Lecture Notes on

4AID4-05
Database Management System

Unit 5

Department of Artificial Intelligence & Data Science
Jaipur Engineering College & Research Centre, Jaipur

Neelkamal Chaudhary

Assistant Professor

AI&DS

Vision of the Institute

To become a renowned centre of outcome based learning and work toward academic, professional, cultural and social enrichment of the lives of individuals and communities.

Mission of the Institute

M1: Focus on evaluation of learning outcomes and motivate students to inculcate research aptitude by project based learning.
M2: Identify, based on informed perception of Indian, regional and global needs, the areas of focus and provide platform to gain knowledge and solutions.
M3: Offer opportunities for interaction between academia and industry.
M4: Develop human potential to its fullest extent so that intellectually capable and imaginatively gifted leaders can emerge in a range of professions.

Vision Of The Department

To prepare students in the field of Artificial Intelligence and Data Science for competing with the global perspective through outcome based education, research and innovation.

Mission Of The Department

1. To impart outcome based education in the area of AI&DS.
2. To provide platform to the experts from institutions and industry of repute to transfer the knowledge to students for providing competitive and sustainable solutions.
3. To provide platform for innovation and research.

Program Outcomes (PO)

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and Artificial Intelligence & Data Science specialization to the solution of complex Artificial Intelligence & Data Science problems.
2. **Problem analysis**: Identify, formulate, research literature, and analyze complex Artificial Intelligence & Data Science problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex Artificial Intelligence & Data Science problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of Artificial Intelligence & Data Science experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex Artificial Intelligence & Data Science activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional Artificial Intelligence & Data Science practice.
7. **Environment and sustainability**: Understand the impact of the professional Artificial Intelligence & Data Science in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the Artificial Intelligence & Data Science practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings in Artificial Intelligence & Data Science
10. **Communication**: Communicate effectively on complex Artificial Intelligence & Data Science activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the Artificial Intelligence & Data Science and management principles and apply these to one"s own work, as a

member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change in Artificial Intelligence & Data Science.

Program Educational Objectives (PEO)

PEO1: To provide students with the fundamentals of Engineering Sciences with more emphasis in Artificial Intelligence & Data Science by way of analyzing and exploiting engineering challenges.

PEO2: To train students with good scientific and engineering knowledge so as to comprehend, analyze, design, and create novel products and solutions for the real life problems in Artificial Intelligence & Data Science

PEO3: To inculcate professional and ethical attitude, effective communication skills, teamwork skills, multidisciplinary approach, entrepreneurial thinking and an ability to relate engineering issues with social issues for Artificial Intelligence & Data Science.

PEO4: To provide students with an academic environment aware of excellence, leadership, written ethical codes and guidelines, and the self-motivated life-long learning needed for a successful professional career in Artificial Intelligence & Data Science.

PEO5: To prepare students to excel in Industry and Higher education by Educating Students along with High moral values and Knowledge in Artificial Intelligence & Data Science.

COURSE OUTCOME: After studying this subject, student will be able

CO-1	Design an ER model for an enterprise
CO-2	Perform and analysis Query database using Relational Algebra, Relational Calculus and SQL
CO-3	Apply normalization based on functional dependency.
C0-4	Illustrate for serialzability among concurrent transactions and apply concurrency control protocols, and Outline database recovery techniques

CO_PO Mapping

SUBJECT CODE	**subject name**		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
4AID4-05	Database Management System	CO-1	3	3	3	3	3	2	1	2	1	2	2	2
		CO-2	3	3	3	3	3	2	1	1	1	2	2	2
		CO-3	3	3	3	3	3	2	1	1	1	2	2	2
		CO-4	3	3	3	2	2	2	1	1	1	2	2	2

