

Normal Forms

CSIR81

Types of Keys

- Attributes : ***Name , Roll_No , Address*** etc.
- Candidate Key:
 - ❖ ***Minimum set/combination of attributes which can uniquely identify a row/tuple in a table***
 - ❖ ***Can contain NULL attribute(s)***

Types of Keys

- Prime Attributes: ***Attributes which form candidate keys***
- Primary Key:
 - ❖ ***Any candidate key can be chosen as Primary Key***
 - ❖ ***Can't contain NULL attribute***
- Super Key:
 - ❖ ***Adding one or more attributes to the candidate key generates super keys***
 - ❖ ***A candidate key is a minimal super key but vice versa is not true***

Functional Dependencies

- Reflexivity : *If key Y can be derived by key X , then $X \rightarrow Y$*
- Augmentation: *If $X \rightarrow Y$ then, $XZ \rightarrow YZ$*
- Transitivity: *If $X \rightarrow Y$, $Y \rightarrow Z$ then, $X \rightarrow Z$*
- Union: *If $X \rightarrow Y$, $X \rightarrow Z$ then, $X \rightarrow YZ$*
- Decomposition: *If $X \rightarrow YZ$ then, $X \rightarrow Y$ and $X \rightarrow Z$*
- Pseudo Transitivity: *If $X \rightarrow Y$, $WY \rightarrow Z$ then, $WX \rightarrow Z$*
- Composition: *If $X \rightarrow Y$, $Z \rightarrow W$ then, $XZ \rightarrow YW$*

1st NF | No Multi valued Attribute

Convert the below table into 1st NF :

RollNo	Name	Course
1	Happy	C , C++
2	Sad	Java
3	Angry	C,DBMS

Method 1:

RollNo	Name	Course
1	Happy	C
1	Happy	C++
2	Sad	Java
3	Angry	C
3	Angry	DBMS

1st NF

- Method 2:

RollNo	Name	Course1	Course2
1	Happy	C	C++
2	Sad	Java	NULL
3	Angry	C	DBMS

1st NF

- Method 3:

RollNo	Name
1	Happy
2	Sad
3	Angry

RollNo (Foreign Key)	Course
1	C
1	C++
2	Java
3	C
3	DBMS

2nd NF

- A relation should be in 1st NF
- No partial dependency:
 - ❖ ***All non-prime attributes*** should be ***fully functional dependent*** on candidate key
 - ❖ Example: Let (A, B) be the candidate key in a relation and there exists a dependency such that $B \rightarrow C$ then it's a partial dependency

2nd NF

- Convert the following relation into 2nd NF:

CustomerID	StoreID	Location
1	1	Delhi
1	3	Mumbai
2	1	Delhi
3	2	Chennai
4	3	Mumbai

Candidate Key: (**CustomerID, StoreID**)

Prime Attributes : {**CustomerID, StoreID**}

Non-Prime Attributes: {**Location**}

2nd NF

Solution:

StoreID	Location
1	Delhi
2	Chennai
3	Mumbai

CustomerID	StoreID (Foreign Key)
1	1
1	3
2	1
3	2
4	3

3rd NF

- A relation should be in 2nd NF
- No Transitive Dependency
 - ❖ *Non-Prime Attribute should not be derived by another Non-Prime Attribute*

3rd NF

- Convert below relation into 3rd NF:

RollNo	State	Country
1	Punjab	India
2	Rajasthan	India
3	Texas	USA
4	Florida	USA
5	Kerala	India
6	Punjab	India

- Candidate Key: (*RollNo*), Non-Prime Attributes: {*State*, *Country*}

3rd NF

- Solution: Decompose it into two relation ***R(RollNo, State)*** and ***R(State, Country)***

State	Country
Punjab	India
Rajasthan	India
Texas	USA
Florida	USA
Kerala	India

RollNo	State (Foreign Key)
1	Punjab
2	Rajasthan
3	Texas
4	Florida
5	Kerala
6	Punjab

BCNF

- A relation should be in 3rd NF
- For any dependency $A \rightarrow B$, A should be the super key
 - ❖ A cannot be a non-prime attribute, if B is a prime attribute

BCNF

Convert below relation into BCNF:

RollNo	Subject	Faculty
1	Java	P.Java
1	C++	P.Cpp
2	Java	P.Java2
3	C#	P.CHash
4	Java	P.Java

Relation : ***R(RollNo, Subject, Faculty)***

Candidate Keys: ***{(RollNo, Subject)}***

Prime Attributes: ***{RollNo, Subject}***

Non-Prime Attribute: ***{Faculty}***

Not in BCNF because : ***{Faculty(Not a Superkey) → Subject(Prime Attribute)}***

BCNF

- Solution:

FacultyID	Faculty	Subject
1	P.Java	Java
2	P.Cpp	C++
3	P.Java2	Java
4	P.CHash	C#

RollNo	FacultyID (Foreign Key)
1	1
1	2
2	3
3	4
4	1

Now this relation is in BCNF

4th NF | Multi-Valued Dependency

A table is said to have a **multi-valued dependency** if:

- For a single value of A in the dependency $A \twoheadrightarrow B$, multiple values of B exist.
- A table should have ***at least 3 columns***.
- For the relation $R(A, B, C)$, if A and B have a multi-valued dependency, then B and C should be independent of each other.

