**Transaction in DBMS**

**What does a Transaction mean in DBMS?**

Transaction in Database Management Systems (DBMS) can be defined as a set of logically related operations. It is the result of a request made by the user to access the contents of the database and perform operations on it. It consists of various operations and has various states in its completion journey. It also has some specific properties that must be followed to keep the database consistent.

**Operations of Transaction**

A user can make different types of requests to access and modify the contents of a database. So, we have different types of operations relating to a transaction. They are discussed as follows:

**i) Read(X)**

A read operation is used to read the value of X from the database and store it in a buffer in the main memory for further actions such as displaying that value. Such an operation is performed when a user wishes just to see any content of the database and not make any changes to it. For example, when a user wants to check his/her account’s balance, a read operation would be performed on user’s account balance from the database.

**ii) Write(X)**

A write operation is used to write the value to the database from the buffer in the main memory. For a write operation to be performed, first a read operation is performed to bring its value in buffer, and then some changes are made to it, e.g. some set of arithmetic operations are performed on it according to the user’s request, then to store the modified value back in the database, a write operation is performed. For example, when a user requests to withdraw some money from his account, his account balance is fetched from the database using a read operation, then the amount to be deducted from the account is subtracted from this value, and then the obtained value is stored back in the database using a write operation.

**iii) Commit**

This operation in transactions is used to maintain integrity in the database. Due to some failure of power, hardware, or software, etc., a transaction might get interrupted before all its operations are completed. This may cause ambiguity in the database, i.e. it might get inconsistent before and after the transaction. To ensure that further operations of any other transaction are performed only after work of the current transaction is done, a commit operation is performed to the changes made by a transaction permanently to the database.

**iv) Rollback**

This operation is performed to bring the database to the last saved state when any transaction is interrupted in between due to any power, hardware, or software failure. In simple words, it can be said that a rollback operation does undo the operations of transactions that were performed before its interruption to achieve a safe state of the database and avoid any kind of ambiguity or inconsistency.

**States of transactions**

**i) Active**

It is the first stage of any transaction when it has begun to execute. The execution of the transaction takes place in this state. Operations such as insertion, deletion, or updation are performed during this state. During this state, the data records are under manipulation and they are not saved to the database, rather they remain somewhere in a buffer in the main memory.

**ii) Partially Committed**

This state of transaction is achieved when it has completed most of the operations and is executing its final operation. It can be a signal to the commit operation, as after the final operation of the transaction completes its execution, the data has to be saved to the database through the commit operation. If some kind of error occurs during this state, the transaction goes into a failed state, else it goes into the Committed state.

**iii) Commited**

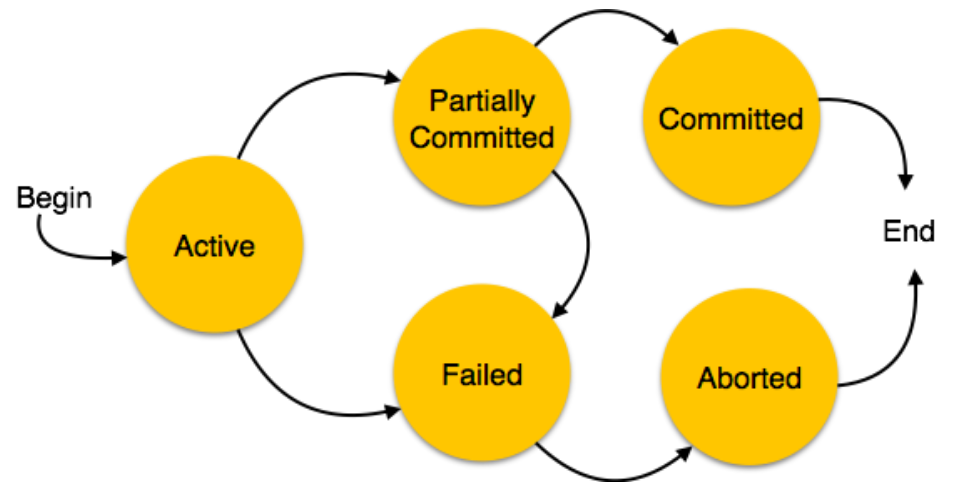
This state of transaction is achieved when all the transaction-related operations have been executed successfully along with the Commit operation, i.e. data is saved into the database after the required manipulations in this state. This marks the successful completion of a transaction.

**iv) Failed**

If any of the transaction-related operations cause an error during the active or partially committed state, further execution of the transaction is stopped and it is brought into a failed state. Here, the database recovery system makes sure that the database is in a consistent state.

**v) Aborted**

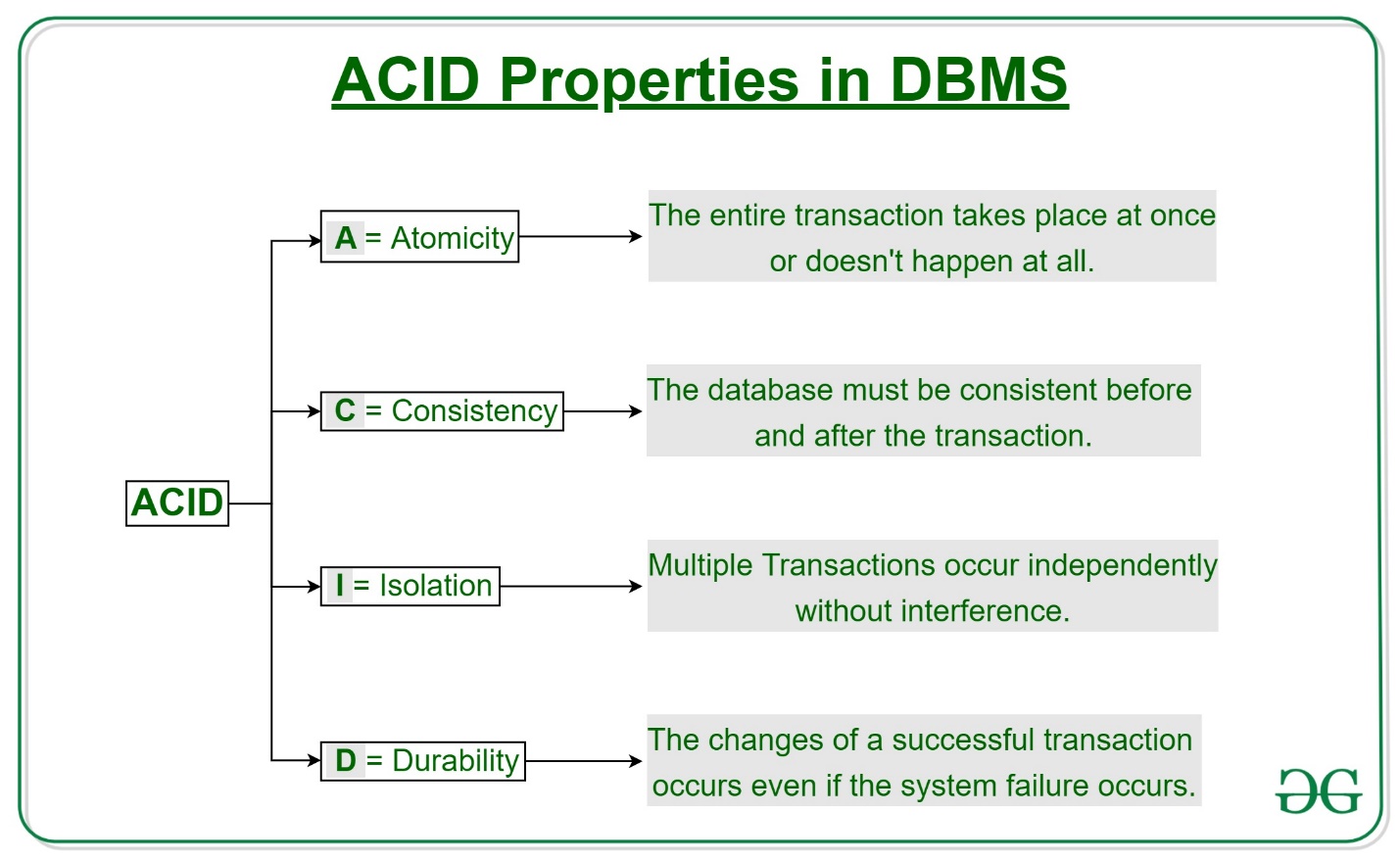
If the error is not resolved in the failed state, then the transaction is aborted and a rollback operation is performed to bring database to the the last saved consistent state. When the transaction is aborted, the database recovery module either restarts the transaction or kills it.



