

## Lesson 12 – Chapter 22

# Database Recovery

*Established in the Unified Field, perform action*



## WHOLENESS OF THE LESSON

Database recovery is the process of restoring the database to a correct state in the event of a failure.

Science & Technology of Consciousness: Through the regular practice of the TM technique, one avoids failures in life.



# Database Recovery

**Process of restoring the database to a correct state in the event of a failure.**

- **Need for Recovery Control**

- Two types of storage: volatile (main memory) and nonvolatile.
- Volatile storage does not survive system crashes.
- Stable storage represents information that has been replicated in several nonvolatile storage media with independent failure modes.



# Types of Failures

- **System crashes** due to h/w or s/w errors, resulting in loss of main memory.
- **Media failures** such as head crash, resulting in loss of parts of secondary storage.
- **Application software errors** causing one or more transactions to fail.
- **Natural physical disasters** such as floods, fires.
- **Carelessness** or unintentional destruction of data or facilities by operators or users.
- **Sabotage**, intentional corruption or destruction of data, h/w, or s/w facilities.



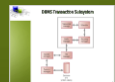
## Types of Failures contd...

- Whatever the cause of the failure, there are two principal effects that we need to consider:
  - The loss of main memory, including the database buffers;
  - The loss of the disk copy of the database.
- We need to discuss the concepts and techniques that can minimize these effects and allow recovery from failure.



# Transactions and Recovery

- Transactions represent basic unit of recovery.
- Recovery manager is responsible for Atomicity and Durability.
- The recovery manager has to ensure that on recovery from failure, either all the effects of a given transaction are permanently recorded in the database or none of them are.
- The situation is complicated by the fact that database writing is not an atomic (single-step) action, and it is therefore possible for a transaction to have committed but for its effects not to have been permanently recorded in the database simply because they have not yet reached the database.





# Transactions Revision

- A database is updated by processing *transactions* that result in changes to one or more records.
- A user's program may carry out many operations on the data retrieved from the database, but the DBMS is only concerned with data *read/written* from/to the database.
- The DBMS's abstract view of a user program is a sequence of transactions (*reads* and *writes*).
- To understand database recovery, we must first understand the concept of *transaction integrity*.



# Transactions

- A transaction is considered a logical unit of work
  - START Statement: `BEGIN TRANSACTION`
  - END Statement: `COMMIT`
  - Execution errors: `ROLLBACK`
- Assume we want to transfer \$100 from one bank (A) account to another (B):  
  
`UPDATE Account_A SET Balance=Balance-100;`  
  
`UPDATE Account_B SET Balance=Balance+100;`
- We want these two operations to appear as a ***single atomic action***.





## Transactions contd..

- Why do we want these two operations to appear as a **single atomic action**?
  - To avoid **inconsistent states** of the database **in-between** the two updates
  - And obviously we cannot allow the first UPDATE to be executed and the second not or vice versa.
- Transactions guarantee that, if a failure occurs before the transaction reaches its planned termination, then those previous transaction updates will be **undone**.



# Transaction Recovery

- **COMMIT** establishes a Commit Point or Synch Point
  - A point at which we assume that the database is in a correct state
- **ROLLBACK** has to roll back the database to the state that it had before the Transaction started.



# Atomicity

```
UPDATE Account_A SET Balance= Balance -100;  
1.    Read(A)  
2.    A = A - 100  
3.    Write(A)  
UPDATE Account_B SET Balance= Balance +100;  
4.    Read(B)  
5.    B = B + 100  
6.    write(B)
```

- Transaction may fail after step 3 and before step 6 (failure could be due to s/w or h/w)
- The system should ensure that updates of a partially executed transaction are not reflected in the database.



# Consistency

```
UPDATE Account_A SET Balance= Balance -100;  
1.   Read(A)  
2.   A = A - 100  
3.   Write(A)  
UPDATE Account_A SET Balance= Balance +100;  
4.   Read(B)  
5.   B = B + 100  
6.   write(B)
```

- In this example, the sum of A and B is unchanged by the execution of the transaction.