4th NF | Multi-Valued Dependency

- Example:

EMPLOYEE_ID	DEPARTMENT	HOBBY
E901	HR	Badminton
E901	Sales	Reading
E902	Marketing	Cricket
E903	Finance	Football

4th NF | Multi-Valued Dependency

A relation ***R*** is in 4th NF if

- ***R is in BCNF***
- ***R doesn't have Multivalued Dependency***

5th NF | Join Dependency

- Consider a table ***R***:

E_Name	Company	Product
Happy	Comp1	Jeans
Sad	Comp2	Jacket
Angry	Comp3	TShirt

- Decompose the above table into three tables ***R1***, ***R2***, ***R3***

5th NF | Join Dependency

E_Name	Company
Happy	Comp1
Sad	Comp2
Angry	Comp3

R1

E_Name	Product
Happy	Jeans
Sad	Jacket
Angry	TShirt

R2

Company	Product
Comp1	Jeans
Comp2	Jacket
Comp3	TShirt

R3

5th NF | Join Dependency

E_Name	Company	Product
Happy	Comp1	Jeans
Sad	Comp2	Jacket
Angry	Comp3	TShirt

$R1 \bowtie R2$

E_Name	Company	Product
Happy	Comp1	Jeans
Sad	Comp2	Jacket
Angry	Comp3	TShirt

$R1 \bowtie R2 \bowtie R3$

If $R1 \bowtie R2 \bowtie R3 = R$, there is **join dependency** (Above Example has Join Dependency)

A relation R is in 5th NF if it satisfies the following conditions:

- *R should already be in 4th NF*
- *It cannot be further decomposed loseless or can't have join dependency*

Example Question 1

Relation R has eight attributes $ABCDEFGH$. Fields of R contain only atomic values. $\mathbf{FDs} = \{CH \rightarrow G, A \rightarrow BC, B \rightarrow CFH, E \rightarrow A, F \rightarrow EG\}$ is a set of functional dependencies (FDs). How many candidate keys does the relation R have?

- Solution: $A^+ = B^+ = E^+ = F^+ = ABCEFGH$
 $AD^+ = BD^+ = ED^+ = FD^+ = ABCDEFGH$
4 Candidate Keys $\{AD, BD, ED, FD\}$

Example Question 2

For the relation $R(ABCDEFGH)$ with $FDs = \{CH \rightarrow G, A \rightarrow BC, B \rightarrow CFH, E \rightarrow A, F \rightarrow EG\}$. The relation R is

- a) In 1st NF ,but not in 2nd NF
- b) In 2nd NF , but not in 3rd NF
- c) In 3rd NF , but not in BCNF
- d) In BCNF

*The table is not in 2nd NF as the non-prime attributes are dependent on subsets of candidate keys. The candidate keys are **AD**, **BD**, **ED** and **FD**. In all of the $FDs\{A \rightarrow BC, B \rightarrow CFH, F \rightarrow EG\}$, the non-prime attributes are dependent on a partial candidate key.*

Example Question 3

What is the highest NF for the relation $R(ABCD)$ with $FDs = \{AB \rightarrow D, CB \rightarrow D, A \rightarrow C, C \rightarrow A\}$?

Solution : 3rd NF

Here there are two Candidate keys, AB and CB . Now $AB \rightarrow D$ and $CB \rightarrow D$ satisfy BCNF as LHS is superkey in both. But, $A \rightarrow C$ and $C \rightarrow A$, doesn't satisfy BCNF. Hence , we check for 3rd NF for the FDs $A \rightarrow C$ and $C \rightarrow A$. As C and A are on RHS of both the FDs are prime attributes, they satisfy 3rd NF.

File Organization and indexing (B and B+ Trees)

Roll No: 106119030

Files In DBMS:

- A database consist of a huge amount of data. The data is grouped within a table in RDBMS, and each table have related records.
- A user can see that the data is stored in form of tables, but in actual this huge amount of data is stored in physical memory in form of files.

File – A file is named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks.

File Organization:

What is File Organization?

File Organization refers to the logical relationships among various records that constitute the file, particularly with respect to the means of identification and access to any specific record.

Storing the files in certain order is called file Organization.

Types of File organization:

Various methods have been introduced to Organize files.

The types are:

- Sequential File Organization
- Heap File Organization
- Hash File Organization
- B+ Tree File Organization
- Clustered File Organization

These particular methods have advantages and disadvantages on the basis of access or selection . Thus, it is all upon the programmer to decide the best suited file Organization method according to the requirements.

