ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System Assignment #9**

**Submitted by:**

Bishal Pandey  
013BSCCSIT016

**Submitted to:**

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

**# Database recovery**

* There are many situations in which a transaction may not reach a commit or abort point.
  + - An operating system crash can terminate the DBMS processes
    - The DBMS can crash
    - The system might lose power
    - A disk may fail or other hardware may fail.
    - 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).
* Two approaches are discussed here: Manual Reprocessing and Automated Recovery.

**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 arisesa

**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.

**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**

Data must be in RAM for DBMS to operate on it!

• Buffer Mgr hides the fact that not all data is in RAM 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:

• 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! More on Buffer Management

• Requestor of page must eventually unpin it, and indicate whether page has been modified: – dirty bit is used for this.

• Page in pool may be requested many times, – a pin count is used. – To pin a page, pin\_count++ – A page is a candidate for replacement iff pin\_count == 0 (“unpinned”)

• CC & recovery may entail additional I/O when a frame is chosen for replacement. (Write A head Log protocol; more later.)

**Transaction log**

Every SQL Server database has a transaction log that records all transactions and the database modifications made by each transaction. The transaction log must be truncated on a regular basis to keep it from filling up. However, some factors can delay log truncation, so monitoring log size is important. Some operations can be minimally logged to reduce their impact on transaction log size.

The transaction log is a critical component of the database and, if there is a system failure, the transaction log might be required to bring your database back to a consistent state. The transaction log should never be deleted or moved unless you fully understand the ramifications of doing this.

**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) and Roll forward**

* 1. To maintain atomicity, a transaction’s operations are redone or undone.
     1. **Undo**: Restore all BFIMs on to disk (Remove all AFIMs).
     2. **Redo**: Restore all AFIMs on to disk.
  2. Database recovery is achieved either by performing only Undos or only Redos or by a combination of the two. These operations are recorded in the log as they happen.

**Check Pointing, shadow paging**

When more than one transaction are being executed in parallel, the logs are interleaved. At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering. To ease this situation, most modern DBMS use the concept of 'checkpoints'. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Problems in recovery procedure as discussed earlier :

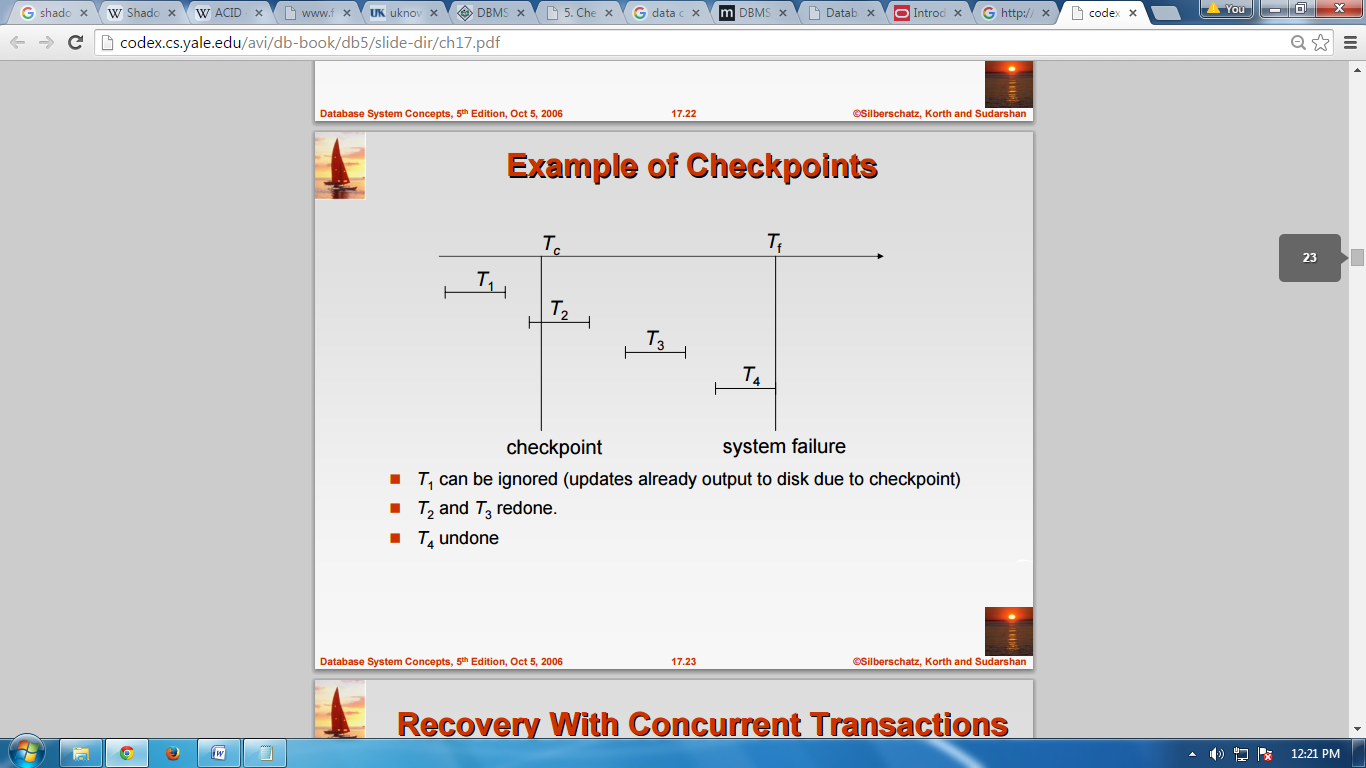
* Searching the entire log is time-consuming
* We might unnecessarily redo transactions which have already
* Output their updates to the database.

