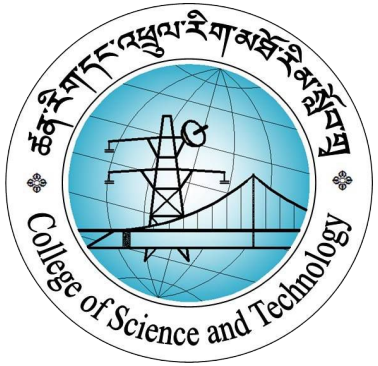


Royal University of Bhutan

# DBS101 Database Systems Fundamentals

## Lesson 27

# Learning Outcomes

1. Understand recovery mechanisms.
2. Analyse database recovery algorithms.
3. Explain the importance of database logging in recovery.

# Why are recovery systems so important in DBMS ???

# Why are recovery systems so important in DBMS ???

- System failures
- Transaction failures
- Human errors
- Security breaches
- Hardware upgrades
- Natural disasters
- Compliance regulations
- Data corruption

# Why are recovery systems so important in DBMS ???

**Transactions must be atomic**

How do you ensure that?

What is a technique you can use?

# Why are recovery systems so important in DBMS ???

**Transactions must be atomic**

How do you ensure that?

What is a technique you can use?

- **Log Based Recovery (The most common technique)**

# Log Records

The log is a sequence of log records, recording all the update activities in the database.

Any operation which is performed on the database is recorded on the log. Prior to performing any modification to the database, an updated log record is created to reflect that modification.

# Log Records

An update log record represented as: <Ti, Xj, V1, V2> has these fields:

1. Transaction identifier(Ti): Unique Identifier of the transaction that performed the write operation.
2. Data item(Xj): Unique identifier of the data item written.
3. Old value(V1): Value of data item prior to write.
4. New value(V2): Value of data item after write operation.

# Log Records

Other types of log records are:

1. <Ti start>: It contains information about when a transaction Ti starts.
2. <Ti commit>: It contains information about when a transaction Ti commits.
3. <Ti abort>: It contains information about when a transaction Ti aborts.

Read Blog: Postgresql Database logs 10 minutes

# Database Modification

If a transaction does not modify the database until it has committed, it is said to use the **deferred-modification technique.**

If database modifications occur while the transaction is still active, the transaction is said to use the **immediate-modification technique.**

The recovery algorithms we describe from now on will support immediate recovery.

# Database Modification

A recovery algorithm must take into account a variety of factors, including:

- The possibility that a transaction may have committed although some of its database modifications exist only in the disk buffer in main memory and not in the database on disk.
- The possibility that a transaction may have modified the database while in the active state and, as a result of a subsequent failure, may need to abort.

# Database Modification

**A database log must be created before a database modification**,this allows for the database to have both old and new modified values.This allows for the system to perform undo and redo operations.

The **undo operation** using a log record sets the data item specified in the log record to the old value contained in the log record.

The **redo operation** using a log record sets the data item specified in the log record to the new value contained in the log record.

# Read 2 slides; Undo and Redo Transactions: 5 minutes

# Undo Transactions

Undo(Ti):

- Restores the value of all data items updated by transaction Ti to their old values by using the log records.
- Writes special redo-only log records to record the updates performed as part of the undo process.
- Does not need to store the old value in the redo-only log records, as the "old value" is the value written by the transaction being rolled back, and the "new value" is the original value being restored.
- Maintains the order in which undo operations are performed to ensure correctness.
- Writes a <Ti abort> log record after completing the undo operation for transaction Ti.

# Redo Transactions

Redo(Ti):

- Sets the value of all data items updated by transaction Ti to their new values using the log records.
- Maintains the order in which updates are redone, as applying them in a different order can lead to incorrect final values.
- Instead of redoing each transaction separately, a single scan of the log is performed, and redo actions are carried out for each log record as it is encountered, ensuring the order is preserved and improving efficiency.

# System Crash Scenario

Transactions that need to be **undone** are identified by the presence of **<Ti start>** in the log **without** a corresponding **<Ti commit> or <Ti abort>**.

Transactions that need to be **redone** are identified by the **presence of <Ti start> and either <Ti commit> or <Ti abort>** in the log.

If <Ti abort> is present, the redo-only log records written during the undo operation are used to redo the transaction, effectively undoing the transaction's modifications.

# Example

- If the crash occurs before <Ti commit>, undo(Ti) is performed to restore the original values.
- If the crash occurs after <Ti commit>, redo(Ti) is performed to ensure the transaction's effects are applied.
- If the crash occurs in the middle of a transaction, undo is performed for the incomplete transaction, and redo is performed for any committed transactions.

# Checkpoints

A checkpoint is a mechanism that periodically records the state of the database system, including information about active transactions, on stable storage (e.g., disk). The main purposes of checkpoints are:

- To limit the portion of the log that needs to be examined during recovery after a crash. Without checkpoints, the entire log would need to be scanned to identify transactions that need to be undone or redone.
- To allow the log to be truncated, discarding log records prior to the checkpoint, as they are no longer needed for recovery.

# Checkpoints

During a checkpoint, the following steps are performed:

- All log records in memory are written to stable storage.
- All modified buffer blocks (with updates from transactions) are written to disk.
- A special checkpoint log record is written, containing a list of active transactions at the time of the checkpoint.

After a crash, the recovery process only needs to consider transactions since the last checkpoint, significantly reducing the recovery time and effort.

