MODULE VI

OVERVIEW OF CONCURRENCY CONTROL AND RECOVERY

- A DBMS is **single-user** if at most one user at a time can use the system, and it is multiuser if many users can use the system-and hence access the database-concurrently.
- Single-user DBMSs are mostly restricted to personal computer systems;
- Most other DBMSs are multiuser. For example, an airline reservations system is used by hundreds of travel agents and reservation clerks concurrently. Systems in banks, insurance agencies, stock exchanges, supermarkets, and the like are also operated on by many users who submit transactions **concurrently** to the system.
- Multiple users can access databases-and use computer systems-simultaneously because of the concept of **multiprogramming**, which allows the computer to execute multiple programs-or processes-at the same time.
- However, multiprogramming operating systems execute some commands from one process, then suspend that process and execute some commands from the next process, and so on.
- A process is resumed at the point where it was suspended whenever it gets its turn to use the CPU again. Hence, concurrent execution of processes is actually **interleaved.**
- If the computer system has multiple hardware processors (crus), **parallel processing** of multiple processes is possible, as illustrated by processes C and D in Figure 21.1

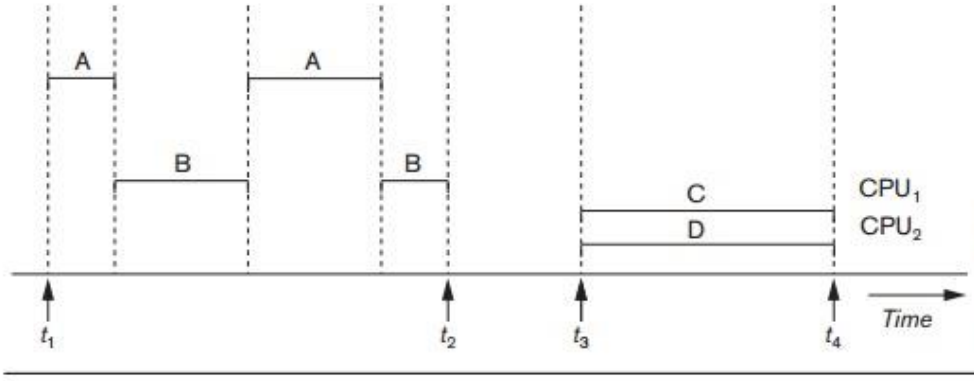


Figure 21.1
Interleaved processing versus parallel processing of concurrent transactions.

Transaction

- Transaction boundaries: **Begin and End transaction.**
- Basic operations on an item X:
  - **read_item(X):** Reads a database item named X into a program variable. To simplify our notation, we assume that *the program variable is also named X.*
  - **write_item(X)**: Writes the value of program variable X into the database item

named X.

READ OPERATIONS command includes the following steps:

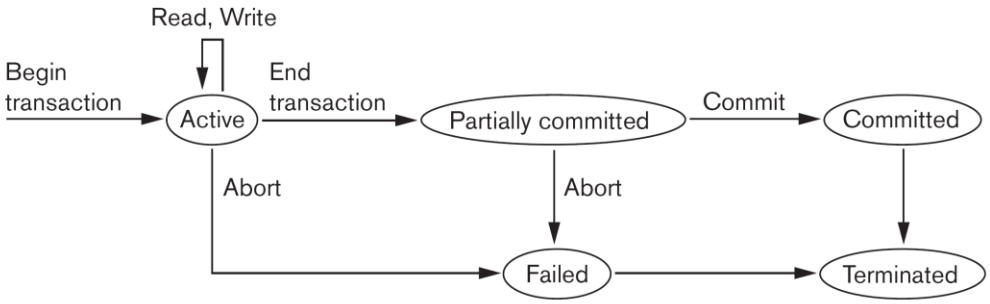
1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the buffer to the program variable named X.

WRITE OPERATIONS

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if it is not already in some main memory buffer).
3. Copy item X from the program variable named X into its correct location in the buffer.
4. Store the updated block from the buffer back to disk (either immediately or at some later point in time).

Transaction State

Figure 21.4
State transition diagram illustrating the states for transaction execution.



• **BEGIN_TRANSACTION**: This marks the beginning of transaction execution.

• **READ OR WRITE:** These specify read or write operations on the database items that are executed as part of a transaction.

• **END_TRANSACTION:** This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. However, at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it violates serializability .

• **COMMIT_TRANSACTION:** This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be **undone.**

• **ROLLBACK (OR ABORT):** This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

State transition diagram shown above describes how a transaction moves through its execution states.

(a) A transaction goes into an **active state** immediately after it starts execution, where it can issue READ and WRITE operations.

(b) When the transaction ends, it moves to the **partially committed state**.

(c) At this point, some recovery protocols need to ensure that a system failure will not result in an inability to record the changes of the transaction permanently .

(d) Once this check is successful, the transaction is said to have reached its **commit point** and enters the **committed state**.

(e) Once a transaction is committed, it has concluded its execution successfully and all its changes must be recorded permanently in the database.

(f) However, a transaction can go to the **failed state** if one of the **checks fails** or if the transaction is **aborted during its active state**.

(g) The transaction may then have to be rolled back to undo the effect of its WRITE operations on the database.

(h) The **terminated state** corresponds to the transaction leaving the system.

ACID PROPERTIES

Transactions should possess several properties. These are often called the ACID properties:

Atomicity-

- This property ensures that either the transaction occurs completely or it does not occur at all.
- In other words, it ensures that no transaction occurs partially.
- That is why, it is also referred to as "**All or nothing rule**".
- It is the responsibility of Transaction Control Manager to ensure atomicity of the transactions.

2. Consistency-

- This property ensures that integrity constraints are maintained.
- In other words, it ensures that the database remains consistent before and after the transaction.
- It is the responsibility of DBMS and application programmer to ensure consistency of the database.

3. Isolation-

- This property ensures that multiple transactions can occur simultaneously without causing any inconsistency.
- During execution, each transaction feels as if it is getting executed alone in the system.
- A transaction does not realize that there are other transactions as well getting executed parallely.
- Changes made by a transaction becomes visible to other transactions only after they are written in the memory.
- The resultant state of the system after executing all the transactions is same as the state that would be achieved if the transactions were executed serially one after the other.
- It is the responsibility of concurrency control manager to ensure isolation for all the transactions.

4. Durability-

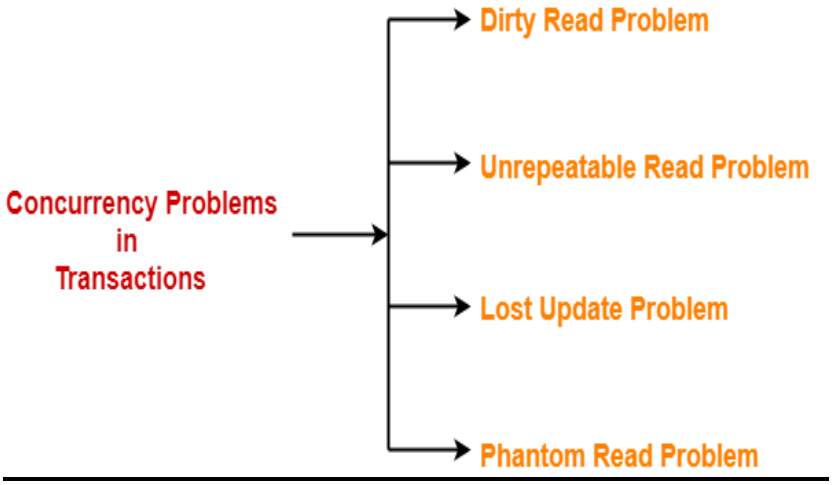
- This property ensures that all the changes made by a transaction after its successful execution are written successfully to the disk.
- It also ensures that these changes exist permanently and are never lost even if there occurs a failure of any kind.
- It is the responsibility of recovery manager to ensure durability in the database.

Transaction Notation

- Notation focuses on the read and write operations
- Can also write in shorthand notation:
  - T1: b1; r1(X); w1(X); r1(Y); w1(Y); e1;
  - T2: b2; r2(Y); w2(Y); e2;
- bi and ei specify transaction boundaries (begin and end)

Concurrency problems

- When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems.
- Such problems are called as concurrency problems.

