**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**Database Management System**

**Theory Assignment #9**

**SUBMITTED BY:**

**Siddhant Rimal**

**013BSCCSIT039**

**SUBMITTED TO**

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

**Date of Submission: October 1st 2015**

Database 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).

9.1. Purpose of Database Recovery

* + To bring the database into the last consistent state, which existed prior to the failure.
  + To preserve transaction properties (Atomicity, Consistency, Isolation and Durability).

Example:

* + - If the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have incorrect value. Thus, the database must be restored to the state before the transaction modified any of the accounts.

**9.2. Types of failure**

To see where the problem has occurred, we generalize a failure into various categories, as follows −

Transaction failure

A transaction has to abort when it fails to execute or when it reaches a point from where it can’t go any further. This is called transaction failure where only a few transactions or processes are hurt.

Reasons for a transaction failure could be −

* **Logical errors** − Where a transaction cannot complete because it has some code error or any internal error condition.
* **System errors** − Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.

System Crash

There are problems − external to the system − that may cause the system to stop abruptly and cause the system to crash. For example, interruptions in power supply may cause the failure of underlying hardware or software failure.

Examples may include operating system errors.

Disk Failure

In early days of technology evolution, it was a common problem where hard-disk drives or storage drives used to fail frequently.

Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or a part of disk storage.

9.3. The storage hierarchy

■ **primary storage:** Fastest media but volatile (cache, main  
memory).  
■ **secondary storage:** next level in hierarchy, non­volatile,  
moderately fast access time  
● also called **on­line storage**  
● E.g. flash memory, magnetic disks  
■ **tertiary storage:** lowest level in hierarchy, non­volatile, slow  
access time  
● also called **off­line storage**  
● E.g. magnetic tape, optical storage

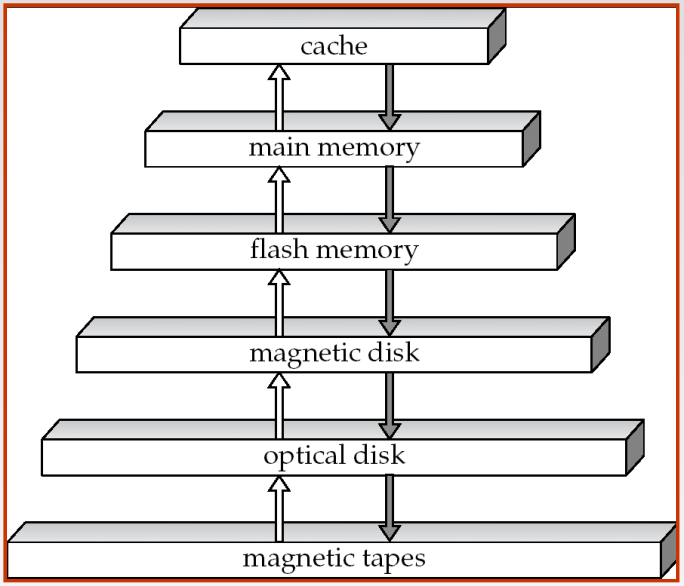


Fig: Storage Hierarchy

9.4. Buffer Management

Database buffer can be implemented either  
● in an area of real main­memory reserved for the database, or  
● in virtual memory  
■ Implementing buffer in reserved main­memory has drawbacks:  
● Memory is partitioned before­hand between database buffer and  
applications, limiting flexibility.  
● Needs may change, and although operating system knows best  
how memory should be divided up at any time, it cannot change  
the partitioning of memory.

Database buffers are generally implemented in virtual memory in spite  
of some drawbacks:  
● When operating system needs to evict a page that has been  
modified, the page is written to swap space on disk.  
● When database decides to write buffer page to disk, buffer page  
may be in swap space, and may have to be read from swap space  
on disk and output to the database on disk, resulting in extra I/O!  
4 Known as **dual paging** problem.  
● Ideally when OS needs to evict a page from the buffer, it should  
pass control to database, which in turn should  
1. Output the page to database instead of to swap space (making  
sure to output log records first), if it is modified  
2. Release the page from the buffer, for the OS to use  
Dual paging can thus be avoided, but common operating systems  
do not support such functionality.

9.5. Transaction Log  
  
In the field of databases in 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 to guarantee ACID properties over crashes or hardware failures.  
  
The transaction log supports the following operations:

* Recovery of individual transactions.
* Recovery of all incomplete transactions when SQL Server is started.
* Rolling a restored database, file, filegroup, or page forward to the point of failure.
* Supporting transactional replication.
* Supporting high availability and disaster recovery solutions: AlwaysOn Availability Groups, database mirroring, and log shipping.