**ACID Properties in DBMS**

The **ACID** properties provide a mechanism to ensure the correctness and consistency of a database in a way such that each transaction is a group of operations that acts as a single unit, produces consistent results, acts in isolation from other operations, and updates that it makes are durably stored.

The acronym ACID stands for Atomicity, Consistency, Isolation, and Durability.

****

**Atomicity:**

Atomicity ensures that a transaction is treated as a single, indivisible unit of work. Either all the operations within the transaction are completed successfully, or none of them are. There is no midway i.e. transactions do not occur partially.If any part of the transaction fails, the entire transaction is rolled back to its original state, ensuring data consistency and integrity. It involves the following two operations.   
—**Abort**: If a transaction aborts, changes made to the database are not visible.   
—**Commit**: If a transaction commits, changes made are visible.   
Atomicity is also known as the ‘All or nothing rule’.

Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.



If the transaction fails after completion of **T1** but before completion of **T2**.( say, after **write(X)** but before **write(Y)**), then the amount has been deducted from **X** but not added to **Y**. This results in an inconsistent database state. Therefore, the transaction must be executed in its entirety in order to ensure the correctness of the database state.

**Consistency:**

Consistency ensures that a transaction takes the database from one consistent state to another consistent state. The database is in a consistent state both before and after the transaction is executed. Constraints, such as unique keys and foreign keys, must be maintained to ensure data consistency.  
Total **before T** occurs = **500 + 200 = 700**.   
Total **after T occurs** = **400 + 300 = 700**.   
Therefore, the database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result, T is incomplete.

**Isolation:**

Isolation ensures that multiple transactions can execute concurrently without interfering with each other. Each transaction must be isolated from other transactions until it is completed. This isolation prevents dirty reads, non-repeatable reads, and phantom reads.   
Let **X**= 500, **Y** = 500.   
Consider two transactions **T** and **T”.**



Suppose **T** has been executed till **Read (Y)** and then **T’’** starts. As a result, interleaving of operations takes place due to which **T’’** reads the correct value of **X** but the incorrect value of **Y** and sum computed by   
**T’’: (X+Y = 50, 000+500=50, 500)**   
is thus not consistent with the sum at end of the transaction:   
**T: (X+Y = 50, 000 + 450 = 50, 450)**.   
This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

**Durability:**

Durability ensures that once a transaction is committed, its changes are permanent and will survive any subsequent system failures. The transaction’s changes are saved to the database permanently, and even if the system crashes, the changes remain intact and can be recovered. Thus, the effects of the transaction are never lost.   
  
**Some important points:**

| **Property** | **Responsibility for maintaining properties** |
| --- | --- |
| Atomicity | Transaction Manager |
| Consistency | Application programmer |
| Isolation | Concurrency Control Manager |
| Durability | Recovery Manager |

**Advantages of ACID Properties in DBMS:**

1. Data Consistency: ACID properties ensure that the data remains consistent and accurate after any transaction execution.
2. Data Integrity: ACID properties maintain the integrity of the data by ensuring that any changes to the database are permanent and cannot be lost.
3. Concurrency Control: ACID properties help to manage multiple transactions occurring concurrently by preventing interference between them.
4. Recovery: ACID properties ensure that in case of any failure or crash, the system can recover the data up to the point of failure or crash.

**Disadvantages of ACID Properties in DBMS:**

1. Performance: The ACID properties can cause a performance overhead in the system, as they require additional processing to ensure data consistency and integrity.
2. Scalability: The ACID properties may cause scalability issues in large distributed systems where multiple transactions occur concurrently.
3. Complexity: Implementing the ACID properties can increase the complexity of the system and require significant expertise and resources.  
   Overall, the advantages of ACID properties in DBMS outweigh the disadvantages. They provide a reliable and consistent approach to data
4. management, ensuring data integrity, accuracy, and reliability. However, in some cases, the overhead of implementing ACID properties can cause performance and scalability issues. Therefore, it’s important to balance the benefits of ACID properties against the specific needs and requirements of the system.

# Serializability in DBMS

**Transaction Schedules**

When multiple transaction requests are made at the same time, we need to decide their order of execution. Thus, a transaction schedule can be defined as a chronological order of execution of multiple transactions. There are broadly two types of transaction schedules discussed as follows,

**i) Serial Schedule**

In this kind of schedule, when multiple transactions are to be executed, they are executed serially, i.e. at one time only one transaction is executed while others wait for the execution of the current transaction to be completed. This ensures consistency in the database as transactions do not execute simultaneously. But, it increases the waiting time of the transactions in the queue, which in turn lowers the throughput of the system, i.e. number of transactions executed per time. To improve the throughput of the system, another kind of schedule are used which has some more strict rules which help the database to remain consistent even when transactions execute simultaneously.

**ii) Non-Serial Schedule/Parallel Schedule**

To reduce the waiting time of transactions in the waiting queue and improve the system efficiency, we use nonserial schedules which allow multiple transactions to start before a transaction is completely executed. This may sometimes result in inconsistency and errors in database operation. So, these errors are handled with specific algorithms to maintain the consistency of the database and improve CPU throughput as well. Non-Serial Schedules are also sometimes referred to as parallel schedules as transactions execute in parallel in this kind of schedules.

**Serializable**

Serializability in DBMS is the property of a nonserial schedule that determines whether it would maintain the database consistency or not. The nonserial schedule which ensures that the database would be consistent after the transactions are executed in the order determined by that schedule is said to be Serializable Schedules. The serial schedules always maintain database consistency as a transaction starts only when the execution of the other transaction has been completed under it. Thus, serial schedules are always serializable.

## What is a serializable schedule, and what is it used for?

If a non-serial schedule can be transformed into its corresponding serial schedule, it is said to be serializable. Simply said, a non-serial schedule is referred to as a serializable schedule if it yields the same results as a serial timetable.

### Non-serial Schedule

A schedule where the transactions are overlapping or switching places. As they are used to carry out actual database operations, multiple transactions are running at once. It’s possible that these transactions are focusing on the same data set. Therefore, it is crucial that non-serial schedules can be serialized in order for our database to be consistent both before and after the transactions are executed.

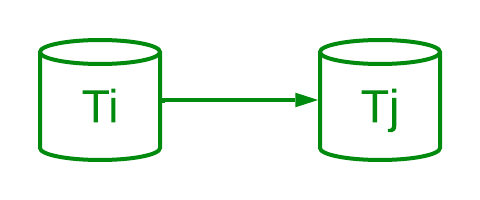
#### Example:

| **Transaction-1** | **Transaction-2** |
| --- | --- |
| R(a) |  |
| W(a) |  |
|  | R(b) |
|  | W(b) |
| R(b) |  |
|  | R(a) |
| W(b) |  |
|  | W(a) |

We can observe that Transaction-2 begins its execution before Transaction-1 is finished, and they are both working on the same data, i.e., “a” and “b”, interchangeably. Where “R”-Read, “W”-Write

## Serializability testing

We can utilize the Serialization Graph or Precedence Graph to examine a schedule’s serializability. A schedule’s full transactions are organized into a Directed Graph, what a serialization graph is.