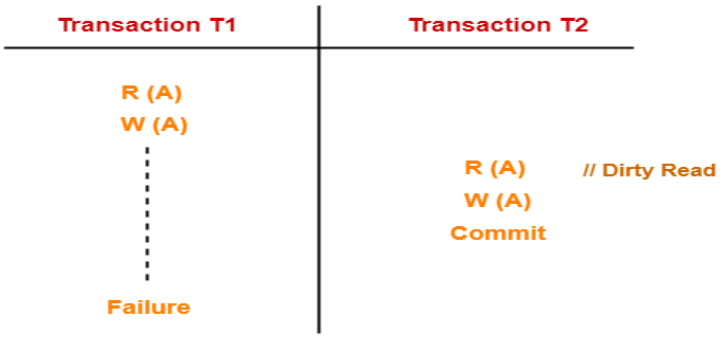


1. Dirty Read Problem-

Reading the data written by an uncommitted transaction is called as dirty read.

This read is called as dirty read because-

- There is always a chance that the uncommitted transaction might roll back later.
- Thus, uncommitted transaction might make other transactions read a value that does not even exist.
- This leads to inconsistency of the database.
- Dirty read does not lead to inconsistency always.
- It becomes problematic only when the uncommitted transaction fails and rolls backs later due to some reason.



Here,

- T1 reads the value of A.
- T1 updates the value of A in the buffer.
- T2 reads the value of A from the buffer.
- T2 writes the updated the value of A.
- T2 commits.
- T1 fails in later stages and rolls back.

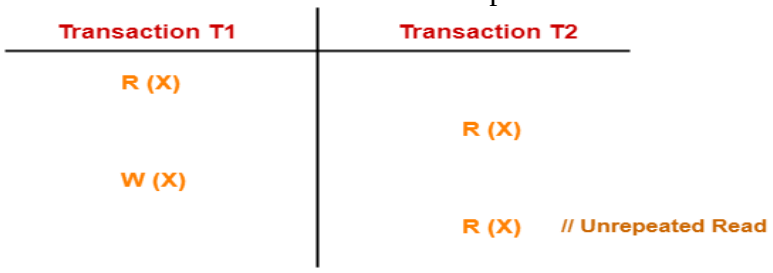
In this example,

- T2 reads the dirty value of A written by the uncommitted transaction T1.
- T1 fails in later stages and roll backs.
- Thus, the value that T2 read now stands to be incorrect.

Therefore, database becomes inconsistent.

2. Unrepeatable Read Problem-

- This problem occurs when a transaction gets to read unrepeated i.e. different values of the same variable in its different read operations even when it has not updated its value.



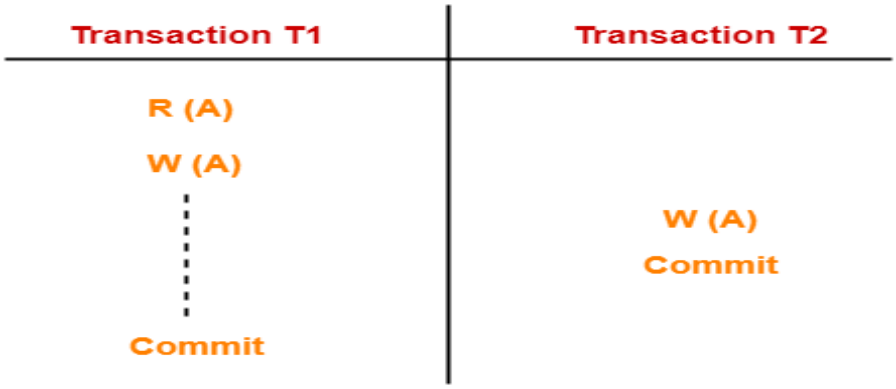
Here,

- T1 reads the value of X (= 10 say).
- T2 reads the value of X (= 10).
- T1 updates the value of X (from 10 to 15 say) in the buffer.
- T2 again reads the value of X (but = 15).

In this example,

- T2 gets to read a different value of X in its second reading.
- T2 wonders how the value of X got changed because according to it, it is running in isolation.

3. Lost Update Problem-(write-write conflict problem)



Here,

- T1 reads X.
- T2 reads X.
- T1 deletes X.
- T2 tries reading X but does not find it.

In this example,

- T2 finds that there does not exist any variable X when it tries reading X again.
- T2 wonders who deleted the variable X because according to it, it is running in isolation.

Avoiding Concurrency Problems-

- To ensure consistency of the database, it is very important to prevent the occurrence of above problems.
- **Concurrency Control Protocols** help to prevent the occurrence of above problems and maintain the consistency of the database.

Schedules in DBMS

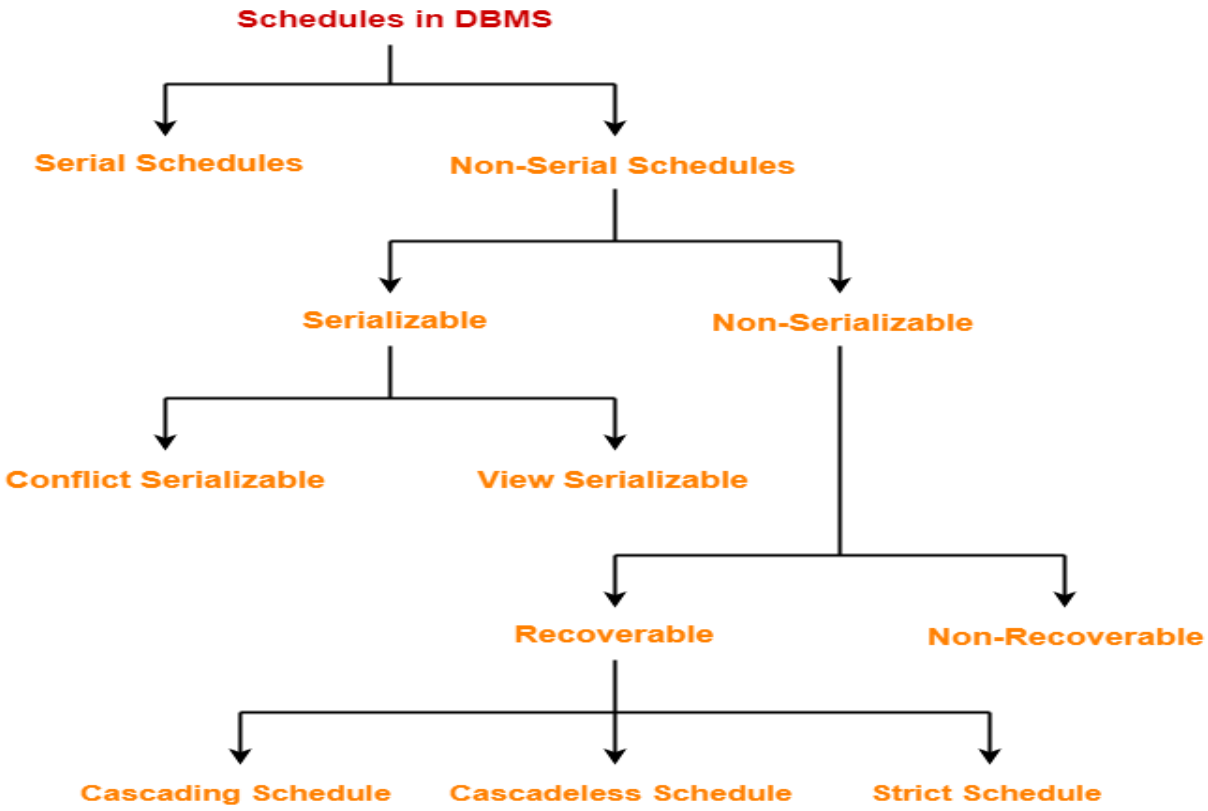
- The order in which the operations of multiple transactions appear for execution is called as a schedule.
- Schedules can also be displayed in more compact notation
- Order of operations from left to right
- Include only read (r) and write (w) operations, with transaction id (1, 2, …) and item name (X, Y, …)
- Can also include other operations such as b (begin), e (end), c (commit), a (abort)

- For n transactions T1, T2, ..., Tn, where each Ti has mi read and write operations, the number of possible schedules is (! is *factorial* function):

  (m1 + m2 + … + mn)! / ((m1)! * (m2)! * … * (mn)!)

