**ST.XAVIER,S COLLEGE**

**Maitighar, Kathmandu**



**Database Management System Assignment #6**

Yub Raj Basnet

013BScCSIT048 (4th Semester)

**Submitted to**

|  |  |
| --- | --- |
| Er. Sanjya Kumar Yadav  (Lecturer, St.Xavier’s College ) |  |

1. **Database Recovery**

**Purpose of Data recovery.**

As a backup administrator, your principal duty is to devise, implement, and manage a backup and recovery strategy. In general, the purpose of a backup and recovery strategy is to protect the database against data loss and reconstruct the database after data loss. Typically, backup administration tasks include the following:

* Planning and testing responses to different kinds of failures
* Configuring the database environment for backup and recovery
* Setting up a backup schedule
* Monitoring the backup and recovery environment
* Troubleshooting backup problems
* Recovering from data loss if the need arises

**Types of Database Failures:**

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.

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).

The figure in the side 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**

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.

A buffer is an 8-KB page in memory, the same size as a data or index page. Thus, the buffer cache is divided into 8-KB pages. The buffer manager manages the functions for reading data or index pages from the database disk files into the buffer cache and writing modified pages back to disk. A page remains in the buffer cache until the buffer manager needs the buffer area to read in more data. Data is written back to disk only if it is modified. Data in the buffer cache can be modified multiple times before being written back to disk.

The buffer manager only performs reads and writes to the database. Other file and database operations such as open, close, extend, and shrink are performed by the database manager and file manager components.

1. **Transaction Log**

Transaction logs are a vital yet often overlooked component of database architecture. They are often forgotten because they are not something actively maintained like the schema contained within a database. In this article we’ll examine how transaction logs are used in Microsoft SQL Server, maintenance and potential problems with them, how they can be used to restore a database, and finally, optimizing them for performance.

Transaction logs can present problems because they are often forgotten about until an issue occurs. The log continues to grow as operations are performed within the database. While the log continues to grow, the available disk space decreases. Unless routine action is taken to prevent it, the transaction log will eventually consume all available space allocated to it. If the log is configured to grow indefinitely as is the default, it will grow to consume all available physical disk space where it is stored. Either scenario causes the database to stop functioning.

A technique often used to perform recovery is the transaction log or journal.

* records information about the progress of transactions in a log since the last consistent state.
* the database therefore knows the state of the database before and after each transaction.
* every so often database is returned to a consistent state and the log may be truncated to remove committed transactions.
* when the database is returned to a consistent state the process is often referred to as `checkpointing'.
  1. **Data Updates**
  2. **Data Caching**

Caching in applications is going to vary from one platform to another. An object cache is a mechanism that you can use to put commonly used objects into memory so that you don't need to pay the cost to retrieve the data and recreate them.

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.

In three tier architecture, the application tier and data tier can be in different hosts. Throughput of the application is affected by the network speed. This network overhead will be avoided by having the database at the application tier. As commercial databases are heavy weight, it is not practically feasible to have the application and the database at the same host. There are lot of light-weight databases available on the market, which can be used to cache data from the commercial databases. The benefits of data caching are as follows:

1. Scalability: distribute query workload from backend to multiple cheap front-end systems.
2. 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 is not.
3. Availability: by continued service for applications that depend only on cached tables even if the backend server is unavailable.
4. 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. **Transition Roll Back (Undo) and Roll Forward**

In [database](https://en.wikipedia.org/wiki/Database) technologies, a rollback is an operation which returns the database to some previous state. Rollbacks are important for database [integrity](https://en.wikipedia.org/wiki/Data_integrity), because they mean that the database can be restored to a clean copy even after erroneous operations are performed. They are crucial for recovering from database server crashes; by rolling back any [transaction](https://en.wikipedia.org/wiki/Database_transaction) which was active at the time of the crash, the database is restored to a consistent state.

The rollback feature is usually implemented with a [transaction log](https://en.wikipedia.org/wiki/Database_log), but can also be implemented via [multi version concurrency control](https://en.wikipedia.org/wiki/Multiversion_concurrency_control).

In SQL, ROLLBACK is a command that causes all data changes since the last BEGIN WORK, or START TRANSACTION to be discarded by the relational database management systems (RDBMS), so that the state of the data is "rolled back" to the way it was before those changes were made.

A ROLLBACK statement will also release any existing savepoints that may be in use.

