UNIT 4: TRANSACTION MANAGEMENT

TRASACTION CONCEPT:

- A transaction is an event which occurs on the database. Generally a transaction reads a value from the database or writes a value to the database.
- In other words a transaction is a program unit whose execution may or may not change the Contents of Database.
- A transaction can both read and write on the database
- A read operation does not change the image of the database in any way. But a write operation, whether performed with the intention of inserting, updating or deleting data from the database, changes the image of the database.
- Hence transactions bring the database from an image which existed before the transaction occurred (called the Before Image or BFIM) to an image which exists after the transaction occurred (called the After Image or AFIM).

Ex: we have two accounts A and B, each containing Rs 1000/-. We now start a transaction to deposit Rs 100/- from account A to Account B.

```
Read A;
A = A – 100;
Write A;
Read B;
B = B + 100;
Write B;
```

PROPERTIES OF TRANSACTION

There are 4 properties named ACID

Atomicity A
Consistency C
Isolation I
Durability D

Atomicity: (all or nothing): A transaction is said to be Atomic if a transaction always executes all its actions in one step or not executes any actions at all. It means either all or none of the transactions operations are performed.

Consistency: (No violation of Integrity constraints) A transaction must preserve the Consistency of a database after the execution. This property holds for each transaction.

Isolation: (concurrent changes invisibles) If several transactions are executed concurrently the results must be same as if they were executed serially in some order. The data used during the execution of a transaction cannot be used by a 2 nd transaction until the 1 st one complete.

Durability: (committed update persist) It means once a transaction Commits, the system must guarantee that the results of its operations will never be lost, in spite of some other failures.

Transactions Support in SQL(TCL)

COMMIT command

COMMIT command is used to permanently save any transaction into the database.

When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.

To avoid that, we use the COMMIT command to mark the changes as permanent.

Following is commit command's syntax,

```
COMMIT;
```

ROLLBACK command

This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.

If we have used the UPDATE command to make some changes into the database, and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command.

Following is rollback command's syntax,

```
ROLLBACK TO savepoint_name;
```

SAVEPOINT command

SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

Following is savepoint command's syntax,

```
SAVEPOINT savepoint_name;
```

TRANSACTION STATE

There are the following six states in which a transaction may exist:

Active: The initial state when the transaction has just started execution.

Partially Committed: At any given point of time if the transaction is executing properly, then it is going towards it COMMIT POINT. The values generated during the execution are all stored in volatile storage.

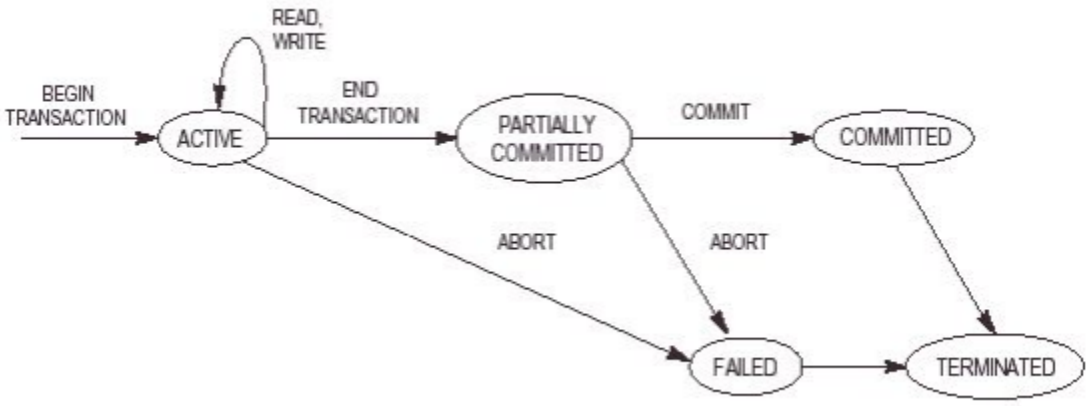
Failed: If the transaction fails for some reason. The temporary values are no longer required, and the transaction is set to ROLLBACK. It means that any change made to the database by this transaction up to the point of the failure must be undone. If the failed transaction has withdrawn Rs. 100/- from account A, then the ROLLBACK operation should add Rs 100/- to account A.

Aborted: When the ROLLBACK operation is over, the database reaches the BFIM. The transaction is now said to have been aborted.

Committed: If no failure occurs then the transaction reaches the COMMIT POINT. All the temporary values are written to the stable storage and the transaction is said to have been committed.

Terminated: Either committed or aborted, the transaction finally reaches this state.

The whole process can be described using the following diagram:



IMPLEMENTATION OF ACID PROPERTIES:

Let T1 be a transaction that transfers Rs. 50 from A/C A to A/C B.

A/C A= 1000 A/C B=2000

T1 Schedule
Read (A, a)
a=a-50
Write (A, a)
Read (B, b)
B=b+50

Write (B, b)

IMPLEMENTATION OF AUTOMOCITY:

To maintain atomicity of transaction, database system keeps track of old values of any write operation and if the transaction does not complete its execution, the old values are restored to make it appear as the transaction never executed.

Now, suppose there is a power failure just after instruction 3 (Write A) has been complete. What happens now? After the system recovers the AFIM will show Rs 900/- in A, but the same Rs 1000/- in B. It would be said that Rs 100/- evaporated in thin air for the power failure. Clearly such a situation is not acceptable.

The solution is to keep every value calculated by the instruction of the transaction not in any stable storage (hard disc) but in a volatile storage (RAM), until the transaction completes its last instruction. When we see that there has not been any error we do something known as a COMMIT operation. Its job is to write every temporarily calculated value from the volatile storage on to the stable storage. In this way, even if power fails at instruction 3, the post recovery image of the database will show accounts A and B both containing Rs 1000/-, as if the failed transaction had never occurred.

IMPLEMENTATION OF CONSISTENCY:

To maintain consistency of a transaction, the A/C A and A/C B should not be changed. Careful mechanisms are created in order to prevent mismanagement of these sharable resources, so that there should not be any change in the way a transaction performs. A transaction which deposits Rs 100/- to account A must deposit the same amount whether it is acting alone or in conjunction with another transaction that may be trying to deposit or withdraw some amount at the same time.

IMPLEMENTATION OF ISOLATION:

To maintain this property, data used during the execution of a transaction cannot be used by a second transaction until the first one is completed. In case multiple transactions are executing concurrently and trying to access a sharable resource at the same time, the system should create an ordering in their execution so that they should not create any anomaly in the value stored at the sharable resource. There are several ways to achieve this and the most popular one is using some kind of locking mechanism. Another solution is Execute each transaction serially one after the other.

IMPLEMENTATION OF DURABILITY:

To maintain this property, once a transaction completes successfully, all updates carried out on the database persist, even if there is a system failure after the transaction completes the execution. There are several ways Atomicity and Durability can be implemented. One of them is called **Shadow Copy**. In this scheme a database pointer is used to point to the BFIM of the database. During the transaction, all the temporary changes are recorded into a Shadow Copy, which is an exact copy of the original database plus the changes made by the transaction, which is the AFIM. Now, if the transaction is required to COMMIT, then the database pointer is updated to point to the AFIM copy, and the BFIM copy is discarded. On the other hand, if the transaction is not committed, then the database pointer is not updated. It keeps pointing to the BFIM, and the AFIM is discarded. This is a simple scheme, but takes a lot of memory space and time to implement.

Concurrent Execution

A **schedule** is a list of actions ;ike read, write, commit and abort from a set of transactions and the irder in which two actions of transaction appears or collection of many transactions which is implemented as a unit. Depending upon how these transactions are arranged in within a schedule, a schedule can be of two types:

1. **Serial:** The transactions are executed one after another, in a non-preemptive manner. In Serial schedule, there is no question of sharing a single data item among many transactions, because not more than a single transaction is executing at any point of time. However, a serial schedule is inefficient in the sense that the transactions suffer for having a longer waiting time and response time, as well as low amount of resource utilization.

EX: Let us consider there are two transactions T1 and T2, whose instruction sets are given as following.

```
T1
Read A;
A = A – 100;
Write A;
Read B;
B = B + 100;
Write B;
T2
Read A;
Temp = A * 0.1;
Read C;
C = C + Temp;
Write C;
```

T2 is a new transaction which deposits to account C 10% of the amount in account A.
If we prepare a serial schedule, then either T1 will completely finish before T2 can begin, or T2 will completely finish before T1 can begin. However, if we want to create a concurrent schedule, then some Context Switching need to be made, so that some portion of T1 will be executed, then some portion of T2 will be executed and so on. For example say we have prepared the following

concurrent schedule or Inter leaved Schedule. The transactions are executed in a preemptive, time shared method. In concurrent schedule, CPU time is shared among two or more transactions in order to run them concurrently. However, this creates the possibility that more than one transaction may need to access a single data item for read/write purpose and the database could contain inconsistent value if such accesses are not handled properly. Hence these transactions are used for improving troughput and response time.

Ex:

T1	T2
R(A)	
W(A)	
	R(B)
	W(B)
R(C)	
W(C)	

It is clear that a proper Context Switching is very important in order to maintain the Consistency and Isolation properties of the transactions.

SERIALIZABILITY

When multiple transactions are running concurrently then there is a possibility that the database may be left in an inconsistent state. The **Serializability** which is ensuring Isolation property of database. When several concurrent transactions are trying to access the same data item, the instructions within these concurrent transactions must be ordered in some way so as there are no problem in accessing and releasing the shared data item. A serializable schedule always leaves the database in consistent state.