- Generally very large number of possible schedules
- Some schedules are easy to recover from after a failure, while others are not
- Some schedules produce correct results, while others produce incorrect results

Types of Schedules-



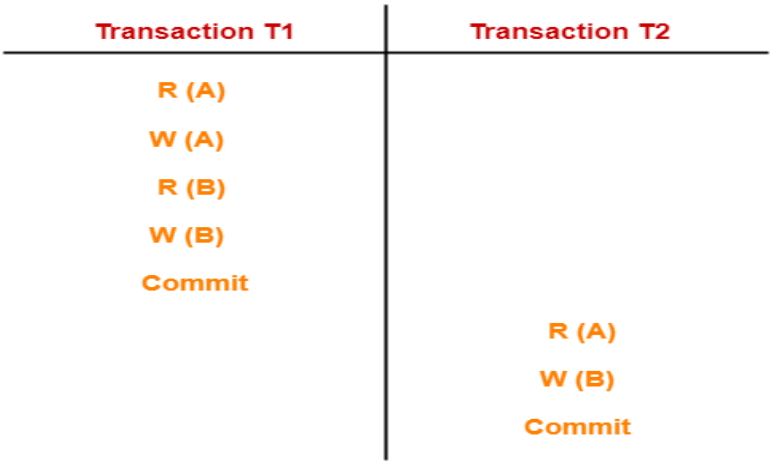
Serial Schedules-

- In serial schedules,
- All the transactions execute serially one after the other.
- When one transaction executes, no other transaction is allowed to execute.

Characteristics-

- Serial schedules are always-

- Consistent
- Recoverable
- Cascadeless
- Strict



Non-Serial Schedules-

- In non-serial schedules,
- Multiple transactions execute concurrently.
- Operations of all the transactions are inter leaved or mixed with each other.

Characteristics-

- Non-serial schedules are **NOT** always-
- Consistent
- Recoverable
- Cascadeless
- Strict

Transaction T1	Transaction T2
R (A)	
W (B)	
	R (A)
R (B)	
W (B)	
Commit	
	R (B)
	Commit

Serializability in DBMS-

- Some non-serial schedules may lead to inconsistency of the database.
- Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.

Serializable Schedules-

If a given non-serial schedule of 'n' transactions is equivalent to some serial schedule of 'n' transactions, then it is called as a **serializable schedule**.

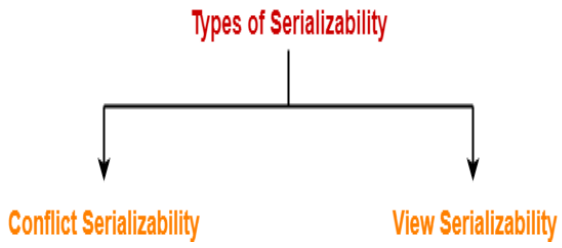
Characteristics-

- Serializable schedules behave exactly same as serial schedules.

Thus, serializable schedules are always-

- Consistent
- Recoverable
- Casacadeless
- Strict

Types of Serializability-



Conflict Serializability-

- If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called as a **conflict serializable schedule**.

Conflicting Operations-

Two operations are called as **conflicting operations** if all the following conditions hold true for them-

1. Both the operations belong to different transactions
2. Both the operations are on the same data item.
3. At least one of the two operations is a write operation.

Transaction T1	Transaction T2
R1 (A)	
W1 (A)	
	R2 (A)
R1 (B)	

In this schedule,

- W1 (A) and R2 (A) are called as conflicting operations.
- This is because all the above conditions hold true for them.

Checking Whether a Schedule is Conflict Serializable Or Not-

Follow the following steps to check whether a given non-serial schedule is conflict serializable or not-

Step-01: Find and list all the conflicting operations.

Step-02: Start creating a precedence graph by drawing one node for each transaction.

Step-03:

- Draw an edge for each conflict pair such that if X_i (V) and Y_j (V) forms a conflict pair then draw an edge from T_i to T_j.
- This ensures that T_i gets executed before T_j.

Step-04:

- Check if there is any cycle formed in the graph.
- If there is no cycle found, then the schedule is conflict serializable otherwise not.

If there is no cycle found, then the schedule is conflict serializable otherwise not.

PRACTICE PROBLEMS BASED ON CONFLICT SERIALIZABILITY-

Problem-01:

Check whether the given schedule S is conflict serializable or not-

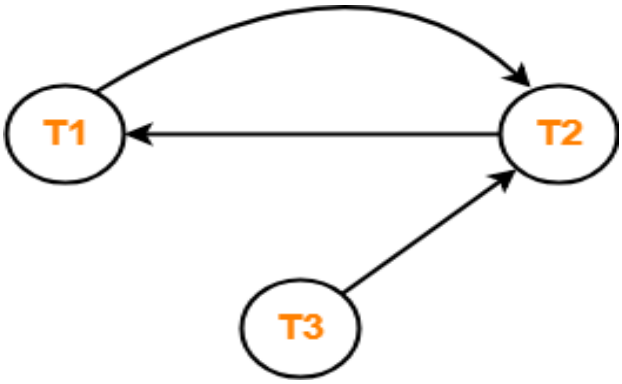
S : $R_1(A)$, $R_2(A)$, $R_1(B)$, $R_2(B)$, $R_3(B)$, $W_1(A)$, $W_2(B)$

Solution-

Step-01: List all the conflicting operations and determine the dependency between the transactions-

- $R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
- $R_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
- $R_3(B)$, $W_2(B)$ $(T_3 \rightarrow T_2)$

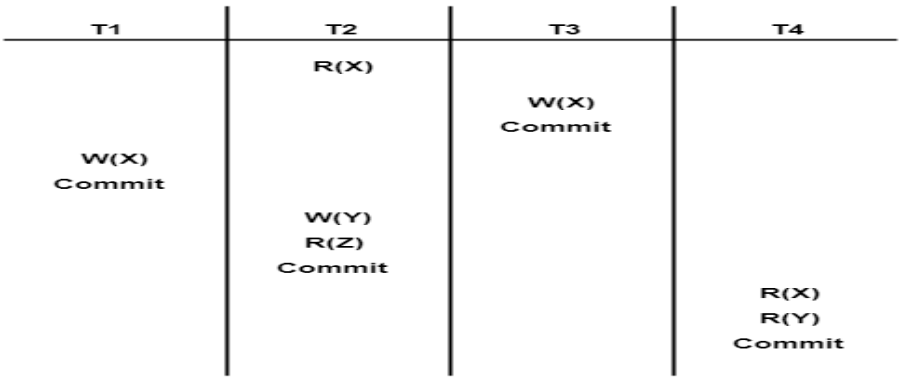
Step-02: Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

Problem-02:

Check whether the given schedule S is conflict serializable



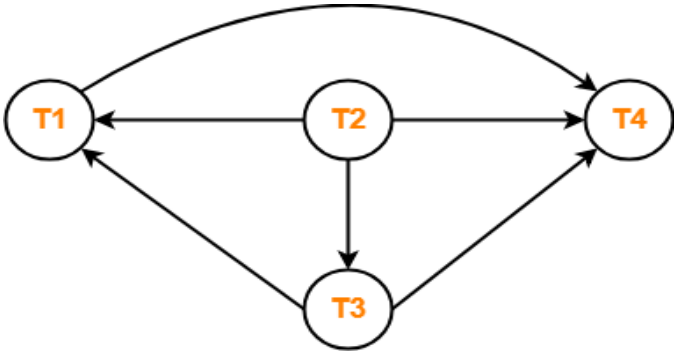
Solution-

Checking Whether S is Conflict Serializable Or Not-

Step-01:

- List all the conflicting operations and determine the dependency between the transactions-
- $R_2(X)$, $W_3(X)$ $(T_2 \rightarrow T_3)$
- $R_2(X)$, $W_1(X)$ $(T_2 \rightarrow T_1)$
- $W_3(X)$, $W_1(X)$ $(T_3 \rightarrow T_1)$
- $W_3(X)$, $R_4(X)$ $(T_3 \rightarrow T_4)$
- $W_1(X)$, $R_4(X)$ $(T_1 \rightarrow T_4)$
- $W_2(Y)$, $R_4(Y)$ $(T_2 \rightarrow T_4)$

Step-02: Draw the precedence graph-



- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.