# Isolation

<u>Transaction T1</u>	<u>Transaction T2</u>
1. <code>Read(A)</code>	
2. <code>A = A - 100</code>	
3. <code>Write(A)</code>	
	<code>Read(A), Read(B), Write(A+B)</code>
4. <code>Read(B)</code>	
5. <code>B = B + 100</code>	
6. <code>write(B)</code>	

- What will T2 "see"? - Database changes should not be revealed to users until after transaction has completed
- Isolation can be ensured trivially by running transactions **serially**.
- However, executing multiple transactions concurrently has significant benefits
  - Keep CPU humming when disk I/O takes place.



# Durability

```
UPDATE Account_A SET Balance= Balance -100;  
1.    Read(A)  
2.    A = A - 100  
3.    Write(A)  
UPDATE Account_A SET Balance= Balance +100;  
4.    Read(B)  
5.    B = B + 100  
6.    write(B)
```


- Database changes are permanent (once the transaction was committed)




# Transactions and Recovery

## Steps taken by DBMS for Read Operation

- find the address of the disk block that contains the record with PK value  $x$ ;
- transfer the disk block salary data into a database buffer in main memory;
- copy the salary data from the database buffer into the variable *salary*.



```
read(staffNo = x, salary)
salary = salary * 1.1
write(staffNo = x, new_salary)
```



## Steps taken by DBMS for Write Operation

- find the address of the disk block that contains the record with PK value  $x$ ;
- transfer the disk block address into a database buffer in main memory;
- copy the salary data from the variable *salary* into the database buffer;
- write the database buffer back to disk.



# Transactions and Recovery contd..

- Any update operation is regarded as permanent only when the **buffers\*** are **flushed** to the secondary storage.
- This flushing of the buffers to the database can be triggered by a specific command (e.g. **transaction commit**) or automatically when the **buffers become full**.
- The explicit writing of the buffers to secondary storage is known as **force-writing**.

\*The database buffers occupy an area in main memory from which data is transferred to and from secondary storage.