Ex:

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)
	COMMIT
COMMIT	

There are two types of serializability.

1. Conflict Serializability
2. View Serializability

1. Conflict Serializability

Two instructions of two different transactions may want to access the same data item in order to perform a read/write operation. Conflict Serializability deals with detecting whether the instructions are conflicting in any way, and specifying the order in which these two instructions

will be executed in case there is any conflict. A schedule is called conflict serializable if we can convert it into a serial schedule after swapping its non-conflicting operations.

An Anomalies due to Interleaved Execution (or) CONFLICT OF OPERATIONS

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

Following are the three problems in concurrency control.

1.Reading Uncommitted data or Dirty Read or WR

2. Unrepetable Reads or RW Conflict

3. Overwriting Uncommotted Data or Lost Update or WW conflict

1. Dirty Read

The dirty read occurs in the case when one transaction updates an item of the database, and then the transaction fails for some reason. The updated database item is accessed by another transaction before it is changed back to the original value.

Ex:	T1	T2
	R(A)	
	W(A)	
		R(A)
		W(A)
		R(B)
		W(B)
		COMMIT
	R(B)	
	W(B)	
	COMMIT	

2. Unrepetable Reads or RW Conflict or Inconsistent retrival

When a transaction calculates some summary function over a set of data while the other transactions are updating the data, then the Inconsistent Retrievals Problem occurs.

A transaction T1 reads a record and then does some other processing during which the transaction T2 updates the record. Now when the transaction T1 reads the record, then the new value will be inconsistent with the previous value.

Example: Suppose two transactions operate on three accounts.

Account-1	Account-2	Account-3
Balance = 200	Balance = 250	Balance = 150

Transaction-X	Time	Transaction-Y
——	t1	——
Read Balance of Acc-1 sum <-- 200 Read Balance of Acc-2	t2	——
Sum <-- Sum + 250 = 450	t3	——
——	t4	Read Balance of Acc-3
——	t5	Update Balance of Acc-3 150 --> 150 - 50 --> 100
——	t6	Read Balance of Acc-1
——	t7	Update Balance of Acc-1 200 --> 200 + 50 --> 250
Read Balance of Acc-3	t8	COMMIT
Sum <-- Sum + 250 = 550	t9	——

- Transaction-X is doing the sum of all balance while transaction-Y is transferring an amount 50 from Account-1 to Account-3.
- Here, transaction-X produces the result of 550 which is incorrect. If we write this produced result in the database, the database will become an inconsistent state because the actual sum is 600.
- Here, transaction-X has seen an inconsistent state of the database.

3. Lost update problem

When two transactions that access the same database items contain their operations in a way that makes the value of some database item incorrect, then the lost update problem occurs.

If two transactions T1 and T2 read a record and then update it, then the effect of updating of the first record will be overwritten by the second update.

Example:

Transaction-X	Time	Transaction-Y
——	t1	——
Read A	t2	——
——	t3	Read A
Update A	t4	——
——	t5	Update A
——	t6	——

- At time t2, transaction-X reads A's value.

- At time t3, Transaction-Y reads A's value.
- At time t4, Transactions-X writes A's value on the basis of the value seen at time t2.
- At time t5, Transactions-Y writes A's value on the basis of the value seen at time t3.
- So at time T5, the update of Transaction-X is lost because Transaction y overwrites it without looking at its current value.
- Such type of problem is known as Lost Update Problem as update made by one transaction is lost here.

If a schedule S can be transformed in to a schedule S by a series of swaps of non-conflicting instructions, then S and S` are conflict equivalent. In other words a schedule S is conflict serializable if it is conflict equivalent to a serial schedule.

VIEW SERIALIZABILITY:

View-serializability of a schedule is defined by equivalence to a serial schedule (no overlapping transactions) with the same transactions, such that respective transactions in the two schedules read and write the same data values ("view" the same data values). View Serializability is a process to find out that a given schedule is view serializable or not. To check whether a given schedule is view serializable, we need to check whether the given schedule is View Equivalent to its serial schedule.

Every conflict serializable schedule is view serializable. Any view serializable schedule that is not conflict serializable contains a BLIND WRITE.

VIEW EQUIVALENT
Two schedules T1 and T2 are said to be view equivalent, if they satisfy all the following conditions:

1. Initial Read: Initial read of each data item in transactions must match in both schedules.
2. Final Write: Final write operations on each data item must match in both the schedules.
3. Update Read: If in schedule S1, the transaction T1 is reading a data item updated by T2 then in schedule S2, T1 should read the value after the write operation of T2 on same data item.

View Serializable
If a schedule is view equivalent to its serial schedule then the given schedule is said to be View Serializable.

EX:

T1	T2
R(X)	
W(X)	

R(X)
W(X)
R(Y)
W(Y)
R(Y)
W(Y)

Serial Schedule of the above given schedule:
As we know that in Serial schedule a transaction only starts when the current running transaction is finished. So the serial schedule of the above given schedule would look like this:

T1 T2
----- ------
R(X)
W(X)
R(Y)
W(Y)
R(X)
W(X)
R(Y)
W(Y)

If we can prove that the given schedule is View Equivalent to its serial schedule then the given schedule is called view Serializable.

SCHEDULES INVOLVING ABORTED TRANSACTIONS

Ex:	**T1**	**T2**
	R(A)	
	W(A)	
		R(A)
		W(A)
		R(B)
		W(B)
		Commit
	Abort	

RECOVERABILITY

Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But some other transaction may also have used value produced by the failed transaction. So we also have to rollback those transactions.

Recoverable schedule is-if a transaction Tj reads a data item previously written by a transaction Ti , then the commit operation of Ti must appear before the commit operation of Tj.

EX: The following schedule is not recoverable if T9 commits immediately after the read(A)

operation.

T_8	T_9
read (A)	
write (A)	
	read (A)
	commit
read (B)	

If T8 should abort, T9 would have read (and possibly shown to the user) an inconsistent database state. Hence, database must ensure that schedules are recoverable.

CASCADING ROLLBACKS

Cascading rollback – a single transaction failure leads to a series of transaction rollbacks. Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable)

T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)
abort		

If T10 fails, T11 and T12 must also be rolled back.

TESTING OF SERIALIZABILITY

A precedence graph, also named conflict graph and serializability graph, is used in the context of testing of serializability and concurrency control in databases.

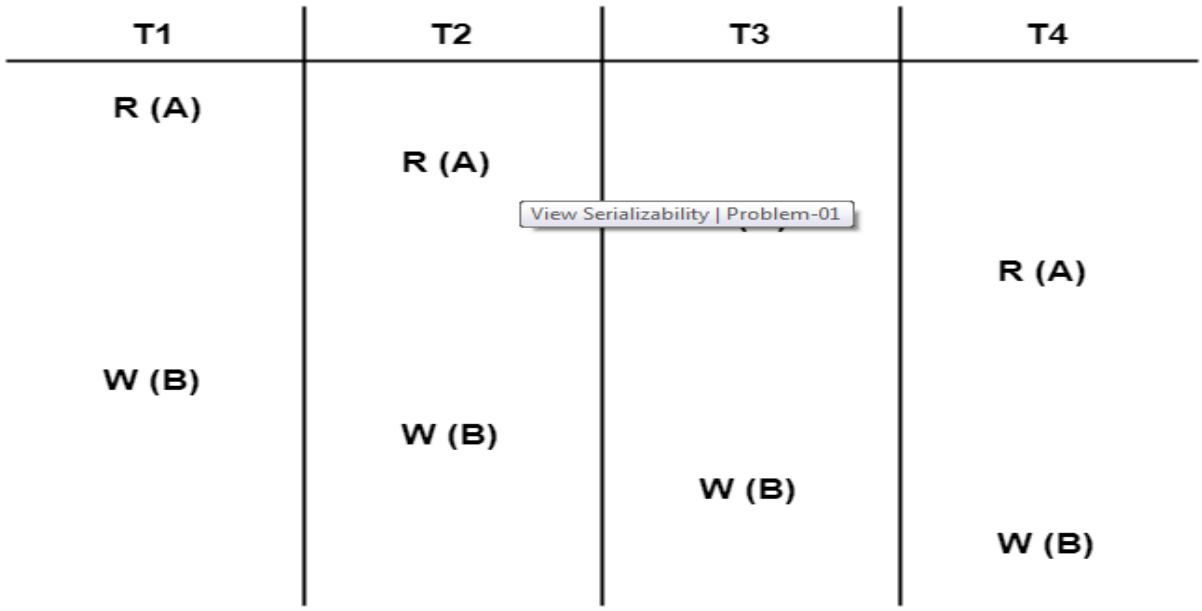
The precedence graph for a schedule S contains:

- A node for each committed transaction in S
- An arc from Ti to Tj if an action of Ti precedes and conflicts with one of Tj's actions.

The Schedule S is serializable if there is no cycle in the precedence graph.

