



**TKR COLLEGE OF ENGINEERING & TECHNOLOGY**  
**AUTONOMOUS**

Approved By AICTE. Affiliated to JNTUH. Accredited By NBA

Accredited by NAAC with 'A' Grade

"Recognized under 2(F) and 12(B) of UGC Act 1956"

**Time/Hr : 12.30pm-1.20pm / 4<sup>th</sup>**

**Date : 17.01.2022**

**Dept / Year : DS / II**

# **DATABASE MANAGEMENT SYSTEMS**

## **UNIT - 4 Transaction Management**

### **Topic : Multi-version Schemes & Recovery System**

**Online Platform : Google Class**

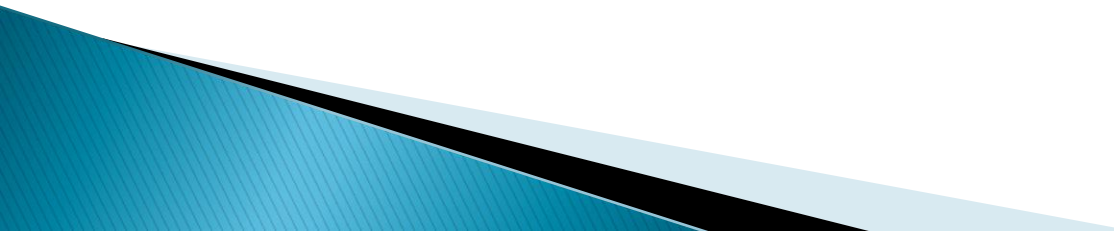
**Online Class Delivery by,**

**Dr.S.A Kalaiselvan,**

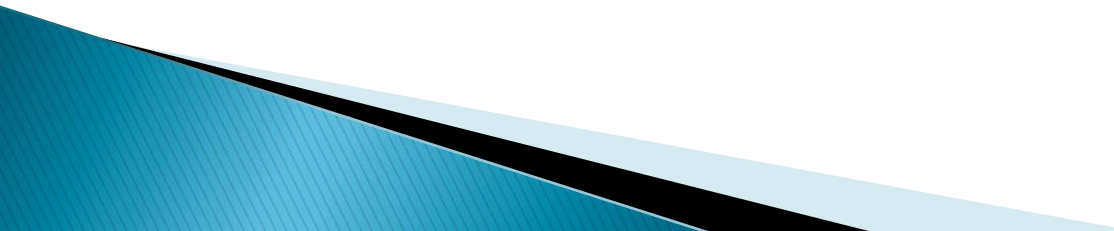
**Professor / CSE,**

**TKR College of Engineering & Technology.**

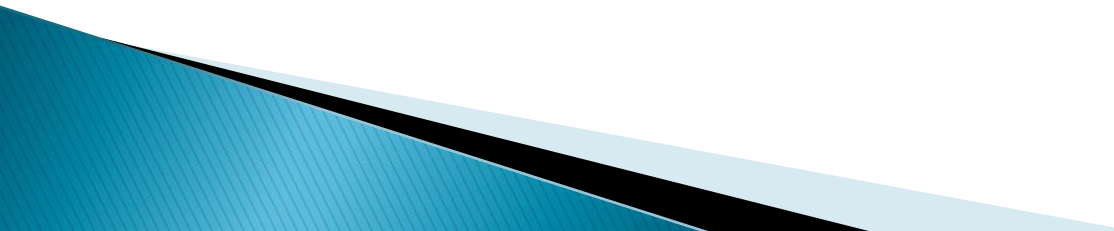
## 1.1.3 Multi-version Schemes

- ▶ The concurrency-control schemes discussed thus far ensure serializability by either delaying an operation or aborting the transaction that issued the operation.
  - ▶ Multiversion schemes keep old versions of data item to increase concurrency.
  - ▶ For example, a read operation may be delayed because the appropriate value has not been written yet; or it may be rejected because the value that it was supposed to read has already been overwritten.
- 

# Contd..

- ▶ These difficulties could be avoided if old copies of each data item were kept in a system.
  - ▶ In multiversion concurrency-control schemes, each  $\text{write}(Q)$  operation creates a new version of  $Q$ . When a transaction issues a  $\text{read}(Q)$  operation, the concurrency-control manager selects one of the versions of  $Q$  to be read.
  - ▶ The concurrency-control scheme must ensure that the version to be read is selected in a manner that ensures serializability
- 

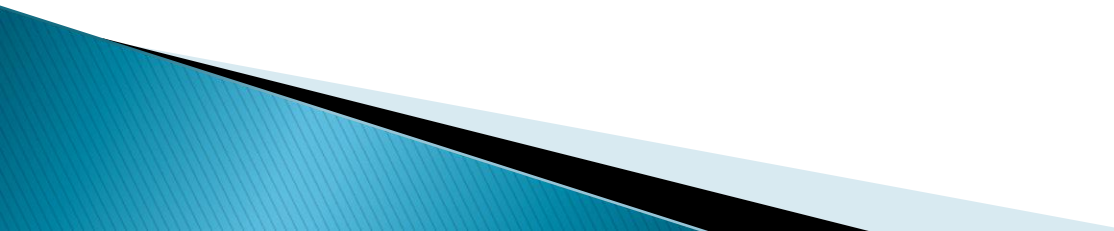
# Contd..

- ▶ An obvious drawback of multiversion techniques is that more storage is needed to maintain multiple versions of the database items. However, older versions may have to be maintained anyway - for example, for recovery purposes.
  - ▶ In addition, some database applications require older versions to be kept to maintain a history of the evolution of data item values. The extreme case is a temporal database, which keeps track of all changes and the times at which they occurred.
- 

# Contd..

- ▶ Several multiversion concurrency control schemes have been proposed. We discuss two schemes here, one based on timestamp ordering and the other based on 2PL.
  - ▶ **Multiversion Technique Based on Timestamp Ordering**
  - ▶ **Multiversion Two-Phase Locking Using Certify Locks**
- ▶ *Multiversion Technique Based on Timestamp Ordering* : In this method, several versions  $X_1, X_2, \dots, X_k$  of each data item  $X$  are maintained. For each version, the value of version  $X_i$  and the following two timestamps are kept:

# Contd..

- ▶ Each version  $X_i$  contains three data fields:
  - ▶ Content is the value of version  $X_k$  .
  - ▶  $\text{read\_TS}(X_i)$ . The read timestamp of  $X_i$  is the largest of all the timestamps of transactions that have successfully read version  $X_i$ .
  - ▶  $\text{write\_TS}(X_i)$ . The write timestamp of  $X_i$  is the timestamp of the transaction that wrote the value of version  $X_i$ .
- 

# Contd..

- ▶ Whenever a transaction  $T$  is allowed to execute a  $\text{write\_item}(X)$  operation, a new version  $X$  of item  $X$  is created, with both the  $\text{write\_TS}(X)$  and the  $\text{read\_TS}(X)$  set to  $\text{TS}(T)$ .
- ▶ Correspondingly, when a transaction  $T$  is allowed to read the value of version  $X$ , the value of  $\text{read\_TS}(X)$  is set to the larger of the current  $\text{read\_TS}(X)$  and  $\text{TS}(T)$ .

To ensure serializability, the following rules are used:

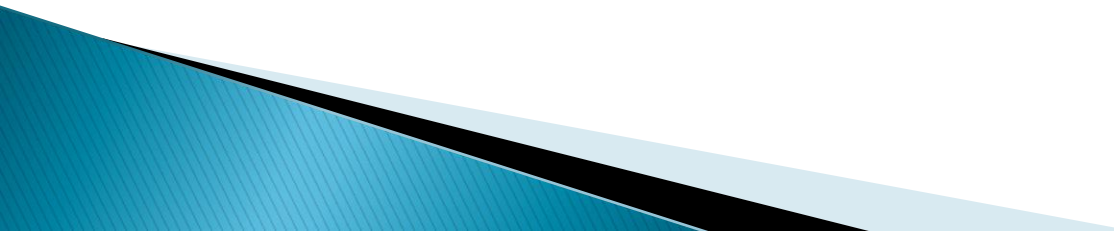
- If transaction  $T$  issues a  $\text{write\_item}(X)$  operation, and version of  $X$  has the highest  $\text{write\_TS}(X_i)$  of all versions of  $X$  that is also less than or equal to  $\text{TS}(T)$ , and  $\text{read\_TS}(X_i) > \text{TS}(T)$ , then abort and roll back transaction  $T$ .

