction completes its execution, it has the same state (committed or aborted) at every site it has run. Such protocols ensure what we call commit atomicity. They are called commit protocols. The other type of protocols, denoted recovery protocols, determines the actions to be triggered after failures in a multi-database environment.

**Multi Database Recovery Procedure:**

No global procedure is needed for sub transaction failures. If a sub transaction of a global transaction G fails for any reason before G is globally committed, the multi-database 2PC assures the atomicity of G by aborting its sub transactions. On the other hand, if G is globally committed, their sub transactions are not subject to sub transaction failures. Database systems do not unilaterally abort any transaction after it has entered its (possibly simulated) prepared to commit state.

To describe multi-database from site failures, we assume that a global recovery process is associated with the MDBS and each of the participating LDBS. The global recovery process at the MDBS sit can access the global log, while a global recovery process at a LDBS site has access to the local agent logs. Global recovery processes execute the recovery procedures presented next to ReDo the sub transactions of globally committed multi-database transactions which were interrupted by a site failure. Furthermore, global recovery processes at the LDBS sites may send messages to the global recovery process at the MDBS site inquiring the state of global transactions. The following two phase scheme restores multi-database consistency in the event of *LDBS site failure.*

**Local Recovery Phase:** When the LDBS host computer comes up after a site failure, the DBA of the local DBMS performs the following actions (either interactively or through a startup procedure executing under his ID):

* 1. Starts the DBMS
  2. Invokes the local DBMS recovery
  3. (if necessary) revokes access to the local database from all users

At the end of the local recovery phase all locally uncommitted transaction are rolled back and the updates of all locally committed transactions are installed in the database. Both local and global transactions are prevented from accessing the LDBS.

**Multi-database Recovery Phase:** On completion of local recovery, the local database is consistent, as far as the local DBMS is concerned. However, it may be inconsistent from the point of view of the MDBS. To restore multi-database consistency the DBA performs the following actions:

1. Executes the MDBS restart procedure described next
2. Grant access to the local database to all users

REFERENCES:

[1] “The Database Recovery Process: Basic Concepts”. *Oracle Help Center.* Link: http://docs.oracle.com/cd/B19306\_01/backup.102/b14192/intro003.htm

[2] Fachbereich Informatik, University of Kaiserslautern, West Germany Link: http://web.stanford.edu/class/cs340v/papers/recovery.pdf

[3] Oracle8i Concepts. Link: http://www.csee.umbc.edu/portal/help/oracle8/server.815/a67781/c28recov.htm