Sequential File organization:

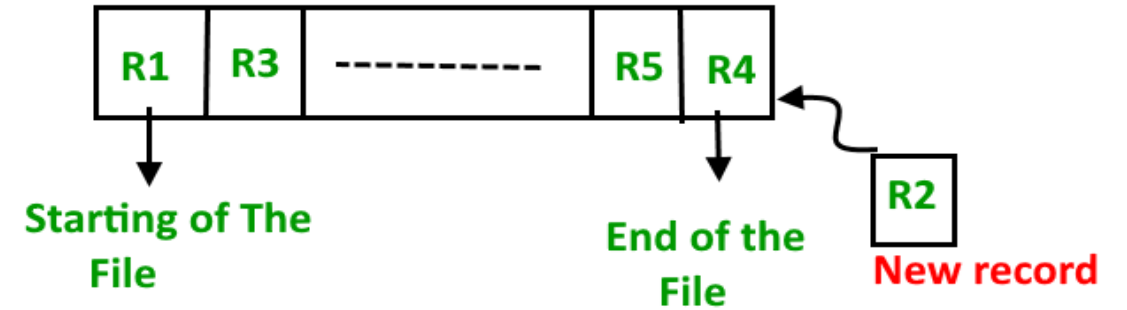
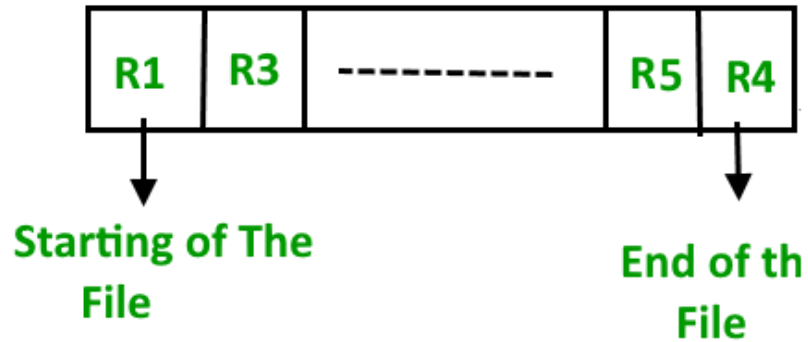
The easiest method for file Organization is Sequential method. In this method the file are stored one after another in a sequential manner.

There are two ways to implement this method:

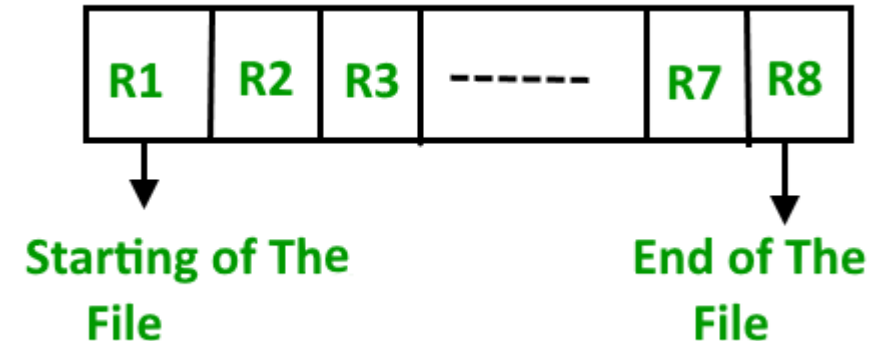
- Pile File Method – This method is quite simple, in which we store the records in a sequence i.e one after other in the order in which they are inserted into the tables.
- Sorted File Method –In this method, As the name itself suggest whenever a new record has to be inserted, it is always inserted in a sorted (ascending or descending) manner. Sorting of records may be based on any primary key or any other key.

Examples for sequential file organization

Pile File Method:



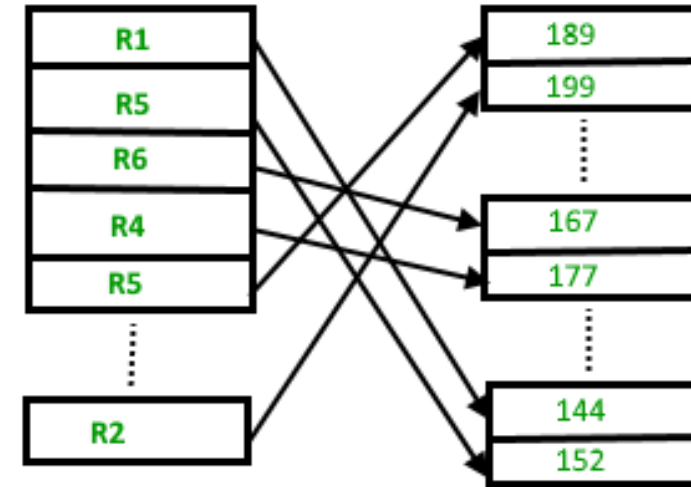
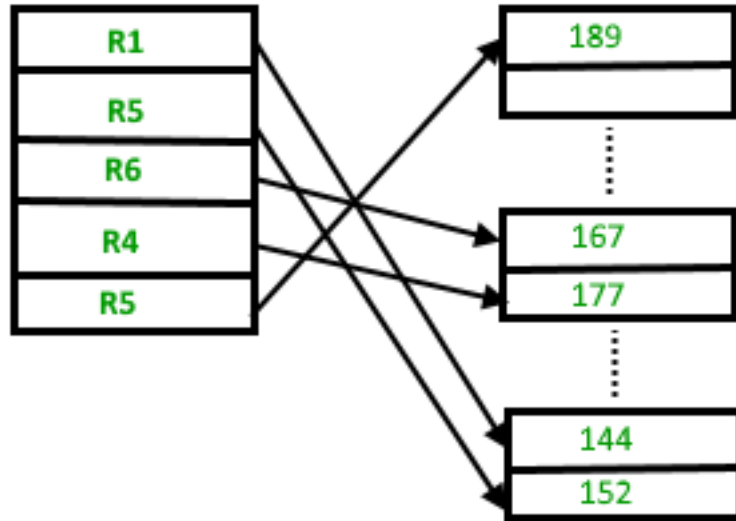
Sorted File Method:



Heap File organization:

- Heap File Organization works with data blocks. In this method records are inserted at the end of the file, into the data blocks.
- No Sorting or Ordering is required in this method.
- If a data block is full, the new record is stored in some other block, Here the other data block need not be the very next data block, but it can be any block in the memory.
- It is the responsibility of DBMS to store and manage the new records.

Example



If we want to search, delete or update data in heap file Organization the we will traverse the data from the beginning of the file till we get the requested record. Thus if the database is very huge, searching, deleting or updating the record will take a lot of time.

Hash file organization:

It becomes very inefficient to search all the index values and reach the desired data. So in order to make the process better we use hashing in order to find the index of data block in which we are going to place or retrieve the record.

Two types of Hash file organization:

- Static Hashing
- Dynamic hashing

Static hashing:

In static hashing, when a search-key value is provided, the hash function always computes the same address. For example, if we want to generate an address for `STUDENT_ID = 104` using mod (5) hash function, it always results in the same bucket address 4.

There will not be any changes to the bucket address here.

Hence the number of data buckets in the memory for this static hashing remain constant throughout.

Dynamic hashing

The drawback of static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks.

In Dynamic hashing, data buckets grows or shrinks (added or removed dynamically) as the records increases or decreases.

Dynamic hashing is also known as extended hashing.

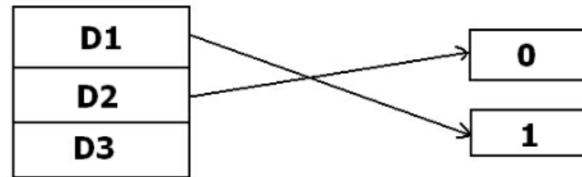
In dynamic hashing, the hash function is made to produce a large number of values.

Example for dynamic hashing:

Assume that, there are three data records D1, D2 and D3 . The hash function generates three addresses 1001, 0101 and 1010 respectively.

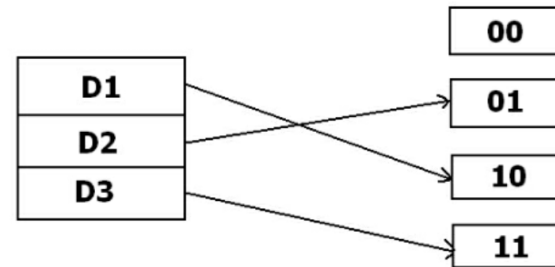
This method of storing considers only part of this address – especially only first one bit to store the data. So it tries to load three of them at a

h(D1) -> 1001
h(D2) -> 0101
h(D3) -> 1010



But the problem is that No bucket address is remaining for D3. The bucket has to grow dynamically to accommodate D3. So it changes the address have 2 bits rather than 1 bit, and then it updates the existing data to have 2 bit address. Then it tries to accommodate D3.

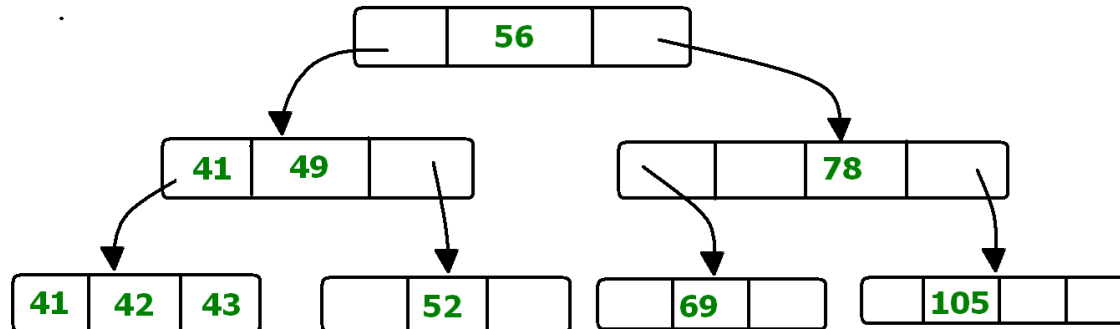
h(D1) -> 1001
h(D2) -> 0101
h(D3) -> 1010



B+ Tree File organization:

It uses a tree like structure to store records in File. It uses the concept of Key indexing where the primary key is used to sort the records.