9.6. Data Updates

* 1. **Immediate Update**: As soon as a data item is modified in cache, the disk copy is updated.
  2. **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.
  3. **Shadow update**: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.
  4. **In-place update**: The disk version of the data item is overwritten by the cache version.

9.7. Data Caching  
  
The database cache 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. Database Caching is the overall process of caching the ongoing data on the database cache so that it can be accessed faster.

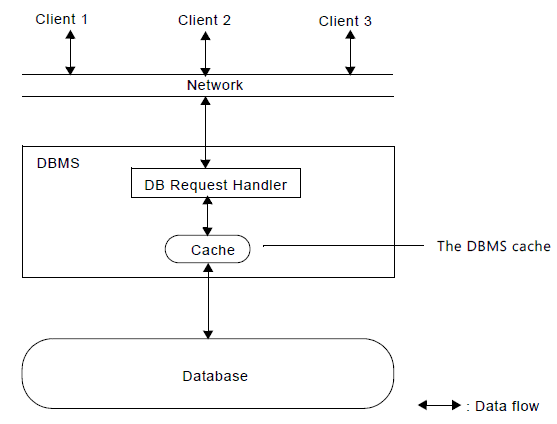
The DBMS cache is transparent to the user. For example, when a user requests data, the data is automatically copied into the cache and stored there. If the data is modified, it is automatically copied back to the physical disk. These data transfers take place automatically. The user does not need to know about the cache.

For example, three users send requests to the DBMS. When user 2 sends a request to read data from the database, the request handler determines whether the desired data can be fetched directly from the cache or whether it must be fetched from a disk.

At the same time, another user can modify a record in a table in the database. The modified data will be written to the DBMS cache, and not to the disk. When this user completes the write transaction (that is, commits the changes), the data in the cache that was modified during the transaction is written to the disk. The cache is then said to be flushed.

Advantages

* **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



9.8. Transaction Rollback & Roll forward

In [database](https://en.wikipedia.org/wiki/Database) technologies, a **rollback** is an operation which returns the database to some previous state. Rollbacks are important for database [integrity](https://en.wikipedia.org/wiki/Data_integrity), because they mean that the database can be restored to a clean copy even after erroneous operations are performed. They are crucial for recovering from database server crashes; by rolling back any[transaction](https://en.wikipedia.org/wiki/Database_transaction) which was active at the time of the crash, the database is restored to a consistent state.

The rollback feature is usually implemented with a [transaction log](https://en.wikipedia.org/wiki/Database_log), but can also be implemented via [multiversion concurrency control](https://en.wikipedia.org/wiki/Multiversion_concurrency_control).

Similarly, 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.

9.9. Check Pointing, Shadow Paging  
  
**Checkpointing** is a technique to add fault tolerance into computing systems. It basically consists of saving a snapshot of the application's state, so that it can restart from that **point** in case of failure.

In computer science, **shadow paging** is a technique for providing atomicity and durability (two of the ACID properties) in database systems. A page in this context refers to a unit of physical storage (probably on a hard disk), typically of the order of 210 to 216 bytes.

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 tranasacation two page tables are maintained, one called a shadow page table and current page table. When a tranasaction 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.

**9.10. Recovery Scheming   
(WAL: Write Ahead Logging)**

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 traditional rollback journal works by writing a copy of the original unchanged database content into a separate rollback journal file and then writing changes directly into the database file. In the event of a crash or [ROLLBACK](https://www.sqlite.org/lang_transaction.html), the original content contained in the rollback journal is played back into the database file to revert the database file to its original state. The [COMMIT](https://www.sqlite.org/lang_transaction.html) occurs when the rollback journal is deleted.

The WAL approach inverts this. The original content is preserved in the database file and the changes are appended into a separate WAL file. A [COMMIT](https://www.sqlite.org/lang_transaction.html) occurs when a special record indicating a commit is appended to the WAL. Thus a COMMIT can happen without ever writing to the original database, which allows readers to continue operating from the original unaltered database while changes are simultaneously being committed into the WAL. Multiple transactions can be appended to the end of a single WAL file.

**9.11. Failure with loss of Non volatile storage (General Concepts)**

So far we assumed no loss of non­volatile storage  
■ Technique similar to checkpointing used to deal with loss of nonvolatile storage  
● 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  
4 Output all log records currently residing in main memory onto  
stable storage.  
4 Output all buffer blocks onto the disk.  
4 Copy the contents of the database to stable storage.  
4 Output a record <**dump**> to log on stable storage.

**9.12. Recovery in Multi-database System**

To recover from disk failure  
● restore database from most recent dump.  
● Consult the log and redo all transactions that committed after  
the dump  
■ Can be extended to allow transactions to be active during dump;  
known as **fuzzy dump** or **online dump**

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.