Problem-01:

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



Solution-

- We know, if a schedule is conflict serializable, then it is surely view serializable.
- So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

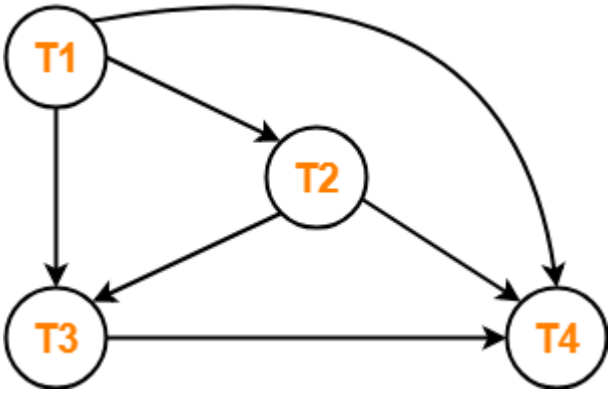
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- $W_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
- $W_1(B)$, $W_3(B)$ $(T_1 \rightarrow T_3)$
- $W_1(B)$, $W_4(B)$ $(T_1 \rightarrow T_4)$
- $W_2(B)$, $W_3(B)$ $(T_2 \rightarrow T_3)$
- $W_2(B)$, $W_4(B)$ $(T_2 \rightarrow T_4)$
- $W_3(B)$, $W_4(B)$ $(T_3 \rightarrow T_4)$

Step-02:

Draw the precedence graph-



- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.
- Thus, we conclude that the given schedule is also view serializable.

Problem-02:

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

T1	T2	T3
R (A)		
	R (A)	
		W (A)
W (A)		

Solution-

- We know, if a schedule is conflict serializable, then it is surely view serializable.
- So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

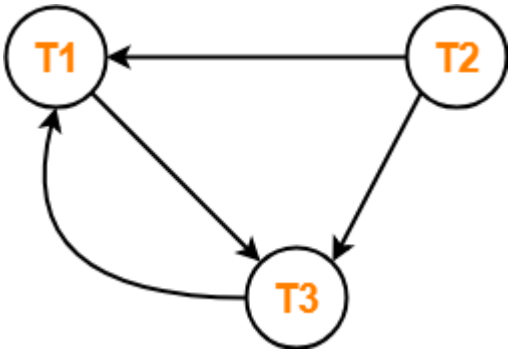
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- $R_1(A)$, $W_3(A)$ $(T_1 \rightarrow T_3)$
- $R_2(A)$, $W_3(A)$ $(T_2 \rightarrow T_3)$
- $R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
- $W_3(A)$, $W_1(A)$ $(T_3 \rightarrow T_1)$

Step-02:

Draw the precedence graph-



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

CONCURRENCY CONTROL

- In the concurrency control, the multiple transactions can be executed simultaneously.
- It may affect the transaction result. It is highly important to maintain the order of execution of those transactions.
- Concurrency control protocols ensure atomicity, isolation, and serializability of concurrent transactions.

The concurrency control protocol can be divided into three categories:

- Lock based protocol
- Time-stamp protocol
- Validation based protocol

Problems of concurrency control same as 3 conflict operations like dirty read etc….

Lock-Based Protocol

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. A LOCK is a small bookkeeping object associated with a database object.

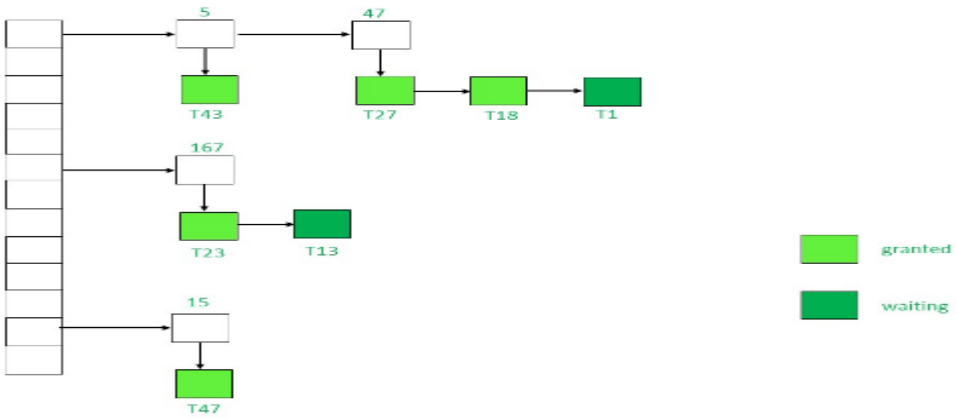
Locking protocols are used in database management systems as a means of concurrency control. A LOCKING PROTOCOL is a set of rules to be followed by each transaction to ensure that actions of several transactions might be interleaved.

Multiple transactions may request a lock on a data item simultaneously. Hence, we require a mechanism to manage the locking requests made by transactions. Such a mechanism is called as **LOCK Management**. Lock management is to track of the locks issued to trasactions by lock manager. Lock Manager maintains a lock table,

Data structure used in Lock Manager –
The data structure required for implementation of locking is called as **Lock table**.

1. It is a hash table where name of data items are used as hashing index.
2. Each locked data item has a linked list associated with it.
3. Every node in the linked list represents the transaction which requested for lock, mode of lock requested (mutual/exclusive) and current status of the request (granted/waiting).
4. Every new lock request for the data item will be added in the end of linked list as a new node.
5. Collisions in hash table are handled by technique of separate chaining.

In the above figure, the locked data items present in lock table are 5, 47, 167 and 15.



There are two types of locks:

1. Shared lock (S):

- 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 (X):

- 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.

Lock Compatibility Matrix –

	S	X
S	✔	✗
X	✗	✗

- 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(X) 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. Then the lock is granted.

Upgrade / Downgrade locks : A transaction that holds a lock on an item **A** is allowed under certain condition to change the lock state from one state to another.

Upgrade: A S(A) can be upgraded to X(A) if T_i is the only transaction holding the S-lock on element A.

Downgrade: We may downgrade X(A) to S(A) when we feel that we no longer want to write on data-item A. As we were holding X-lock on A, we need not check any conditions.

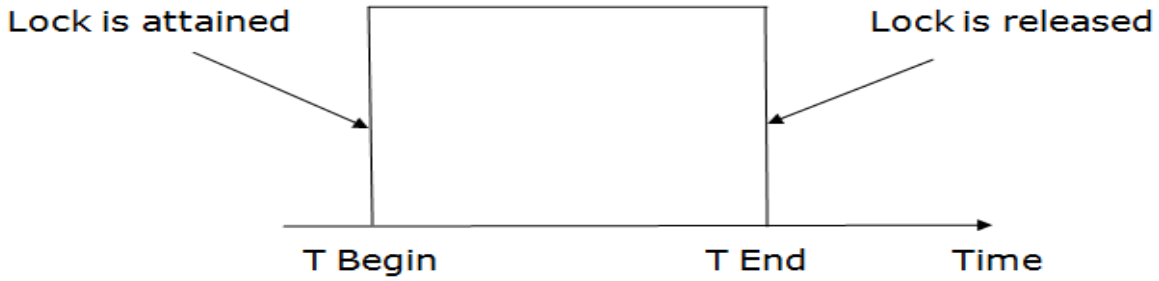
There are four types of lock protocols available:

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.

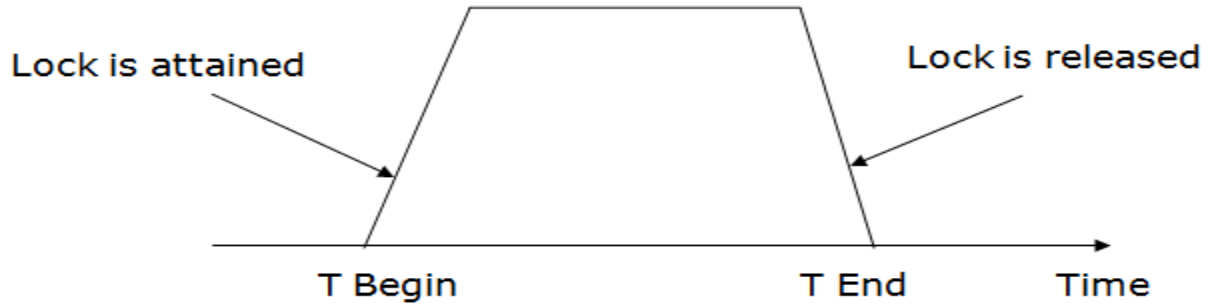
2. Pre-claiming Lock Protocol

- 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:

	T1	T2
0	LOCK-S(A)	
1		LOCK-S(A)
2	LOCK-X(B)	
3	———	———
4	UNLOCK(A)	
5		LOCK-X(C)
6	UNLOCK(B)	
7		UNLOCK(A)
8		UNLOCK(C)
9	———	———

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:

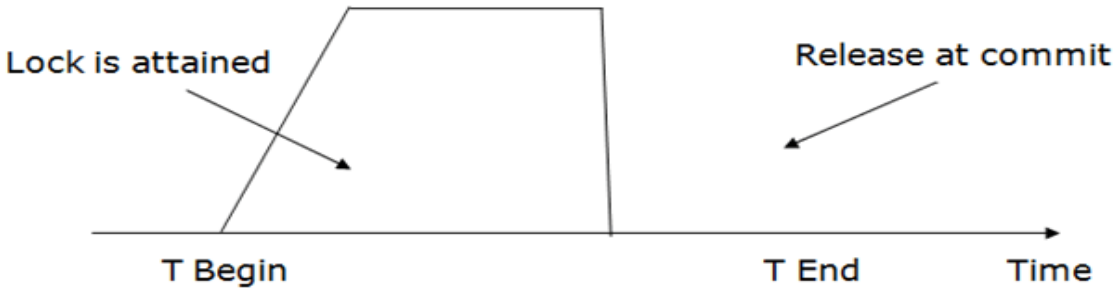
- **Growing phase:** from step 2-6
- **Shrinking phase:** from step 8-9
- **Lock point:** at 6

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.

