ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**Database Management System**

**Theory Assignment**

**SUBMITTED BY:**

Samrat Maskey

013BSCCSIT034

**SUBMITTED TO**

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

**An Introduction to 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**

* To bring the database into the last consistent state, which existed prior to the failure [5].
* To preserve transaction properties:
  + Atomicity
  + Consistency
  + Isolation
  + Durability

**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.

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

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

**Transactions roll back and roll forward**

Database buffers in the buffer cache in the SGA are written to disk only when necessary, using a least-recently-used algorithm. Because of the way that the DBWn process uses this algorithm to write database buffers to datafiles, datafiles might contain some data blocks modified by uncommitted transactions and some data blocks missing changes from committed transactions.

Two potential problems can result if an instance failure occurs:

* Data blocks modified by a transaction might not be written to the datafiles at commit time and might only appear in the redo log. Therefore, the redo log contains changes that must be reapplied to the database during recovery.
* After the roll forward phase, the datafiles may contain changes that had not been committed at the time of the failure. These uncommitted changes must be rolled back to ensure transactional consistency. These changes were either saved to the datafiles before the failure, or introduced during the roll forward phase.

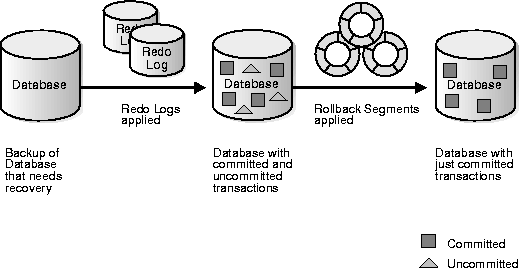
To solve this dilemma, two separate steps are generally used by Oracle for a successful recovery of a system failure: rolling forward with the redo log (cache recovery) and rolling back with the rollback segments (transaction recovery).

**Rollback:** The Rollback transaction is a transaction which rolls back the transaction to the beginning of the transaction. The transaction can be rolled back completely by specifying the transaction name in the Rollback statement or to cancel any changes to a database during current transaction. It is permissible to use before Commit transaction [4].

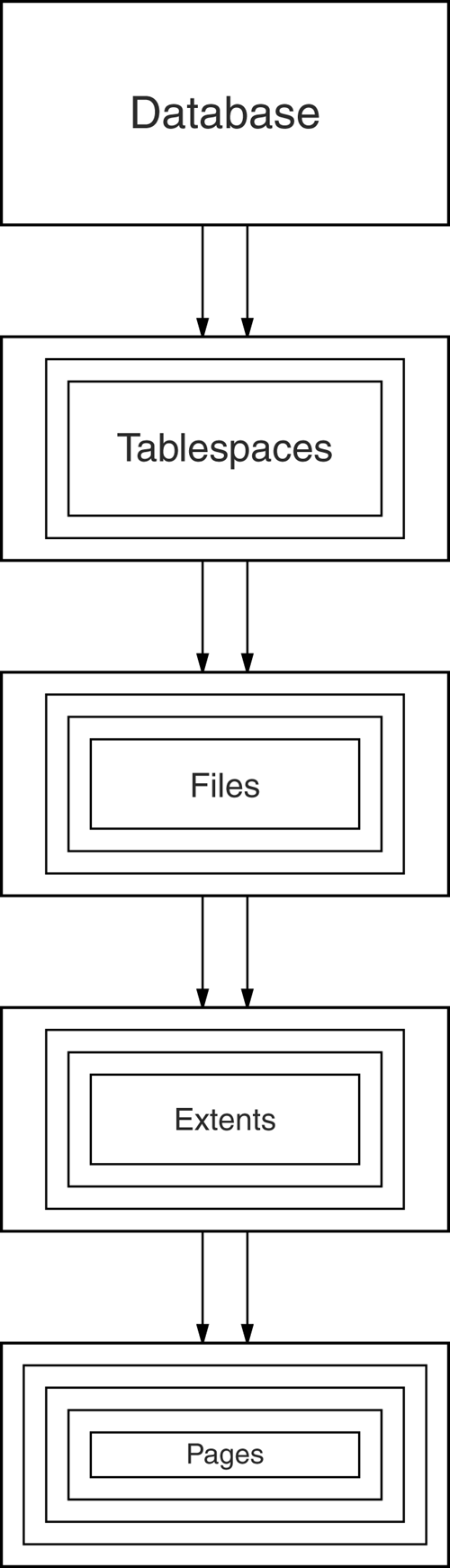
**Roll forward:** Recovering a database by applying different transactions that recorded in the database log files. It is nothing but re-doing the changes made by a transaction i.e. after the committed transaction and to over write the changed value again to ensure consistency [4].