# Contd..

- ▶ If transaction  $T$  issues a  $\text{read\_item}(X)$  operation, find the version of  $X$  that has the highest  $\text{write\_TS}(X_i)$  of all versions of  $X$  that is also less than or equal to  $\text{TS}(T)$ ; then return the value of  $X_i$  to transaction  $T$ , and set the value of  $\text{read\_TS}(X_i)$  to the larger of  $\text{TS}(T)$  and the current  $\text{read\_TS}(X_i)$ .
- ▶ *Multiversion Two-Phase Locking Using Certify Locks* : In this multiple-mode locking scheme, there are three locking modes for an item: read, write, and certify, instead of just the two modes (read, write) discussed previously.



# Contd..

- ▶ Hence, the state of LOCK(X) for an item X can be one of read-locked, write-locked, certify-locked, or unlocked.
  - ▶ In the standard locking scheme, with only read and write locks, a write lock is an exclusive lock. We can describe the relationship between read and write locks in the standard scheme by means of the lock compatibility table shown in Figure.
  - ▶ An entry of Yes means that if a transaction T holds the type of lock specified in the column header.
- 

# Contd..

(a)		Read	Write
	Read	Yes	No
	Write	No	No

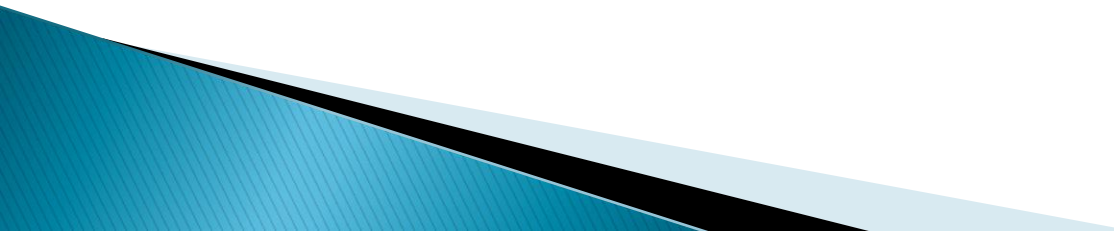
  

(b)		Read	Write	Certify
	Read	Yes	Yes	No
	Write	Yes	No	No
	Certify	No	No	No

**Figure 22.6**

Lock compatibility tables.  
(a) A compatibility table for read/write locking scheme.  
(b) A compatibility table for read/write/certify locking scheme.

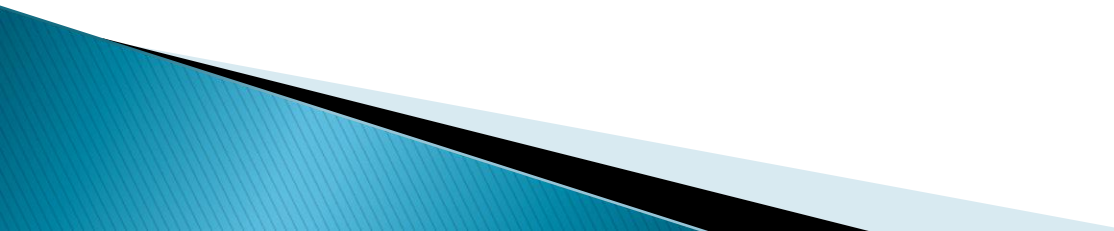
# 1.14 Recovery System

- ▶ A computer system, like any other device, is subject to failure from a variety of causes: disk crash, power outage, software error, a fire in the machine room, even sabotage.
  - ▶ In any failure, information may be lost. Therefore, the database system must take actions in advance to ensure that the atomicity and durability properties of transactions.
  - ▶ An integral part of a database system is a recovery scheme that can restore the database to the consistent state that existed before the failure.
- 

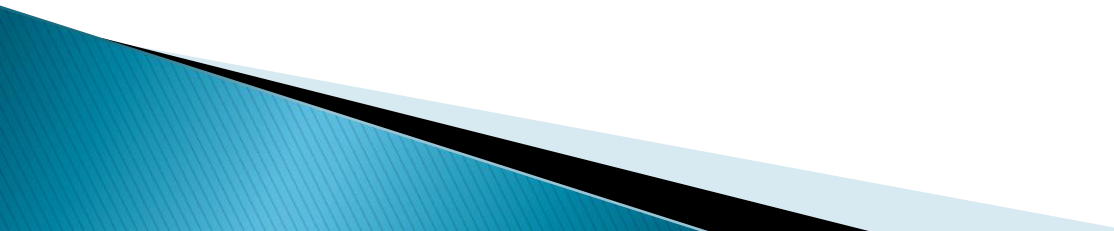
# Contd..

- ▶ The recovery scheme must also provide high availability; that is, it must minimize the time for which the database is not usable after a failure.

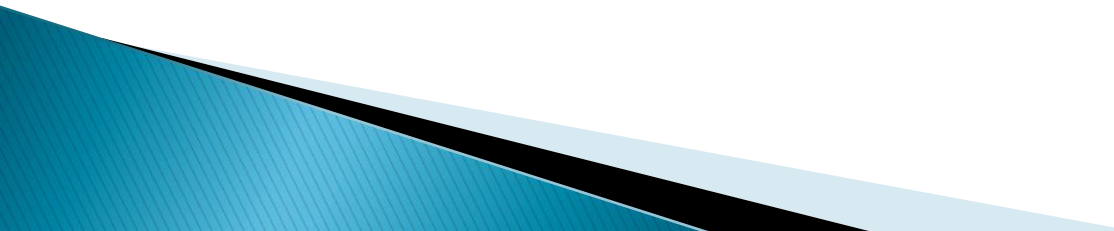
## *1.14.1 Failure Classification:*

- ▶ There are various types of failure that may occur in a system, each of which needs to be dealt with in a different manner.
  - ▶ These are following types of failure:
- 

# Contd..

- ▶ **Transaction failure**
  - ▶ **System crash**
  - ▶ **Disk failure**
- 
- ▶ *Transaction failure*: There are two types of errors that may cause a transaction to fail:
    - ▶ *Logical error*: The transaction can no longer continue with its normal execution because of some internal condition, such as bad input, data not found, overflow, or resource limit exceeded.
- 

# Contd..

- ▶ *System error*: The system has entered an undesirable state (for example, deadlock), as a result of which a transaction cannot continue with its normal execution. The transaction, however, can be re-executed at a later time.
  - ▶ *System crash*: There is a hardware malfunction, or a bug in the database software or the operating system, that causes the loss of the content of volatile storage (Internal Memory), and brings transaction processing to a halt. The content of nonvolatile storage (HDD/Pendrive) remains intact, and is not corrupted.
- 

# Contd..

- ▶ The assumption that hardware errors and bugs in the software bring the system to a halt, but do not corrupt the nonvolatile storage contents, is known as the fail-stop assumption.
- ▶ Well-designed systems have numerous internal checks, at the hardware and the software level, that bring the system to a halt when there is an error. Hence, the fail-stop assumption is a reasonable one.
- ▶ *Disk failure*: A disk block loses its content as a result of either a head crash or failure during a data-transfer operation. Copies of the data on other disks, or archival backups on tertiary media, such as DVD or tapes, are used to recover from the failure.