B+ Tree is very much similar to binary search tree.



Cluster file organization:

In cluster file organization, two or more related tables/records are stored within same file known as clusters. These files will have two or more tables in the same data block and the key attributes which are used to map these table together are stored only once.

Thus it lowers the cost of searching and retrieving various records in different files as they are now combined and kept in a single cluster.

Example:

For example we have two tables or relation Employee and Department. These table are related to each other. Therefore these table are allowed to combine using a join operation and can be seen in a cluster file.


EMPLOYEE

EMP ID	EMP_NAME	EMP_ADD	DEP_ID
01	JOE	CAPE TOWN	D_101
02	ANNIE	FRANSISCO	D_103
03	PETER	CROY CITY	D_101
04	JOHN	FRANSISCO	D_102
05	LUNA	TOKYO	D_106
06	SONI	W.LAND	D_105
07	SAKACHI	TOKYO	D_104
08	MARY	NOVI	D_101

DEPARTMENT

DEP_ID	DEP_NAME
D_101	ECO
D_102	CS
D_103	JAVA
D_104	MATHS
D_105	BIO
D_106	CIVIL

CLUSTER KEY



DEP_ID	DEP_NAME	EMP ID	EMP_NAME	EMP_ADD
D_101	ECO	01	JOE	CAPE TOWN
		02	PETER	CROY CITY
		03	MARY	NOVI
D_102	CS	04	JOHN	FRANSISCO
D_103	JAVA	05	ANNIE	FRANSISCO
D_104	MATHS	06	SAKACHI	TOKYO
D_105	BIO	07	SONI	W.LAND
D_106	CIVIL	08	LUNA	TOKYO

DEPARTMENT + EMPLOYEE

Cluster key is the key with which joining of the table is performed.

B Trees:

When it comes to storing and searching large amounts of data, traditional binary search trees can become impractical due to their poor performance and high memory usage.

B-Trees, also known as B-Tree or Balanced Tree, are a type of self-balancing tree that was specifically designed to overcome these limitations.

Each node in a B-Tree can contain multiple keys, which allows the tree to have a larger branching factor and thus a shallower height. This shallow height leads to less disk I/O, which results in faster search and insertion operations.

B-Trees maintain balance by ensuring that each node has a minimum number of keys, so the tree is always balanced. This balance guarantees that the time complexity for operations such as insertion, deletion, and searching is always $O(\log n)$.