Streamline recovery procedure by periodically performing

* Output all log records currently residing in main memory onto stable storage.
* Output all modified buffer blocks to the disk.
* Write a log record < checkpoint> onto stable storage.

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

* Scan backwards from end of log to find the most recent <checkpoint> record
* Continue scanning backwards till a record <Ti start> 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 <Ti commit>, execute undo(Ti).
* Scanning forward in the log, for all transactions starting from Ti or later with a <Ti commit>, execute redo(Ti).



*Fig. example of check points*

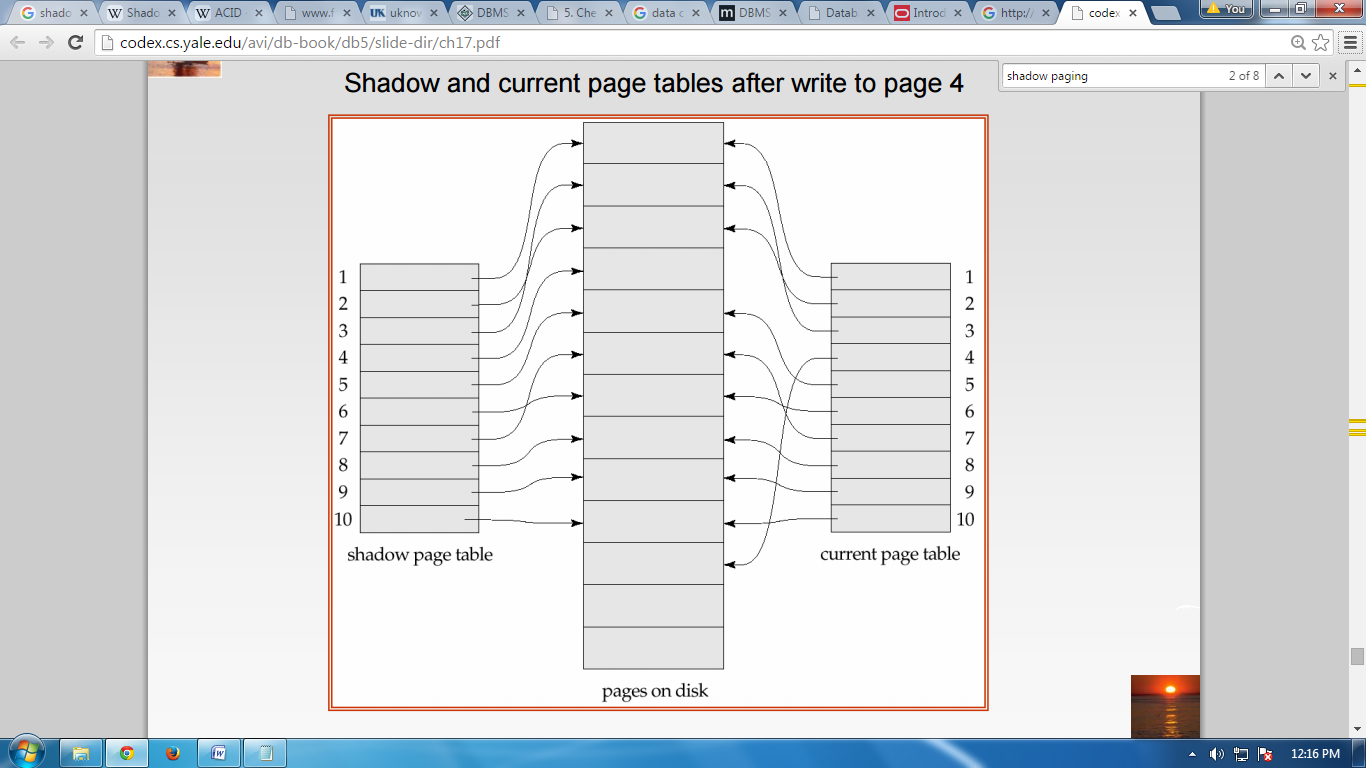
**Shadow Paging**

Shadow paging is an alternative to log-based recovery; this scheme is useful if transactions execute serially.

* 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.
* The update is performed on the copy.



*Fig. shadow paging*

**Recovery schemes (WAL: Write Ahead logging Protous)**

WAL is a log-based file system recovery scheme in which modifications to permanent data are written to a log file, the WAL log. The log file contains change records (Redo records) which represent the updates. At key moments, such as transaction commit, the WAL log is forced to disk. In the case of a reset or crash, Redo records can be used to transform the old copy of a permanent data block on disk into the version that existed in memory at the time of the reset.

By maintaining the WAL log, the permanent data blocks that were modified no longer have to be written to disk as each block is modified. Only the Redo records in the WAL log must be written to disk. This allows a write cache of permanent data blocks to be maintained.

WAL protects all permanent tables and all system tables but is not used to protect the Transient Journal (TJ), since TJ records are stored in the WAL log. WAL also is not used to protect spool or volatile tables.

The WAL log is maintained as a separate logical file system from the normal table area. Whole cylinders are allocated to the WAL log, and it has its own index structure.

The WAL log data is a sequence of WAL log records and includes the following:

**•**Redo records, used for updating disk blocks and insuring file system consistency during restarts.

**•**TJ records used for transaction rollback.

**Failure with loss of Non-volatile storage (general concepts)**

Technique similar to check pointing used to deal with loss of non-volatile 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 check pointing 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.” 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

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 periodically dumped 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.

**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.