4AID4-05: Database Management System

Credit: 3 **Max. Marks: 100(IA:30, ETE:70)**

3L+0T+0P **End Term Exam: 3 Hours**

SN	Contents	Hours
1	**Introduction:** Objective, scope and outcome of the course.	**1**
2	**Introduction to database systems:** Overview and History of DBMS. File System v/s DBMS.Advantage of DBMS Describing and Storing Data in a DBMS.Queries in DBMS.Structure of a DBMS. **Entity Relationship model:** Overview of Data Design Entities, Attributes and Entity Sets, Relationship and Relationship Sets. Features of the ER Model- Key Constraints, Participation Constraints, Weak Entities, Class Hierarchies, Aggregation, Conceptual Data Base, and Design with ER Model- Entity v/s Attribute, Entity vs Relationship Binary vs Ternary Relationship and Aggregation v/s ternary Relationship Conceptual Design for a Large Enterprise.	**7**
3	**Relationship Algebra and Calculus:** Relationship Algebra Selection and Projection, Set Operations, Renaming, Joints, Division, Relation Calculus, Expressive Power of Algebra and Calculus. **SQL queries programming and Triggers:** The Forms of a Basic SQL Query, Union, and Intersection and Except, Nested Queries, Correlated Nested Queries, Set-Comparison Operations, Aggregate Operators, Null Values and Embedded SQL, Dynamic SQL, ODBC and JDBC, Triggers and Active Databases.	**8**
4	**Schema refinement and Normal forms:** Introductions to Schema Refinement, Functional Dependencies, Boyce-Codd Normal Forms, Third Normal Form, Normalization-Decomposition into BCNF Decomposition into 3-NF.	**8**
5	**Transaction Processing:** Introduction-Transaction State, Transaction properties, Concurrent Executions. Need of Serializability, Conflict vs. View Serializability, Testing for Serializability, Recoverable Schedules, Cascadeless Schedules.	**8**
6	**Concurrency Control: Implementation of Concurrency:** Lock-based protocols, Timestamp-based protocols, Validation-based protocols, Deadlock handling, **Database Failure and Recovery:** Database Failures, Recovery Schemes: Shadow Paging and Log-based Recovery, Recovery with Concurrent transactions.	**8**
	Total	**40**

Unit 5
DBMS

Concurrency Control in Database Management System is a procedure of managing simultaneous operations without conflicting with each other.
It ensures that Database transactions are performed concurrently and accurately to produce correct results without violating data integrity of the respective Database.
Concurrent access is quite easy if all users are just reading data.

There is no way they can interfere with one another.

Though for any practical Database, it would have a mix of READ and WRITE operations and hence the concurrency is a challenge.

DBMS Concurrency Control is used to address such conflicts, which mostly occur with a multi-user system.

Therefore, Concurrency Control is the most important element for proper functioning of a Database Management System where two or more database transactions are executed simultaneously, which require access to the same data.

Potential problems of Concurrency

Here, are some issues which you will likely to face while using the DBMS Concurrency Control method:

- **Lost Updates** occur when multiple transactions select the same row and update the row based on the value selected
- Uncommitted dependency issues occur when the second transaction selects a row which is updated by another transaction (**dirty read**)
- **Non-Repeatable Read** occurs when a second transaction is trying to access the same row several times and reads different data each time.
- **Incorrect Summary issue** occurs when one transaction takes summary over the value of all the instances of a repeated data-item, and second transaction update few instances of that specific data-item. In that situation, the resulting summary does not reflect a correct result.

Why use Concurrency method?

Reasons for using Concurrency control method is DBMS:

- To apply Isolation through mutual exclusion between conflicting transactions
- To resolve read-write and write-write conflict issues
- To preserve database consistency through constantly preserving execution obstructions
- The system needs to control the interaction among the concurrent transactions. This control is achieved using concurrent-control schemes.
- Concurrency control helps to ensure serializability

Example

Assume that two people who go to electronic kiosks at the same time to buy a movie ticket for the same movie and the same show time.