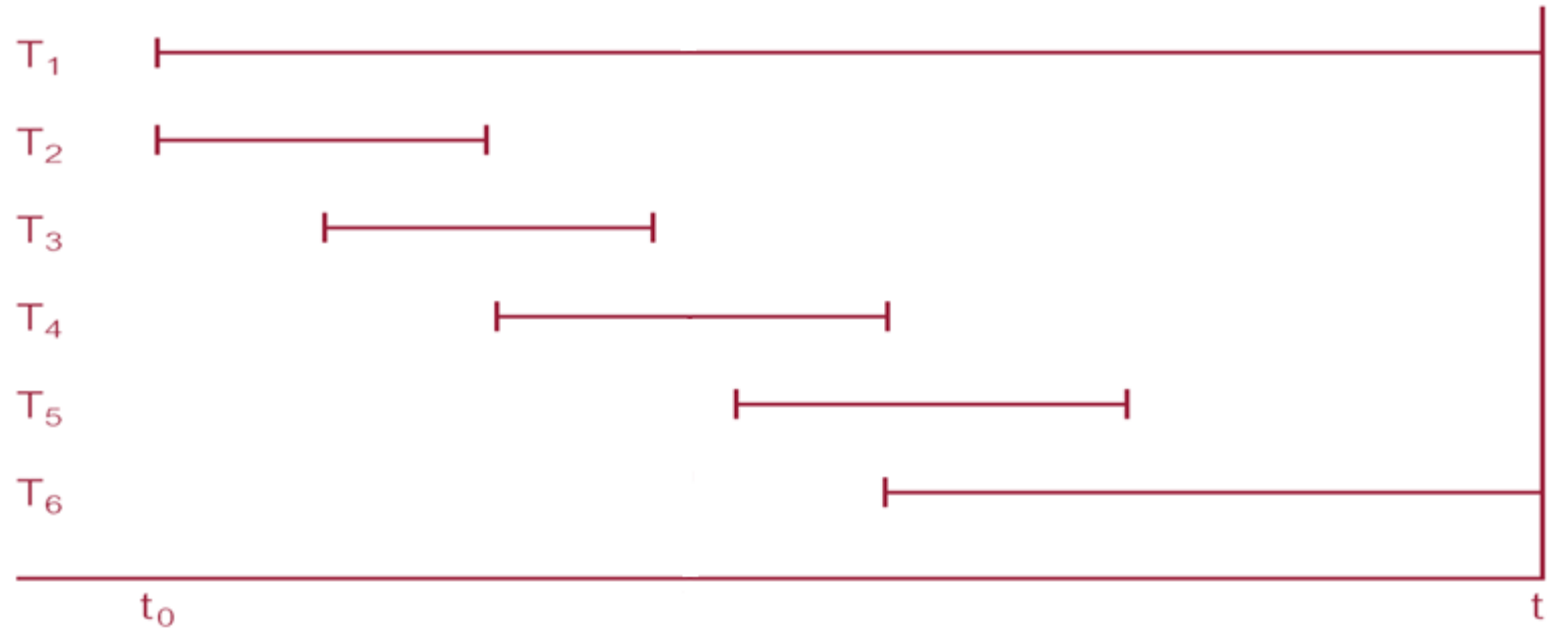


## Transactions and Recovery contd..

- If failure occurs between commit and database buffers being flushed to secondary storage then, to ensure durability, recovery manager has to *redo (rollforward)* transaction's updates.
- If transaction had not committed at failure time, recovery manager has to *undo (rollback)* any effects of that transaction for atomicity.
  - Partial undo - only one transaction has to be undone.
  - Global undo - all active transactions have to be undone.



## Example – Use of UNDO/REDO



Concurrently  
executing  
transactions

- DBMS starts at time  $t_0$ , but fails at time  $t_f$ . Assume that data for transactions  $T_2$  and  $T_3$  have been written to secondary storage.
- $T_1$  and  $T_6$  have to be undone. In absence of any other information, recovery manager has to redo  $T_2$ ,  $T_3$ ,  $T_4$ , and  $T_5$ .



# Recovery Facilities

- DBMS should provide following facilities to assist with recovery:
  - **Backup mechanism**, which makes periodic backup copies of database.
  - **Logging facilities**, which keeps track of current state of transactions and database changes.
  - **Checkpoint facility**, which enables updates to database in progress to be made permanent.
  - **Recovery manager**, which allows DBMS to restore database to consistent state following a failure.



# Backup Mechanism

- DBMS should provide a mechanism to allow backup copies of the DB and log file to be made at regular intervals without a need to stop the system first.
- Backup copy is used in the event when DB is damaged or destroyed.
- Backup can be a complete copy of the DB or an incremental backup.
- Backup is stored offline on optical disk.



# Log File

- To keep track of database transactions, the DBMS maintains a special file called a **log** (or **journal**) that contains information about all updates to the database.
- The log may contain the following data:
  - **Transaction records**
  - **Checkpoint records**
- Often used for other purposes (e.g. performance monitoring and auditing).



## Log File contd..

- **Transaction records contain:**
  - Transaction identifier
  - Type of log record (transaction start, insert, update, delete, abort, commit)
  - Identifier of data item affected by database action (insert, delete, and update operations).
  - Before-image of data item
  - After-image of data item
  - Log management information such as a pointer to previous and next log records for that transaction (all operations).



# Sample Log File

	Tid	Time	Operation	Object	Before image	After image	pPtr	nPtr
1	T1	10:12	START				0	2
2	T1	10:13	UPDATE	STAFF SL21	(old value)	(new value)	1	8
3	T2	10:14	START				0	4
4	T2	10:16	INSERT	STAFF SG37		(new value)	3	5
5	T2	10:17	DELETE	STAFF SA9	(old value)		4	6
6	T2	10:17	UPDATE	PROPERTY PG16	(old value)	(new value)	5	9
7	T3	10:18	START				0	11
8	T1	10:18	COMMIT				2	0
9		10:19	CHECKPOINT	T2, T3				
10	T2	10:19	COMMIT				6	0
11	T3	10:20	INSERT	PROPERTY PG4		(new value)	7	12
12	T3	10:21	COMMIT				11	0



## Log File contd..

- Log file may be duplexed or triplexed (i.e. two or three separate copies are maintained).
- Log files sometimes split into two separate random-access files.
- It should be noted that the log file is a potential bottleneck and the speed of the writes to the log file can be critical in determining the overall performance of the database system.





# Write-Ahead Log Protocol

- It is essential that log records are written before the corresponding write to the database. This is known as the **write-ahead log protocol**.
- If updates were made to the database first and failure occurred before the log record was written, then the recovery manager would have no way of undoing (or redoing) the operation.
- Under the write-ahead log protocol, the recovery manager can safely assume that if there is no transaction commit record in the log file for a particular transaction, then that transaction was still active at the time of failure and must therefore be undone.