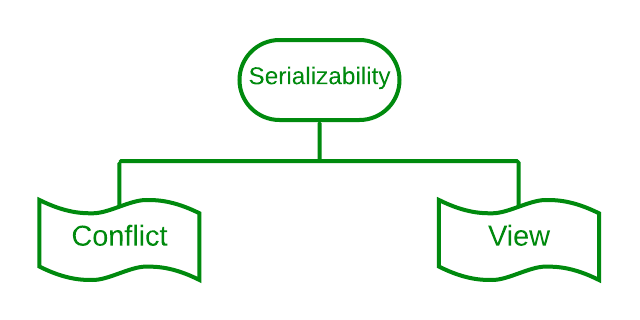


*Precedence Graph*

It can be described as a Graph G(V, E) with vertices V = “V1, V2, V3,…, Vn” and directed edges E = “E1, E2, E3,…, En”. One of the two operations—READ or WRITE—performed by a certain transaction is contained in the collection of edges. Where Ti -> Tj, means Transaction-Ti is either performing read or write before the transaction-Tj.

## Types of Serializability

There are two ways to check whether any non-serial schedule is serializable.



### 1. Conflict serializability

[Conflict serializability](https://www.geeksforgeeks.org/conflict-serializability-in-dbms/) refers to a subset of serializability that focuses on maintaining the consistency of a database while ensuring that identical data items are executed in an order. In a DBMS each transaction has a value and all the transactions, in the database rely on this uniqueness. This uniqueness ensures that no two operations with the conflict value can occur simultaneously.

For example lets consider an order table and a customer table as two instances. Each order is associated with one customer even though a single client may place orders. However there are restrictions for achieving conflict serializability in the database. Here are a few of them.

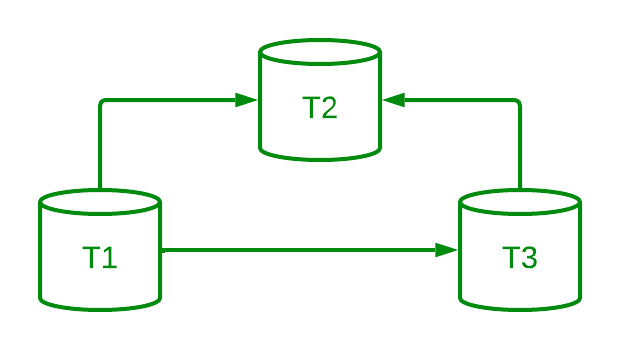
1. Different transactions should be used for the two procedures.
2. The identical data item should be present in both transactions.
3. Between the two operations, there should be at least one write operation.

#### Example

Three transactions—t1, t2, and t3—are active on a schedule “S” at once. Let’s create a graph of precedence.

| **Transaction – 1 (t1)** | **Transaction – 2 (t2)** | **Transaction – 3 (t3)** |
| --- | --- | --- |
| R(a) |  |  |
|  | R(b) |  |
|  |  | R(b) |
|  | W(b) |  |
| W(a) |  |  |
|  |  | W(a) |
|  | R(a) |  |
|  | W(a) |  |

It is a conflict serializable schedule as well as a serial schedule because the graph (a DAG) has no loops. We can also determine the order of transactions because it is a serial schedule.



*DAG of transactions*

As there is no incoming edge on Transaction 1, Transaction 1 will be executed first. T3 will run second because it only depends on T1. Due to its dependence on both T1 and T3, t2 will finally be executed.

Therefore, the serial schedule’s equivalent order is: t1 –> t3 –> t2

**Note:** A schedule is unquestionably consistent if it is conflicting serializable. A non-conflicting serializable schedule, on the other hand, might or might not be serial. We employ the idea of View Serializability to further examine its serial behavior.

### 2. View Serializability

[View serializability](https://www.geeksforgeeks.org/view-serializability-in-dbms/) is a kind of operation in a serializable in which each transaction should provide some results, and these outcomes are the output of properly sequentially executing the data item. The view serializability, in contrast to conflict serialized, is concerned with avoiding database inconsistency. The view serializability feature of DBMS enables users to see databases in contradictory ways.

To further understand view serializability in DBMS, we need to understand the schedules S1 and S2. The two transactions T1 and T2 should be used to establish these two schedules. Each schedule must follow the three transactions in order to retain the equivalent of the transaction. These three circumstances are listed below.

1. The first prerequisite is that the same kind of transaction appears on every schedule. This requirement means that the same kind of group of transactions cannot appear on both schedules S1 and S2. The schedules are not equal to one another if one schedule commits a transaction but it does not match the transaction of the other schedule.
2. The second requirement is that different read or write operations should not be used in either schedule. On the other hand, we say that two schedules are not similar if schedule S1 has two write operations whereas schedule S2 only has one. The number of the write operation must be the same in both schedules, however there is no issue if the number of the read operation is different.
3. The second to last requirement is that there should not be a conflict between either timetable. execution order for a single data item. Assume, for instance, that schedule S1’s transaction is T1, and schedule S2’s transaction is T2. The data item A is written by both the transaction T1 and the transaction T2. The schedules are not equal in this instance. However, we referred to the schedule as equivalent to one another if it had the same number of all write operations in the data item.

## What is view equivalency?

Schedules (S1 and S2) must satisfy these two requirements in order to be viewed as equivalent:

1. The same piece of data must be read for the first time.For instance, if transaction t1 is reading “A” from the database in schedule S1, then t1 must also read A in schedule S2.
2. The same piece of data must be used for the final write. As an illustration, if transaction t1 updated A last in S1, it should also conduct final write in S2.
3. The middle sequence need to follow suit. As an illustration, if in S1 t1 is reading A, and t2 updates A, then in S2 t1 should read A, and t2 should update A.

View Serializability refers to the process of determining whether a schedule’s views are equivalent.

#### Example

We have a schedule “S” with two concurrently running[transactions](https://www.geeksforgeeks.org/sql-transactions/), “t1” and “t2.”

**Schedule – S:**

| **Transaction-1 (t1)** | **Transaction-2 (t2)** |
| --- | --- |
| R(a) |  |
| W(a) |  |
|  | R(a) |
|  | W(a) |
| R(b) |  |
| W(b) |  |
|  | R(b) |
|  | W(b) |

By switching between both transactions’ mid-read-write operations, let’s create its view equivalent schedule (S’).

**Schedule – S’:**

| **Transaction-1 (t1)** | **Transaction-2 (t2)** |
| --- | --- |
| R(a) |  |
| W(a) |  |
| R(b) |  |
| W(b) |  |
|  | R(a) |
|  | W(a) |
|  | R(b) |
|  | W(b) |

It is a view serializable schedule since a view similar schedule is conceivable.

**Note**: A conflict serializable schedule is always viewed as serializable, but vice versa is not always true.

## Advantages of Serializability

1. **Execution is predictable:**In serializable, the DBMS’s threads are all performed simultaneously. The DBMS doesn’t include any such surprises. In [DBMS](https://www.geeksforgeeks.org/dbms/), no data loss or corruption occurs and all variables are updated as intended.
2. DBMS executes each thread independently, making it much simpler to understand and troubleshoot each database thread. This can greatly simplify the debugging process. The concurrent process is therefore not a concern for us.
3. **Lower Costs:** The cost of the hardware required for the efficient operation of the database can be decreased with the aid of the serializable property. It may also lower the price of developing the software.
4. **Increased Performance:**Since serializable executions provide developers the opportunity to optimize their code for performance, they occasionally outperform non-serializable equivalents.

For a DBMS transaction to be regarded as serializable, it must adhere to the [ACID properties](https://www.geeksforgeeks.org/acid-properties-in-dbms/). In DBMS, serializability comes in a variety of forms, each having advantages and disadvantages of its own. Most of the time, choosing the best sort of serializability involves making a choice between performance and correctness.

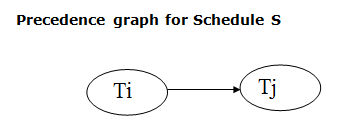
Making the incorrect choice for serializability might result in database issues that are challenging to track down and resolve. You should now have a better knowledge of how serializability in DBMS functions and the different types that are available thanks to this guide.

Testing of Serializability

Serialization Graph is used to test the Serializability of a schedule.

