**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System**

**Assignment # 9**

**Submitted by:**

Biraj Poudyal  
013BSCCSIT015

**Submitted to:**

|  |  |
| --- | --- |
| **Er. Sanjay Kumar Yadav**  Lecturer  St. Xavier’s College |  |

**Date of Submission:** September 10, 2015

**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).
* Two approaches are discussed here: Manual Reprocessing and Automated Recovery.

**Manual Reprocessing**

In a Manual Reprocessing recovery approach, the database is periodically backed up (a database *save*) and all transactions applied since the last save are recorded

If the system crashes, the latest database backup set is restored and all of the transactions are re-applied (by users) to bring the database back up to the point just before the crash.

Several shortcomings to the Manual Reprocessing approach:

* 1. Time required to re-apply transactions
  2. Transactions might have other (physical) consequences
  3. Re-applying concurrent transactions in the same original sequence may not be possible.

### Automated Recovery

* As with the manual recovery approach we also make periodic backups of the database (time consuming operation).
* In the Automated Recovery approach, we introduce a Log file – this is a file separate from the data that records all of the changes made to the database by transactions.
* This transaction log Includes information helpful to the recovery process such as: A transaction identifier, the date and time, the user running the transaction, *before images* and *after images.*
* **Purpose of database 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**

When a transaction is submitted for execution ( for example when the *save* button is pressed) the system checks whether all operations involved in the transaction are successfully completed the transaction has had no effect on the database or any other transaction. If either of these checks fail then the system will generate an error message depending on the nature of the failure.

**Types of Failure**

Failures may be

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

**Reasons for Failure**

Failure may be caused by a number of things.

|  |  |
| --- | --- |
| **A System Crash** | A hardware, software or network error causes the transaction to fail. |
| **Transaction or System error** | Some operation in the transaction may cause the failure or the user may interrupt the transaction. |
| **Local Errors or Exceptions** | Conditions occur during the transaction that results in transaction cancellation. |
| **Concurrency Control Enforcement** | Several transactions may be in deadlock so the transaction may be aborted to be restarted later. |
| **Disk Failure** | Read Write error on the physical disk. |
| **Physical Problems** | This can be any range of physical problems, such as power failure, mounting wrong disk or tape by operator, wiring problems etc |
| **Catastrophe Situations** | Large scale threats to the system and the data for example fire, cyclone, security breaches etc. |

Transaction errors, system errors, system crashes, concurrency problems and local errors or exceptions are the more common causes of system failure.  The system must be able to recover from such failures without loss of data.

* **The storage 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**
  + We need to use disk storage for the database, and to transfer blocks of data between MM and disk.
  + We also want to minimize the number of such transfers, as they are time-consuming.
  + One way is to keep as many blocks as possible in MM.
  + Usually, we cannot keep all blocks in MM, so we need to manage the allocation of available MM space.
  + The **buffer** is the part of MM available for storage of **copies** of disk blocks.
  + The subsystem responsible for the allocation of buffer space is called the **buffer manager**.
  + The buffer manager handles all requests for blocks of the database.
  + If the block is already in MM, the address in MM is given to the requestor.
  + If not, the buffer manager must read the block in from disk (possibly displacing some other block if the buffer is full) and then pass the address in MM to the requestor.
* The buffer manager must use some sophisticated techniques in order to provide good service:
  + **Replacement Strategy** - When there is no room left in the buffer, some block must be removed to make way for the new one. Typical operating system memory management schemes use a ``least recently used'' (**LRU**) method. (Simply remove the block least recently referenced.) This can be improved upon for database applications.
  + **Pinned Blocks** - For the database to be able to recover from crashes, we need to restrict times when a block maybe written back to disk. A block not allowed to be written is said to be **pinned**. Many operating systems do not provide support for pinned blocks, and such a feature is essential if a database is to be ``crash resistant''.
  + **Forced Output of Blocks** - Sometimes it is necessary to write a block back to disk even though its buffer space is not needed. (Called the **forced output** of a block.) This is due to the fact that MM contents (and thus the buffer) are lost in a crash, while disk data usually survives.
* **Transaction log**

In the field of [databases](https://en.wikipedia.org/wiki/Database) in [computer science](https://en.wikipedia.org/wiki/Computer_science), a transaction log (also transaction journal, database log, binary log or audit trail) is a history of actions executed by a[database management system](https://en.wikipedia.org/wiki/Database_management_system) to guarantee [ACID](https://en.wikipedia.org/wiki/ACID) properties over [crashes](https://en.wikipedia.org/wiki/Crash_(computing)) or hardware failures. Physically, a log is a [file](https://en.wikipedia.org/wiki/Computer_file) listing changes to the database, stored in a stable storage format.