However, there is only one seat left in for the movie show in that particular theatre. Without concurrency control in DBMS, it is possible that both moviegoers will end up purchasing a ticket. However, concurrency control method does not allow this to happen. Both moviegoers can still access information written in the movie seating database. But concurrency control only provides a ticket to the buyer who has completed the transaction process first.

Concurrency Control Protocols

Different concurrency control protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose. Following are the Concurrency Control techniques in DBMS:

- Lock-Based Protocols
- Two Phase Locking Protocol
- Timestamp-Based Protocols
- Validation-Based Protocols

Lock-based Protocols

Lock Based Protocols in DBMS is a mechanism in which a transaction cannot Read or Write the data until it acquires an appropriate lock. Lock based protocols help to eliminate the concurrency problem in DBMS for simultaneous transactions by locking or isolating a particular transaction to a single user.
A lock is a data variable which is associated with a data item. This lock signifies that operations that can be performed on the data item. Locks in DBMS help synchronize access to the database items by concurrent transactions.

All lock requests are made to the concurrency-control manager. Transactions proceed only once the lock request is granted.

Binary Locks: A Binary lock on a data item can either locked or unlocked states.

Shared/exclusive: This type of locking mechanism separates the locks in DBMS based on their uses. If a lock is acquired on a data item to perform a write operation, it is called an exclusive lock.

1. Shared Lock (S):

A shared lock is also called a Read-only lock. With the shared lock, the data item can be shared between transactions. This is because you will never have permission to update data on the data item.

For example, consider a case where two transactions are reading the account balance of a person. The database will let them read by placing a shared lock. However, if another transaction wants to update that account's balance, shared lock prevent it until the reading process is over.

2. Exclusive Lock (X):

With the Exclusive Lock, a data item can be read as well as written. This is exclusive and can't be held concurrently on the same data item. X-lock is requested using lock-x instruction. Transactions may unlock the data item after finishing the 'write' operation.

For example, when a transaction needs to update the account balance of a person. You can allows this transaction by placing X lock on it. Therefore, when the second transaction wants to read or write, exclusive lock prevent this operation.

3. Simplistic Lock Protocol

This type of lock-based protocols allows transactions to obtain a lock on every object before beginning operation. Transactions may unlock the data item after finishing the 'write' operation.

4. Pre-claiming Locking

Pre-claiming lock protocol helps to evaluate operations and create a list of required data items which are needed to initiate an execution process. In the situation when all locks are granted, the transaction executes. After that, all locks release when all of its operations are over.

Starvation

Starvation is the situation when a transaction needs to wait for an indefinite period to acquire a lock.

Following are the reasons for Starvation:

- When waiting scheme for locked items is not properly managed
- In the case of resource leak
- The same transaction is selected as a victim repeatedly

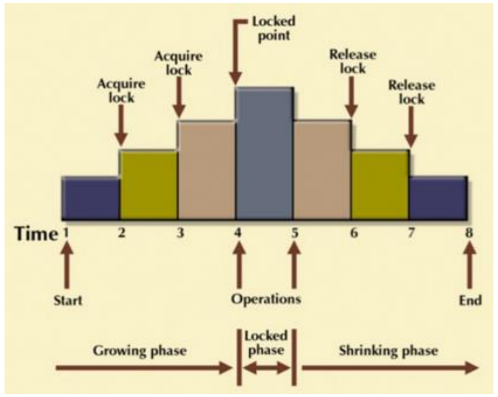
Deadlock

Deadlock refers to a specific situation where two or more processes are waiting for each other to release a resource or more than two processes are waiting for the resource in a circular chain.

Two Phase Locking Protocol

Two Phase Locking Protocol also known as 2PL protocol is a method of concurrency control in DBMS that ensures serializability by applying a lock to the transaction data which blocks other transactions to access the same data simultaneously. Two Phase Locking protocol helps to eliminate the concurrency problem in DBMS.
This locking protocol divides the execution phase of a transaction into three different parts.

