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** | |

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.

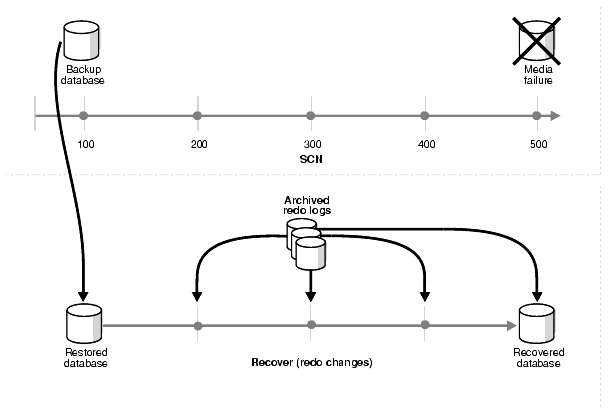
In any of these situations, data in the database may become inconsistent or lost.

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.

We have already described the storage system. In brief, the storage structure can be divided into two categories −

* **Volatile storage** − As the name suggests, a volatile storage cannot survive system crashes. Volatile storage devices are placed very close to the CPU; normally they are embedded onto the chipset itself. For example, main memory and cache memory are examples of volatile storage. They are fast but can store only a small amount of information.
* **Non-volatile storage** − These memories are made to survive system crashes. They are huge in data storage capacity, but slower in accessibility. Examples may include hard-disks, magnetic tapes, flash memory, and non-volatile (battery backed up) RAM.

**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). To understand how pages of memory from the buffer cache are assigned when using NUMA, see [Growing and Shrinking the Buffer Pool Under NUMA](https://technet.microsoft.com/en-us/library/ms345403(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 dynamic memory allocation on Microsoft Windows XP 32-bit and Windows 2003 32-bit platforms when AWE is enabled. Dynamic memory allocation allows the Database Engine to efficiently acquire and release memory in the buffer cache to support the current workload. For more information see, [Dynamic Memory Management](https://technet.microsoft.com/en-us/library/ms178145(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. You can use these views to monitor a variety of operating system resources that are specific to SQL Server. For example, you can use the [sys.dm\_os\_buffer\_descriptors](https://technet.microsoft.com/en-us/library/ms173442(v=sql.105).aspx) view to monitor the pages in the buffer cache. For more information, see [SQL Server Operating System Related Dynamic Management Views (Transact-SQL)](https://technet.microsoft.com/en-us/library/ms176083(v=sql.105).aspx).

**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;**

* When applying the last log backup, you can do either of the following:
  + Recover the database as part of the last BACKUP LOG statement:
  1. **DATA UPDATES**
  2. **DATA CACHING**