Ex:

T1	**T2**
X(A)	
R(A)	
W(A)	



It does not have cascading abort as 2PL does.

There are 2 methods of locking

1. **Optimistic locking** is when you check if the record was updated by someone else before you commit the transaction.

More specifically, Optimistic concurrency control transactions involve these phases:

- **Begin**: Record a timestamp marking the transaction's beginning.
- **Modify**: Read database values, and tentatively write changes.
- **Validate**: Check whether other transactions have modified data that this transaction has used (read or written). This includes transactions that completed after this transaction's start time, and optionally, transactions that are still active at validation time.
- **Commit/Rollback**: If there is no conflict, make all changes take effect. If there is a conflict, resolve it, typically by aborting the transaction, although other resolution schemes are possible.

2. **Pessimistic locking** is when you take an exclusive lock so that no one else can start modifying the record.

DEADLOCK

Transaction T1 sets an exclusive lock on object A, T2 sets an exclusive lock on B, T1 requests an exclusive lock on B and is queued and T2 requests an exclusive lock on A and is queued. Now T1 is waiting for T2 to release its lock and t2 is waiting for t1 to release its lock. Such a cycle of transactions waiting for locks to be released is called a deadlock.

Dead lock is detected by PRECERANCE GRAPH. Deadlock can be prevented by 2 techniques
1. Wait- Die

2. Wound-Wait

Performance of Locking:

Locking Suffers with the problem of **Thrashing. The** Solutions for this problem is

1. By locking the smallest sized objects possible.

2. By reducing the time that transaction hold locks

3. By reducing **hot spots.** A hot spot is a database object that is frequently accessed and modified.

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- free.

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 (T_i): 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.

Thomas write Rule

Thomas Write Rule provides the guarantee of serializability order for the protocol. It improves the Basic Timestamp Ordering Algorithm.

The basic Thomas write rules are as follows:

- If TS(T) < R_TS(X) then transaction T is aborted and rolled back, and operation is rejected.
- If TS(T) < W_TS(X) then don't execute the W_item(X) operation of the transaction and continue processing.
- If neither condition 1 nor condition 2 occurs, then allowed to execute the WRITE operation by transaction Ti and set W_TS(X) to TS(T).

If we use the Thomas write rule then some serializable schedule can be permitted that does not conflict serializable as illustrate by the schedule in a given figure:

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

Fig:A Serializable Schedule that is not Conflict Serializable

In the above figure, T1's read and precedes T1's write of the same data item. This schedule does not conflict serializable.

Thomas write rule checks that T2's write is never seen by any transaction. If we delete the write operation in transaction T2, then conflict serializable schedule can be obtained which is shown in below figure.

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

Figure: A Conflict Serializable Schedule

Multiple Granularity

Let's start by understanding the meaning of granularity.

Granularity: It is the size of data item allowed to lock.

Multiple Granularity:

- It can be defined as hierarchically breaking up the database into blocks which can be locked.
- The Multiple Granularity protocol enhances concurrency and reduces lock overhead.
- It maintains the track of what to lock and how to lock.
- It makes easy to decide either to lock a data item or to unlock a data item. This type of hierarchy can be graphically represented as a tree.

For example: Consider a tree which has four levels of nodes.

- The first level or higher level shows the entire database.
- The second level represents a node of type area. The higher level database consists of exactly these areas.
- The area consists of children nodes which are known as files. No file can be present in more than one area.

- Finally, each file contains child nodes known as records. The file has exactly those records that are its child nodes. No records represent in more than one file.
- Hence, the levels of the tree starting from the top level are as follows:
    1. Database
    2. Area
    3. File
    4. Record

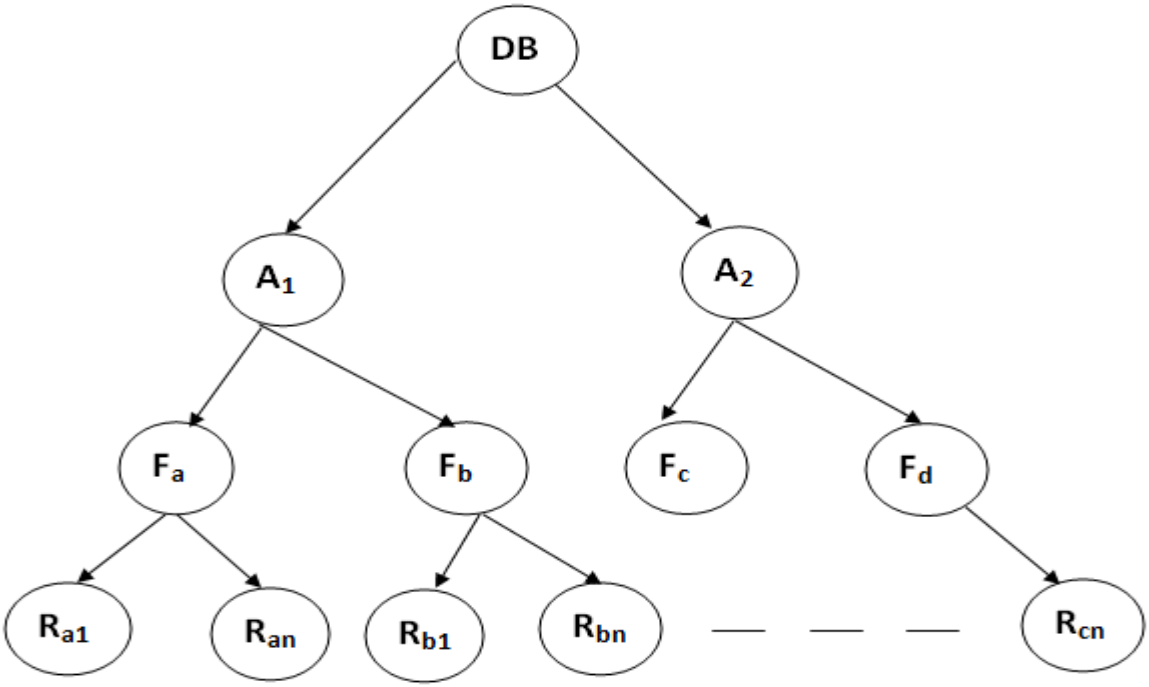


Figure: Multi Granularity tree Hierarchy

In this example, the highest level shows the entire database. The levels below are file, record, and fields.

There are three additional lock modes with multiple granularity:

Intention Mode Lock

Intention-shared (IS): It contains explicit locking at a lower level of the tree but only with shared locks.

Intention-Exclusive (IX): It contains explicit locking at a lower level with exclusive or shared locks.

Shared & Intention-Exclusive (SIX): In this lock, the node is locked in shared mode, and some node is locked in exclusive mode by the same transaction.

Compatibility Matrix with Intention Lock Modes: The below table describes the compatibility matrix for these lock modes:

	IS	IX	S	SIX	X
IS	✓	✓	✓	✓	X
IX	✓	✓	X	X	X
S	✓	X	✓	X	X
SIX	✓	X	X	X	X
X	X	X	X	X	X

It uses the intention lock modes to ensure serializability. It requires that if a transaction attempts to lock a node, then that node must follow these protocols:

- Transaction T1 should follow the lock-compatibility matrix.
- Transaction T1 firstly locks the root of the tree. It can lock it in any mode.
- If T1 currently has the parent of the node locked in either IX or IS mode, then the transaction T1 will lock a node in S or IS mode only.
- If T1 currently has the parent of the node locked in either IX or SIX modes, then the transaction T1 will lock a node in X, SIX, or IX mode only.
- If T1 has not previously unlocked any node only, then the Transaction T1 can lock a node.
- If T1 currently has none of the children of the node-locked only, then Transaction T1 will unlock a node.

Observe that in multiple-granularity, the locks are acquired in top-down order, and locks must be released in bottom-up order.

- If transaction T1 reads record R_{a9} in file F_a, then transaction T1 needs to lock the database, area A_1 and file F_a in IX mode. Finally, it needs to lock R_{a2} in S mode.
- If transaction T2 modifies record R_{a9} in file F_a, then it can do so after locking the database, area A_1 and file F_a in IX mode. Finally, it needs to lock the R_{a9} in X mode.
- If transaction T3 reads all the records in file F_a, then transaction T3 needs to lock the database, and area A in IS mode. At last, it needs to lock F_a in S mode.
- If transaction T4 reads the entire database, then T4 needs to lock the database in S mode.

RECOVERY AND AUTOMICITY

When a system crashes, it many have several transactions being executed and various files opened for them to modifying data items. As we know that transactions are made of various operations, which are atomic in nature.

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

There are two types of techniques, which can help DBMS in recovering as well as maintaining the atomicity of transaction:

- Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
- Maintaining shadow paging, where are the changes are done on a volatile memory and later the actual database is updated.

1.Log-Based Recovery

- The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.
- If any operation is performed on the database, then it will be recorded in the log.
- But the process of storing the logs should be done before the actual transaction is applied in the database.

Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.

- When the transaction is initiated, then it writes 'start' log.

1. <Tn, Start>

- When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.

1. <Tn, City, 'Noida', 'Bangalore' >

- When the transaction is finished, then it writes another log to indicate the end of the transaction.

1. <Tn, Commit>

There are two approaches to modify the database:

1. Deferred database modification:

- The deferred modification technique occurs if the transaction does not modify the database until it has committed.
- In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.

2. Immediate database modification:

- The Immediate modification technique occurs if database modification occurs while the transaction is still active.
- In this technique, the database is modified immediately after every operation. It follows an actual database modification.