# Checkpointing

- The information in the log file is used to recover from a database failure.
- One difficulty with this scheme is that when a failure occurs, we may not know how far back in the log to search and we may end up redoing transactions that have been safely written to the database.
- To limit the amount of searching and subsequent processing that we need to carry out on the log file, we can use a technique called **checkpointing**.
- **Checkpoint** - Point of synchronization between the database and the transaction log file. All buffers are force-written to secondary storage.
- Checkpoints are scheduled at predetermined intervals.



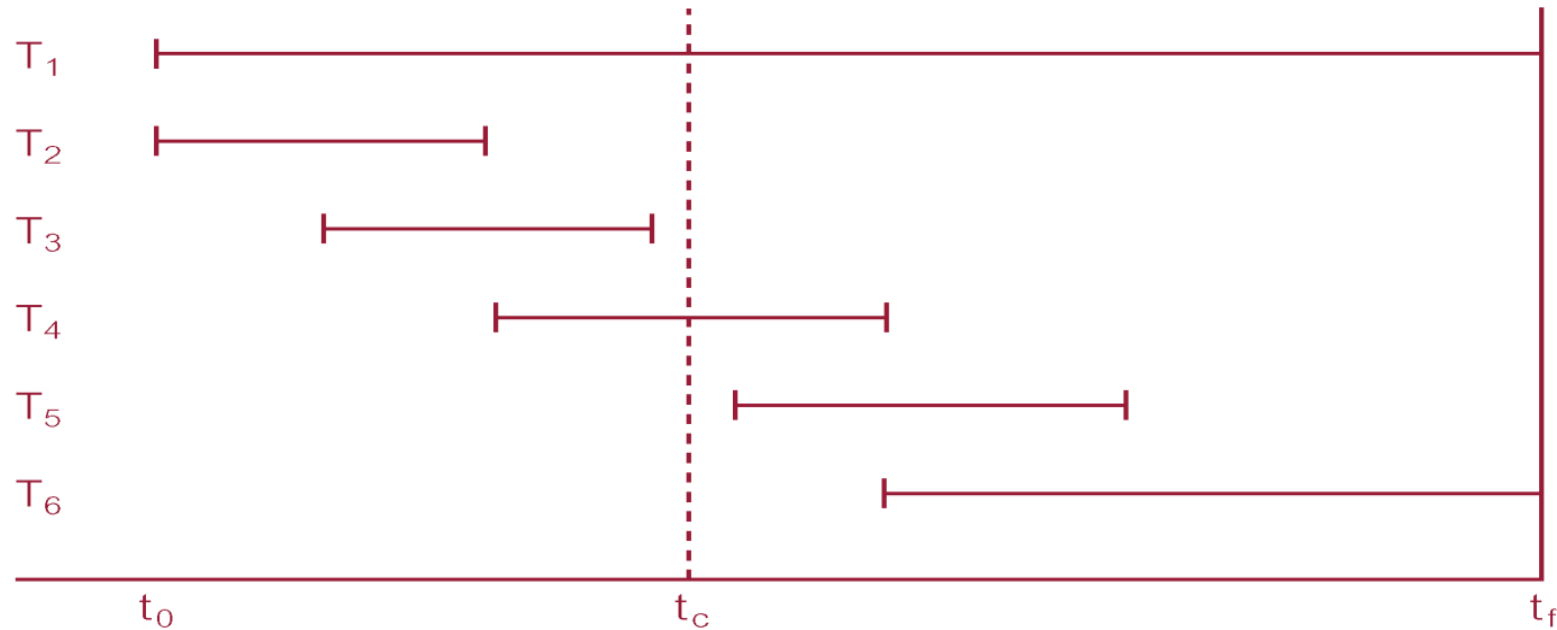
# Checkpointing contd..