[Figure 32-1](http://www.csee.umbc.edu/portal/help/oracle8/server.815/a67781/c28recov.htm#3868) illustrates rolling forward and rolling back, the two steps necessary to recover from any type of system failure.

**Figure 32-1 Basic Recovery Steps: Rolling Forward and Rolling Back**



**The Storage Hierarchy:**

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.

Pages

Depending on the DBMS, a page is also called a data block, a [block](http://codeidol.com/community/sql/the-storage-hierarchy/3437/app02.html#gloss19), a [blocking unit](http://codeidol.com/community/sql/the-storage-hierarchy/3437/app02.html#gloss21), a [control interval](http://codeidol.com/community/sql/the-storage-hierarchy/3437/app02.html#gloss48), and a [row group](http://codeidol.com/community/sql/the-storage-hierarchy/3437/app02.html#gloss197).

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. They are:

* 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.
* The choice of page sizes is restricted to certain multiples of 1024 (1KB), in a range between 1024 and 65536—that is, between 1KB and 64KB.
* The optimum page size is related to the disk system's attributes. Smaller page sizes like 2KB were once the norm, but disks' capacity tends to increase over time, so now 8KB is reasonable, while 16KB is what we'll upgrade to soon.
* Pages contain an integral number of rows. Even for the rare DBMSs that allow large rows to overflow into later pages, the very strong recommendation is that you should avoid overflow.

**Buffer Management**

DB

MAIN MEMORY

DISK

disk page

free frame

Page Requests from Higher Levels

BUFFER POOL

choice of frame dictated

by **replacement policy**

**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.
  + If requests can be predicted (e.g., sequential scans), pages can be pre-fetched several pages at a time.

**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 recovery requirement 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 the transaction 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.

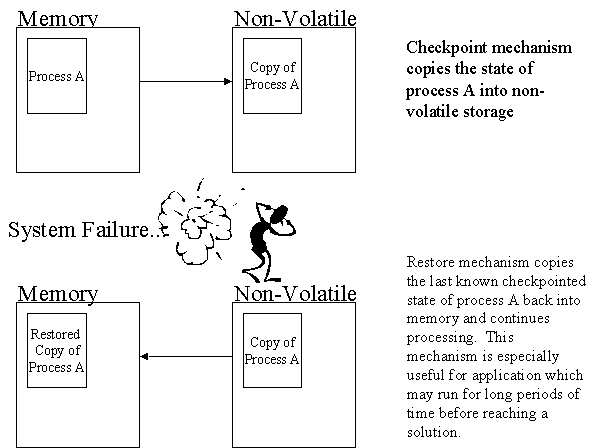
While the DBMS executes transactions that modify the database, it also automatically updates the transaction log. The transaction log stores:

* 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.

**Check pointing**

Checkpoint-Recovery is a common technique for imbuing a program or system with fault tolerant qualities, and grew from the ideas used in systems which employ transaction processing. It allows systems to recover after some fault interrupts the system, and causes the task to fail, or be aborted in some way. While many systems employ the technique to minimize lost processing time, it can be used more r from faults in a critical application or task.