Recovery using Log records

When the system is crashed, then the system consults the log to find which transactions need to be undone and which need to be redone.

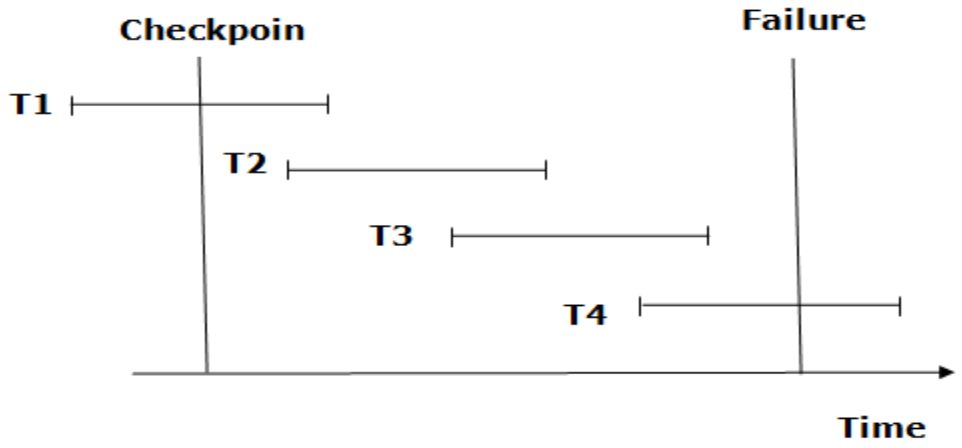
1. If the log contains the record <Ti, Start> and <Ti, Commit> or <Ti, Commit>, then the Transaction Ti needs to be redone.
2. If log contains record<T_n, Start> but does not contain the record either <Ti, commit> or <Ti, abort>, then the Transaction Ti needs to be undone.

2. Checkpoint

- 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 using Checkpoint

In the following manner, a recovery system recovers the database from this failure:



- The recovery system reads log files from the end to start. It reads log files from T4 to T1.
- Recovery system maintains two lists, a redo-list, and an undo-list.
- The transaction is put into redo state if the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>. In the redo-list and their previous list, all the transactions are removed and then redone before saving their logs.
- **For example:** In the log file, transaction T2 and T3 will have <Tn, Start> and <Tn, Commit>. The T1 transaction will have only <Tn, commit> in the log file. That's why the transaction is committed after the checkpoint is crossed. Hence it puts T1, T2 and T3 transaction into redo list.
- The transaction is put into undo state if the recovery system sees a log with <Tn, Start> but no commit or abort log found. In the undo-list, all the transactions are undone, and their logs are removed.
- **For example:** Transaction T4 will have <Tn, Start>. So T4 will be put into undo list since this transaction is not yet complete and failed amid.

3. Recovery with Concurrent Transaction

- Whenever more than one transaction is being executed, then the interleaved of logs occur. During recovery, it would become difficult for the recovery system to backtrack all logs and then start recovering.
- To ease this situation, 'checkpoint' concept is used by most DBMS.

RECOVERING FROM FAILURE OF NON-VOLATILE STORAGE

To recover from disk failure

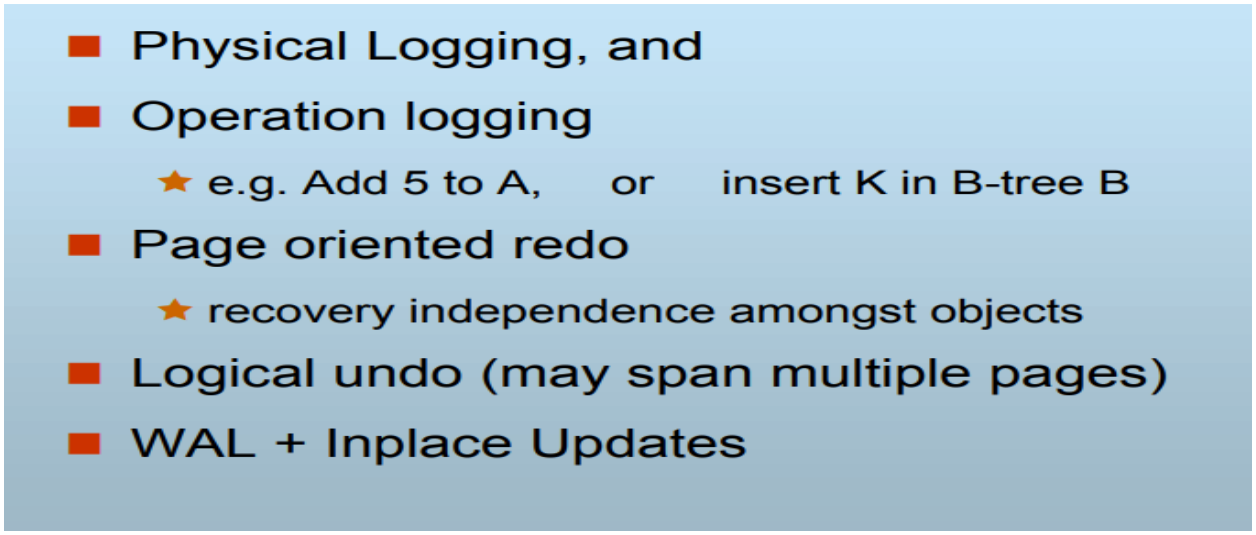
1.restore database from most recent dump.

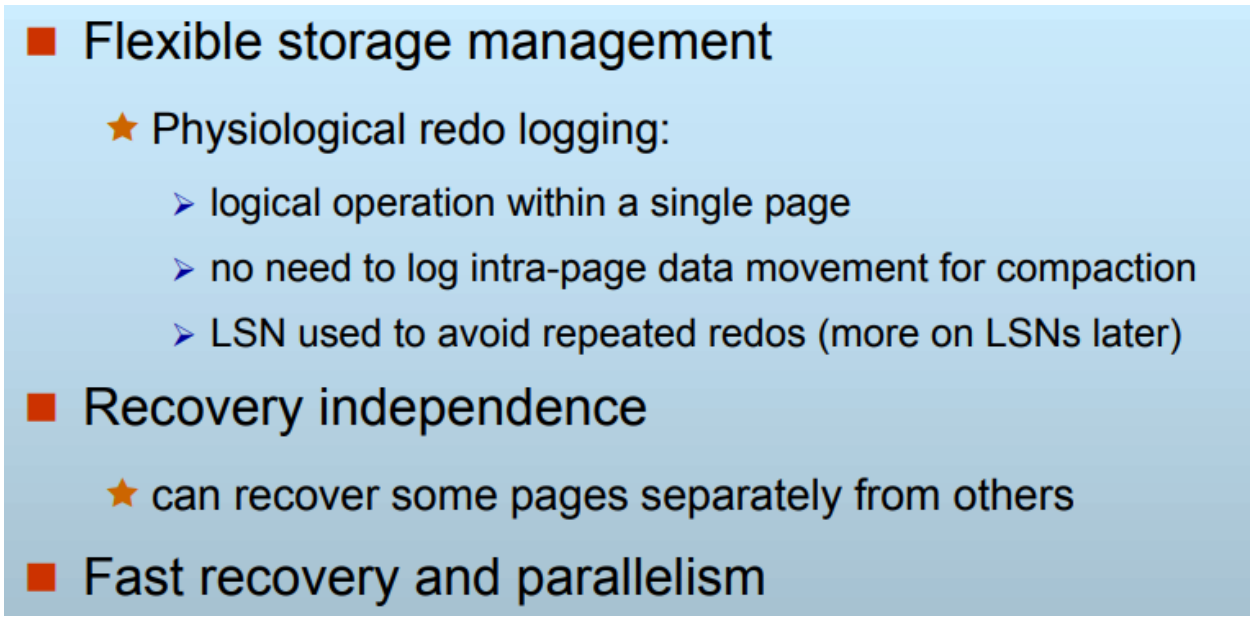
2.Consult the log and redo all transactions that committed after the dump

ARIES RECOVERY ALGORITHM

ARIES (Algorithm for Recovery and Isolation Exploiting Semantics) recovery is based on the Write Ahead Logging (WAL) protocol.