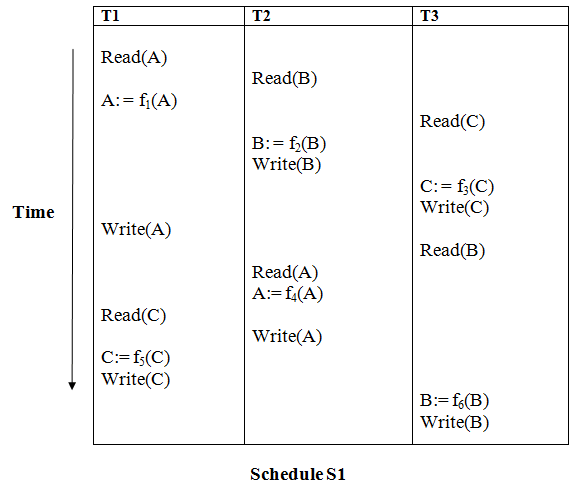
Assume a schedule S. For S, we construct a graph known as precedence graph. This graph has a pair G = (V, E), where V consists a set of vertices, and E consists a set of edges. The set of vertices is used to contain all the transactions participating in the schedule. The set of edges is used to contain all edges Ti ->Tj for which one of the three conditions holds:

1. Create a node Ti → Tj if Ti executes write (Q) before Tj executes read (Q).
2. Create a node Ti → Tj if Ti executes read (Q) before Tj executes write (Q).
3. Create a node Ti → Tj if Ti executes write (Q) before Tj executes write (Q).



* If a precedence graph contains a single edge Ti → Tj, then all the instructions of Ti are executed before the first instruction of Tj is executed.
* If a precedence graph for schedule S contains a cycle, then S is non-serializable. If the precedence graph has no cycle, then S is known as serializable.

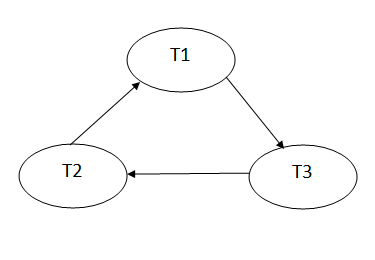
**For example:**



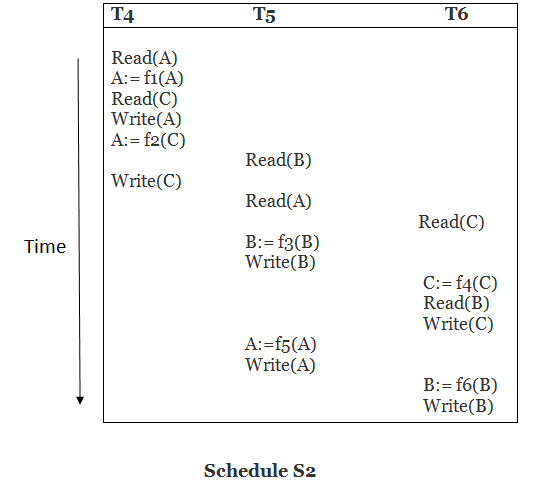
**Explanation:**

**Read(A):** In T1, no subsequent writes to A, so no new edges  
**Read(B):** In T2, no subsequent writes to B, so no new edges  
**Read(C):** In T3, no subsequent writes to C, so no new edges  
**Write(B):** B is subsequently read by T3, so add edge T2 → T3  
**Write(C):** C is subsequently read by T1, so add edge T3 → T1  
**Write(A):** A is subsequently read by T2, so add edge T1 → T2  
**Write(A):** In T2, no subsequent reads to A, so no new edges  
**Write(C):** In T1, no subsequent reads to C, so no new edges  
**Write(B):** In T3, no subsequent reads to B, so no new edges

Precedence graph for schedule S1:



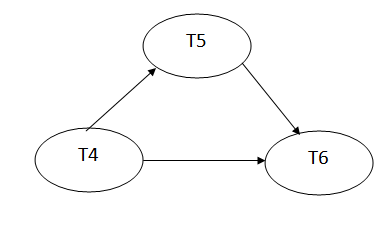
The precedence graph for schedule S1 contains a cycle that's why Schedule S1 is non-serializable.



**Explanation:**

**Read(A):** In T4,no subsequent writes to A, so no new edges  
**Read(C):** In T4, no subsequent writes to C, so no new edges  
**Write(A):** A is subsequently read by T5, so add edge T4 → T5  
**Read(B):** In T5,no subsequent writes to B, so no new edges  
**Write(C):** C is subsequently read by T6, so add edge T4 → T6  
**Write(B):** A is subsequently read by T6, so add edge T5 → T6  
**Write(C):** In T6, no subsequent reads to C, so no new edges  
**Write(A):** In T5, no subsequent reads to A, so no new edges  
**Write(B):** In T6, no subsequent reads to B, so no new edges

Precedence graph for schedule S2:



The precedence graph for schedule S2 contains no cycle that's why ScheduleS2 is serializable.

Lock-Based Protocol

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock:

**1. Shared lock:**

* It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.
* It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.

**2. Exclusive lock:**

* In the exclusive lock, the data item can be both reads as well as written by the transaction.
* This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

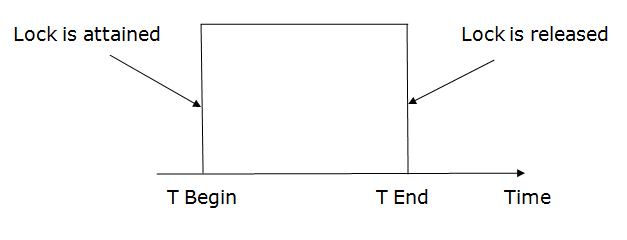
There are four types of lock protocols available:

1. Simplistic lock protocol

It is the simplest way of locking the data while transaction. Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it. It will unlock the data item after completing the transaction.

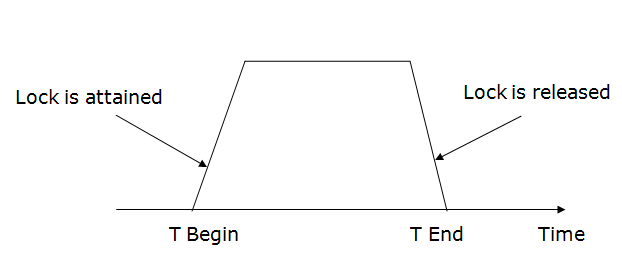
2. Pre-claiming Lock Protocol

* Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.
* Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.
* If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.
* If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.



3. Two-phase locking (2PL)

* The two-phase locking protocol divides the execution phase of the transaction into three parts.
* In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
* In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.
* In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.



There are two phases of 2PL:

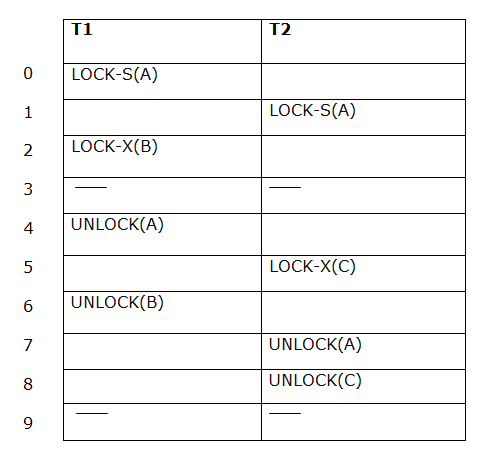
**Growing phase:** In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

**Shrinking phase:** In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

In the below example, if lock conversion is allowed then the following phase can happen:

1. Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.
2. Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

**Example:**



The following way shows how unlocking and locking work with 2-PL.