- In the first phase, when the transaction begins to execute, it requires permission for the locks it needs.
- The second part is where the transaction obtains all the locks. When a transaction releases its first lock, the third phase starts.
- In this third phase, the transaction cannot demand any new locks. Instead, it only releases the acquired locks.



The Two-Phase Locking protocol allows each transaction to make a lock or unlock request in two steps:

- **Growing Phase**: In this phase transaction may obtain locks but may not release any locks.
- **Shrinking Phase**: In this phase, a transaction may release locks but not obtain any new lock

It is true that the 2PL protocol offers serializability. However, it does not ensure that deadlocks do not happen.

In the above-given diagram, you can see that local and global deadlock detectors are searching for deadlocks and solve them with resuming transactions to their initial states.

Strict Two-Phase Locking Method

Strict-Two phase locking system is almost similar to 2PL. The only difference is that Strict-2PL never releases a lock after using it. It holds all the locks until the commit point and releases all the locks at one go when the process is over.

Centralized 2PL

In Centralized 2 PL, a single site is responsible for lock management process. It has only one lock manager for the entire DBMS.

Primary copy 2PL

Primary copy 2PL mechanism, many lock managers are distributed to different sites. After that, a particular lock manager is responsible for managing the lock for a set of data items. When the primary copy has been updated, the change is propagated to the slaves.

Distributed 2PL
In this kind of two-phase locking mechanism, Lock managers are distributed to all sites. They are responsible for managing locks for data at that site. If no data is replicated, it is equivalent to primary copy 2PL. Communication costs of Distributed 2PL are quite higher than primary copy 2PL

Timestamp-based Protocols

Timestamp based Protocol in DBMS is an algorithm which uses the System Time or Logical Counter as a timestamp to serialize the execution of concurrent transactions. The Timestamp-based protocol ensures that every conflicting read and write operations are executed in a timestamp order.
The older transaction is always given priority in this method. It uses system time to determine the time stamp of the transaction. This is the most commonly used concurrency protocol.
Lock-based protocols help you to manage the order between the conflicting transactions when they will execute. Timestamp-based protocols manage conflicts as soon as an operation is created.
Example:
Suppose there are there transactions T1, T2, and T3.
T1 has entered the system at time 0010
T2 has entered the system at 0020
T3 has entered the system at 0030
Priority will be given to transaction T1, then transaction T2 and lastly Transaction T3.
Advantages:

- Schedules are serializable just like 2PL protocols
- No waiting for the transaction, which eliminates the possibility of deadlocks!

Disadvantages:
Starvation is possible if the same transaction is restarted and continually aborted

Validation Based Protocol

Validation based Protocol in DBMS also known as Optimistic Concurrency Control Technique is a method to avoid concurrency in transactions. In this protocol, the local copies of the transaction data are updated rather than the data itself, which results in less interference while execution of the transaction.
The Validation based Protocol is performed in the following three phases:

1. Read Phase
2. Validation Phase
3. Write Phase

Read Phase
In the Read Phase, the data values from the database can be read by a transaction but the write operation or updates are only applied to the local data copies, not the actual database.

Validation Phase
In Validation Phase, the data is checked to ensure that there is no violation of serializability while applying the transaction updates to the database.

Write Phase
In the Write Phase, the updates are applied to the database if the validation is successful, else; the updates are not applied, and the transaction is rolled back.

Characteristics of Good Concurrency Protocol
An ideal concurrency control DBMS mechanism has the following objectives:

- Must be resilient to site and communication failures.
- It allows the parallel execution of transactions to achieve maximum concurrency.
- Its storage mechanisms and computational methods should be modest to minimize overhead.
- It must enforce some constraints on the structure of atomic actions of transactions.

DBMS - Data Recovery

Crash Recovery

DBMS is a highly complex system with hundreds of transactions being executed every second. The durability and robustness of a DBMS depends on its complex architecture and its underlying hardware and system software. If it fails or crashes amid transactions, it is expected that the system would follow some sort of algorithm or techniques to recover lost data.