FEATURES:





- Transaction Rollback
  - Total vs partial (up to a savepoint)
  - Nested rollback - partial rollback followed by another (partial/total) rollback
- Fine-grain concurrency control
  - supports tuple level locks on records, and key value locks on indices

NOTATIONS:

- LSN: Log Sequence Number
  - = logical address of record in the log
- Page LSN: stored in page
  - LSN of most recent update to page
- PrevLSN: stored in log record
  - identifies previous log record for that transaction

DATA STRUCTURES USED:

1. LOG RECORD STRUCTURE--Uses log sequence number (LSN) to identify log records , Stores LSNs in pages to identify what updates have already been applied to a database page

2. TRANSACTION TABLE---Physiological redo

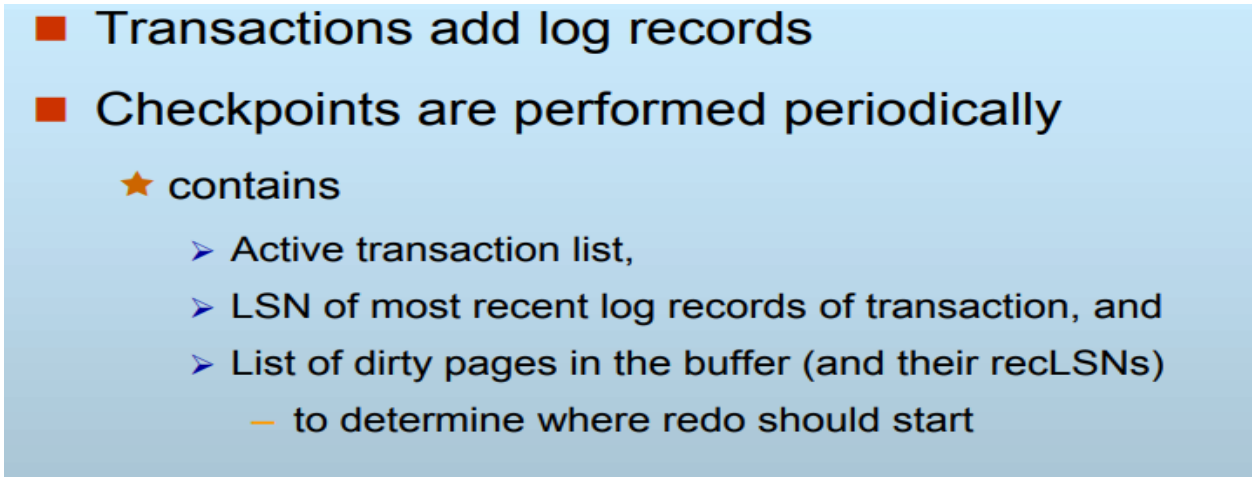
3.Dirty page table to avoid unnecessary redos during recovery

4.Fuzzy checkpointing that only records information about dirty pages, and does not require dirty pages to be written out at checkpoint time

BUFFER MANAGER

- Fix, unfix and fix_new (allocate and fix new pg)
- Aries uses **steal policy** - uncommitted writes may be output to disk (contrast with **no-steal** policy)
- Aries uses **no-force** policy (updated pages need not be forced to disk before commit)
- dirty page: buffer version has updated not yet reflected on disk
  - dirty pages written out in a continuous manner to disk

NORMAL PROCESSING



Every update operation writes a log record which is one of

1. An undo-only log record: Only the before image is logged. Thus, an undo operation can be done to retrieve the old data.

2. A redo-only log record: Only the after image is logged. Thus, a redo operation can be attempted.

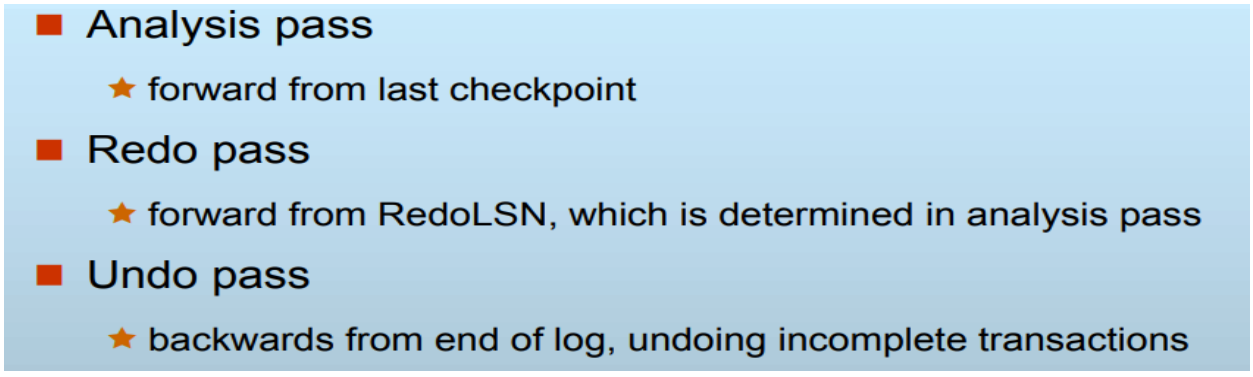
3. An undo-redo log record. Both before image and after images are logged.

Every log record is assigned a unique and monotonically increasing log sequence number (LSN). Every data page has a page LSN field that is set to the LSN of the log record corresponding to the last update on the page. WAL requires that the log record corresponding to an update make it to stable storage before the data page corresponding to that update is written to disk. For performance reasons, each log write is not immediately forced to disk. A log tail is maintained in main memory to buffer log writes. The log tail is flushed to disk when it gets full. A transaction cannot be declared committed until the commit log record makes it to disk.

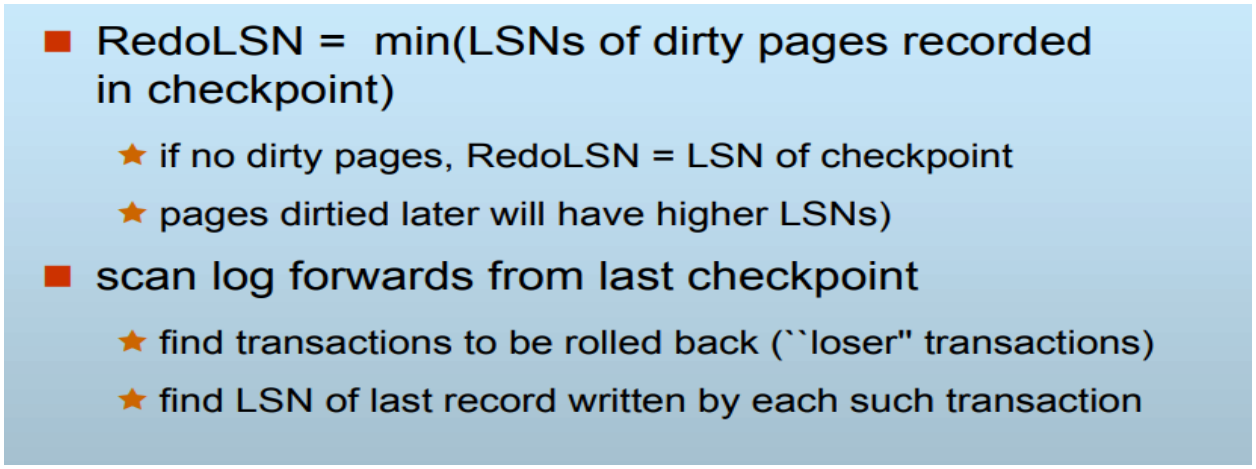
Once in a while the recovery subsystem writes a checkpoint record to the log. The checkpoint record contains the transaction table (which gives the list of active transactions) and the dirty page table (the list of data pages in the buffer pool that have not yet made it to disk). A master log record is maintained separately, in stable storage, to store the LSN of the latest checkpoint record that made it to disk. On restart, the recovery subsystem reads the master log record to find the checkpoint's

LSN, reads the checkpoint record, and starts recovery from there on.

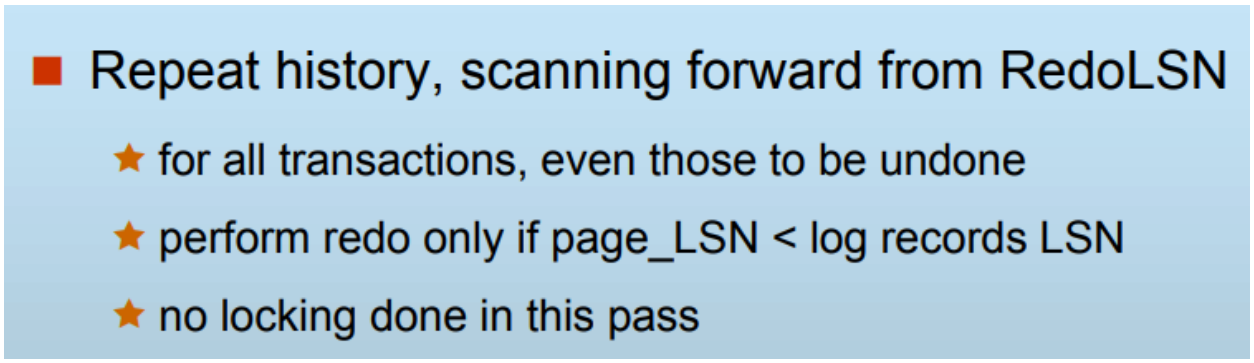
The actual recovery process consists of three passes:



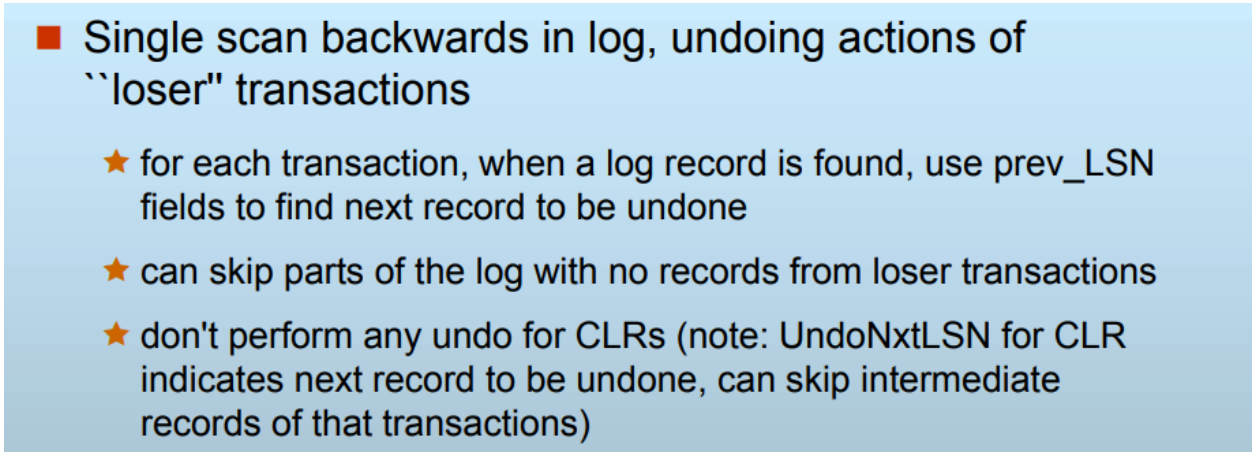
1. Analysis. The recovery subsystem determines the earliest log record from which the next pass must start. It also scans the log forward from the checkpoint record to construct a snapshot of what the system looked like at the instant of the crash.