**Transaction T1:**

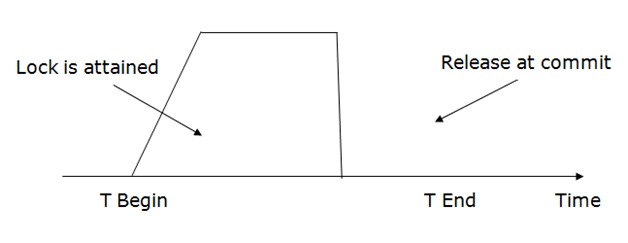
* **Growing phase:** from step 1-3
* **Shrinking phase:** from step 5-7
* **Lock point:** at 3

**Transaction T2:**

* **Growing phase:** from step 2-6
* **Shrinking phase:** from step 8-9
* **Lock point:** at 6

4. Strict Two-phase locking (Strict-2PL)

* The first phase of Strict-2PL is similar to 2PL. In the first phase, after acquiring all the locks, the transaction continues to execute normally.
* The only difference between 2PL and strict 2PL is that Strict-2PL does not release a lock after using it.
* Strict-2PL waits until the whole transaction to commit, and then it releases all the locks at a time.
* Strict-2PL protocol does not have shrinking phase of lock release.



It does not have cascading abort as 2PL does.

Timestamp Ordering Protocol

* The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.
* The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.
* The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.
* Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.
* The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

**Basic Timestamp ordering protocol works as follows:**

1. Check the following condition whenever a transaction Ti issues a **Read (X)** operation:

* If W\_TS(X) >TS(Ti) then the operation is rejected.
* If W\_TS(X) <= TS(Ti) then the operation is executed.
* Timestamps of all the data items are updated.

2. Check the following condition whenever a transaction Ti issues a **Write(X)** operation:

* If TS(Ti) < R\_TS(X) then the operation is rejected.
* If TS(Ti) < W\_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

**Where,**

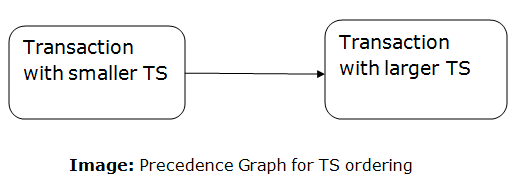
**TS(TI)** denotes the timestamp of the transaction Ti.

**R\_TS(X)** denotes the Read time-stamp of data-item X.

**W\_TS(X)** denotes the Write time-stamp of data-item X.

Advantages and Disadvantages of TO protocol:

* TO protocol ensures serializability since the precedence graph is as follows:



* TS protocol ensures freedom from deadlock that means no transaction ever waits.
* But the schedule may not be recoverable and may not even be cascade-

Validation Based Protocol

Validation phase is also known as optimistic concurrency control technique. In the validation based protocol, the transaction is executed in the following three phases:

1. **Read phase:** In this phase, the transaction T is read and executed. It is used to read the value of various data items and stores them in temporary local variables. It can perform all the write operations on temporary variables without an update to the actual database.
2. **Validation phase:** In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability.
3. **Write phase:** If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back.

Here each phase has the following different timestamps:

**Start(Ti):** It contains the time when Ti started its execution.

**Validation (Ti):** It contains the time when Ti finishes its read phase and starts its validation phase.

**Finish(Ti):** It contains the time when Ti finishes its write phase.

* This protocol is used to determine the time stamp for the transaction for serialization using the time stamp of the validation phase, as it is the actual phase which determines if the transaction will commit or rollback.
* Hence TS(T) = validation(T).
* The serializability is determined during the validation process. It can't be decided in advance.
* While executing the transaction, it ensures a greater degree of concurrency and also less number of conflicts.
* Thus it contains transactions which have less number of rollbacks.

**Database Recovery Techniques in DBMS**

Database Systems like any other computer system, are subject to failures but the data stored in them must be available as and when required. When a database fails it must possess the facilities for fast recovery. It must also have atomicity i.e. either transactions are completed successfully and committed (the effect is recorded permanently in the database) or the transaction should have no effect on the database.

**Types of Recovery Techniques in DBMS**

Database recovery techniques are used in database management systems (DBMS) to restore a database to a consistent state after a failure or error has occurred. The main goal of recovery techniques is to ensure data integrity and consistency and prevent data loss.

There are mainly two types of recovery techniques used in DBMS

* **Rollback/Undo Recovery Technique**
* **Commit/Redo Recovery Technique**

**Rollback/Undo Recovery Technique**

The rollback/undo recovery technique is based on the principle of backing out or undoing the effects of a transaction that has not been completed successfully due to a system failure or error. This technique is accomplished by undoing the changes made by the transaction using the log records stored in the transaction log. The transaction log contains a record of all the transactions that have been performed on the database. The system uses the log records to undo the changes made by the failed transaction and restore the database to its previous state.

**Commit/Redo Recovery Technique**

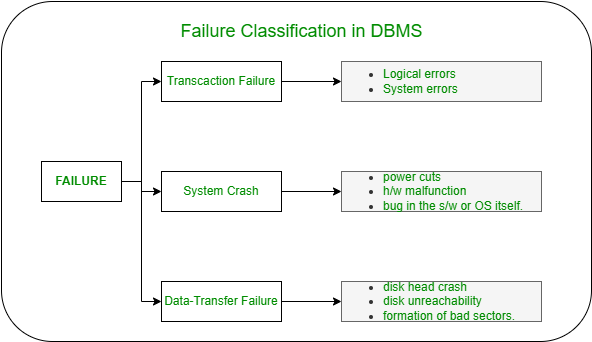
The commit/redo recovery technique is based on the principle of reapplying the changes made by a transaction that has been completed successfully to the database. This technique is accomplished by using the log records stored in the transaction log to redo the changes made by the transaction that was in progress at the time of the failure or error. The system uses the log records to reapply the changes made by the transaction and restore the database to its most recent consistent state.

**Failure Classification in DBMS**

Failure in terms of a database can be defined as its inability to execute the specified transaction or loss of data from the database. A DBMS is vulnerable to several kinds of failures and each of these failures needs to be managed differently. There are many reasons that can cause database failures such as network failure, system crash, natural disasters, carelessness, sabotage(corrupting the data intentionally), software errors, etc.

**Failure Classification in DBMS**

A failure in [DBMS](https://www.geeksforgeeks.org/introduction-of-dbms-database-management-system-set-1/) can be classified as:



*Failure Classification in DBMS*

**Transaction Failure:**

If a transaction is not able to execute or it comes to a point from where the transaction becomes incapable of executing further then it is termed as a failure in a transaction.

**Reason for a transaction failure in DBMS:**

1. **Logical error:** A logical error occurs if a transaction is unable to execute because of some mistakes in the code or due to the presence of some internal faults.
2. **System error:** Where the termination of an active transaction is done by the database system itself due to some system issue or because the database management system is unable to proceed with the transaction. *For example*– The system ends an operating transaction if it reaches a deadlock condition or if there is an unavailability of resources.

**System Crash:**

A system crash usually occurs when there is some sort of hardware or software breakdown. Some other problems which are external to the system and cause the system to abruptly stop or eventually crash include failure of the transaction, operating system errors, power cuts, main memory crash, etc.

These types of failures are often termed soft failures and are responsible for the data losses in the volatile memory. It is assumed that a system crash does not have any effect on the data stored in the non-volatile storage and this is known as the *fail-stop assumption*.

**Data-transfer Failure:**

When a disk failure occurs amid data-transfer operation resulting in loss of content from disk storage then such failures are categorized as data-transfer failures. Some other reason for disk failures includes disk head crash, disk unreachability, formation of bad sectors, read-write errors on the disk, etc.

# RAID (Redundant Arrays of Independent Disks)

RAID is a technique that makes use of a combination of multiple disks instead of using a single disk for increased performance, data redundancy, or both. The term was coined by David Patterson, Garth A. Gibson, and Randy Katz at the University of California, Berkeley in 1987.

## ****Why Data Redundancy?****

Data redundancy, although taking up extra space, adds to disk reliability. This means, that in case of disk failure, if the same data is also backed up onto another disk, we can retrieve the data and go on with the operation. On the other hand, if the data is spread across multiple disks without the RAID technique, the loss of a single disk can affect the entire data.