# Contd..

- ▶ To determine how the system should recover from failures, we need to identify the failure modes of those devices used for storing data.
- ▶ Next, we must consider how these failure modes affect the contents of the database. We can then propose algorithms to ensure database consistency and transaction atomicity despite failures. These algorithms, known as recovery algorithms, have two parts:
  - ▶ *Actions taken during normal transaction processing to ensure that enough information exists to allow recovery from failures.*





**TKR COLLEGE OF ENGINEERING & TECHNOLOGY**  
**AUTONOMOUS**

Approved By AICTE. Affiliated to JNTUH. Accredited By NBA

Accredited by NAAC with 'A' Grade

"Recognized under 2(F) and 12(B) of UGC Act 1956"

**Time/Hr : 03.40pm-04.30pm / 7<sup>th</sup>**

**Date : 18.01.2022**

**Dept / Year : DS / II**

# **DATABASE MANAGEMENT SYSTEMS**

## **UNIT - 4 Transaction Management**

### **Topic : Recovery System & Recovery and Atomicity**

**Online Platform : Google Class**

**Online Class Delivery by,**

**Dr.S.A Kalaiselvan,**


**Professor / CSE,**

**TKR College of Engineering & Technology.**

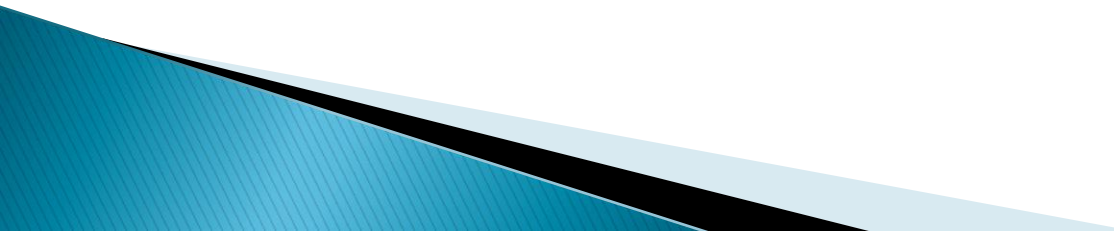
# Contd..

- ▶ *Actions taken after a failure to recover the database contents to a state that ensures database consistency, transaction atomicity, and durability.*

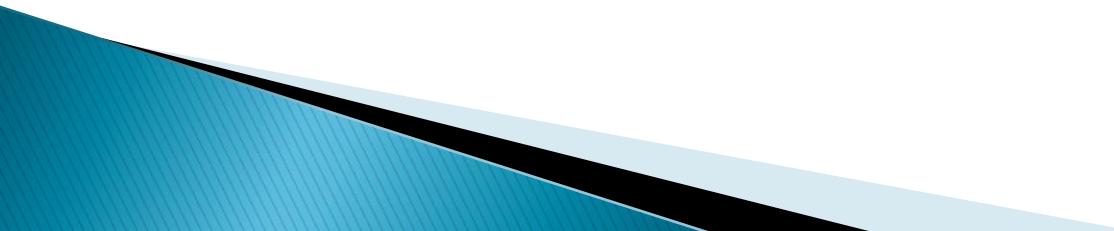
## ***1.14.2 Storage***

- ▶ There are various data items in the database may be stored and accessed in a number of different storage media.
  - ▶ The storage media can be distinguished by their relative speed, capacity, and resilience to failure. We identified three categories of storage:
- 

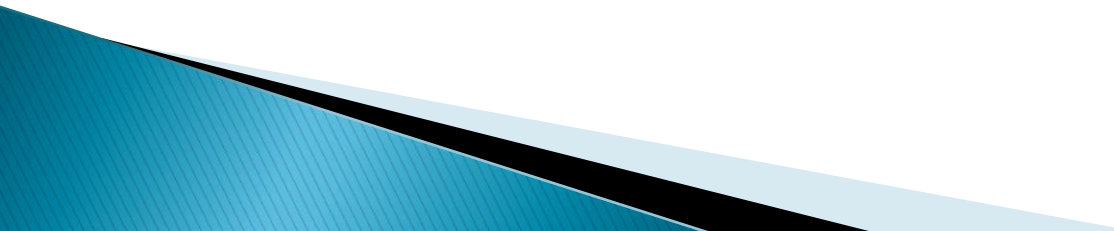
# Contd..

- ▶ Volatile storage
  - ▶ Nonvolatile storage
  - ▶ Stable storage
- 
- ▶ Stable storage or, more accurately, an approximation thereof, plays a critical role in recovery algorithms.
  - ▶ Stable storage is a classification of computer data storage technology that guarantees atomicity for any given write operation and allows software to be written that is robust against some hardware and power failures.
- 

# Contd..

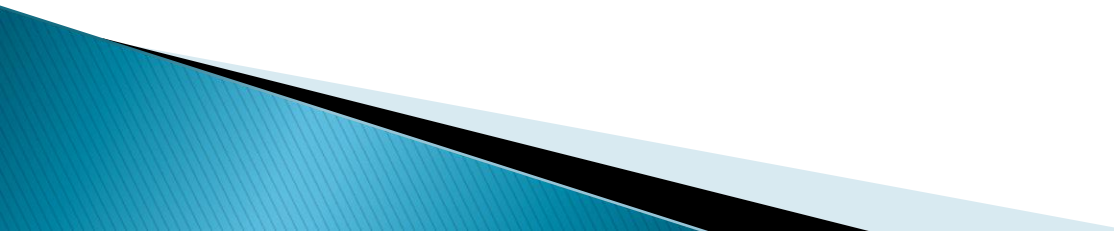
- ▶ Stable-Storage Implementation: To implement stable storage, we need to replicate the needed information in several nonvolatile storage media (usually disk) with independent failure modes, and to update the information in a controlled manner to ensure that failure during data transfer does not damage the needed information.
  - ▶ Here few steps how storage media can be protected from failure during data transfer. Block transfer between memory and disk storage can result in:
- 

# Contd..

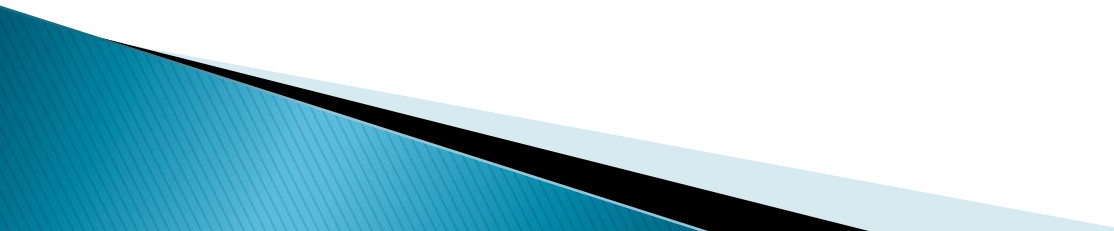
- ▶ *Successful completion.* The transferred information arrived safely at its destination.
  - ▶ *Partial failure.* A failure occurred in the midst of transfer, and the destination block has incorrect information.
  - ▶ *Total failure.* The failure occurred sufficiently early during the transfer that the destination block remains intact.
  - ▶ We require that, if a data-transfer failure occurs, the system detects it and invokes a recovery procedure to restore the block to a consistent state.
- 

# Contd..

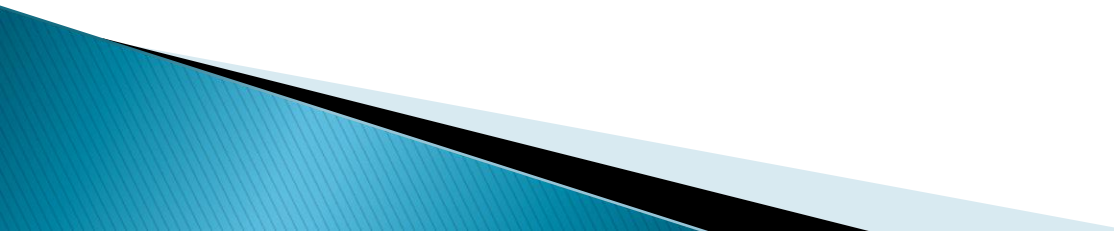
- ▶ To do so, the system must maintain two physical blocks for each logical database block; in the case of mirrored disks, both blocks are at the same location; in the case of remote backup, one of the blocks is local, whereas the other is at a remote site. An output operation is executed as follows:

1. Write the information onto the first physical block.
  2. When the first write completes successfully, write the same information onto the second physical block.
  3. The output is completed only after the second write completes successfully.
- 

# Contd..

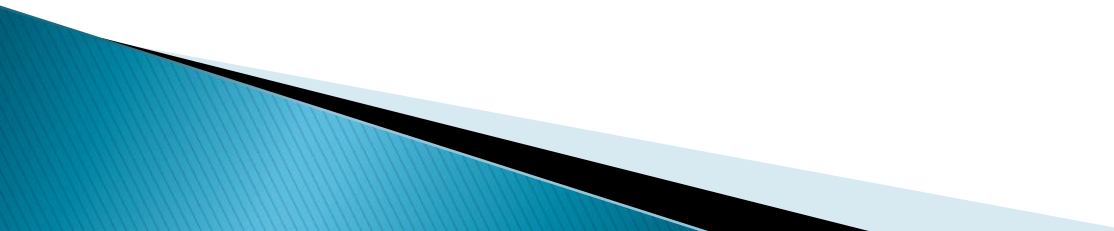
- ▶ If the system fails while blocks are being written, it is possible that the two copies of a block are inconsistent with each other.
  - ▶ During recovery, for each block, the system would need to examine two copies of the blocks. If both are the same and no detectable error exists, then no further actions are necessary.
  - ▶ Data Access: The database system resides permanently on nonvolatile storage (usually disks) and only parts of the database are in memory at any time. The database is partitioned into fixed-length storage units called blocks.
- 

# Contd..

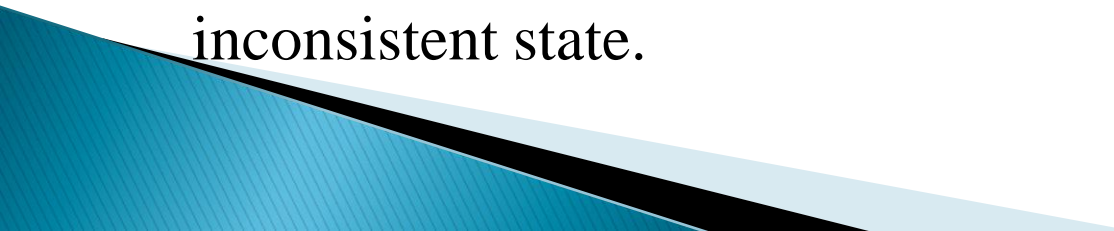
- ▶ Blocks are the units of data transfer to and from disk, and may contain several data items.
  - ▶ The blocks residing on the disk are referred to as physical blocks; the blocks residing temporarily in main memory are referred to as buffer blocks. The area of memory where blocks reside temporarily is called the disk buffer.
- 



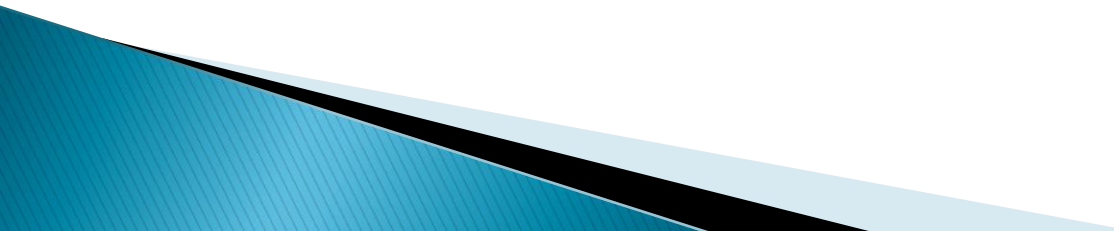
## 1.14.3 Recovery and Atomicity

- ▶ When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items.
  - ▶ Transactions are made of various operations, which are atomic in nature. But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.
  - ▶ When a DBMS recovers from a crash, it should maintain the following
- 

# Contd..

- ▶ It should check the states of **all the transactions**, which were being **executed**.
  - ▶ A transaction may be in the **middle** of some **operation**; the DBMS must ensure the atomicity of the transaction in this case.
  - ▶ It should check whether the transaction can be completed now or it needs to be **rolled back**.
  - ▶ No transactions would be allowed **to leave the DBMS** in an inconsistent state.
- 

# Contd..

- ▶ There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction.
    - ▶ Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
    - ▶ Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.
- 



**TKR COLLEGE OF ENGINEERING & TECHNOLOGY**  
**AUTONOMOUS**

Approved By AICTE. Affiliated to JNTUH. Accredited By NBA

Accredited by NAAC with 'A' Grade

"Recognized under 2(F) and 12(B) of UGC Act 1956"

**Time/Hr : 10.40am-11.30am / 2<sup>nd</sup>**

**Date : 19.01.2022**

**Dept / Year : DS / II**

# **DATABASE MANAGEMENT SYSTEMS**

## **UNIT - 4 Transaction Management**

### **Topic : Recovery and Atomicity**

**Online Platform : Google Class**

**Online Class Delivery by,**

**Dr.S.A Kalaiselvan,**

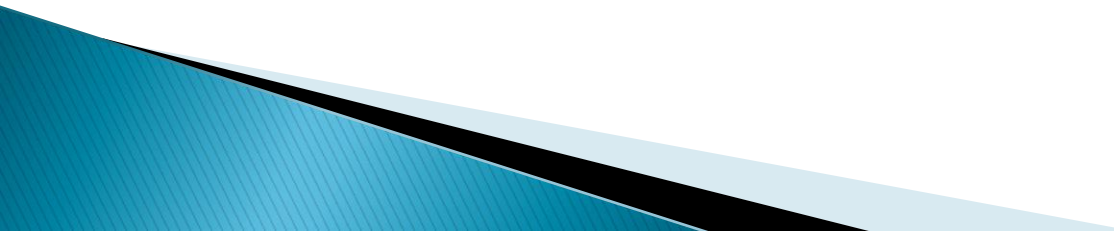
**Professor / CSE,**

**TKR College of Engineering & Technology.**

# Contd..

- ▶ Log Records: The most widely used structure for recording database modifications is the log. The log is a sequence of log records, recording all the update activities in the database.
- ▶ There are several types of log records. An update log record describes a single database write. It has these fields:
  - ▶ **Transaction identifier**, which is the unique identifier of the transaction that performed the write operation.
  - ▶ **Data-item identifier**, which is the unique identifier of the data item written. Typically, it is the location on disk of the data item, consisting of the block identifier of the block on which the data item resides, and an offset within the block.

# Contd..

- ▶ **Old value**, which is the value of the data item prior to the write.
  - ▶ **New value**, which is the value that the data item will have after the write.
  - ▶ We represent an update log record as  $\langle T_i, X_j, V_1, V_2 \rangle$ , indicating that transaction  $T_i$  has performed a write on data item  $X_j$ .  $X_j$  had value  $V_1$  before the write, and has value  $V_2$  after the write.
- 

# Contd..

- ▶ Other special log records exist to record significant events during transaction processing, such as the start of a transaction and the commit or abort of a transaction. Among the types of log records are:
  - ▶  $\langle T_i \text{ start} \rangle$ . Transaction  $T_i$  has started.
  - ▶  $\langle T_i \text{ commit} \rangle$ . Transaction  $T_i$  has committed.
  - ▶  $\langle T_i \text{ abort} \rangle$ . Transaction  $T_i$  has aborted.
- ▶ Log records to be useful for recovery from system and disk failures, the log must reside in stable storage.