Properties of B Trees

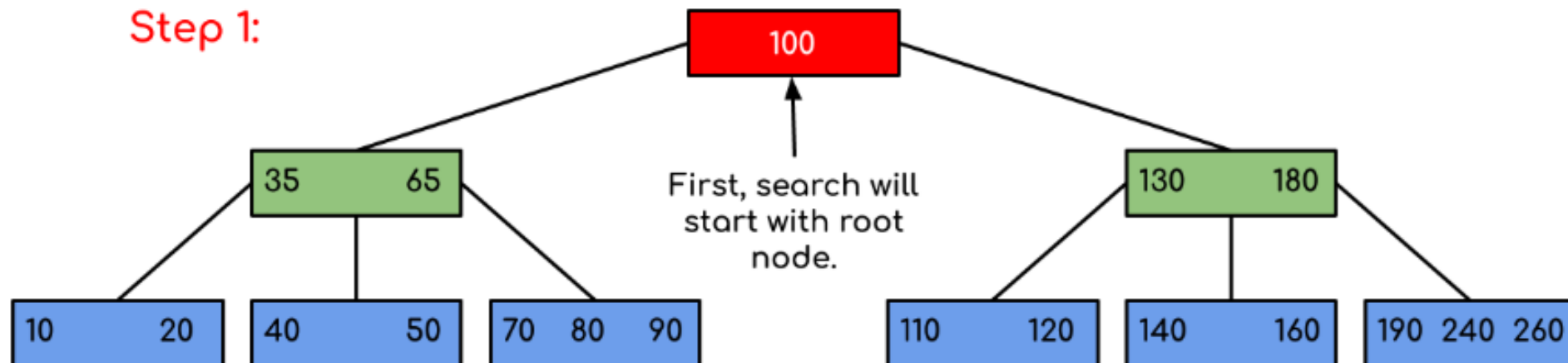
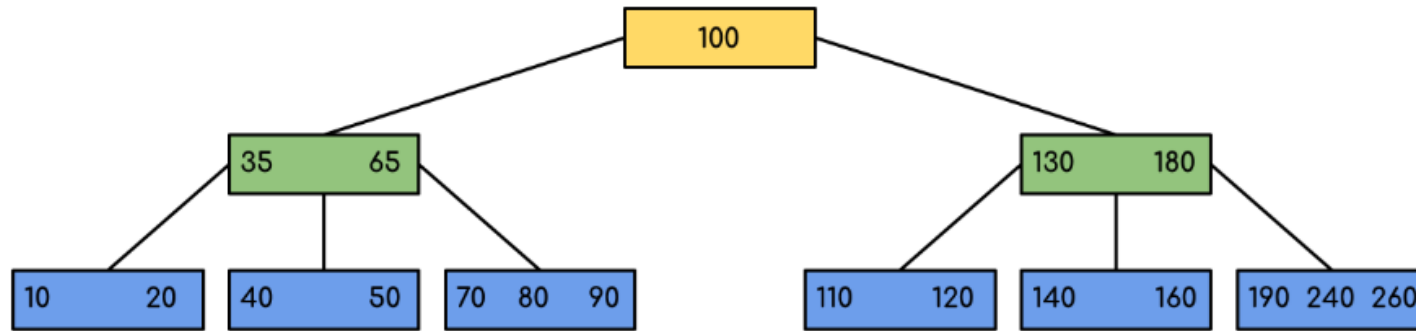
- All leaves are at the same level.
- B-Tree is defined by the term minimum degree ' t '. The value of ' t ' depends upon disk block size.
- Every node except the root must contain at least $t-1$ keys. The root may contain a minimum of **1** key.
- All nodes (including root) may contain at most **$(2*t - 1)$** keys.
- Number of children of a node is equal to the number of keys in it plus **1**.
- All keys of a node are sorted in increasing order. The child between two keys **k_1** and **k_2** contains all keys in the range from **k_1** and **k_2** .
- B-Tree grows and shrinks from the root which is unlike Binary Search Tree. Binary Search Trees grow downward and also shrink from downward.
- Like other balanced Binary Search Trees, the time complexity to search, insert and delete is $O(\log n)$.
- Insertion of a Node in B-Tree happens only at Leaf Node

Search in B-Tree:

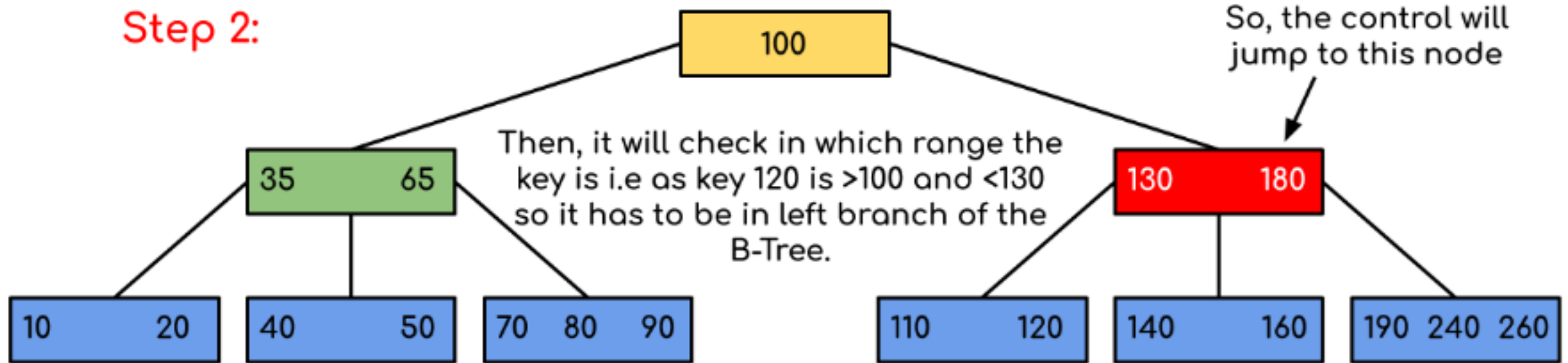
- Start from the root and recursively traverse down.
- For every visited non-leaf node,
 - If the node has the key, we simply return the node.
 - Otherwise, we recur down to the appropriate child (The child which is just before the first greater key) of the node.
- If we reach a leaf node and don't find k in the leaf node, then return NULL.