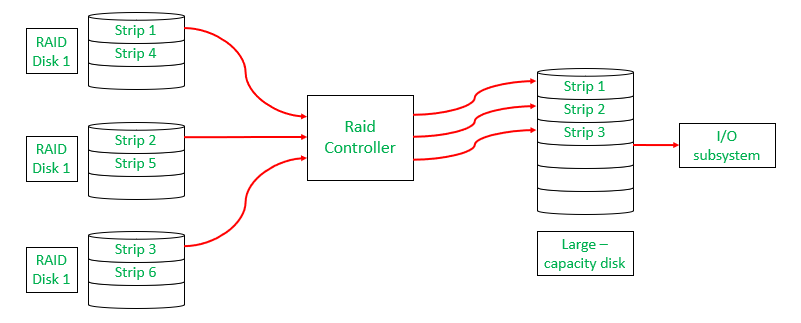
## ****Key Evaluation Points for a RAID System****

* **Reliability:**How many disk faults can the system tolerate?
* **Availability:** What fraction of the total session time is a system in uptime mode, i.e. how available is the system for actual use?
* **Performance:**How good is the response time? How high is the throughput (rate of processing work)? Note that performance contains a lot of parameters and not just the two.
* **Capacity:** Given a set of N disks each with B blocks, how much useful capacity is available to the user?

RAID is very transparent to the underlying system. This means, that to the host system, it appears as a single big disk presenting itself as a linear array of blocks. This allows older technologies to be replaced by RAID without making too many changes to the existing code.

## ****Different RAID Levels****

1. [RAID-0 (Stripping)](https://www.geeksforgeeks.org/difference-between-raid-0-and-raid-1/)
2. [RAID-1 (Mirroring)](https://www.geeksforgeeks.org/difference-between-raid-0-and-raid-1/)
3. [RAID-2 (Bit-Level Stripping with Dedicated Parity)](https://www.geeksforgeeks.org/difference-between-raid-2-and-raid-3/)
4. [RAID-3 (Byte-Level Stripping with Dedicated Parity)](https://www.geeksforgeeks.org/difference-between-raid-2-and-raid-3/)
5. [RAID-4 (Block-Level Stripping with Dedicated Parity)](https://www.geeksforgeeks.org/difference-between-raid-3-and-raid-4/)
6. RAID-5 (Block-Level Stripping with Distributed Parity)
7. RAID-6 (Block-Level Stripping with two Parity Bits)



*Raid Controller*

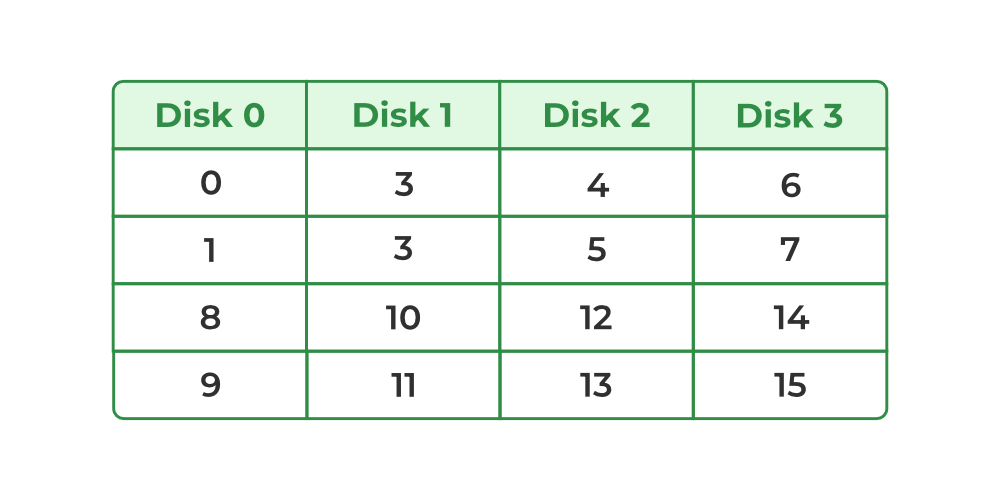
### ****1. RAID-0 (Stripping)****

* Blocks are “stripped” across disks.



*RAID-0*

* In the figure, blocks “0,1,2,3” form a stripe.
* Instead of placing just one block into a disk at a time, we can work with two (or more) blocks placed into a disk before moving on to the next one.



*Raid-0*

#### **Evaluation**

* **Reliability:** 0   
  There is no duplication of data. Hence, a block once lost cannot be recovered.
* **Capacity:** N\*B   
  The entire space is being used to store data. Since there is no duplication, N disks each having B blocks are fully utilized.

#### **Advantages**

1. It is easy to implement.
2. It utilizes the storage capacity in a better way.

#### **Disadvantages**

1. A single drive loss can result in the complete failure of the system.
2. Not a good choice for a critical system.

### ****2. RAID-1 (Mirroring)****

* More than one copy of each block is stored in a separate disk. Thus, every block has two (or more) copies, lying on different disks.



*Raid-1*

* The above figure shows a RAID-1 system with mirroring level 2.
* RAID 0 was unable to tolerate any disk failure. But RAID 1 is capable of reliability.

#### **Evaluation**

Assume a RAID system with mirroring level 2.

* **Reliability:**1 to N/2   
  1 disk failure can be handled for certain because blocks of that disk would have duplicates on some other disk. If we are lucky enough and disks 0 and 2 fail, then again this can be handled as the blocks of these disks have duplicates on disks 1 and 3. So, in the best case, N/2 disk failures can be handled.
* **Capacity:**N\*B/2   
  Only half the space is being used to store data. The other half is just a mirror of the already stored data.

#### **Advantages**

1. It covers complete redundancy.
2. It can increase data security and speed.

#### **Disadvantages**

1. It is highly expensive.
2. Storage capacity is less.

### ****3. RAID-2 (Bit-Level Stripping with Dedicated Parity)****

* In Raid-2, the error of the data is checked at every bit level. Here, we use [Hamming Code Parity Method](https://www.geeksforgeeks.org/hamming-code-in-computer-network/) to find the error in the data.
* It uses one designated drive to store parity.
* The structure of Raid-2 is very complex as we use two disks in this technique. One word is used to store bits of each word and another word is used to store error code correction.
* It is not commonly used.

#### **Advantages**

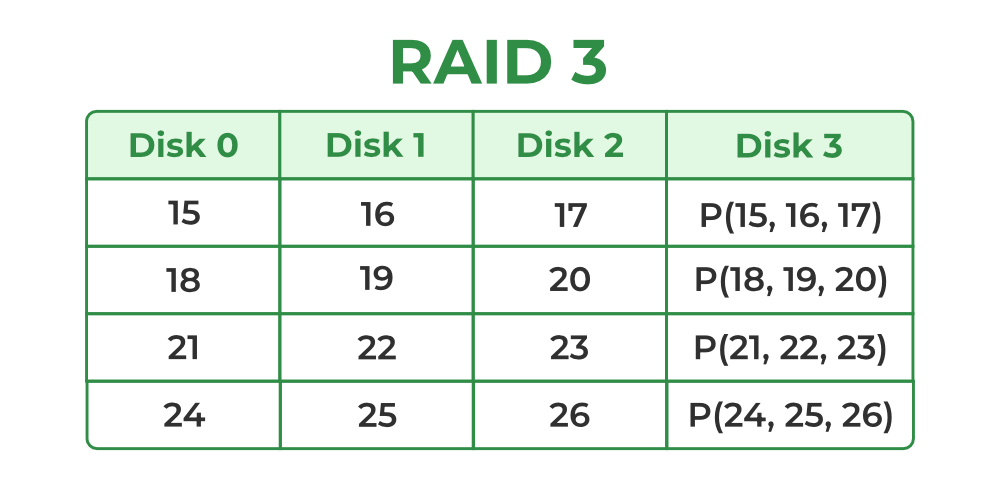
1. In case of Error Correction, it uses hamming code.
2. It Uses one designated drive to store parity.

#### **Disadvantages**

1. It has a complex structure and high cost due to extra drive.
2. It requires an extra drive for error detection.

### ****4. RAID-3 (Byte-Level Stripping with Dedicated Parity)****

* It consists of byte-level striping with dedicated parity striping.
* At this level, we store parity information in a disc section and write to a dedicated parity drive.
* Whenever failure of the drive occurs, it helps in accessing the parity drive, through which we can reconstruct the data.