Failure Classification

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.

Storage Structure

We have already described the storage system. In brief, the storage structure can be divided into two categories −

- **Volatile storage** − As the name suggests, a volatile storage cannot survive system crashes. Volatile storage devices are placed very close to the CPU; normally they are embedded onto the chipset itself. For example, main memory and cache memory are examples of volatile storage. They are fast but can store only a small amount of information.
- **Non-volatile storage** − These memories are made to survive system crashes. They are huge in data storage capacity, but slower in accessibility. Examples may include hard-disks, magnetic tapes, flash memory, and non-volatile (battery backed up) RAM.

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 −

- 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.

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.

Log-based Recovery

Log is a sequence of records, which maintains the records of actions performed by a transaction. It is important that the logs are written prior to the actual modification and stored on a stable storage media, which is failsafe.

Log-based recovery works as follows −

- The log file is kept on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log about it.

<T_n, Start>

- When the transaction modifies an item X, it write logs as follows −

<T_n, X, V_1, V_2>

It reads T_n has changed the value of X, from V_1 to V_2.

- When the transaction finishes, it logs −

<T_n, commit>

The database can be modified using two approaches −

- **Deferred database modification** − All logs are written on to the stable storage and the database is updated when a transaction commits.
- **Immediate database modification** − Each log follows an actual database modification. That is, the database is modified immediately after every operation.

Recovery with Concurrent Transactions

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

Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all. Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Recovery

When a system with concurrent transactions crashes and recovers, it behaves in the following manner −

Database Recovery

Purpose of Database Recovery

To bring the database into the last consistent state,which existed prior to the failure.

To preserve transaction properties (Atomicity,Consistency, Isolation and Durability).

Example:

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

Database Recovery

Types of Failure

The database may become unavailable for use due to

- **Transaction failure**: Transactions may fail because of incorrect input, deadlock, incorrect synchronization.
- **System failure**: System may fail because of addressing error, application error, operating systemfault, RAM failure, etc.
- **Media failure**: Disk head crash, power disruption,etc.

Database Recovery

Transaction Log

- For recovery from any type of failure data values prior to modification (BFIM - BeFore Image) and the new value after modification (AFIM – AFter Image) are required.
- These values and other information is stored in a sequentialfile called Transaction log. A sample log is given below. Back P and Next P point to the previous and next log records of the same transaction.

T ID	**Back P**	**Next P**	**Operation**	**Data item**	**BFIM**	**AFIM**
T1	0	1	Begin			
T1	1	4	Write	X	X = 100	X = 200
T2	0	8	Begin			
T1	2	5	W	Y	Y = 50	Y = 100
T1	4	7	R	M	M = 200	M = 200
T3	0	9	R	N	N = 400	N = 400
T1	5	nil	End			

Database Recovery

Data Update

- **Immediate Update**: As soon as a data item is modified incache, 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 doesnot overwrite its disk copy but is written at a separate disk location.
- **In-place update**: The disk version of the data item isoverwritten by the cache version.

Database Recovery

Data Caching

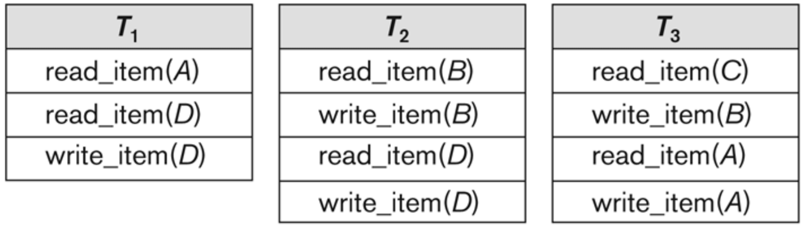
- Data items to be modified are first stored into databasecache by the Cache Manager (CM) and after modification they are flushed (written) to the disk.
- The flushing is controlled by **Modified** and **Pin-Unpin**
  bits.
  - **Pin-Unpin**: Instructs the operating system not toflush the data item.
  - **Modified**: Indicates the AFIM of the data item.