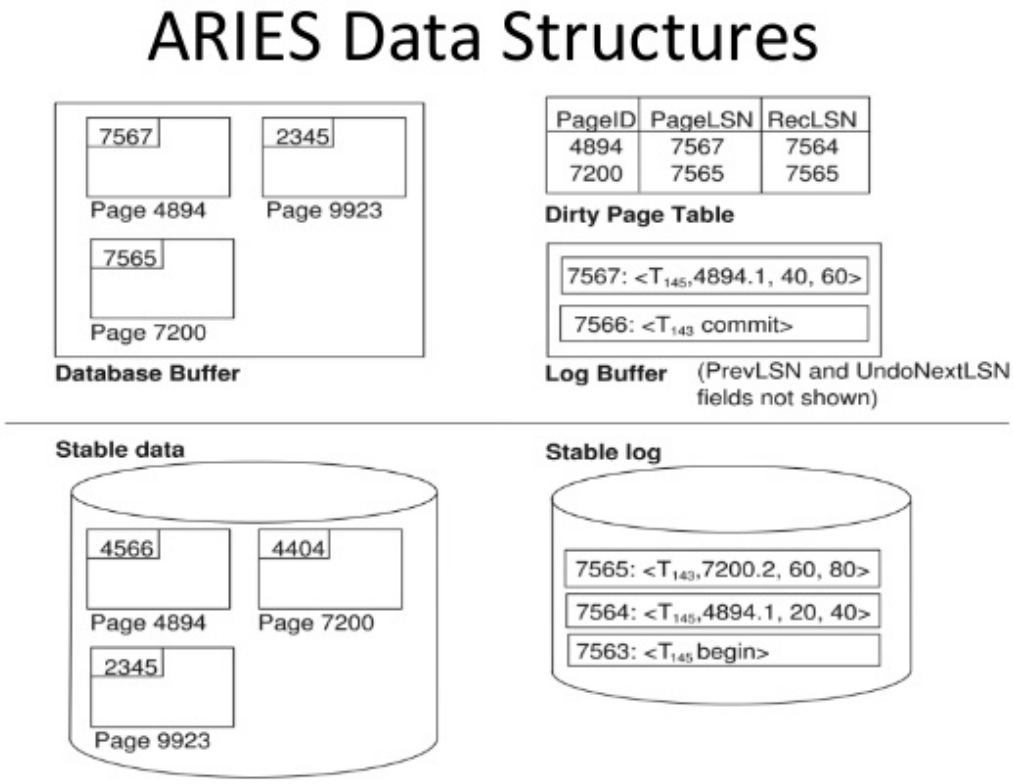
2. Redo. Starting at the earliest LSN determined in pass (1) above, the log is read forward and each update redone.



3. Undo. The log is scanned backward and updates corresponding to loser transactions are undone



.ARIES is a state of the art recovery method that .Incorporates numerous optimizations to reduce overheads during normal processing and to speed up recovery PROCESS.



.ARIES is a state of the art recovery method that .Incorporates numerous optimizations to reduce overheads during normal processing and to speed up recovery PROCESS.

RAID

RAID is a technology that is used to increase the performance and/or reliability of data storage. The abbreviation stands for either *Redundant Array of Inexpensive Disks* or *Redundant Array of Independent Drives*. A RAID system consists of two or more drives working in parallel. These can be hard discs, but there is a trend to also use the technology for SSD (Solid State Drives). There are different RAID levels, each optimized for a specific situation. These are not standardized by an industry group or standardization committee. This explains why companies sometimes come up with their own unique numbers and implementations. This article covers the following RAID levels:

- **RAID 0** – striping
- **RAID 1** – mirroring
- **RAID** 2 – striping with parity
- **RAID** 3 – striping with double parity
- **RAID** 4 – combining mirroring and striping

The software to perform the RAID-functionality and control the drives can either be located on a separate controller card (a hardware RAID controller) or it can simply be a driver. Some versions of Windows, such as Windows Server 2012 as well as Mac OS X, include software RAID functionality. Hardware RAID controllers cost more than pure software, but they also offer better performance, especially with RAID 5 and 6.

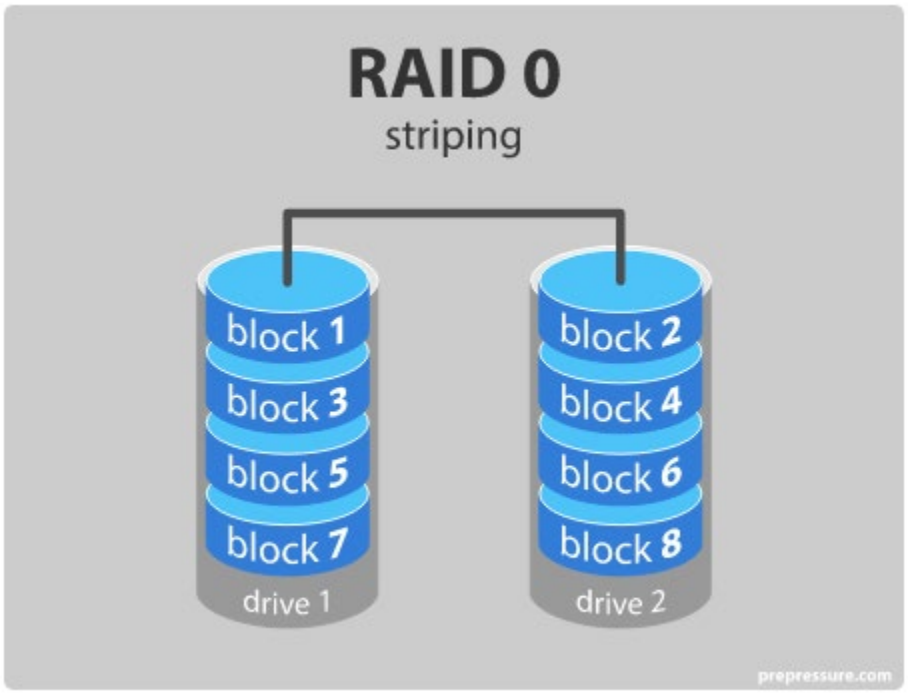
RAID-systems can be used with a number of interfaces, including SCSI, IDE, SATA or FC (fiber channel.) There are systems that use SATA disks internally, but that have a FireWire or SCSI-interface for the host system.

Sometimes disks in a storage system are defined as JBOD, which stands for *'Just a Bunch Of Disks'*. This means that those disks do not use a specific RAID level and acts as stand-alone disks. This is often done for drives that contain swap files or spooling data.

Below is an overview of the most popular RAID levels:

RAID level 0 – Striping

In a RAID 0 system data are split up into blocks that get written across all the drives in the array. By using multiple disks (at least 2) at the same time, this offers superior I/O performance. This performance can be enhanced further by using multiple controllers, ideally one controller per disk.



Advantages

- RAID 0 offers great performance, both in read and write operations. There is no overhead caused by parity controls.
- All storage capacity is used, there is no overhead.
- The technology is easy to implement.

Disadvantages

- RAID 0 is not fault-tolerant. If one drive fails, all data in the RAID 0 array are lost. It should not be used for mission-critical systems.

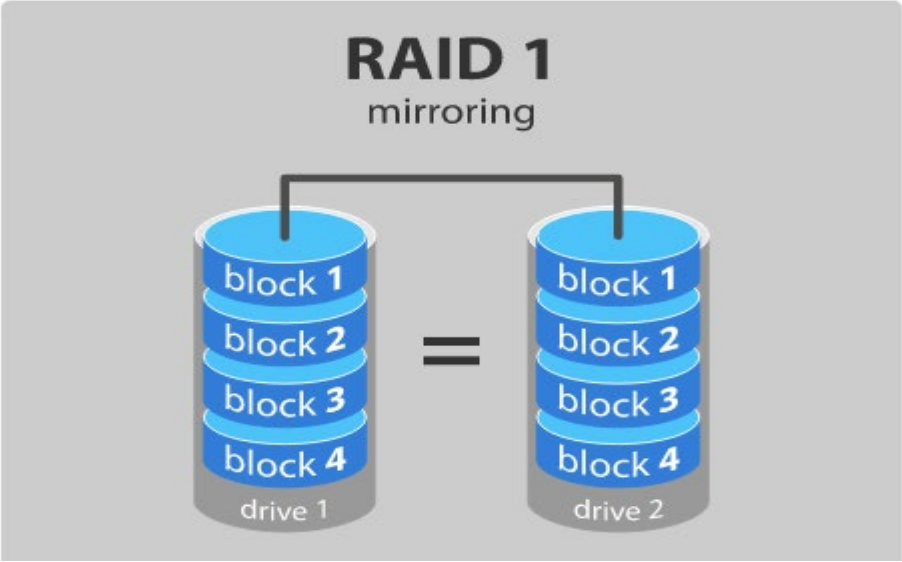
Ideal use

RAID 0 is ideal for non-critical storage of data that have to be read/written at a high speed, such as on an image retouching or video editing station.