Non-Serializable Schedules-

- A non-serial schedule which is not serializable is called as a non-serializable schedule.
- A non-serializable schedule is not guaranteed to produce the the same effect as produced by some serial schedule on any consistent database.

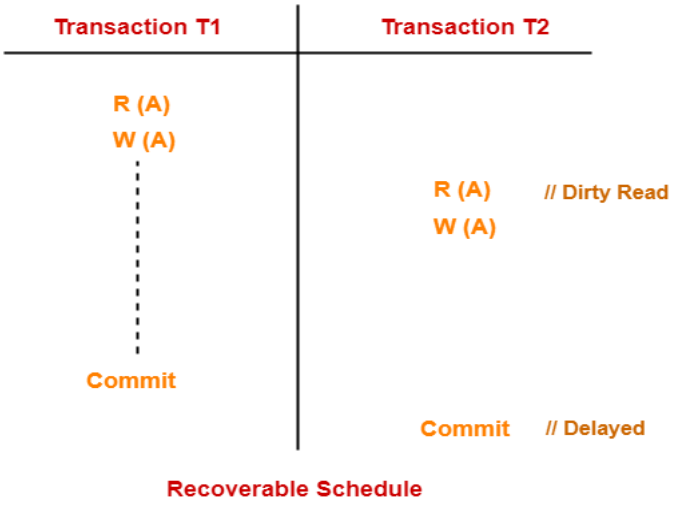
Characteristics-

- Non-serializable schedules-
- may or may not be consistent
- may or may not be recoverable

RecoverableSchedules-

If in a schedule,A transaction performs a dirty read operation from an uncommitted transaction
And its commit operation is delayed till the uncommitted transaction either commits or roll backs then such a schedule is known as a **Recoverable Schedule**.

- The commit operation of the transaction that performs the dirty read is delayed.
- This ensures that it still has a chance to recover if the uncommitted transaction fails later.



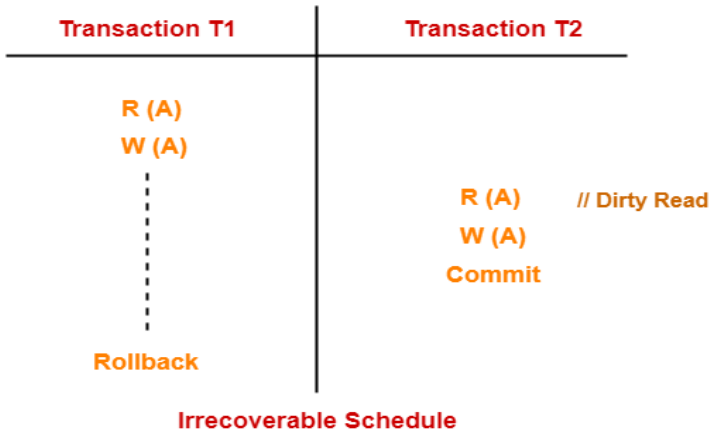
Here,

- T2 performs a dirty read operation.
- The commit operation of T2 is delayed till T1 commits or roll backs.
- T1 commits later.
- T2 is now allowed to commit.
- In case, T1 would have failed, T2 has a chance to recover by rolling back.

Irrecoverable Schedules-

If in a schedule,

- A transaction performs a dirty read operation from an uncommitted transaction
- And commits before the transaction from which it has read the value
- then such a schedule is known as an **Irrecoverable Schedule**.



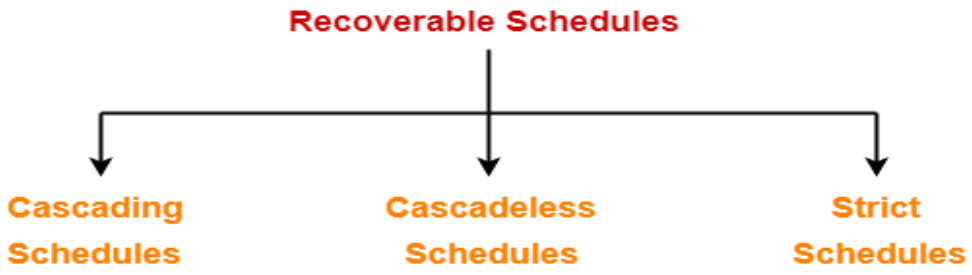
Here,

- T2 performs a dirty read operation.
- T2 commits before T1.
- T1 fails later and roll backs.
- The value that T2 read now stands to be incorrect.
- T2 can not recover since it has already committed.

Cascading Rollback | Cascadeless Schedule

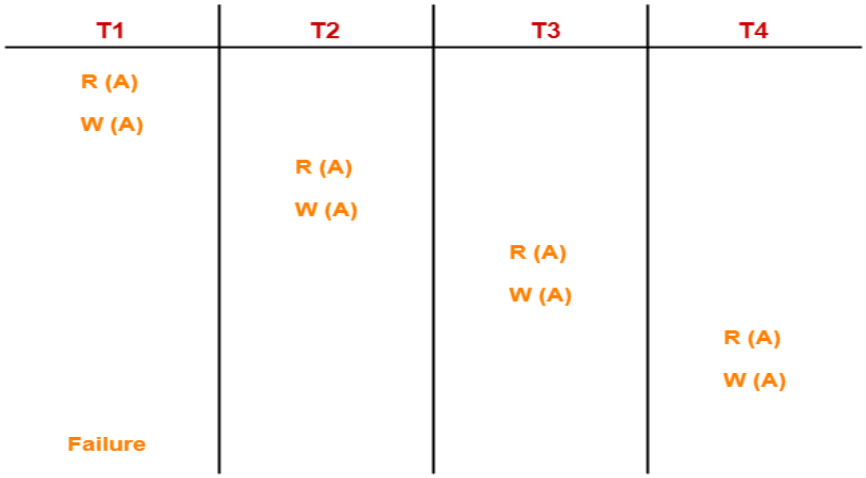
If in a schedule,

- A transaction performs a dirty read operation from an uncommitted transaction
- And its commit operation is delayed till the uncommitted transaction either commits or roll backs
- then such a schedule is called as a **Recoverable Schedule**.



Cascading Schedule-

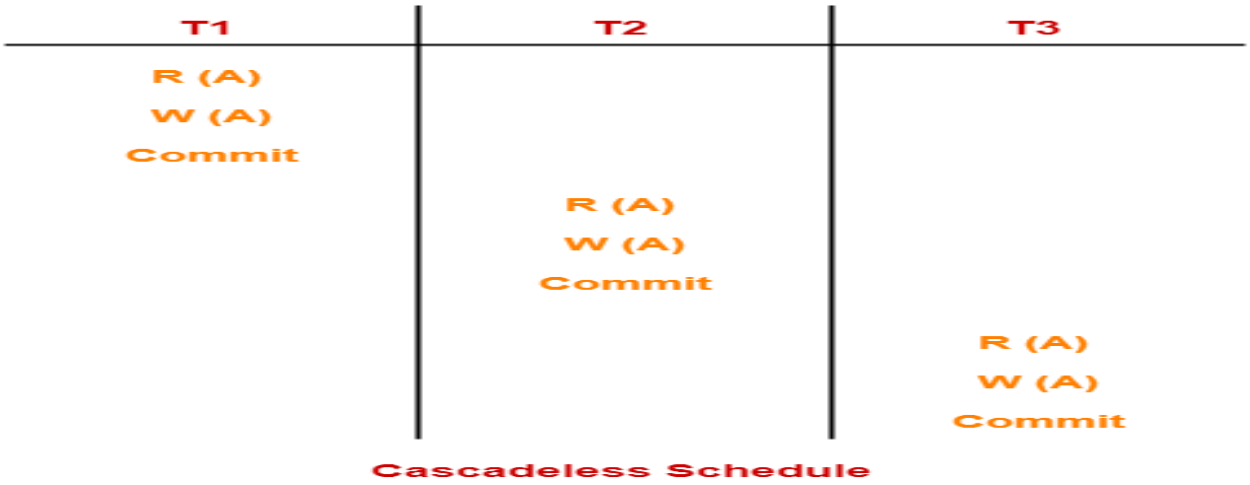
- If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a **Cascading Schedule** or **Cascading Rollback** or **Cascading Abort**.
- It simply leads to the wastage of CPU time.