*Raid-3*

* Here Disk 3 contains the Parity bits for Disk 0, Disk 1, and Disk 2. If data loss occurs, we can construct it with Disk 3.

#### **Advantages**

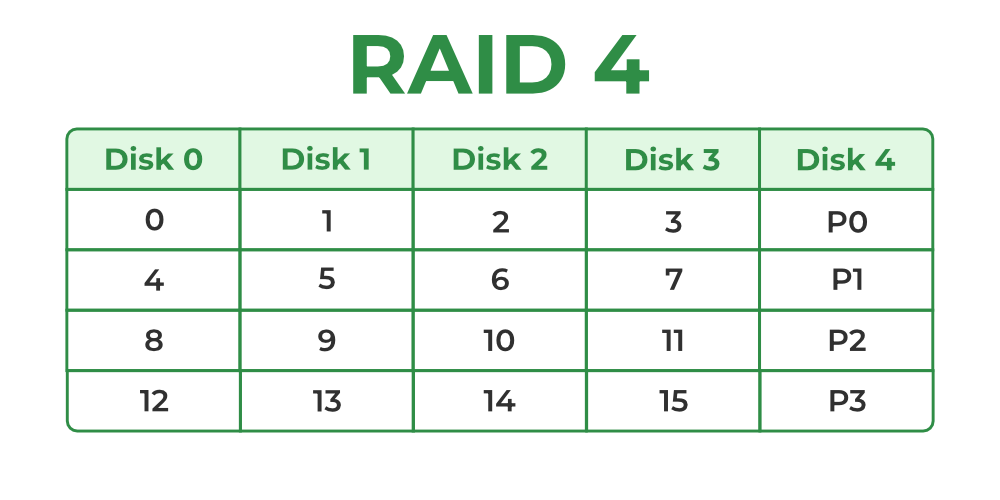
1. Data can be transferred in bulk.
2. Data can be accessed in parallel.

#### **Disadvantages**

1. It requires an additional drive for parity.
2. In the case of small-size files, it performs slowly.

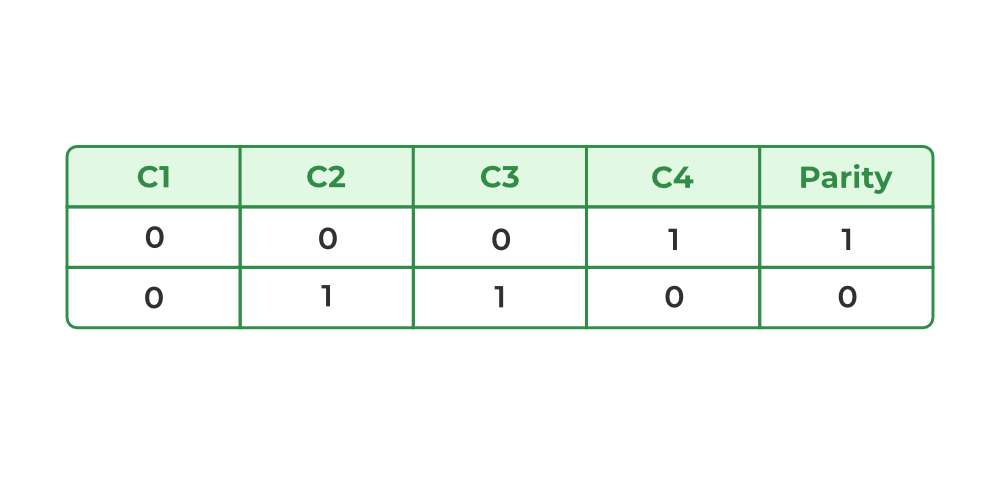
### ****5. RAID-4 (Block-Level Stripping with Dedicated Parity)****

* Instead of duplicating data, this adopts a parity-based approach.



*Raid-4*

* In the figure, we can observe one column (disk) dedicated to parity.
* Parity is calculated using a simple XOR function. If the data bits are 0,0,0,1 the parity bit is XOR(0,0,0,1) = 1. If the data bits are 0,1,1,0 the parity bit is XOR(0,1,1,0) = 0. A simple approach is that an even number of ones results in parity 0, and an odd number of ones results in parity 1.



*Raid-4*

* Assume that in the above figure, C3 is lost due to some disk failure. Then, we can recompute the data bit stored in C3 by looking at the values of all the other columns and the parity bit. This allows us to recover lost data.

#### **Evaluation**

* **Reliability:** 1   
  RAID-4 allows recovery of at most 1 disk failure (because of the way parity works). If more than one disk fails, there is no way to recover the data.
* **Capacity:** (N-1)\*B   
  One disk in the system is reserved for storing the parity. Hence, (N-1) disks are made available for data storage, each disk having B blocks.

#### **Advantages**

1. It helps in reconstructing the data if at most one data is lost.

#### **Disadvantages**

1. It can’t help in reconstructing when more than one data is lost.

### ****6. RAID-5 (Block-Level Stripping with Distributed Parity)****

* This is a slight modification of the RAID-4 system where the only difference is that the parity rotates among the drives.



*Raid-5*

* In the figure, we can notice how the parity bit “rotates”.
* This was introduced to make the random write performance better.

#### **Evaluation**

* **Reliability:** 1   
  RAID-5 allows recovery of at most 1 disk failure (because of the way parity works). If more than one disk fails, there is no way to recover the data. This is identical to RAID-4.
* **Capacity:** (N-1)\*B   
  Overall, space equivalent to one disk is utilized in storing the parity. Hence, (N-1) disks are made available for data storage, each disk having B blocks.

#### **Advantages**

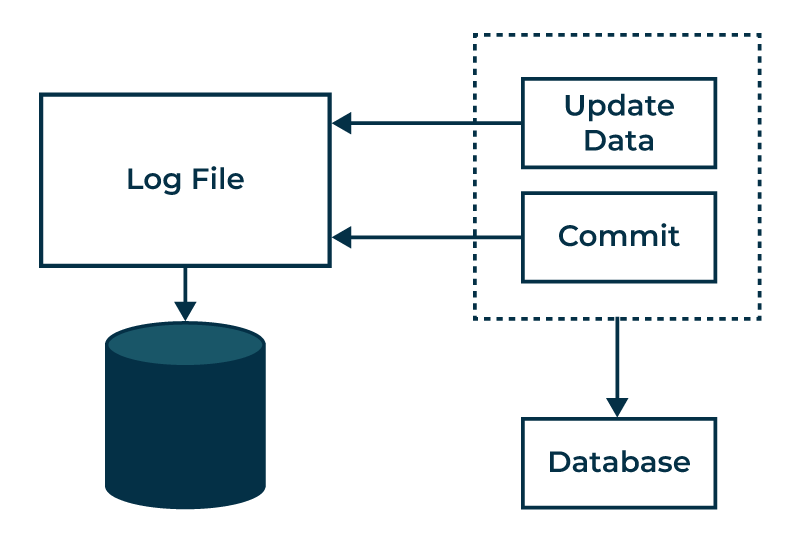
1. Data can be reconstructed using parity bits.
2. It makes the performance better.

#### **Disadvantages**

1. Its technology is complex and extra space is required.
2. If both discs get damaged, data will be lost forever.

**Log based Recovery in DBMS**

The atomicity property of DBMS states that either all the operations of transactions must be performed or none. The modifications done by an aborted transaction should not be visible to the database and the modifications done by the committed transaction should be visible. To achieve our goal of atomicity, the user must first output stable storage information describing the modifications, without modifying the database itself. This information can help us ensure that all modifications performed by committed transactions are reflected in the database. This information can also help us ensure that no modifications made by an aborted transaction persist in the database.



*Log based Recovery in DBMS*

**Log and log records**

The log is a sequence of log records, recording all the updated activities in the database. In stable storage, logs for each transaction are maintained. Any operation which is performed on the database is recorded on the log. Prior to performing any modification to the database, an updated log record is created to reflect that modification. 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 types of log records are:

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

**Undo and Redo Operations**

Because 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 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 the transaction is still active, it is said to use immediate modification technique.

