ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**Database Management System Assignment #9**

**SUBMITTED BY:**

**Hemanchal Joshi**

**013BSCCSIT021**

**SUBMITTED TO:**

|  |  |
| --- | --- |
| **Er. Sanjay Kr. Yadav**  Lecturer |  |
| **Department of Computer Science** | |

**Submission date:** October 4, 2015

1. **DATABASE RECOVERY**

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.

**PURPOSE OF DATA RECOVERY**

There are many situations in which a transaction may not reach a commit or abort point.

* 1. An operating system crash can terminate the DBMS processes
  2. The DBMS can crash
  3. The system might lose power
  4. A disk may fail or other hardware may fail.
  5. Human error can result in deletion of critical data.

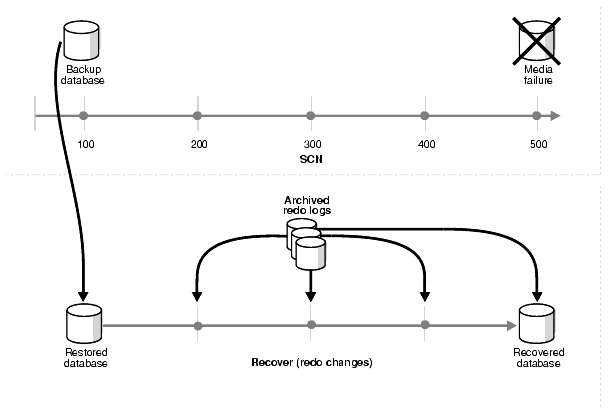
In any of these situations, data in the database may become inconsistent or lost.

For example, if a transaction has completed 30 out of 40 scheduled writes to the database when the DBMS crashes, then the database may be in an inconsistent state as only part of the transaction’s work was completed.

Database Recovery is the process of restoring the database and the data to a consistent state. This may include restoring lost data up to the point of the event (e.g. system crash).

The main purpose of data recovery is:

* To restore a datafile 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 datafile (also called performing recovery on a datafile), is to take a restored copy of the datafile 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 datafiles.

****

**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.

**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 datafiles, online redo log files, and control files.

**User Error**

A database administrator can do little to prevent user errors such as 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 such as 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 such as local area networks and phone lines 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.

**Database Instance Failure**

Database instance failure occurs when a problem arises that prevents an Oracle database instance 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 statement.

**THE STORE 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.

**BUFFER MANAGEMENT**

The interval between SQL Server startup and when the buffer cache obtains its memory target is called ramp-up. During this time, read requests fill the buffers as needed. For example, a single-page read request fills a single buffer page. This means the ramp-up depends on the number and type of client requests. Ramp-up is expedited by transforming single-page read requests into aligned eight-page requests. This allows the ramp-up to finish much faster, especially on machines with a lot of memory.

Because the buffer manager uses most of the memory in the SQL Server process, it cooperates with the memory manager to allow other components to use its buffers. The buffer manager interacts primarily with the following components:

* Resource manager to control overall memory usage and, in 32-bit platforms, to control address space usage.
* Database manager and the SQL Server Operating System (SQLOS) for low-level file I/O operations.
* Log manager for write-ahead logging.

The buffer manager supports the following features:

* The buffer manager is non-uniform memory access (NUMA) aware. Buffer cache pages are distributed across hardware NUMA nodes, which allows a thread to access a buffer page that is allocated on the local NUMA node rather than from foreign memory. For more information, see [How SQL Server Supports NUMA](https://technet.microsoft.com/en-us/library/ms180954(v=sql.105).aspx).
* The buffer manager supports Hot Add Memory, which allows users to add physical memory without restarting the server. For more information, see [Hot Add Memory](https://technet.microsoft.com/en-us/library/ms175490(v=sql.105).aspx).
* The buffer manager supports large pages on 64-bit platforms. The page size is specific to the version of Windows. For more information, see the Windows documentation.
* The buffer manager provides additional diagnostics that are exposed through dynamic management views.

**TRANSACTION LOG**

To apply a transaction log backup to a database, the following must be true:

* Before the most recent full or differential database backup was created, the database must have been using the full recovery model or bulk-logged recovery model.
* The restore sequence must have specified WITH NORECOVERY when restoring earlier backups.
* Log backups must be applied in the sequence in which they were created, without any gaps in the log chain. Except for the last log backup, you must use WITH NORECOVERY, as follows:

**RESTORE LOG <database\_name> FROM <backup\_device> WITH NORECOVERY;**

**DATA UPDATES**

The MODIFY statement extends the capabilities of the DATA step by enabling you to modify data accessed by a view descriptor or a SAS data file without creating an additional copy of the data. To use the MODIFY statement with a view descriptor, you must have UPDATE privileges on the view's underlying DBMS table.