Cascading Recoverable Schedule

Cascadeless Schedule-

- If in a schedule, a transaction is not allowed to read a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a **Cascadeless Schedule**.
- In other words,
- Cascadeless schedule allows only committed read operations.
- Therefore, it avoids cascading roll back and thus saves CPU time.



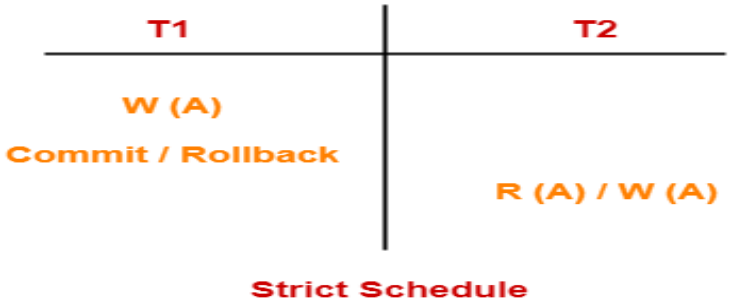
Strict Schedule-

If in a schedule, a transaction is neither allowed to read nor write a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a **Strict Schedule**.

In other words,

- Strict schedule allows only committed read and write operations.

Clearly, strict schedule implements more restrictions than cascadeless schedule.



Concurrency Control Technique

1. Locking protocol
2. Time Stamp based protocol
3. Optimistic protocol
4. Multiversion protocol

Lock-Based Protocols

- A lock is a mechanism to control concurrent access to a data item
- Data items can be locked in two modes :
    - ***exclusive (X) mode***. Data item can be both read as well as written. X-lock is

requested using **lock-X** instruction.

- ***shared (S) mode***. Data item can only be read. S-lock is requested using **lock-S** instruction.

- Lock requests are made to the concurrency-control manager by the programmer. Transaction can proceed only after request is granted.

Lock-compatibility matrix

	S	X
S	true	false
X	false	false

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions

- Any number of transactions can hold shared locks on an item,

  - But if any transaction holds an exclusive on the item no other transaction may hold any lock on the item.

- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted.

The Two-Phase Locking Protocol

This protocol ensures conflict-serializable schedules.

Phase 1: Growing Phase

Transaction may obtain locks

Transaction may not release locks

Phase 2: Shrinking Phase

Transaction may release locks

Transaction may not obtain locks

The protocol assures serializability. It can be proved that the transactions can be serialized in the order of their **lock points** (i.e., the point where a transaction acquired its final lock).

- There can be conflict serializable schedules that cannot be obtained if two-phase locking is used.

- However, in the absence of extra information (e.g., ordering of access to data), two-phase locking is needed for conflict serializability in the following sense:

- Given a transaction T_i that does not follow two-phase locking, we can find a transaction T_j that uses two-phase locking, and a schedule for T_i and T_j that is not conflict serializable.

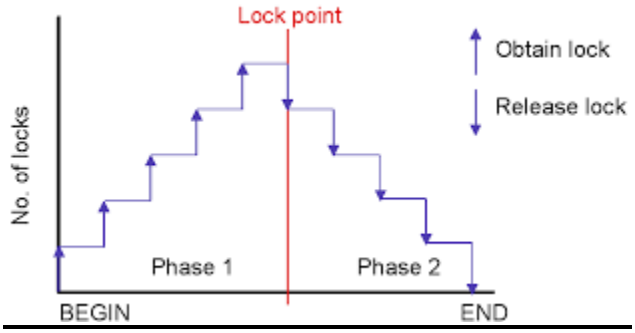
Two-phase locking with lock conversions:

- First Phase:
  - can acquire a lock-S on item
  - can acquire a lock-X on item
  - can convert a lock-S to a lock-X (upgrade)
- Second Phase:
  - can release a lock-S
  - can release a lock-X
  - can convert a lock-X to a lock-S (downgrade)

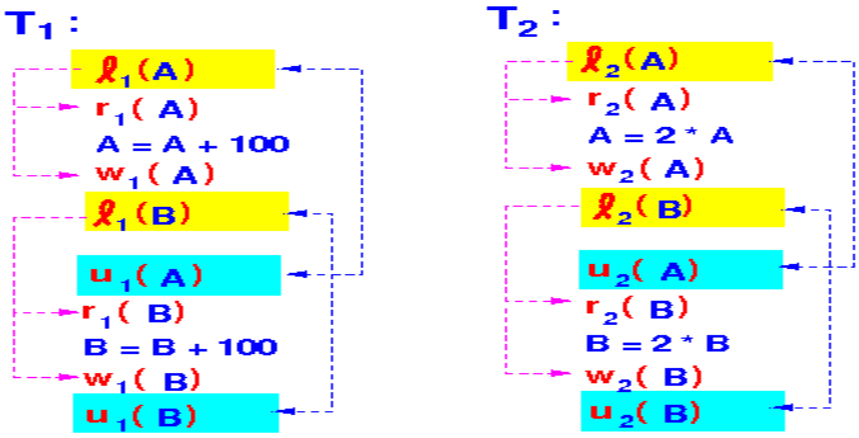
This protocol assures serializability. But still relies on the programmer to insert the various locking instructions.

Difficulties faced by 2-phase locking—deadlock, cascading rollback

Graph of 2PL



Example



All *lock* operations precedes all *unlock* operations

T_3	T_4
lock-x (B)	
read (B)	
$B := B - 50$	
write (B)	
	lock-s (A)
	read (A)
	lock-s (B)
lock-x (A)	

- Neither T_3 nor T_4 can make progress — executing **lock-S**(B) causes T_4 to wait for T_3 to release its lock on B, while executing **lock-X**(A) causes T_3 to wait for T_4 to release its lock on A.
- Such a situation is called a **deadlock**.
    - To handle a deadlock one of T_3 or T_4 must be rolled back and its locks released.
- Two-phase locking *does not* ensure freedom from deadlocks.
- In addition to deadlocks**,** there is a possibility of **starvation.**
- **Starvation** occurs if the concurrency control manager is badly designed. For example:
    - A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item.
    - The same transaction is repeatedly rolled back due to deadlocks.
- Concurrency control manager can be designed to prevent starvation.
- The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil.
- When a deadlock occurs there is a possibility of cascading roll-backs.

Variations of 2PL protocol

1. **Conservative (static) 2PL**
   Acquire all locks before starts after commit only it release locks. Avoid cascading rollback and deadlock.
2. **Strict 2PL**
   Until commit exclusive lock cannot be released. Deadlock may occur.
3. **Rigrous 2PL**
   Shared or exclusive lock cannot be released until commit. Deadlock may occur.

FAILURE CLASSIFICATION.

Types of Failures. Failures are generally classified as transaction, system, and media failures. There are several possible reasons for a transaction to fail in the middle of execution:

1. *A computer failure (system crash):* A **hardware, software, or network error** occurs in the computer system during transaction execution. Hardware crashes are usually media failures-for example, main memory failure.

2. *A transaction or system error*: Some operation in the transaction may cause it to fail, such as **integer overflow or division by zero**. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error.In addition, the user may interrupt the transaction during its execution.

3. *Local errors or exception conditions detected by the transaction*: During transaction execution, **certain conditions** may occur that necessitate cancellation of the transaction. For example, data for the transaction may not be found. Notice that an exception condition**," such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal, to be canceled.** This exception should be programmed in the transaction itself, and hence would not be considered a failure.

4. *Concurrency control enforcement*: The concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in a state of deadlock.

5. *Disk failure*: Some disk blocks may lose their data because of a read or write malfunction or because of **a disk read/write head crash.** This may happen during a read or a write operation of the transaction.

6. *Physical problems and catastrophes:* This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator. Failures are generally classified as transaction, system, and media failures.

STORAGE STRUCTURE.

- **Volatile storage**:
    - does not survive system crashes
    - examples: main memory, cache memory
- **Nonvolatile storage**:
    - survives system crashes
    - examples: disk, tape, flash memory, non-volatile (battery backed up) RAM
    - but may still fail, losing data
- **Stable storage**:
    - a mythical form of storage that survives all failures
    - approximated by maintaining multiple copies on distinct nonvolatile media.

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.