## Checkpointing involves the following operations:

- Writing all log records in main memory to secondary storage;
- Writing the modified blocks in the database buffers to secondary storage;
- Writing a checkpoint record to the log file. This record contains the identifiers of all transactions that are active at the time of the checkpoint.
- **When failure occurs, redo all transactions that have committed since the checkpoint and undo all transactions that were active at the time of crash.**



# Checkpointing Example



- In this example, with checkpoint at time  $t_c$ , changes made by  $T_2$  and  $T_3$  have been written to secondary storage. Thus:
  - only redo  $T_4$  and  $T_5$ ,
  - undo transactions  $T_1$  and  $T_6$ .



# Recovery Techniques

- **If database has been extensively damaged:**
  - Need to restore last backup copy of database and reapply updates of committed transactions using log file.
- **If database is only inconsistent & not physically damaged:**
  - Need to undo changes that caused inconsistency. May also need to redo some transactions to ensure updates reach secondary storage.
  - Do not need backup copy, but can restore database using before- and after-images in the log file.



# Main Recovery Techniques

- Three main recovery techniques:
  - **Deferred Update**
  - **Immediate Update**
  - **Shadow Paging**
- These techniques are applicable only if database is inconsistent and not physically damaged.



# Deferred Update

- Updates are not written to the database until after a transaction has reached its commit point.
- Before reaching the commit point, all transaction updates are recorded in the local transaction workspace (or cache buffers).
- If transaction fails before commit, it will not have modified the database and so no undoing of changes required.
- May be necessary to redo updates of committed transactions as their effects may not have reached database.
- Deferred update is also known as **NO-UNDO/REDO** algorithm.



# Deferred Update – Use of Logs to Protect against System Failures

- When a transaction starts, write a *transaction start* record to the log.
- For all the transaction operations, write a log record containing all the log data specified previously. Do not actually write the update to the database buffers or to the database itself.
- When a transaction is about to commit, write a *transaction commit* log record, write all the log records for the transaction to disk, and then commit the transaction. Use the log records to perform the actual updates to the Database.
- If transaction aborts, ignore the log records for the transaction and do not perform the writes. Add transaction abort record to the log.





## Deferred Update – Use Of Logs To Protect Against System Failures contd..

Starting at the last entry in the log file, we go back to the most recent checkpoint record:

- Any transaction with *transaction start* and *transaction commit* log records or just *transaction commit* log record should be **redone**. The redo procedure performs all the writes to the database using the *after-image* log records for the transactions, *in the order in which they were written to the-log*.
  - This method guarantees that we will update any data item that was not properly updated prior to the failure.
  - If this writing has been performed already, before the failure, the write has no effect on the data item, so there is no damage done if we write the data again.



## Deferred Update – Use Of Logs To Protect Against System Failures contd..

- For any transactions with *transaction start* and *transaction abort* log records, we do nothing, because no actual writing was done to the database.
- The transactions that were active and didn't commit are effectively canceled and must be resubmitted.



# Immediate Update

- Updates are applied to database as they occur without waiting to reach the commit point.
- Need to redo updates of committed transactions following a failure.
- May need to undo effects of transactions that had not committed at time of failure.
- Essential that log records are written before the write to the database. *Write-ahead log protocol*.
- If no "*transaction commit*" record in log, then that transaction was active at failure and must be undone.
- Undo operations are performed *in reverse order in which they were written to log*.
- Immediate update is also known as **UNDO/REDO** algorithm.



# Immediate Update – Use Of Logs To Protect Against System Failures

- When a transaction starts, write a *transaction start* record to the log.
- When a write operation is performed, write a record containing the necessary data to the log file.
- Once the log record is written, write the update to the database buffers.
- The updates to the database itself are written when the buffers are next flushed to secondary storage or when full.
- When the transaction commits, write a *transaction commit* record to the log.



# Immediate Update – Use Of Logs To Protect Against System Failures contd..

Starting at the last entry in the log file, we go back to the most recent checkpoint record:

- For any transaction for which both a *transaction start* and *transaction commit* records appear in the log or if only *transaction commit* record appears then we **redo** using the log records to write the *after-image* of updated fields.
- If the new values have already been written to the db, these writes, although unnecessary, will have no effect. However, any write that did not actually reach the db will now be performed.