Example:

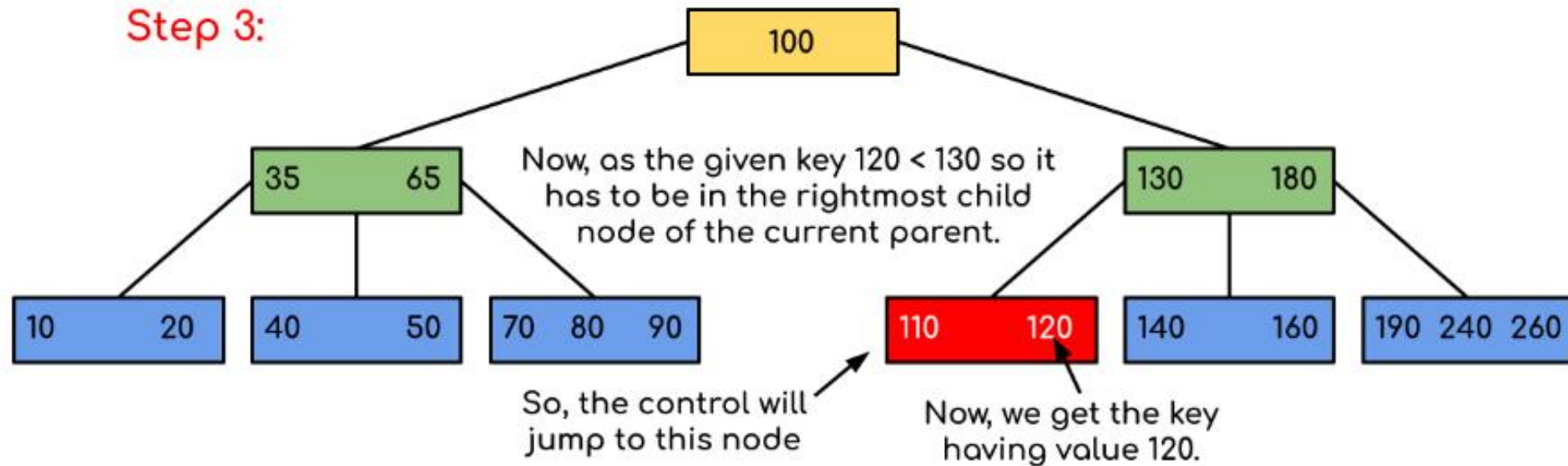
Search 120 in the given B-Tree.



Step 2:



Step 3:

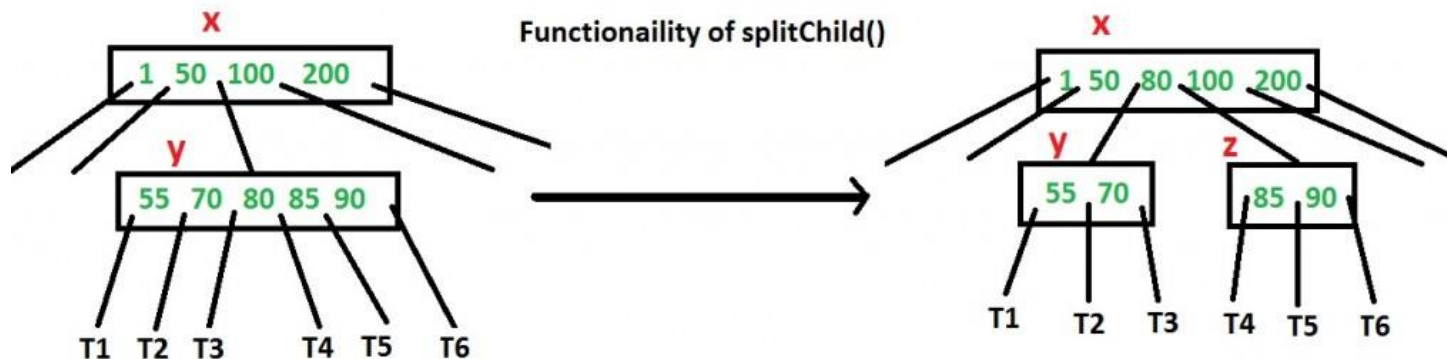


Insertion in B- Tree:

A new key is always inserted at the leaf node. Let the key to be inserted be k. Like BST, we start from the root and traverse down till we reach a leaf node. Once we reach a leaf node, we insert the key in that leaf node.

So before inserting a key to the node, we make sure that the node has extra space.

If there is no extra space we have to split the node and it can be done as follows:



Insertion

1) Initialize x as root.

2) While x is not leaf, do following

..**a)** Find the child of x that is going to be traversed next. Let the child be y.

..**b)** If y is not full, change x to point to y.

..**c)** If y is full, split it and change x to point to one of the two parts of y. If k is smaller than mid key in y, then set x as the first part of y. Else second part of y. When we split y, we move a key from y to its parent x.

3) The loop in step 2 stops when x is leaf. x must have space for 1 extra key as we have been splitting all nodes in advance. So simply insert k to x.

Example of insertion:

Insert a sequence of integers 10, 20, 30, 40, 50, 60, 70, 80 and 90 in an initially empty B-Tree. Assume the max keys in a block is 5.