Atomicity Consistency Isolation Durability (ACID) is a concept referring to a database system’s four transaction properties: atomicity, consistency, isolation and durability.

A database guarantees the following four properties to ensure database reliability, as follows:   
 **Atomicity:** A database follows the all or nothing rule, i.e., the database considers all transaction operations as one whole unit or atom. Thus, when a database processes a transaction, it is either fully completed or not executed at all.  
  
**Consistency:** Ensures that only valid data following all rules and constraints is written in the database. When a transaction results in invalid data, the database reverts to its previous state, which abides by all customary rules and constraints.  
  
**Isolation:** Ensures that transactions are securely and independently processed at the same time without interference, but it does not ensure the order of transactions. For example, user A withdraws $100 and user B withdraws $250 from user Z’s account, which has a balance of $1000. Since both A and B draw from Z’s account, one of the users is required to wait until the other user transaction is completed, avoiding inconsistent data. If B is required to wait, then B must wait until A’s transaction is completed, and Z’s account balance changes to $900. Now, B can withdraw $250 from this $900 balance.  
  
**Durability:** In the above example, user B may withdraw $100 only after user A’s transaction is completed and is updated in the database. If the system fails before A’s transaction is logged in the database, A cannot withdraw any money, and Z’s account returns to its previous consistent state.

* **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**

A cache, in computing, is a data storing technique that provides the ability to access data or files at a higher speed.   
A cache works in both hardware and software to provide similar functionality. In its physical or hardware form, it is a small form factor of internal memory that stores instances of the most frequently executed programs in the main memory to enable faster access when they are requested by the CPU.   
A very commmon example of caching is in a Web browser, where a website's HTML, images, CSS, Javascript, etc is cached locally so that a page will load faster after its first hit.

* **Transaction roll back(undo)&roll forward**
* **Roll forward:** The Rollforward 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. 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.
* **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 . It occurs when the database restarts after an abnormal shutdown. Its 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.
* **Check pointing**

As the problems in recovery procedure as discussed

• Streamline recovery procedure by periodically performing checkpointing

1. Output all log records currently residing in main memory onto stable storage.

2. Output all modified buffer blocks to the disk.

3. Write a log record onto stable storage

During recovery we need to consider only the most recent transaction Ti that started before the checkpoint, and transactions that started after Ti.

• Scan backwards from end of log to find the most recent record

• Continue scanning backwards till a record is found.

• Need only consider the part of log following above start record. Earlier part of log can be ignored during recovery, and can be erased whenever desired.

• For all transactions (starting from Ti or later) with no , execute undo(Ti). (Done only in case of immediate modification.)

• Scanning forward in the log, for all transactions starting from Ti or later with a , execute redo(Ti).

* **Shadow paging**

Alternative to log-based recovery

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, and the update is performed on the copy

* **Recovery schema (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.
* **Failure with loss of non-volatile storage(General Concepts)**
* Periodically dump the entire content of the database to stable storage
* No transaction may be active during the dump procedure; a procedure similar to checkpointing 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. Then log is consulted and all transactions that committed since the dump are redone.
* Can be extended to allow transactions to be active during dump; known as fuzzy or online dump.
* **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 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.

* **REFERENCE:**

http://docs.oracle.com/cd/B28359\_01/backup.111/b28270/rcmintro.htm#CHDCBEDG

https://www.dlsweb.rmit.edu.au/toolbox/knowmang/content/backup/failure\_types.htm#Types

http://codeidol.com/community/sql/the-storage-hierarchy/3437/

http://www.inf.uni-konstanz.de/dbis/teaching/ss04/architektur-von-dbms/download/local/buffermanager.pdf