# Contd..

- ▶ Database Modification: A transaction creates a log record prior to modifying the database.
- ▶ The log records allow the system to undo changes made by a transaction in the event that the transaction must be aborted; they allow the system also to redo changes made by a transaction if the transaction has committed but the system crashed before those changes could be stored in the database on disk.
- ▶ All database modifications must be preceded by the creation of a log record, the system has available both the old value prior to the modification of the data item and the new value that is to be written for the data item. This allows the system to perform undo and redo operations as appropriate.



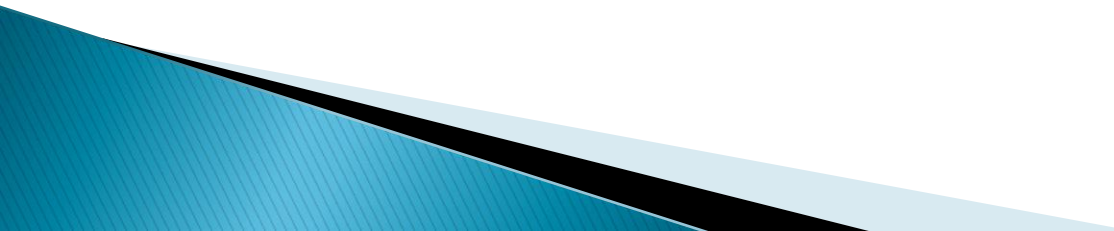
# Contd..

- ▶ **Undo** using a log record sets the data item specified in the log record to the old value.
- ▶ **Redo** using a log record sets the data item specified in the log record to the new value.
- ▶ Concurrency Control and Recovery: If the concurrency control scheme allows a data item X that has been modified by a transaction T1 to be further modified by another transaction T2 before T1 commits, then undoing the effects of T1 by restoring the old value of X (before T1 updated X) would also undo the effects of T2.

# Contd..

- ▶ To avoid such situations, recovery algorithms usually require that if a data item has been modified by a transaction, no other transaction can modify the data item until the first transaction commits or aborts.
- ▶ This requirement can be ensured by acquiring an exclusive lock on any updated data item and holding the lock until the transaction commits.
- ▶ Transaction Commit: We say that a transaction has committed when its commit log record, which is the last log record of the transaction, has been output to stable storage; at that point all earlier log records have already been output to stable storage.

# Contd..

- ▶ Thus, there is enough information in the log to ensure that even if there is a system crash, the updates of the transaction can be redone.
  - ▶ If a system crash occurs before a log record  $\langle T_i \text{ commit} \rangle$  is output to stable storage, transaction  $T_i$  will be rolled back. Thus, the output of the block containing the commit log record is the single atomic action that results in a transaction getting committed.
- 

## Contd..

- ▶ *Using the Log to Redo and Undo Transactions:* Consider our simplified banking system. Let  $T_0$  be a transaction that transfers \$50 from account A to account B:

```
<T0 start>  
<T0, A, 1000, 950>  
<T0, B, 2000, 2050>  
<T0 commit>
```

Figure 16.2 Portion of the system log corresponding to  $T_0$

# Contd..

```
T0: read(A);  
      A := A - 50;  
      write(A);  
      read(B);  
      B := B + 50;  
      write(B).
```

Let  $T_1$  be a transaction that withdraws \$100 from account C:

```
T1: read(C);  
      C := C - 100;  
      write(C).
```

# Contd..

- ▶ State of system log and database corresponding to  $T_0$  and  $T_1$ .

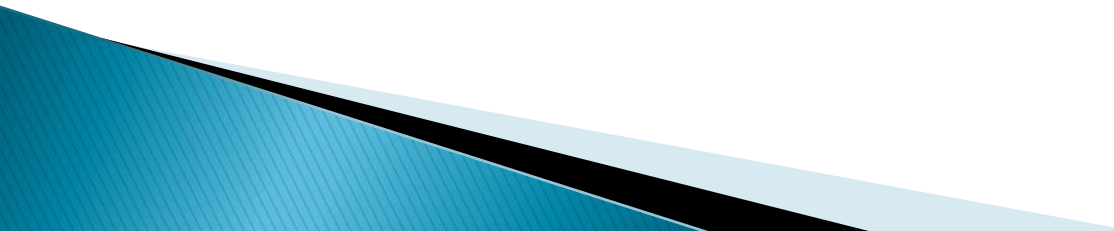
Log	Database
$\langle T_0 \text{ start} \rangle$	
$\langle T_0, A, 1000, 950 \rangle$	
$\langle T_0, B, 2000, 2050 \rangle$	
	$A = 950$
	$B = 2050$
$\langle T_0 \text{ commit} \rangle$	
$\langle T_1 \text{ start} \rangle$	
$\langle T_1, C, 700, 600 \rangle$	
	$C = 600$
$\langle T_1 \text{ commit} \rangle$	

Figure 16.3 State of system log and database corresponding to  $T_0$  and  $T_1$ .

## 1.14.3 Recovery Algorithm

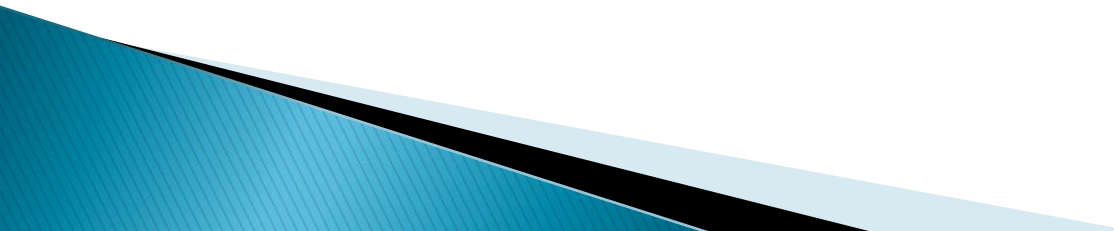
- ▶ We have identified transactions that need to be redone and those that need to be undone, but we have not given a precise algorithm for performing these actions.
- ▶ Transaction Rollback: First consider transaction rollback during normal operation. Rollback of a transaction  $T_i$  is performed as follows:
  - ▶ The log is scanned backward, and for each log record of  $T_i$  of the form  $\langle T_i, X_j, V_1, V_2 \rangle$ .
  - ▶ Once the log record  $\langle T_i \text{ start} \rangle$  is found the backward scan is stopped, and a log record  $\langle T_i \text{ abort} \rangle$  is written to the log.

# Contd..

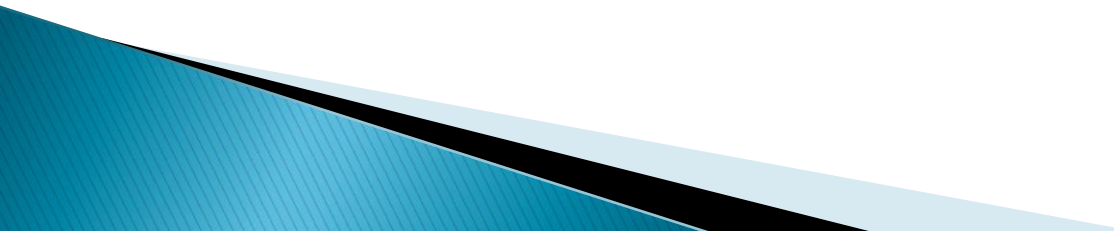
- ▶ Observe that every update action performed by the transaction or on behalf of the transaction, including actions taken to restore data items to their old value.
  - ▶ Recovery After a System Crash: Recovery actions, when the database system is restarted after a crash, take place in two phases:
    - ▶ redo phase
    - ▶ undo phase
- 



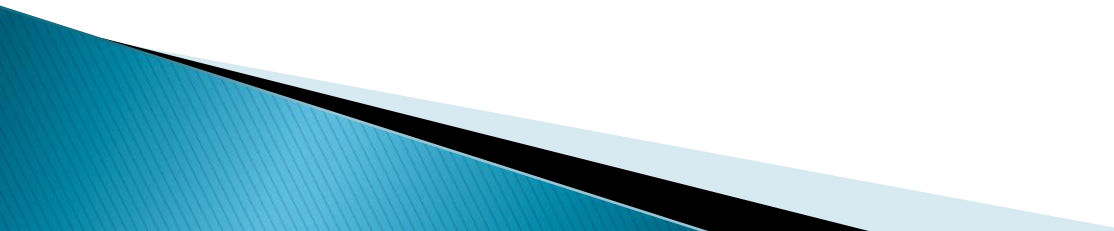
# Contd..