STABLE STORAGE

- Stable storage plays a critical role in recovery algorithms.
- If a data-transfer failure occurs, the system detects it and invokes a recovery procedure to restore the block to a consistent state.
- 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:
- Write the information onto the first physical block.
- When the first write completes successfully, write the same information onto the second physical block.
- The output is completed only after the second write completes successfully.
- RAID systems (is the mirrored disk, which keeps two copies of each block, on separate disks),**cannot** guard against data loss due to disasters such as **fires or flooding.**
- Many systems store archival **backups of tapes offsite** to guard against such disasters.
- However, since tapes cannot be carried offsite continually, updates since the most recent time that tapes were carried offsite could be lost in such a disaster.
- More secure systems keep a copy of each block of stable storage at a remote site, writing it out over a computer network, in addition to storing the block on a local disk system.
- Since **the blocks are output to a remote system** as and when they are output to local storage, once an output operation is complete, **the output is not lost**, **even in the event of a disaster such as a fire or flood.**
- 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.
- If the **system detects an error in one block**, then it replaces its content with the content of the other block.
- If **both blocks contain no detectable error** ,but they **differ in content ,**then the system
- **replaces** the content of **the first block** with the value of the second.
- This recovery procedure ensures that a write to stable storage either succeeds completely(that is, updates all copies)or results in no change.
- The requirement of comparing every corresponding pair of blocks during recovery is expensive to meet.
- We can reduce the cost greatly by keeping track of block writes that are in progress, using a small amount of nonvolatile RAM. On recovery, only blocks for which writes were in progress need to be compared.

LOG BASED RECOVERY

- A **log** is kept on stable storage.
  - The log is a sequence of **log records**, and maintains a record of update activities on the database.

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

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

Other type of log records are:

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

Undo and Redo Operations –Because all database modifications must be preceded by creation of log record, the system has available both the old value prior to modification of data item and new value that is to be written for data item. This allows system to perform redo and undo operations as appropriate:

1. **Undo:** using a log record sets the data item specified in log record to old value.
2. **Redo:** using a log record sets the data item specified in log record to new value.

The database can be modified using two approaches –

1. **Deferred Modification Technique:** If the transaction does not modify the database until it has partially committed, it is said to use deferred modification technique.
2. **Immediate Modification Technique:** If database modification occur while transaction is still active, it is said to use immediate modification technique.

Deferred Modification Technique

The deferred database modification scheme records all modifications to the log, but defers all the writes to after partial commit.

- Assume that transactions execute serially
- Transaction starts by writing record to log.
- A write(X) operation results in a log record being written, where V is the new value for X
  z Note: old value is not needed for this scheme
- The write is not performed on X at this time, but is deferred.
- When Ti partially commits, is written to the log
- Finally, the log records are read and used to actually execute the previously deferred writes.
- During recovery after a crash, a transaction needs to be redone if and only if both and are there in the log.
- Redoing a transaction Ti (redoTi) sets the value of all data items updated by the transaction to the new values.
- Crashes can occur while z the transaction is executing the original updates, or z while recovery action is being taken.

  .

example transactions T_0 and T_1 (T_0 executes before T_1):

T_0	T_1
read (A)	**read** (C)
A: - A - 50	C:- C- 100
Write (A)	**write** (C)
read (B)	
B:- B + 50	
write (B)	

- Below we show the log as it appears at three instances of time

(a)	(b)	(c)
<T_0 start>	<T_0 start>	<T_0 start>
<T_0, A, 950>	<T_0, A, 950>	<T_0, A, 950>
<T_0, B, 2050>	<T_0, B, 2050>	<T_0, B, 2050>
	<T_0 commit>	<T_0 commit>
	<T_1 start>	<T_1 start>
	<T_1, C, 600>	<T_1, C, 600>
		<T_1 commit>

If log on stable storage at time of crash is as in case:

(a) No redo actions need to be taken

(b) redo(T0) must be performed since is present

(c) redo(T0) must be performed followed by redo(T1) since and are present

Checkpoint

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**'.

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.

- The checkpoint is a type of mechanism where all the previous logs are removed from the system and permanently stored in the storage disk.
- The checkpoint is like a bookmark. While the execution of the transaction, such checkpoints are marked, and the transaction is executed then using the steps of the transaction, the log files will be created.
- When it reaches to the checkpoint, then the transaction will be updated into the database, and till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till next checkpoint and so on.
- The checkpoint is used to declare a point before which the DBMS was in the consistent state, and all transactions were committed.

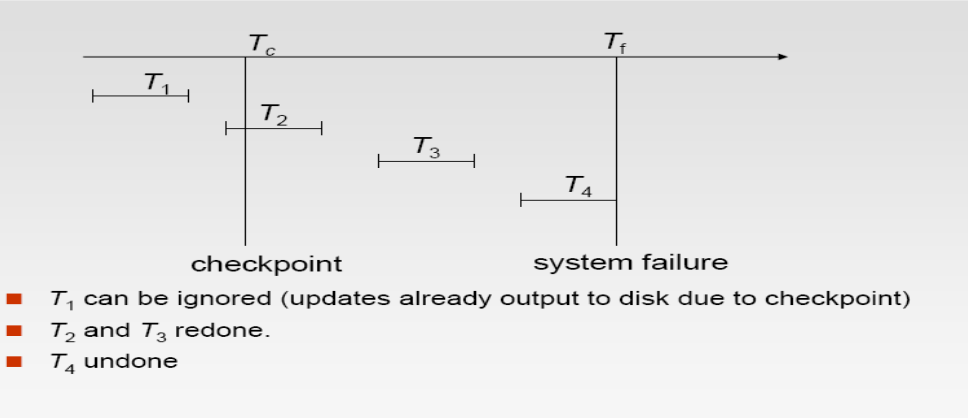
Recovery

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

- The recovery system reads the logs backwards from the end to the last checkpoint.
- It maintains two lists, an undo-list and a redo-list.
- If the recovery system sees a log with $<T_n$, Start> and $<T_n$, Commit> or just $<T_n$, Commit>, it puts the transaction in the redo-list.
- If the recovery system sees a log with $<T_n$, Start> but no commit or abort log found, it puts the transaction in undo-list.

All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

Example



SEMANTIC WEB

Semantic Web is the new generation Web that tries to represent information such that it can be used by machines, not just for display purposes, but for automation, integration, and reuse acrossapplications (Berners-Lee 2000). Furthermore, semantic Web is about explicitly declaring the knowledge embedded in many Web based applications, integrating information in an intelligent way, providing semantic based access to the Internet, and extracting information from texts.

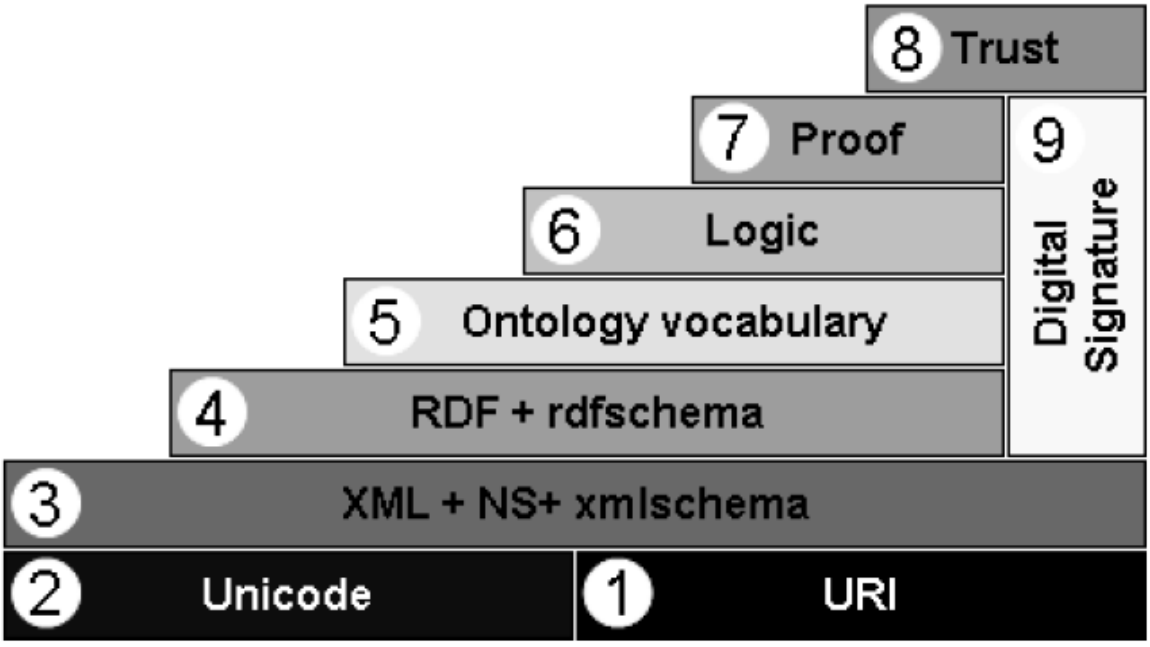
Traditionally, HTML provides the standard of structured document published on the Internet. Though the simplicity of HTML promotes the growth of the Web, it seriously hampered advanced applications such as processing, understanding and semantic interoperability of information contained in several documents. Semantic Web is the new generation Web which makes possible to express information in precise, machine-interpretable form.