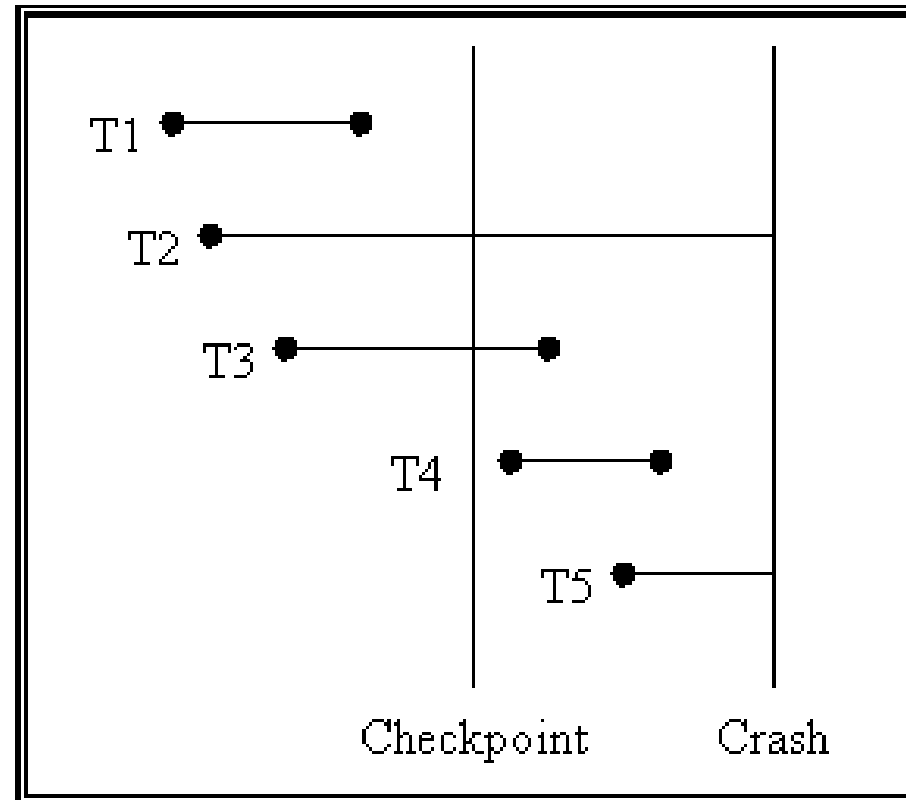
# Immediate Update – Use Of Logs To Protect Against System Failures contd..

- For any transaction for which the log contains a *transaction start* record but not a *transaction commit* record, we need to **undo** that transaction.
  - This time the log records are used to write the before-image of the affected fields, and thus restore the database to its state prior to the transaction's start.
- The undo operations are performed *in the reverse order to which they were written to the log*.



# Example – Deferred Update & Immediate Update

- Assume that database buffers are only written to disk at checkpoints.
- Figure out what will happen to each of the transactions under Deferred Update and Immediate Update protocols when crash occurs.





# Recovery Scenarios

	Deferred Update	Immediate Update
<b>Transaction was committed before checkpoint</b>	No Rollback (no undo) No Roll forward (no redo) No Restart	No Rollback (no undo) No Roll forward (no redo) No Restart
<b>Transaction was committed after checkpoint</b>	Roll forward (Redo)	Roll forward (Redo)
<b>Transaction never committed</b>	No Rollback (no undo) No Roll forward (no redo) Restart is needed	Rollback (Undo) Restart is needed





# MAIN POINTS

1. The DBMS should provide the following facilities to assist with recovery: backup mechanism, logging facilities, checkpoint facility and recovery manager. **Science & Technology of Consciousness:** When activity is undertaken from the level of pure intelligence, no errors or obstacles arise.
2. The main recovery techniques are deferred update (updates not written to database until after commit) and immediate update (updates applied to database as they occur). **Science & Technology of Consciousness:** The regular practice of the TM technique cultures flexibility that allows one to quickly recover from any circumstances.



## UNITY CHART

### CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE:

#### Database Recovery Techniques in case of Failures

1. System failures such as crashes or media failures may cause loss of information that is vital to an enterprise.
2. DBMS Recovery Facilities are designed to restore the database in the event of a failure with no loss of information.

- 
3. Transcendental consciousness is the home of all the laws of nature.
  4. Impulses within the Transcendental Field: These impulses of change move within the eternally stable field of pure consciousness.
  5. Wholeness moving within itself: All activity is conducted by natural law for maximum evolution towards unity consciousness.