Database Recovery

1 Transaction **Roll-back (Undo)** and **Roll-Forward (Redo)**

- To maintain atomicity, a transaction’s operations areredone or undone.
  - **Undo**: Restore all BFIMs on to disk (Remove allAFIMs).
  - **Redo**: Restore all AFIMs on to disk.
- 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 theyhappen.

(a)



T_1	T_2	T_3
read_item(*A*)	read_item(*B*)	read_item(*C*)
read_item(*D*)	write_item(*B*)	write_item(*B*)
write_item(*D*)	read_item(*D*)	read_item(*A*)
	write_item(*D*)	write_item(*A*)

Figure 19.1
Illustrating cascading rollback (a process that never occurs in strict or cascadeless schedules). (a) The read and write operations of three transactions. (b) System log at point of crash. (c) Operations before the crash.

(b)

		A	*B*	*C*	*D*
		30	15	40	20
	[start_transaction,T_3]				
	[read_item,T_3,*C*]				
*	[write_item,T_3,*B*,15,12]		12		
	[start_transaction,T_2]				
	[read_item,T_2,*B*]				
**	[write_item,T_2,*B*,12,18]		18		
	[start_transaction,T_1]				
	[read_item,T_1,*A*]				
	[read_item,T_1,*D*]				
	[write_item,T_1,*D*,20,25]				25
	[read_item,T_2,*D*]				
**	[write_item,T_2,*D*,25,26]				26
	[read_item,T_3,*A*]				

← System crash

Figure 19.1
Illustrating cascading rollback (a process that never occurs in strict or cascadeless schedules).
(a) The read and write operations of three transactions.
(b) System log at point of crash. (c) Operations before the crash.

* T_3 is rolled back because it did not reach its commit point.

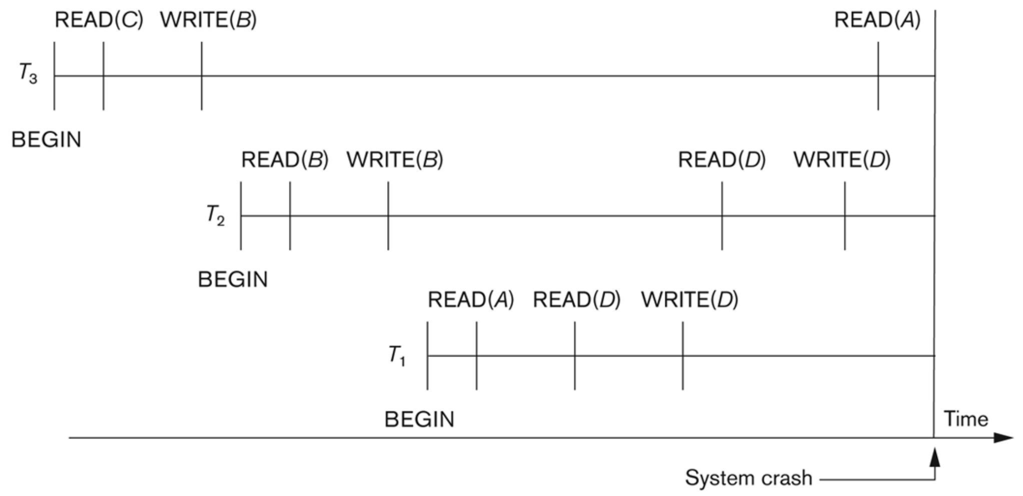
** T_2 is rolled back because it reads the value of item *B* written by T_3.

Database Recovery

Roll-back: One execution of T1, T2 and T3 as recorded inthe log.