It enables intelligent services such as information brokers, search agents and information filters, and also offers greater functionality and interoperability. Semantic Web promotes Web based applications with both semantic and syntactic interoperability. The explicit representation of metainformation, accompanied by domain theories (i.e. ontologies), will enable a Web to provide a qualitatively new level of service. This process may ultimately create extremely knowledgeable systems with various specialized reasoning services.

The aim of the Semantic Web is to allow much more advanced knowledge management systems:

- Knowledge will be organized in conceptual spaces according to its meaning.
- Automated tools will support maintenance by checking for inconsistencies and extracting new knowledge
- Keyword-based search will be replaced by query answering. Requested knowledge will be retrieved, extracted and presented in a human friendly way.
- Query answering over several documents will be supported
- Defining who may view certain parts of information (even parts of documents) will be possible.

The architecture of semantic Web (W3C) is shown in Figure. The semantic Web technologies offer a new approach to managing information and processes, the fundamental principle of which is the creation and use of semantic metadata.



(i) URI

A universal resource identifier (URI) is a formatted string that serves as a means of identifying abstract or physical resource. A URI can be further classified as a locator, a name, or both. Uniform resource locator (URL) refers to the subset of URI that identifies resources via a representation of their primary access mechanism. An uniform resource name (URN) refers to the subset of URI that is required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable

(ii) Unicode

Unicode provides a unique number for every character, independently of the underlying platform, program, or language.

(iii)XML and XML Namespace

XML (eXtensible markup language) with XML namespace and XML schema definitions makes sure that there is a common syntax used in the semantic Web.

(iv) RDF and RDF Schema

On top of XML is the Resource Description Framework (RDF), for representing information about resources in a graph form. RDF is based on triples O-A-V that form a graph data with a relation among object (a resource), an attribute (a property), and a value (a resource).

(v) Ontology

Ontology comprises a set of knowledge terms, including the vocabulary, the semantic interconnections, simple rules of inference and logic for some particular topic. Ontologies applied to the Web are creating the semantic Web.

(vi) Logic, Proof, Trust and Digital Signature

The logic layer is used to enhance the ontology language further and to allow the writing of application-specific declarative knowledge.

The proof layer involves the actual deductive process as well as the representation of proofs in Web languages and proof validation.

Finally, the Trust layer will emerge through the use of digital signatures and other kinds of knowledge, based on recommendations by trusted agents or on rating and certification agencies and consumer bodies.

RDF

At the top of XML, the World Wide Web Consortium (W3C) has developed the Resource Description Framework (RDF) language to standardize the definition and use of metadata. RDF is a simple general-purpose metadata language for representing information in the Web and provides a model for describing and creating relationships between resources. With RDF it is possible to add predefined modeling primitives for expressing semantics of data to a document without making any assumptions about the structure of the document.

The fundamental concepts of RDF are resources, properties and statements.

1. Resources : A resource as an object, can be a thing such as person, a song, or a Web page. Every resource has URI to identify itself.

2. Properties: They are a special kind of resources; they describe relations between resources, for example ―written by‖, ―age‖, ―title‖ and so on.

3. Statements: Statements assert the properties of resources. A statement is an object-attribute-value (O-A-V) tripe, consisting of a resource, a property and a value. Value can be either be resources or literals. It may also be represented as (Subject-Predicate-Object).

The basic structure of RDF is in the form of triples

Object/Resource/Thing ←→ Property/Attribute ←→ Value (is also a resource / literal)

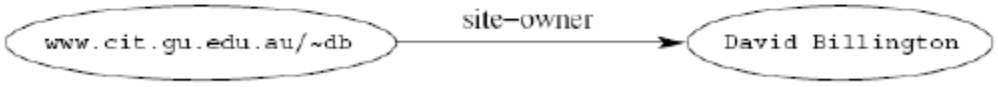
An example of a statement is

David Billington is the owner of the Web page http://www.cit.gu.edu.au/~db.

The simplest way of interpreting this statement is to use the definition and consider the triple

("David Billington", http://www.mydomain.org/site-owner, http://www.cit.gu.edu.au/~db).

The graphical RDF representation



We can think of this triple (x,P,y) as a logical formula P(x,y), where the binary predicate P relates the object x to the object y. In fact, RDF offers only binary predicate (properties).

GIS

A **geographic information system** (**GIS**) is a system designed to capture, store, manipulate, analyze, manage, and present spatial or geographic data. GIS applications are tools that allow users to create interactive queries (user-created searches), analyze spatial information, edit data in maps, and present the results of all these operations.

GIS can refer to a number of different technologies, processes, techniques and methods. It is attached to many operations and has many applications related to engineering, planning, management, transport/logistics, insurance, telecommunications, and business. For that reason, GIS and location intelligence applications can be the foundation for many location-enabled services that rely on analysis and visualization.

GIS can relate unrelated information by using location as the key index variable. Locations or extents in the Earth space–time may be recorded as dates/times of occurrence, and x, y, and z coordinates representing, longitude, latitude, and elevation, respectively. All Earth-based spatial–temporal location and extent references should be relatable to one another and ultimately to a "real" physical location or extent. This key characteristic of GIS has begun to open new avenues of scientific inquiry.

Benefits of GIS

1. Cost savings from greater efficiency
2. Better decision making
3. Improved communication
4. Better record keeping
5. Managing geographically

Real World application of GIS

1. Disaster management
2. Crime statistics
3. Archeology
4. Civic planning
5. Health/medical resourse management
6. transport

Biological Database

Biological databases are libraries of life sciences information, collected from scientific experiments, published literature, high-throughput experiment technology, and computational analysis.They contain information from research areas including genomics, proteomics, metabolomics, microarray gene expression, and phylogenetics. Information contained in biological databases includes gene function, structure, localization (both cellular and chromosomal), clinical effects of mutations as well as similarities of biological sequences and structures.

Biological databases can be broadly classified into sequence, structure and functional databases. Nucleic acid and protein sequences are stored in sequence databases and structure databases store solved structures of RNA and proteins. Functional databases provide information on the physiological role of gene products, for example enzyme activities, mutant phenotypes, or biological pathways.

Model Organism Databases are functional databases that provide species-specific data. Databases are important tools in assisting scientists to analyze and explain a host of biological phenomena from the structure of biomolecules and their interaction, to the whole metabolism of organisms and to understanding the evolution ofspecies. This knowledge helps facilitate the fight against diseases, assists in the development of medications, predicting certain genetic diseases and in discovering basic relationships among species in the history of life.

Biological knowledge is distributed among many different general and specialized databases. This sometimes makes it difficult to ensure the consistency of information. Integrative bioinformatics is one field attempting to tackle this problem by providing unified access. One solution is how biological databases cross-reference to other databases with accession numbers to link their related knowledge together.

Relational database concepts of computer science and Information retrieval concepts of digital libraries are important for understanding biological databases. Biological database design, development, and long-term management is a core area of the discipline of bioinformatics. Data contents include gene sequences, textual descriptions, attributes and ontologyclassifications, citations, and tabular data. These are often described as semi-structured data, and can be represented as tables, key delimited records, and XML structures.