If you want to use RAID 0 purely to combine the storage capacity of twee drives in a single volume, consider mounting one drive in the folder path of the other drive. This is supported in Linux, OS X as well as Windows and has the advantage that a single drive failure has no impact on the data of the second disk or SSD drive.

RAID level 1 – Mirroring

Data are stored twice by writing them to both the data drive (or set of data drives) and a mirror drive (or set of drives). If a drive fails, the controller uses either the data drive or the mirror drive for data recovery and continues operation. You need at least 2 drives for a RAID 1 array.



Advantages

- RAID 1 offers excellent read speed and a write-speed that is comparable to that of a single drive.
- In case a drive fails, data do not have to be rebuild, they just have to be copied to the replacement drive.
- RAID 1 is a very simple technology.

Disadvantages

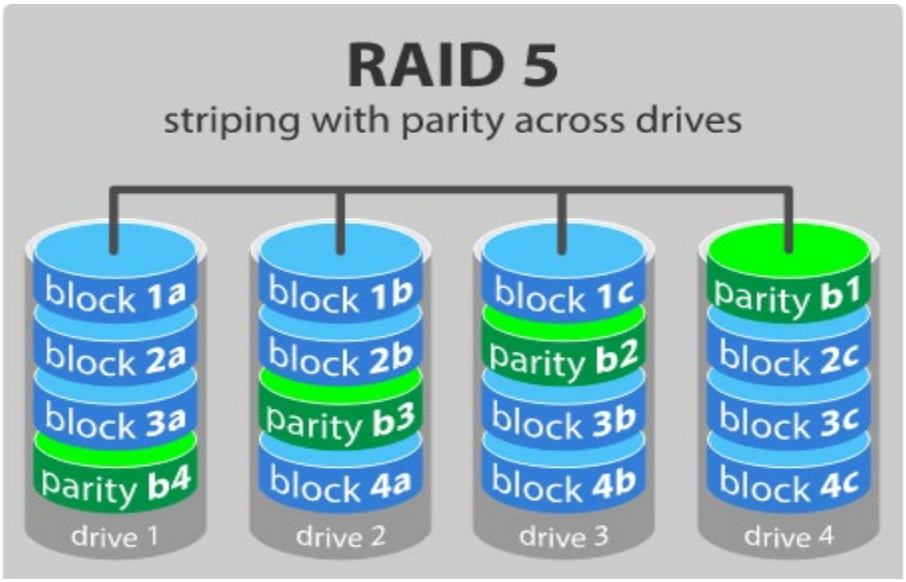
- The main disadvantage is that the effective storage capacity is only half of the total drive capacity because all data get written twice.
- Software RAID 1 solutions do not always allow a hot swap of a failed drive. That means the failed drive can only be replaced after powering down the computer it is attached to. For servers that are used simultaneously by many people, this may not be acceptable. Such systems typically use hardware controllers that do support hot swapping.

Ideal use

RAID-1 is ideal for mission critical storage, for instance for accounting systems. It is also suitable for small servers in which only two data drives will be used.

RAID level 2

RAID 2 is the most common secure RAID level. It requires at least 3 drives but can work with up to 16. Data blocks are striped across the drives and on one drive a parity checksum of all the block data is written. The parity data are not written to a fixed drive, they are spread across all drives, as the drawing below shows. Using the parity data, the computer can recalculate the data of one of the other data blocks, should those data no longer be available. That means a RAID 2 array can withstand a single drive failure without losing data or access to data. Although RAID 2 can be achieved in software, a hardware controller is recommended. Often extra cache memory is used on these controllers to improve the write performance.



Advantages

- Read data transactions are very fast while write data transactions are somewhat slower (due to the parity that has to be calculated).
- If a drive fails, you still have access to all data, even while the failed drive is being replaced and the storage controller rebuilds the data on the new drive.

Disadvantages

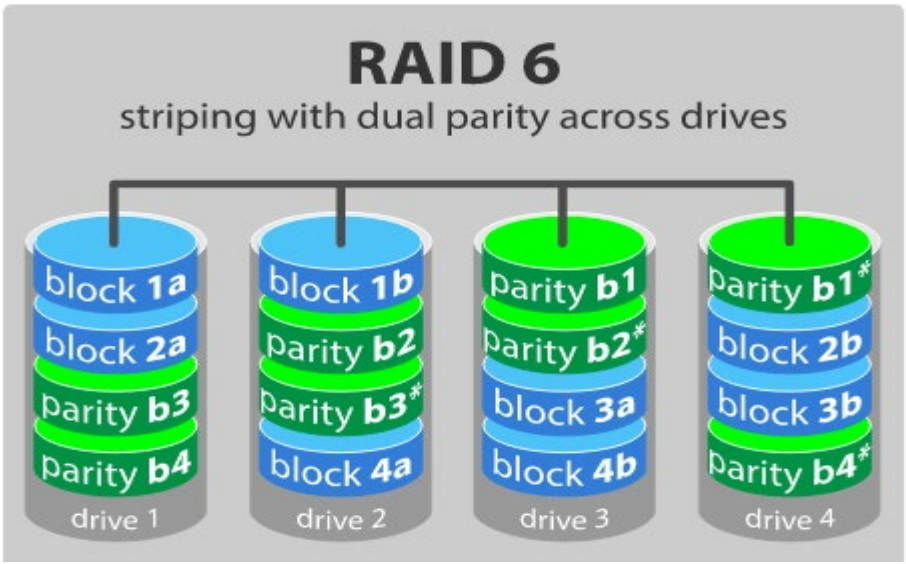
- Drive failures have an effect on throughput, although this is still acceptable.
- This is complex technology. If one of the disks in an array using 4TB disks fails and is replaced, restoring the data (the rebuild time) may take a day or longer, depending on the load on the array and the speed of the controller. If another disk goes bad during that time, data are lost forever.

Ideal use

RAID 5 is a good all-round system that combines efficient storage with excellent security and decent performance. It is ideal for file and application servers that have a limited number of data drives.

RAID level 3 – Striping with double parity

RAID 3 is like RAID 2, but the parity data are written to two drives. That means it requires at least 4 drives and can withstand 2 drives dying simultaneously. The chances that two drives break down at exactly the same moment are of course very small. However, if a drive in a RAID 2systems dies and is replaced by a new drive, it takes hours or even more than a day to rebuild the swapped drive. If another drive dies during that time, you still lose all of your data. With RAID 3, the RAID array will even survive that second failure.



Advantages

- Like with RAID 2, read data transactions are very fast.
- If two drives fail, you still have access to all data, even while the failed drives are being replaced. So RAID 3 is more secure than RAID 2.

Disadvantages

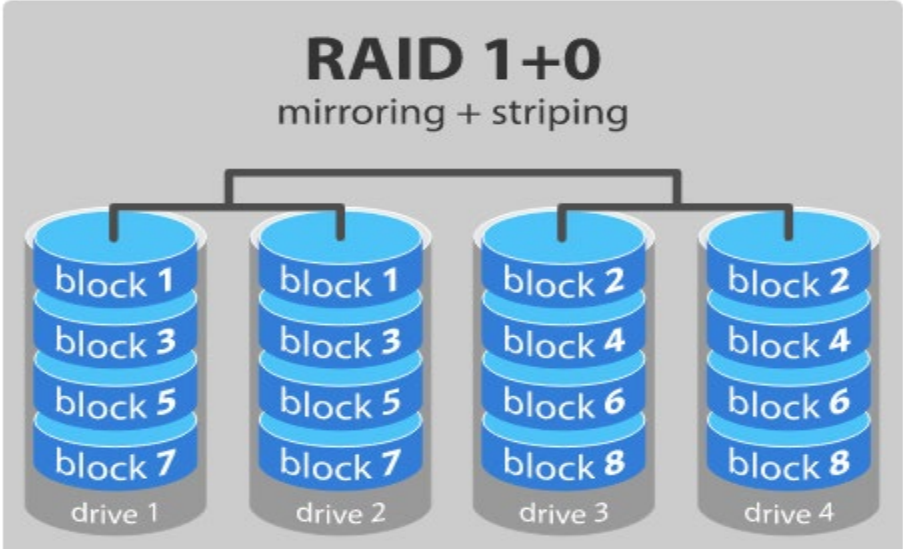
- Write data transactions are slower than RAID 2 due to the additional parity data that have to be calculated. In one report I read the write performance was 20% lower.
- Drive failures have an effect on throughput, although this is still acceptable.
- This is complex technology. Rebuilding an array in which one drive failed can take a long time.

Ideal use

RAID 6 is a good all-round system that combines efficient storage with excellent security and decent performance. It is preferable over RAID 5 in file and application servers that use many large drives for data storage.

RAID level 4 – combining RAID 1 & RAID 0

It is possible to combine the advantages (and disadvantages) of RAID 0 and RAID 1 in one single system. This is a nested or hybrid RAID configuration. It provides security by mirroring all data on secondary drives while using striping across each set of drives to speed up data transfers.



Advantages

- If something goes wrong with one of the disks in a RAID 4 configuration, the rebuild time is very fast since all that is needed is copying all the data from the surviving mirror to a new drive. This can take as little as 30 minutes for drives of 1 TB.

Disadvantages

- Half of the storage capacity goes to mirroring, so compared to large RAID 2 or RAID 3 arrays, this is an expensive way to have redundancy.