The basic idea behind checkpoint-recover is the saving and restoration of system state. By saving the current state of the system periodically or before critical code sections, it provides the baseline information needed for the restoration of lost state in the event of a system failure. While the cost of checkpoint-recovery can be high, by using techniques like memory exclusion, and by designing a system to have as small a critical state as possible may minimize the cost of checkpointing enough to be useful in even cost sensitive embedded applications.[6]

broadly to tolerate and recover

|  |
| --- |
|  |

# Write-Ahead Logging (WAL)

Write-Ahead Logging (WAL) is a standard method for ensuring data integrity. A detailed description can be found in most (if not all) books about transaction processing. Briefly, WAL's central concept is that changes to data files (where tables and indexes reside) must be written only after those changes have been logged, that is, after log records describing the changes have been flushed to permanent storage. If we follow this procedure, we do not need to flush data pages to disk on every transaction commit, because we know that in the event of a crash we will be able to recover the database using the log: any changes that have not been applied to the data pages can be redone from the log records. (This is roll-forward recovery, also known as REDO.)

Using WAL results in a significantly reduced number of disk writes, because only the log file needs to be flushed to disk to guarantee that a transaction is committed, rather than every data file changed by the transaction. The log file is written sequentially, and so the cost of syncing the log is much less than the cost of flushing the data pages. This is especially true for servers handling many small transactions touching different parts of the data store. Furthermore, when the server is processing many small concurrent transactions, one fsync of the log file may suffice to commit many transactions.

By archiving the WAL data we can support reverting to any time instant covered by the available WAL data: we simply install a prior physical backup of the database, and replay the WAL log just as far as the desired time. What's more, the physical backup doesn't have to be an instantaneous snapshot of the database state — if it is made over some period of time, then replaying the WAL log for that period will fix any internal inconsistencies.[7]

**RECOVERY IN MULTIDATABASE SYSTEMS**

To maintain the atomicity of a multi database 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 multidatabase 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.

## The net effect of the two-phase commit protocol is that either all participating databases commit the effect of the transaction or none of them do. In case any of the participants—or the coordinator—fails, it is always possible to recover to a state where either the transaction is committed or it is rolled back. A failure during or before Phase 1 usually requires the transaction to be rolled back, whereas a failure during Phase 2 means that a successful transaction can recover and commit.

## Data updates

* Immediate Update: As soon as a data item is modified in cache, the disk copy is updated.
* 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.
* Shadow update: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.
* In-place update: The disk version of the data item is overwritten by the cache version.

**Data caching**

The Database Management System (DBMS) is a memory buffer which stores copies of portions of the database that the DBMS is currently using. Reading from memory is much faster than reading from the disk. The DBMS therefore returns a record more quickly if it is already stored in cache. As long as the required data is stored in cache, the data is immediately available. When the required data is not stored in cache, it must be copied from the disk and then stored in cache.

**Failure with Loss of No volatile Storage (General Concepts)**

A volatile storage like RAM stores all the active logs, disk buffers, and related data. In addition, it stores all the transactions that are being currently executed. What happens if such a volatile storage crashes abruptly? It would obviously take away all the logs and active copies of the database. It makes recovery almost impossible, as everything that is required to recover the data is lost.

* Following techniques may be adopted in case of loss of volatile storage −
* • We can have checkpoints at multiple stages so as to save the contents of the database periodically.
* • A state of active database in the volatile memory can be periodicallydumped onto a stable storage, which may also contain logs and active transactions and buffer blocks.
* • <dump> can be marked on a log file, whenever the database contents are dumped from a non-volatile memory to a stable one.

**References:**

1. http://searchdisasterrecovery.techtarget.com/definition/data-recovery
2. http://codeidol.com/community/sql/the-storage-hierarchy/3437/
3. http://ora10g.com/8\_1\_failures.html
4. http://riaddba.blogspot.com/2012/11/definition-of-rollback-and-roll-forward.html
5. people.cis.ksu.edu/~hankley/d560/Topics/Finger\_**Recovery**.ppt
6. <http://www.myreadingroom.co.in/images/stories/docs/dbms/the%20transaction%20log.pdf>
7. <http://www.doc.ic.ac.uk/~jab00/dbs/database-lectures-8.html#8.1>
8. <http://users.ece.cmu.edu/~koopman/des_s99/checkpoint/>
9. <http://www.postgresql.org/docs/9.1/static/wal-intro.html>
10. Fundamentals of Database Systems by Elmasri Navathe