Figure 19.1
Illustrating cascading rollback (a process that never occurs in strict or cascadeless schedules).
(a) The read and write operations of three transactions.
(b) System log at point of crash. (c) Operations before the crash.

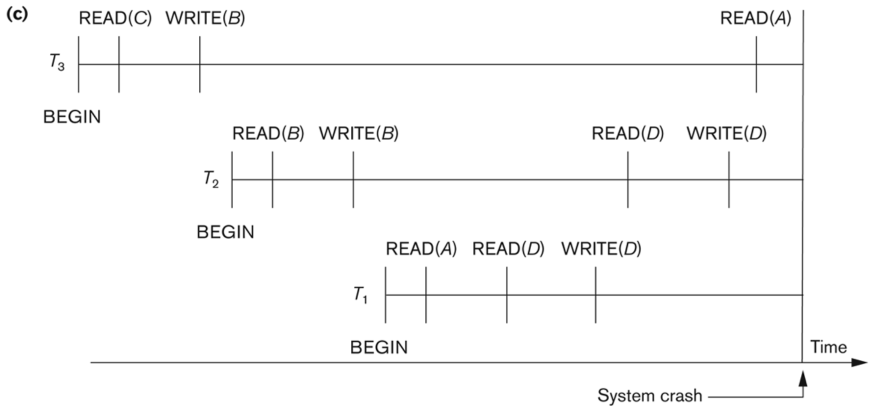
(c)



Database Recovery

Roll-back: One execution of T1, T2 and T3 as recorded inthe log.

Figure 19.1
Illustrating cascading rollback (a process that never occurs in strict or cascadeless schedules). (a) The read and write operations of three transactions. (b) System log at point of crash. (c) Operations before the crash.



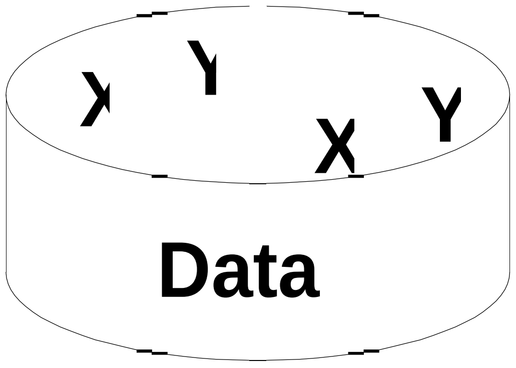
Database Recovery

Shadow Paging

- The AFIM does not overwrite its BFIM but recorded at another place on the disk. Thus, at any time a data itemhas AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.

Database Recovery

Shadow Paging



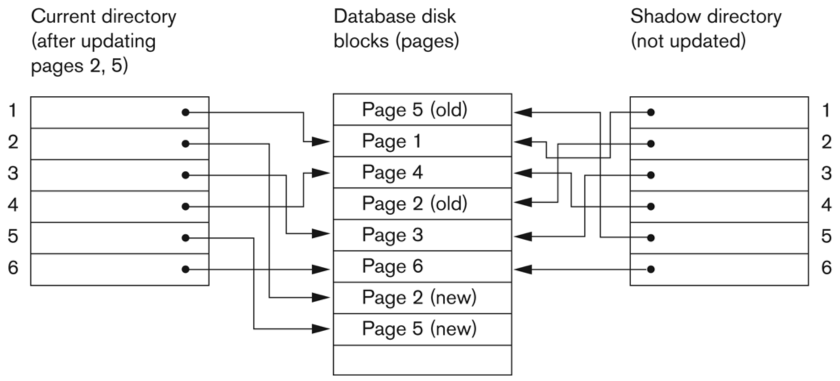

Current directory (after updating pages 2, 5)
Database disk blocks (pages)
Shadow directory (not updated)
1
2
3
4
5
6
Page 5 (old)
Page 1
Page 4
Page 2 (old)
Page 3
Page 6
Page 2 (new)
Page 5 (new)
1
2
3
4
5
6

Figure 19.5
An example of shadow paging.