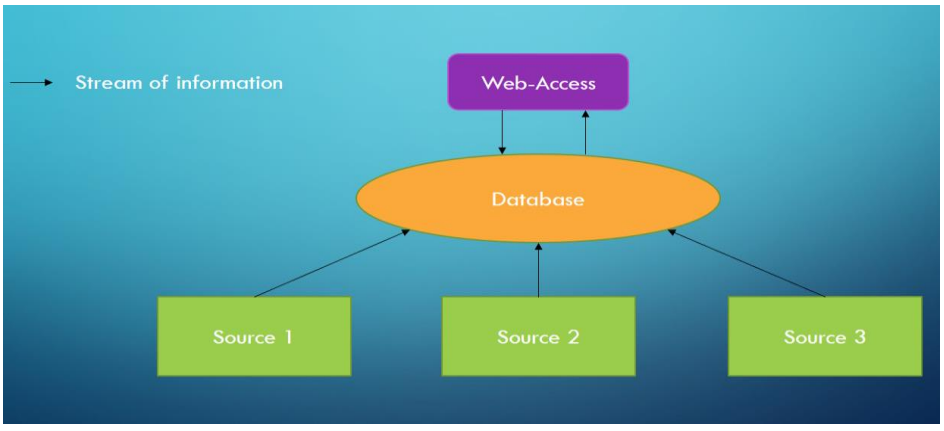
Types of Biological Databases

There are three common concepts of biological databases:

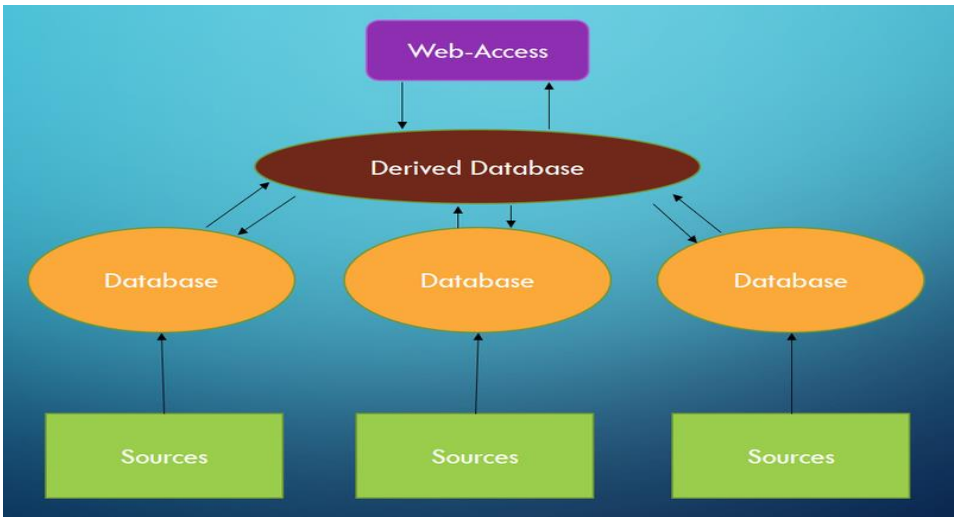
1. Primary Databases,
2. Secondary Databases
3. composite databases.

These three differ in their archive structure. Primary databases often hold only one type of specific data which is stored in their own archive. They upload new data explored in experiments and update entries to ensure the quality of the data.

Secondary databases are databases, which use other databases as their source of information, thus they get their data by requesting.



Concept of Primary databases



Concept of Secondary databases

Big data

Big Data is also **data** but with a **huge size**. Big Data is a term used to describe a collection of data that is huge in size and yet growing exponentially with time. In short such data is so large and complex that none of the traditional data management tools are able to store it or process it efficiently.

Examples Of Big Data

Following are some the examples of Big Data-

1. The **New York Stock Exchange** generates about ***one terabyte*** of new trade data per day.
2. **Social Media**

   The statistic shows that ***500+terabytes*** of new data get ingested into the databases of social media site **Facebook**, every day. This data is mainly generated in terms of photo and video uploads, message exchanges, putting comments etc.

3. A single **Jet engine** can generate ***10+terabytes*** of data in ***30 minutes*** of flight time. With many thousand flights per day, generation of data reaches up to many ***Petabytes.***
4. Weather data
5. Contract data
6. Labour data

7. Maintenance data
8. Financial reporting data
9. Compliance data
10. Clinical trials data
11. Contracts
12. Processing doctor’s notes on diagnosis

Types of Big Data

BigData' could be found in three forms:

1. **Structured**
2. **Unstructured**
3. **Semi-structured**

Structured

Any data that can be stored, accessed and processed in the form of fixed format is termed as a 'structured' data. Over the period of time, talent in computer science has achieved greater success in developing techniques for working with such kind of data (where the format is well known in advance) and also deriving value out of it. However, nowadays, we are foreseeing issues when a size of such data grows to a huge extent, typical sizes are being in the rage of multiple zettabytes.

Examples of Structured Data

An 'Employee' table in a database is an example of Structured Data

Employee_ID	Employee_Name	Gender	Department	Salary_In_lacs
2365	Rajesh Kulkarni	Male	Finance	650000
3398	Pratibha Joshi	Female	Admin	650000

7465	Shushil Roy	Male	Admin	500000
7500	Shubhojit Das	Male	Finance	500000
7699	Priya Sane	Female	Finance	550000

Unstructured

Any data with unknown form or the structure is classified as unstructured data. In addition to the size being huge, un-structured data poses multiple challenges in terms of its processing for deriving value out of it. A typical example of unstructured data is a heterogeneous data source containing a combination of simple text files, images, videos etc. Now day organizations have wealth of data available with them but unfortunately, they don't know how to derive value out of it since this data is in its raw form or unstructured format.

Examples of Un-structured Data

The output returned by 'Google Search'

Semi-structured

Semi-structured data can contain both the forms of data. We can see semi-structured data as a structured in form but it is actually not defined with e.g. a table definition in relational DBMS. Example of semi-structured data is a data represented in an XML file.

Examples Of Semi-structured Data

Personal data stored in an XML file-

```
<rec><name>Prashant Rao</name><sex>Male</sex><age>35</age></rec>
<rec><name>Seema R.</name><sex>Female</sex><age>41</age></rec>
<rec><name>Satish Mane</name><sex>Male</sex><age>29</age></rec>
<rec><name>Subrato Roy</name><sex>Male</sex><age>26</age></rec>
<rec><name>Jeremiah J.</name><sex>Male</sex><age>35</age></rec>
```

Data Growth over the years

Please note that web application data, which is unstructured, consists of log files, transaction history files etc. OLTP systems are built to work with structured data wherein data is stored in relations (tables).

Characteristics of Big Data

(i) Volume – The name Big Data itself is related to a size which is enormous. Size of data plays a very crucial role in determining value out of data. Also, whether a particular data can actually be considered as a Big Data or not, is dependent upon the volume of data. Hence, **'Volume'** is one characteristic which needs to be considered while dealing with Big Data.

(ii) Variety – The next aspect of Big Data is its **variety**.

Variety refers to heterogeneous sources and the nature of data, both structured and unstructured. During earlier days, spreadsheets and databases were the only sources of data considered by most of the applications. Nowadays, data in the form of emails, photos, videos, monitoring devices, PDFs, audio, etc. are also being considered in the analysis applications. This variety of unstructured data poses certain issues for storage, mining and analyzing data.

(iii) Velocity – The term **'velocity'** refers to the speed of generation of data. How fast the data is generated and processed to meet the demands, determines real potential in the data.

Big Data Velocity deals with the speed at which data flows in from sources like business processes, application logs, networks, and social media sites, sensors,Mobile devices, etc. The flow of data is massive and continuous.

(iv) Variability – This refers to the inconsistency which can be shown by the data at times, thus hampering the process of being able to handle and manage the data effectively.

Benefits of Big Data Processing

Ability to process Big Data brings in multiple benefits, such as-

- **Businesses can utilize outside intelligence while taking decisions**

Access to social data from search engines and sites like facebook, twitter are enabling organizations to fine tune their business strategies.

- **Improved customer service**

Traditional customer feedback systems are getting replaced by new systems designed with Big Data technologies. In these new systems, Big Data and natural language processing technologies are being used to read and evaluate consumer responses.

- **Early identification of risk to the product/services, if any**
- **Better operational efficiency**

Big Data technologies can be used for creating a staging area or landing zone for new data before identifying what data should be moved to the data warehouse. In addition, such integration of Big Data technologies and data warehouse helps an organization to offload infrequently accessed data.