In most SQL dialects, ROLLBACKs are connection specific. This means that if two connections are made to the same database, a ROLLBACK made in one connection will not affect any other connections. This is vital for proper concurrency.

The Roll forward is redoing the changes made by a transaction that is after the committed transaction and to over-write the changed value once again to ensure the consistency.

1. **Check Pointing, Shadow Paging**

**Check Pointing**

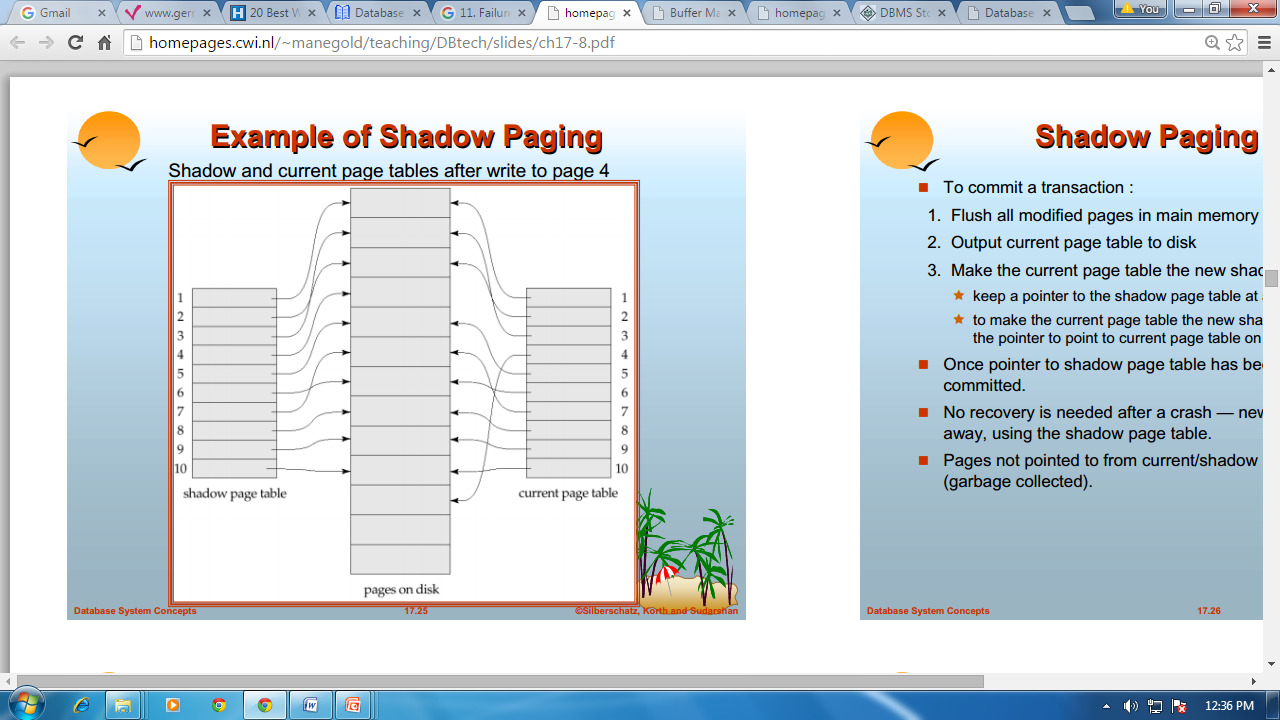
Time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery. The following steps defines a checkpoint operation:

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

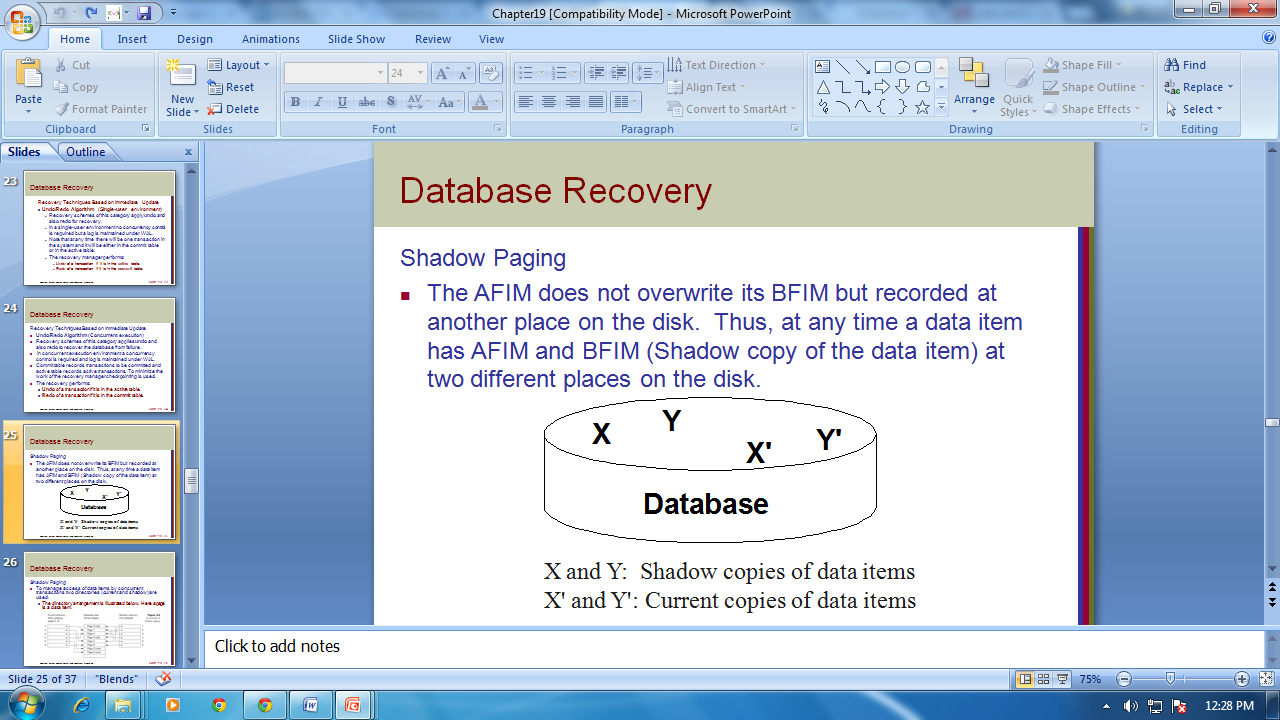
During recovery redo or undo is required to transactions appearing after [checkpoint] record.

**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

**WAL: Write ahead Logging Protocol**

Write-ahead logging (WAL) is a family of techniques for providing [atomicity](https://en.wikipedia.org/wiki/Atomic_(computer_science)) and [durability](https://en.wikipedia.org/wiki/Durability_(database_systems)) (two of the [ACID](https://en.wikipedia.org/wiki/ACID) properties) in [database systems](https://en.wikipedia.org/wiki/Database_system). In a system using WAL, all modifications are written to a [log](https://en.wikipedia.org/wiki/Database_log) before they are applied. Usually both redo and undo information is stored in the log.

The purpose of this can be illustrated by an example. Imagine a program that is in the middle of performing some operation when the machine it is running on loses power. Upon restart, that program might well need to know whether the operation it was performing succeeded, half-succeeded, or failed. If a write-ahead log is used, the program can check this log and compare what it was supposed to be doing when it unexpectedly lost power to what was actually done. On the basis of this comparison, the program could decide to undo what it had started, complete what it had started, or keep things as they are.

Before a block of data in main memory can be output to the database (in nonvolatile storage), all log records pertaining to data in that block must have been output to stable storage. This rule is called the WAL rule. Strictly speaking, the WAL rule requires only that the undo information in the log have been output to stable storage, and permits the redo information to be written later. The difference is relevant in systems where undo information and redo information are stored in separate log records. Write-ahead logging is employed to flush log records to the persistent log file before data pages are written or at commit time.

**Recovery in Multi database systems**

A multi database consists of a collection of autonomous local databases. Systems used to manage multi databases are called multi database systems (MDBSs). In such a system, global transactions are executed under the control of the MDBS. 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. 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. In this work, we will describe a commit and a recovery strategy. They can be implemented to ensure transaction processing reliability in an MDBS which does not use a 2PC protocol. The proposed commit strategy guarantees commit atomicity. In turn, the recovery strategy, denoted ReMT (Recovering Multi database Transactions), enables the MDBS to deal with the specific failures in multi database environments. Our proposal is suitable to a wide variety of multi database applications, such as CAD, CASE, GIS and WFMS.