- ▶ In the **redo phase**, the system replays updates of all transactions by scanning the log forward from the last checkpoint.
  - ▶ The log records that are replayed include log records for transactions that were rolled back before system crash, and those that had not committed when the system crash occurred.
  - ▶ This phase also determines all transactions that were incomplete at the time of the crash, and must therefore be rolled back.
- 

# Contd..

- ▶ At the end of the redo phase, undo-list contains the list of all transactions that are incomplete, that is, they neither committed nor completed rollback before the crash.
  - ▶ In the **undo phase**, the system rolls back all transactions in the undo-list. It performs rollback by scanning the log backward from the end.
  - ▶ Whenever it finds a log record belonging to a transaction in the undolist, it performs undo actions just as if the log record had been found during the rollback of a failed transaction.
- 

# Contd..

- ▶ When the system finds a  $\langle T_i \text{ start} \rangle$  log record for a transaction  $T_i$  in undo-list, it writes a  $\langle T_i \text{ abort} \rangle$  log record to the log, and removes  $T_i$  from undo-list.
  - ▶ The undo phase terminates once undo-list becomes empty, that is, the system has found  $\langle T_i \text{ start} \rangle$  log records for all transactions that were initially in undo-list.
  - ▶ After the undo phase of recovery terminates, normal transaction processing can resume.
- 



**TKR COLLEGE OF ENGINEERING & TECHNOLOGY**  
**AUTONOMOUS**

Approved By AICTE. Affiliated to JNTUH. Accredited By NBA

Accredited by NAAC with 'A' Grade

"Recognized under 2(F) and 12(B) of UGC Act 1956"

**Time/Hr : 02pm-02.50pm / 5<sup>th</sup>**

**Date : 20.01.2022**

**Dept / Year : DS / II**

# **DATABASE MANAGEMENT SYSTEMS**

## **UNIT - 4 Transaction Management**

### **Topic : Buffer Management**

**Online Platform : Google Class**

**Online Class Delivery by,**

**Dr.S.A Kalaiselvan,**

**Professor / CSE,**

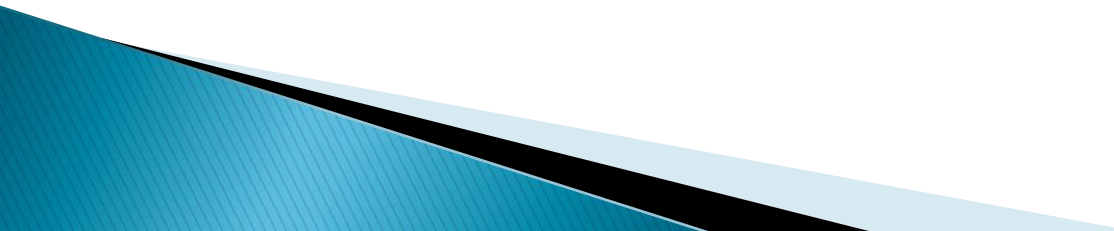
**TKR College of Engineering & Technology.**

## 1.14.4 Buffer Management

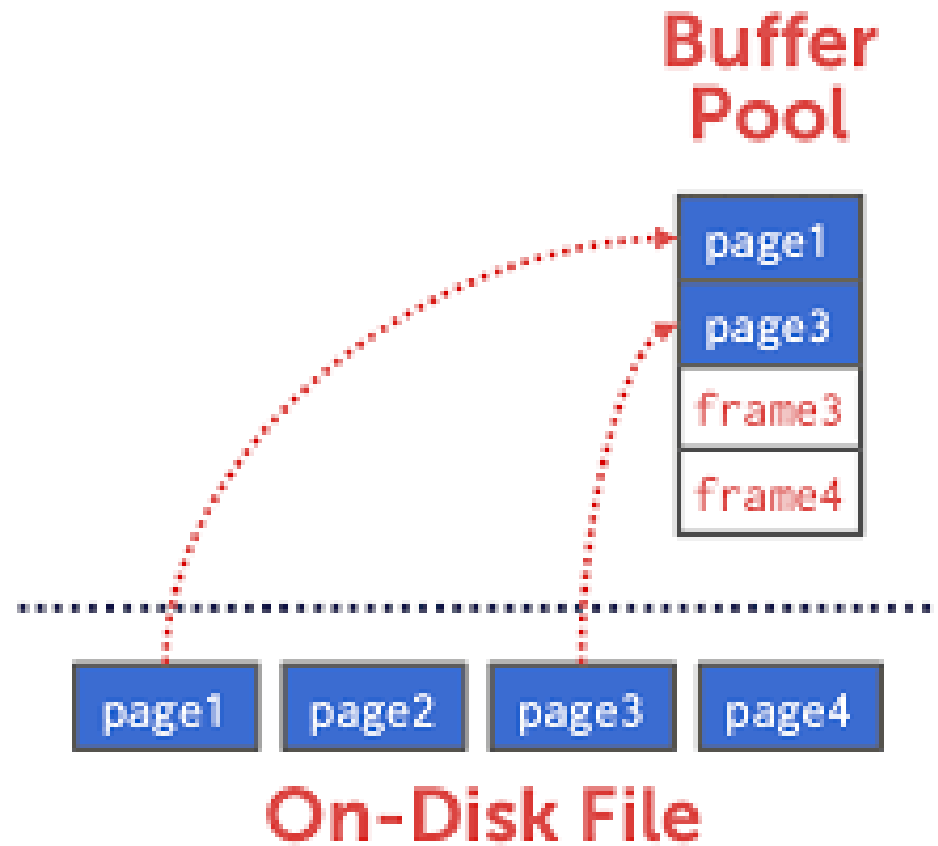
- ▶ The buffer manager is the software layer that is responsible for bringing pages from physical disk to main memory as needed.
- ▶ The buffer manages the available main memory by dividing the main memory into a collection of pages, which we called as buffer pool. The main memory pages in the buffer pool are called frames.
- ▶ A database buffer is a temporary storage area in the main memory. It allows storing the data temporarily when moving from one place to another. A database buffer stores a copy of disk blocks. But, the version of block copies on the disk may be older than the version in the buffer.

# Contd..

## What is Buffer Manager?

- ▶ A Buffer Manager is responsible for allocating space to the buffer in order to store data into the buffer.
  - ▶ If a user request a particular block and the block is available in the buffer, the buffer manager provides the block address in the main memory.
  - ▶ If the block is not available in the buffer, the buffer manager allocates the block in the buffer.
- 

# Contd..

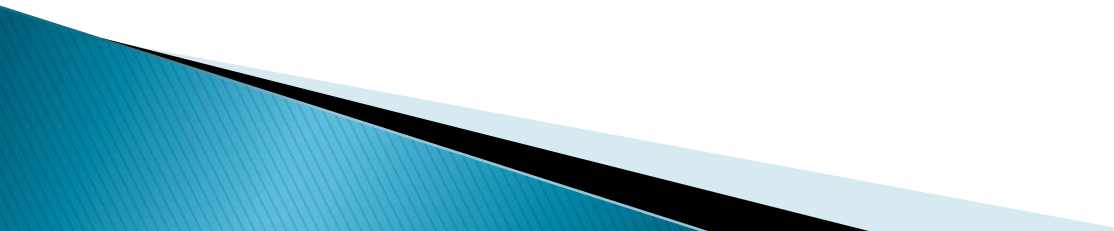


# Contd..

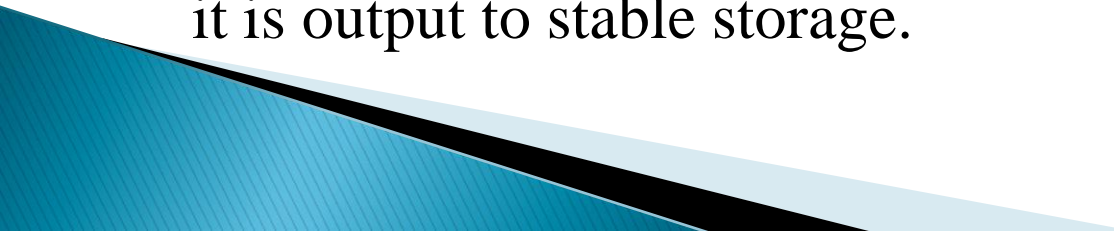
- ▶ If free space is not available, it throws out some existing blocks from the buffer to allocate the required space for the new block.
- ▶ If the user requests such thrown-out blocks, the buffer manager reads the requested block from the disk to the buffer and then passes the address of the requested block to the user in the main memory.
- However, the internal actions of the buffer manager are not visible to the programs that may create any problem in disk-block requests. The buffer manager is just like a virtual machine.