A DBMS trigger may prevent you from modifying observations in a DBMS table. Refer to your DBMS documentation to see if triggers are used in your DBMS.

The UPDATE statement is used to update existing records in a table.

**SQL UPDATE Syntax**

UPDATE *table\_name*  
SET *column1*=*value1*,*column2*=*value2*,...  
WHERE *some\_column*=*some\_value*;

**DATA CACHING**

The cache is the area of RAM that contains data. When you write data to disk, the lazy-write technique in Windows 2000 indicates that the data is written when, in fact, it is still in the cache. There can also be cache memory on the disk controller or on the disk itself. The following information will help you decide whether you want to enable the disk or controller cache:

* Turning on write caching improves disk performance, particularly if the disk is being heavily written to.
* Control of the write-back cache is a firmware function provided by the disk manufacturer. See the documentation supplied with the disk or disk controller. You cannot configure the write-back cache from Windows 2000.
* Write caching does not impact the reliability of the file system's own metadata. NTFS instructs the disk device driver to ensure that metadata writes get written regardless of whether write caching is enabled. Non-metadata is written to the disk normally, so such data can be cached.
* Read caching in the disk has no impact on file system reliability.

**TRANSACTION: ROLL BACK AND ROLL FORWARD**

**Roll forward:** 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. Roll Forward occurs when the database restarts after an abnormal shutdown. It’s a process of going to the log files and applying changes from the log files to the underlying database. In the case where the underlying tables have been restored from an old backup this can involve millions of updates and take several hours.

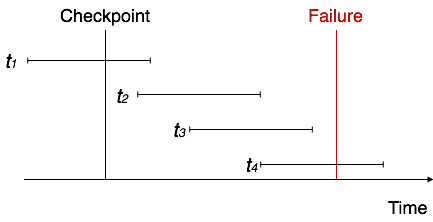
**Roll back:** The Rollback transaction is a transaction which rolls back the transaction to the beginning of the transaction (Rollback *Transaction\_name*). It is possible to use before Commit transaction. Roll Back is the process of undoing changes and reverting to a previous state. This usually occurs either on request when a program detects some logical error and decides the transaction should not take place, or, when the DBMS loses contact with the program before an explicit "COMMIT" has been requested.

**CHECK POINTING**

Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all. Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

**Recovery**

When a system with concurrent transactions crashes and recovers, it behaves in the following manner :-



* The recovery system reads the logs backwards from the end to the last checkpoint.
* It maintains two lists, an undo-list and a redo-list.
* If the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>, it puts the transaction in the redo-list.
* If the recovery system sees a log with <Tn, Start> but no commit or abort log found, it puts the transaction in undo-list.

All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

## SHADOW PAGING

It is inconvenient to maintain logs of all transactions fro the purposes of recovery. An alternative is to use a system of shadow paging. This is where the database is divided into pages that may be stored in any order on the disk. In order to identify the location of any given page, we use something called a page table.

During the life of a transaction two page tables are maintained, one called a shadow page table and current page table. When a transaction begins both of these page tables point to the same locations (are identical). During the lifetime of a transaction the shadow page table doesn't change at all. However during the lifetime of a transaction changes may be made update values etc. So whenever we update a page in the database we always write the updated page to a new location. This means that when we then update our current page table to reflect the changes that have been made.

**RECOVERY SCHEMA (WAL: WRITE AHEAD LOGGING PROTOCOL)**

In computer science, write-ahead logging (WAL) is a family of techniques for providing atomicity and durability (two of the ACID properties) in database systems. In a system using WAL, all modifications are written to a 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.

WAL allows updates of a database to be done in-place. Another way to implement atomic updates is with shadow paging, which is not in-place. The main advantage of doing updates in-place is that it reduces the need to modify indexes and block lists.

* 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.

**FAILURE WITH LOSS OF NON-VOLATILE STORAGE**

A catastrophic failure is one where a stable, secondary storage device gets corrupt. With the storage device, all the valuable data that is stored inside is lost. We have two different strategies to recover data from such a catastrophic failure −Remote backup &minu; Here a backup copy of the database is stored at a remote location from where it can be restored in case of a catastrophe.

Alternatively, database backups can be taken on magnetic tapes and stored at a safer place. This backup can later be transferred onto a freshly installed database to bring it to the point of backup. Grown-up databases are too bulky to be frequently backed up. In such cases, we have techniques where we can restore a database just by looking at its logs. So, all that we need to do here is to take a backup of all the logs at frequent intervals of time. The database can be backed up once a week, and the logs being very small can be backed up every day or as frequently as possible.

**RECOVERY IN MULTIDATABASE SYSTEM**

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 multi database 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.