**Recovery using Log records**

After a system crash has occurred, the system consults the log to determine which transactions need to be redone and which need to be undone.

1. Transaction Ti needs to be undone if the log contains the record <Ti start> but does not contain either the record <Ti commit> or the record <Ti abort>.
2. Transaction Ti needs to be redone if log contains record <Ti start> and either the record <Ti commit> or the record <Ti abort>.

**Use of Checkpoints –** When a system crash occurs, user must consult the log. In principle, that need to search the entire log to determine this information. There are two major difficulties with this approach:

1. The search process is time-consuming.
2. Most of the transactions that, according to our algorithm, need to be redone have already written their updates into the database. Although redoing them will cause no harm, it will cause recovery to take longer.

**Advantages of Log based Recovery**

* **Durability:**In the event of a breakdown, the log file offers a dependable and long-lasting method of recovering data. It guarantees that in the event of a system crash, no committed transaction is lost.
* **Faster Recovery:**Since log-based recovery recovers databases by replaying committed transactions from the log file, it is typically faster than alternative recovery methods.
* **Incremental Backup:** Backups can be made in increments using log-based recovery. Just the changes made since the last backup are kept in the log file, rather than creating a complete backup of the database each time.
* **Lowers the Risk of Data Corruption:** By making sure that all transactions are correctly committed or canceled before they are written to the [database](https://www.geeksforgeeks.org/what-is-database/), log-based recovery lowers the risk of data corruption.

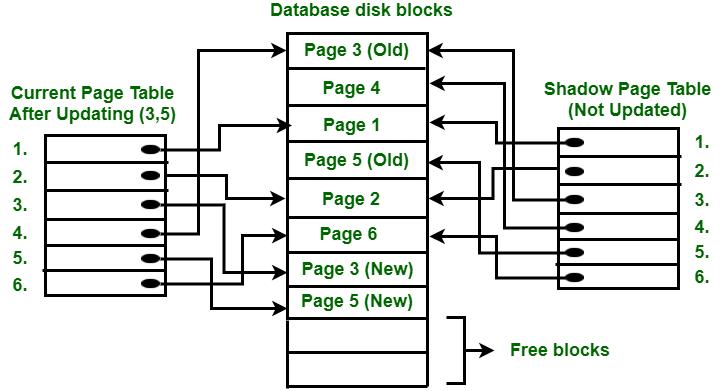
**Disadvantages of Log based Recovery**

* **Additional overhead:** Maintaining the log file incurs an additional overhead on the database system, which can reduce the performance of the system.
* **Complexity:**Log-based recovery is a complex process that requires careful management and administration. If not managed properly, it can lead to data inconsistencies or loss.
* **Storage space:**The log file can consume a significant amount of storage space, especially in a database with a large number of transactions.
* **Time-Consuming:** The process of replaying the transactions from the log file can be time-consuming, especially if there are a large number of transactions to recover.

**Conclusion**

In conclusion, data integrity and system reliability are maintained by log-based recovery in database management systems. It minimizes data loss and ensures reliability by assisting in the continuous restoration of databases following failures.

# Introduction of Shadow Paging

**Shadow Paging**is recovery technique that is used to recover database. In this recovery technique, database is considered as made up of fixed size of logical units of storage which are referred as **pages.** pages are mapped into physical blocks of storage, with help of the **page table**which allow one entry for each logical page of database. This method uses two page tables named **current page table** and **shadow page table**. The entries which are present in current page table are used to point to most recent database pages on disk. Another table i.e., Shadow page table is used when the transaction starts which is copying current page table. After this, shadow page table gets saved on disk and current page table is going to be used for transaction. Entries present in current page table may be changed during execution but in shadow page table it never get changed. After transaction, both tables become identical. This technique is also known as **Cut-of-Place updating.**   
To understand concept, consider above figure. In this 2 write operations are performed on page 3 and 5. Before start of write operation on page 3, current page table points to old page 3. When write operation starts following steps are performed :

1. Firstly, search start for available free block in disk blocks.
2. After finding free block, it copies page 3 to free block which is represented by Page 3 (New).
3. Now current page table points to Page 3 (New) on disk but shadow page table points to old page 3 because it is not modified.
4. The changes are now propagated to Page 3 (New) which is pointed by current page table.

**COMMIT Operation :** To commit transaction following steps should be done :

1. All the modifications which are done by transaction which are present in buffers are transferred to physical database.
2. Output current page table to disk.
3. Disk address of current page table output to fixed location which is in stable storage containing address of shadow page table. This operation overwrites address of old shadow page table. With this current page table becomes same as shadow page table and transaction is committed.

**Failure :** If system crashes during execution of transaction but before commit operation, With this, it is sufficient only to free modified database pages and discard current page table. Before execution of transaction, state of database get recovered by reinstalling shadow page table. If the crash of system occur after last write operation then it does not affect propagation of changes that are made by transaction. These changes are preserved and there is no need to perform redo operation. **Advantages :**

* This method require fewer disk accesses to perform operation.
* In this method, recovery from crash is inexpensive and quite fast.
* There is no need of operations like- Undo and Redo.
* Recovery using this method will be faster.
* **Improved fault tolerance:**Shadow paging provides improved fault tolerance since it isolates transactions from each other. This means that if one transaction fails, it does not affect the other transactions that are currently executing.
* **Increased concurrency:**Since modifications made during a transaction are written to the shadow copy instead of the actual database, multiple transactions can be executed concurrently without interfering with each other. This leads to increased concurrency and better performance.
* **Simplicity:**Shadow paging is a relatively simple technique to implement. It requires minimal modifications to the existing database system, making it easier to integrate into existing systems.
* **No need for log files:** In traditional database systems, log files are used to maintain a record of all changes made to the database. Shadow paging eliminates the need for log files since all changes are made to the shadow copy. This reduces the overhead associated with maintaining log files and makes the system more efficient.

**Disadvantages :**

* Due to location change on disk due to update database it is quite difficult to keep related pages in database closer on disk.
* During commit operation, changed blocks are going to be pointed by shadow page table which have to be returned to collection of free blocks otherwise they become accessible.
* The commit of single transaction requires multiple blocks which decreases execution speed.
* To allow this technique to multiple transactions concurrently it is difficult.
* **Data fragmentation:** The main disadvantage of this technique is the updated Data will suffer from fragmentation as the data is divided up into pages that may or not be in linear order for large sets of related hence, complex storage management strategies.
* **Garbage collection:**Garbage will accumulate in the pages on the disk as data is updated and pages lose any references. For example if i have a page that contains a data item X that is replaced with a new value then a new page will be created. Once the shadow page table is updated nothing will reference the old value of X. The operation to migrate between current and shadow directories must be implemented as an atomic mode.

**Difference between Conflict and View Serializability :**

| **S.no.** | **Conflict Serializability** | **View Serializability** |
| --- | --- | --- |
| 1. | Two schedules are said to be conflict equivalent if all the conflicting operations in both the schedule get executed in the same order. If a schedule is a conflict equivalent to its serial schedule then it is called Conflict Serializable Schedule. | Two schedules are said to be view equivalent if the order of initial read, final write and update operations is the same in both the schedules. If a schedule is view equivalent to its serial schedule then it is called View Serializable Schedule. |
| 2. | If a schedule is view serializable then it may or may not be conflict serializable. | If a schedule is conflict serializable then it is also view serializable schedule. |
| 3. | Conflict equivalence can be easily achieved by reordering the operations of two transactions therefore, Conflict Serializability is easy to achieve. | View equivalence is rather difficult to achieve as both transactions should perform similar actions in a similar manner. Thus, View Serializability is difficult to achieve. |
| 4. | For a transaction T1 writing a value A that no one else reads but later some other transactions say T2 write its own value of A, W(A) cannot be placed under positions where it is never read. | If a transaction T1 writes a value A that no other transaction reads (because later some other transactions say T2 writes its own value of A) W(A) can be placed in positions of the schedule where it is never read. |