# Contd..

- ▶ Log-Record Buffering: So far, we have assumed that every log record is output to stable storage at the time it is created.
  - ▶ Typically, output to stable storage is in units of blocks. In most cases, a log record is much smaller than a block. Thus, the output of each log record translates to a much larger output at the physical level.
  - ▶ The cost of outputting a block to stable storage is sufficiently high that it is desirable to output multiple log records at once.
- 

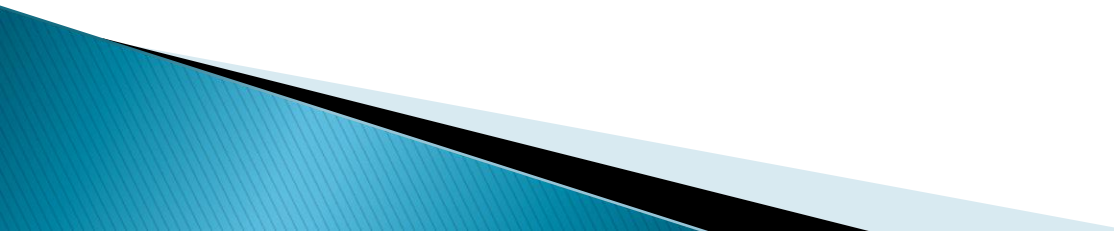
# Contd..

- ▶ To do so, we write log records to a log buffer in main memory, where they stay temporarily until they are output to stable storage.
  - ▶ Multiple log records can be gathered in the log buffer and output to stable storage in a single output operation. The order of log records in the stable storage must be exactly the same as the order in which they were written to the log buffer.
  - ▶ As a result of log buffering, a log record may reside in only main memory (volatile storage) for a considerable time before it is output to stable storage.
- 

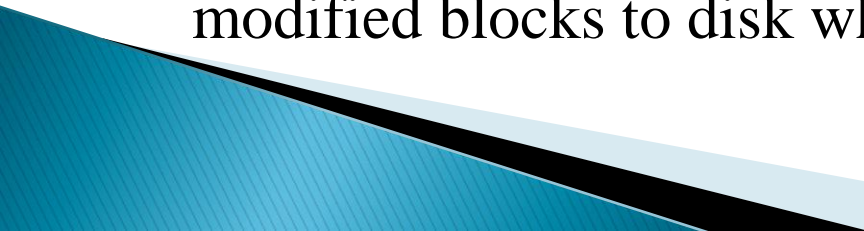
# Contd..

- ▶ Transaction  $T_i$  enters the commit state after the  $\langle T_i \text{ commit} \rangle$  log record has been output to stable storage.
- ▶ Before the  $\langle T_i \text{ commit} \rangle$  log record can be output to stable storage, all log records pertaining to transaction  $T_i$  must have been output to stable storage.
- ▶ Before a block of data in main memory can be output to the database (in nonvolatile storage), all log records pertaining to data in that block must have been output to stable storage.
- ▶ This rule is called the write-ahead logging (WAL) rule

# Contd..

- ▶ The three rules state situations in which certain log records must have been output to stable storage. There is no problem resulting from the output of log records earlier than necessary.
  - ▶ Thus, when the system finds it necessary to output a log record to stable storage, it outputs an entire block of log records, if there are enough log records in main memory to fill a block.
  - ▶ Writing the buffered log to disk is sometimes referred to as a log force.
- 

# Contd..


- ▶ Database Buffering: The system stores the database in nonvolatile storage (disk), and brings blocks of data into main memory as needed. Since main memory is typically much smaller than the entire database, it may be necessary to overwrite a block B1 in main memory when another block B2 needs to be brought into memory
  - ▶ If B1 has been modified, B1 must be output prior to the input of B2.
  - ▶ One might expect that transactions would force-output all modified blocks to disk when they commit.
- 

# Contd..

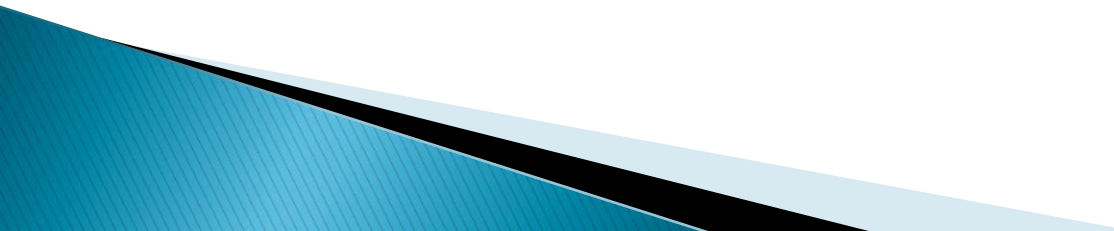
- ▶ Such a policy is called the force policy. The alternative, the no-force policy, allows a transaction to commit even if it has modified some blocks that have not yet been written back to disk.
- ▶ It is important that no writes to the block B1 be in progress while the block is being output, since such a write could violate the write-ahead logging rule. We can ensure that there are no writes in progress by using a special means of locking:
  - ▶ *Before a transaction performs a write on a data item, it acquires an exclusive lock on the block in which the data item resides. The lock is released immediately after the update has been performed.*

# Contd..

The following sequence of actions is taken when a block is to be output


- Obtain an exclusive lock on the block, to ensure that no transaction is performing a write on the block.
  - Output log records to stable storage until all log records pertaining to block B1 have been output.
  - Output block B1 to disk.
  - Release the lock once the block output has completed.
- 
- Locks on buffer blocks can also be used to ensure that buffer blocks are not updated, and log records are not generated, while a checkpoint is in progress.
- 

# Contd..

- ▶ This restriction may be enforced by acquiring exclusive locks on all buffer blocks, as well as an exclusive lock on the log, before the checkpoint operation is performed.
  - ▶ These locks can be released as soon as the checkpoint operation has completed.
  - ▶ Operating System Role in Buffer Management: We can manage the database buffer by using one of two approaches:
- 



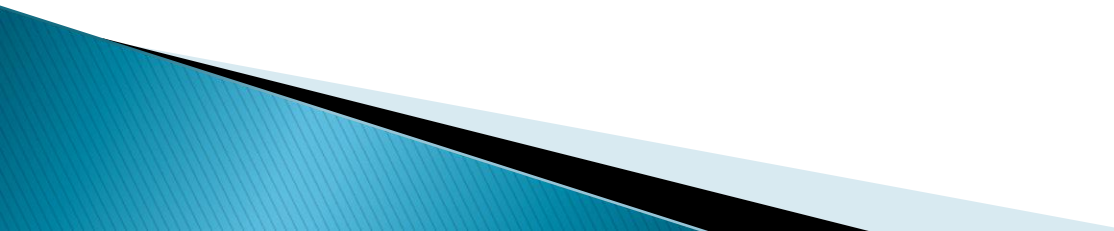
# Contd..