# Fuzzy Checkpoint

A fuzzy checkpoint is a variation of the checkpoint technique where transactions are allowed to continue executing and performing updates while the checkpoint is in progress. This relaxes the requirement of halting all transaction activity during checkpointing, which can be disruptive to the system.

In a fuzzy checkpoint, transactions that start executing after the checkpoint begins are not included in the list of active transactions recorded in the checkpoint log record. Their updates are effectively deferred until after the checkpoint completes.

# Fuzzy Checkpoint

Fuzzy checkpoints provide more flexibility and reduce the impact on transaction processing during checkpointing, but they introduce some additional complexity in the recovery process, as transactions that started after the checkpoint must be treated differently.

The use of checkpoints and fuzzy checkpoints is crucial for efficient database recovery because they significantly reduce the amount of log data that needs to be processed during recovery, thereby improving the overall recovery time and system availability after a crash.

# Log Record Buffering

Log records are temporarily kept in a log buffer in main memory instead of being immediately written to stable storage.

This allows multiple log records to be gathered and output to stable storage in a single operation, reducing overhead.

However, to ensure atomicity, the Write-Ahead Logging (WAL) rule must be enforced, which requires certain log records to be on stable storage before related actions.

# Database Buffering

The no-force policy allows a transaction to commit without writing all its modified database blocks to disk immediately.

The steal policy allows writing modified database blocks to disk even if the modifying transaction has not committed yet.

To ensure WAL during block writes, exclusive locks are acquired, relevant log records are forced to stable storage first, then the block is written, and lock is released.

# Fuzzy Checkpointing

Fuzzy checkpoints allow transactions to continue performing updates during the checkpoint operation instead of being blocked.

The checkpoint record is written before modified buffer blocks are written to disk.

To handle incomplete checkpoints due to crashes, the last-checkpoint location is updated only after all modified blocks are written.

# Failure with Loss of Non-Volatile Storage

To recover from failures that lose non-volatile disk storage, the database is periodically dumped entirely to stable backup storage like tapes.

Recovery first restores the database state from the most recent dump, then redoes changes from the log after that point.

# High Availability Using Remote Backup Systems

A remote backup site maintains a replica updated by shipping log records from the primary site.

If the primary fails, the remote backup takes over by completing recovery using its data copy and received logs, then processes new transactions.

This provides high availability by allowing recovery from disasters affecting the primary site.

# High Availability Using Remote Backup Systems

## Remote Backup System Issues

- Failure detection using multiple communication links, automated or manual transfer of control to the backup.
- Hot-spare configuration allows backup to be almost ready by continuously replaying logs.
- Trade-offs between different commit durability levels - one-safe, two-very-safe, two-safe.

# High Availability Using Remote Backup Systems

## Other High Availability Techniques

- Distributed databases with data replication across sites can provide higher availability than single remote backup.
- At application level, load balancing of requests across servers, and failover to active servers on failures.

# Recovery Algorithms

Recovery algorithms are techniques to ensure database consistency, transaction atomicity, and durability despite failures.

Recovery algorithms have two parts:

- Actions during normal txn processing to ensure that the DBMS can recover from a failure.
- Actions after a failure to recover the database to a state that ensures atomicity, consistency, and durability.

# Recovery Algorithms: ARIES

Algorithms for Recovery and Isolation Exploiting Semantics.

Developed at IBM Research in early 1990s for the DB2 DBMS.

Not all systems implement ARIES exactly as defined in this paper but they're close enough.

# Recovery Algorithms: ARIES

ARIES recovers from a system crash in three passes.

1. **Analysis pass**: This pass determines which transactions to undo, which pages were dirty at the time of the crash, and the LSN from which the redo pass should start.
2. **Redo pass**: This pass starts from a position determined during analysis and performs a redo, repeating history, to bring the database to a state it was in before the crash.
3. **Undo pass**: This pass rollback all transactions that were incomplete at the time of crash.

# Recovery Algorithms: ARIES

Write-Ahead Logging:

- Any change is recorded in log on stable storage before the database change is written to disk.
- Must use STEAL + NO-FORCE buffer pool policies.

Repeating History During Redo:

- On DBMS restart, retrace actions and restore database to exact state before crash.

Logging Changes During Undo:

- Record undo actions to log to ensure action is not repeated in the event of repeated failures.

# Recovery in Main-Memory Databases

Main-memory databases keep data in main memory to support very fast random access for queries and updates.

However, main memory is volatile, so its contents are lost during system failures or shutdowns. Therefore, data must also be stored on persistent or stable storage like disk to allow recovery when the system restarts.

Traditional database recovery algorithms involving logging, checkpointing, and log replay can be applied to main-memory databases.

# Recovery in Main-Memory Databases

Some optimizations are possible for main-memory database recovery:

1. No redo logging is required for index updates since indices can be rebuilt very quickly from the base data after recovery.
2. Only undo log records for aborts need to be kept in memory, not forced to stable storage, reducing logging overhead.
3. Several databases perform just redo logging, taking periodic checkpoints while avoiding updates to uncommitted data on disk or using record versioning. Recovery then reloads the last checkpoint and redoes changes.

# Recovery in Main-Memory Databases

Fast recovery is critical for main-memory databases since the entire database must be loaded into memory and all recovery actions must be completed before any transaction processing can begin. To minimize this recovery time, several main-memory databases perform recovery in parallel across multiple CPU cores.

This parallelization is enabled by partitioning the data and log records, such that each core handles recovery for a particular data partition using the corresponding log records.

# The End .....