**ST.XAVIER,S COLLEGE**

**Maitighar, Kathmandu**



DataBase Management System Assignment #

Urendra Khkaurel

013BScCSIT045(4th Semester)

**Submitted to**

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

**Database Recovery**

**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).

**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 failure**

* + 1. Transaction System
* Caused by errors within the transaction processes.
* Caused by failure of network or operating system or physical threats to the system as a whole.
  + 1. Media
* Failure of hard disk, out of memory errors, out of disk space errors.

**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.

**BUFFER MANAGEMENT**

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.

A hash table is used to figure out what page frame a given disk page (i.e., with a given pageId) occupies. A buffer descriptor object is associated with every page frame in the buffer pool. It contains a dirty bit, the page number, and the pin count for the page occupying that frame.

When a page is requested, the buffer manager brings it in and pins it. The buffer manager does not keep track of all the pages that have been pinned by a transaction. It is up to the various components (that call the buffer manager) to make sure that all pinned pages are subsequently unpinned.

**TRANSACTION LOG**

A transaction log (also transaction journal, database log, binary log or audit trail) is a history of actions executed by a database management system to guarantee ACID properties over crashes or hardware failures. Physically, a log is a file listing changes to the database, stored in a stable storage format.

If, after a start, the database is found in an inconsistent state or not been shut down properly, the database management system reviews the database logs for uncommitted transactions and rolls back the changes made by these transactions. Additionally, all transactions that are already committed but whose changes were not yet materialized in the database are re-applied. Both are done to ensure atomicity and durability of transactions.

The database can be modified using two approaches −

* **Deferred database modification** − All logs are written on to the stable storage and the database is updated when a transaction commits.
* **Immediate database modification** − Each log follows an actual database modification. That is, the database is modified immediately after every operation.

**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**

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.

**Benefits:**

* **Scalability**: distribute query workload from backend to multiple cheap front-end systems.
* **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 are not.
* **Availability**: by continued service for applications that depend only on cached tables even if the backend server is unavailable.
* **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

**Instance Failures**

An instance failure occurs when the instance shuts down without synchronizing all the database files to the same system change number (SCN), requiring a recovery operation the next time the instance is started. Many of the reasons for an instance failure are out of your direct control; in these situations, you can minimize the impact of these failures by tuning instance recovery.

A few causes for instance failure:

* A power outage
* A server hardware failure
* Failure of a database background process
* Emergency shutdown procedures (intentional power outage or SHUTDOWN ABORT)
* **Transactions roll back and roll forward**
* **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].
* **Recovery Schemes (WAL: Write Ahead Logging Protocol)**
* WAL protocol Write-ahead logging (WAL) is a family of techniques for providing atomicity and durability (two of the ACID properties) in database systems [9].
* 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.
* 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.
* ,  
  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 were used, the program could 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.