1. The database system reserves part of main memory to serve as a buffer that it, rather than the operating system, manages. The database system manages data-block transfer in accordance with the requirements.
    - ▶ *This approach has the drawback of limiting flexibility in the use of main memory. The buffer must be kept small enough that other applications have sufficient main memory available for their needs.*
  2. The database system implements its buffer within the virtual memory provided by the operating system.
- 

# Contd..

- ▶ Since the operating system knows about the memory requirements of all processes in the system, ideally it should be in charge of deciding what buffer blocks must be force-output to disk, and when.
- ▶ Fuzzy Checkpointing: If the number of pages in the buffer is large, a checkpoint may take a long time to finish, which can result in an unacceptable interruption in processing of transactions.
- ▶ To avoid such interruptions, the check pointing technique can be modified to permit updates to start once the checkpoint record has been written.

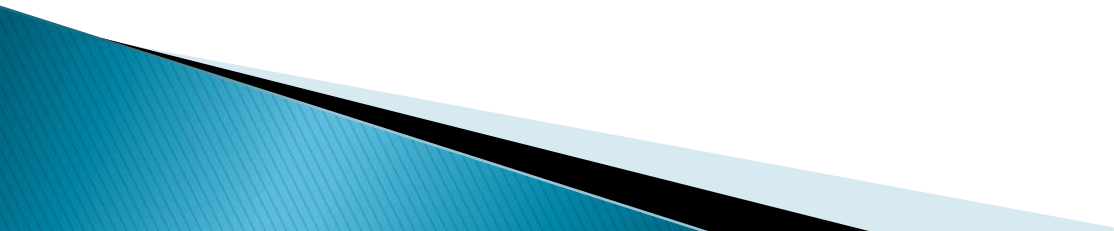
# Contd..

- ▶ Since pages are output to disk only after the checkpoint record has been written, it is possible that the system could crash before all pages are written.
  - ▶ One way to deal with incomplete checkpoints is this: The location in the log of the checkpoint record of the last completed checkpoint is stored in a fixed position, last-checkpoint, on disk.
  - ▶ The system does not update this information when it writes the checkpoint record.
- 

# Contd..

- ▶ The last-checkpoint information is updated only after all buffer blocks in the list of modified buffer blocks have been output to disk.

## 1.14.5 Failure with Loss of Nonvolatile Storage

- ▶ Until now, we have considered only the case where a failure results in the loss of information residing in volatile storage while the content of the nonvolatile storage remains intact.
  - ▶ The basic scheme is to dump the entire contents of the database to stable storage periodically.
  - ▶ For example, we may dump the database to one or more magnetic tapes. If a failure occurs that results in the loss of physical database blocks, the system uses the most recent dump in restoring the database to a previous consistent state.
- 

# Contd..

- ▶ One approach to database dumping requires that no transaction may be active during the dump procedure, and uses a procedure similar to check pointing:
  - ▶ 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 log record <dump> onto the stable storage.

# Contd..

- ▶ To recover from the loss of nonvolatile storage, the system restores the database to disk by using the most recent dump. Then, it consults the log and redoes all the actions since the most recent dump occurred. Notice that no undo operations need to be executed.
- ▶ In case of a partial failure of nonvolatile storage, such as the failure of a single block or a few blocks, only those blocks need to be restored, and redo actions performed only for those blocks.
- ▶ A dump of the database contents is also referred to as an archival dump, since we can archive the dumps and use them later to examine old states of the database.

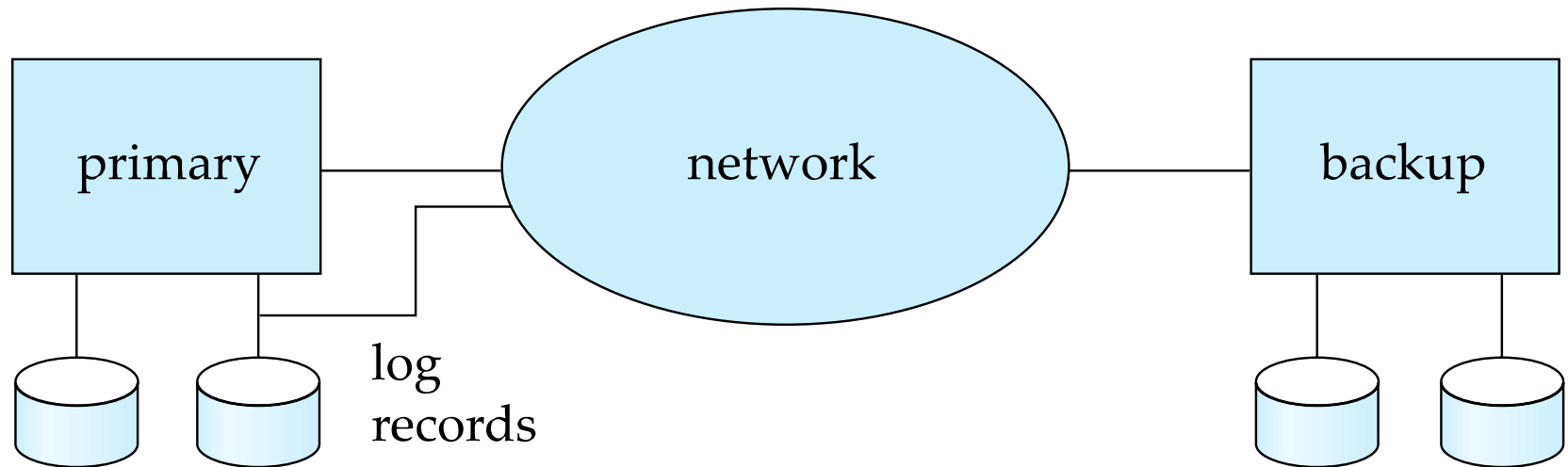
## **1.14.6 Early Lock Release and Logical Undo Operations**

- ▶ Need to study B+tree which comes in UNIT V



## 1.14.7 Remote Backup systems

- ▶ Remote backup systems provide high availability by allowing transaction processing to continue even if the primary site is destroyed.



# Contd..

- ▶ Detection of failure: Backup site must detect when primary site has failed
  - ▶ to distinguish primary site failure from link failure maintain several communication links between the primary and the remote backup.
  - ▶ Heart-beat messages.
- ▶ Transfer of control:
  - ▶ To take over control backup site first perform recovery using its copy of the database and all the long records it has received from the primary.
  - ▶ Thus, completed transactions are redone and incomplete transactions are rolled back.

# Contd..

- ▶ When the backup site takes over processing it becomes the new primary
- ▶ To transfer control back to old primary when it recovers, old primary must receive redo logs from the old backup and apply all updates locally.
- ▶ Time to recover: To reduce delay in takeover, backup site periodically processes the redo log records (in effect, performing recovery from previous database state), performs a checkpoint, and can then delete earlier parts of the log.

# Contd..

- ▶ Hot-Spare configuration permits very fast takeover:
  - ▶ Backup continually processes redo log record as they arrive, applying the updates locally.
  - ▶ When failure of the primary is detected the backup rolls back incomplete transactions, and is ready to process new transactions.
- ▶ Alternative to remote backup: distributed database with replicated data
  - ▶ Remote backup is faster and cheaper, but less tolerant to failure
  - ▶ Ensure durability of updates by delaying transaction commit until update is logged at backup; avoid this delay by permitting lower degrees of durability.

# Contd..

- One-safe: commit as soon as transaction's commit log record is written at primary
  - Problem: updates may not arrive at backup before it takes over.
- Two-very-safe: commit when transaction's commit log record is written at primary and backup
  - Reduces availability since transactions cannot commit if either site fails.
- Two-safe: proceed as in two-very-safe if both primary and backup are active. If only the primary is active, the transaction commits as soon as its commit log record is written at the primary.
  - Better availability than two-very-safe; avoids problem of lost transactions in one-safe

**UNIT 4 COMPLETED**