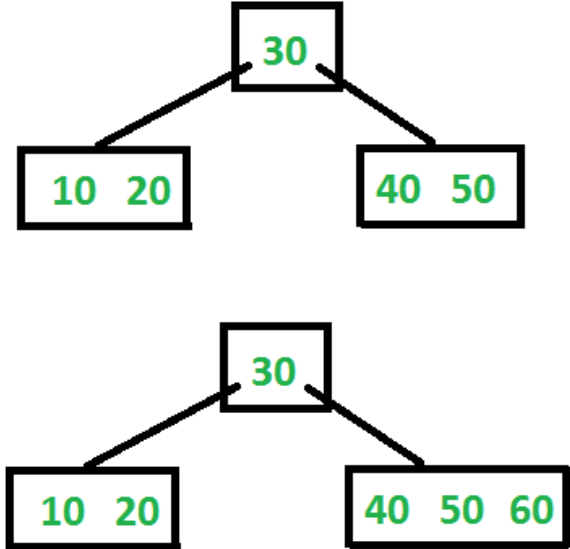
Insert 10



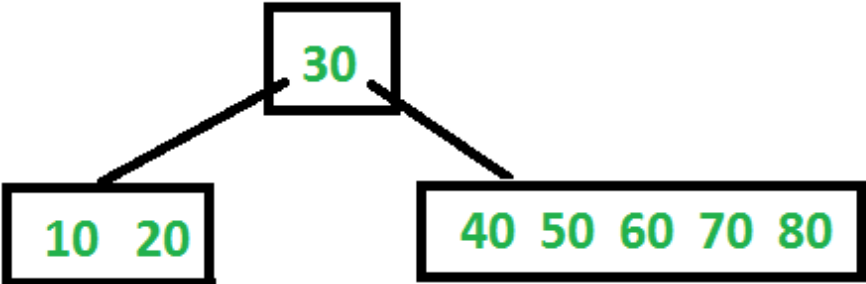
Insert 20, 30, 40 and 50



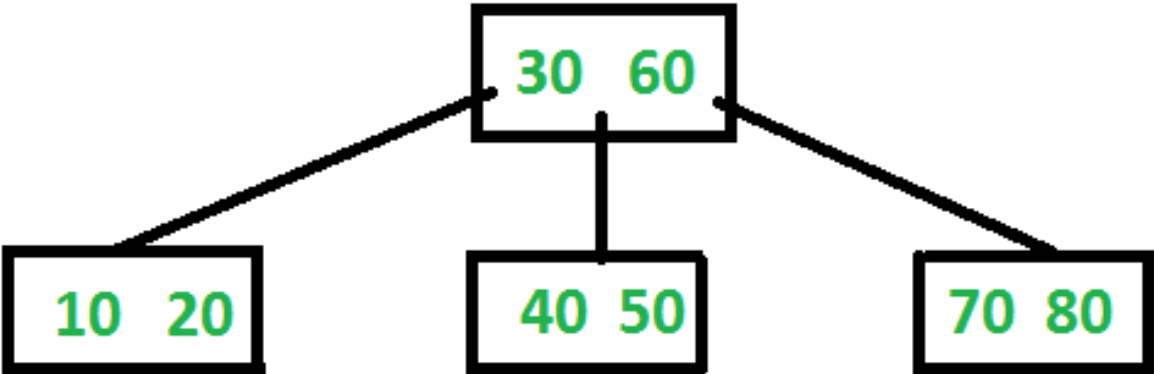
Insert 60



Insert 70 and 80

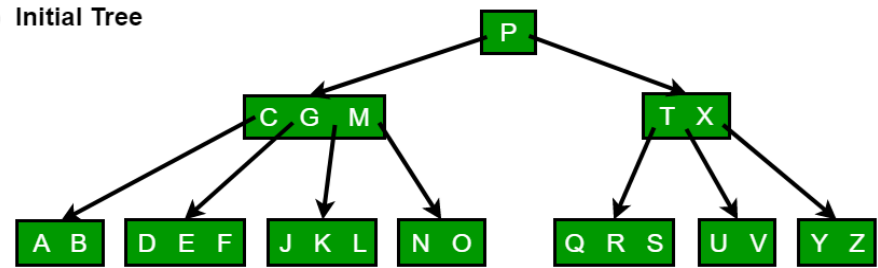


Insert 90

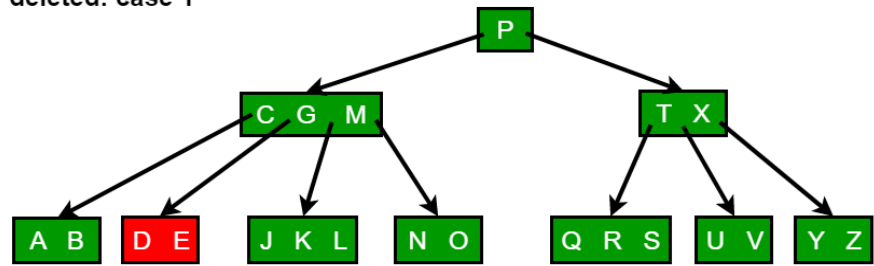


Deletion:

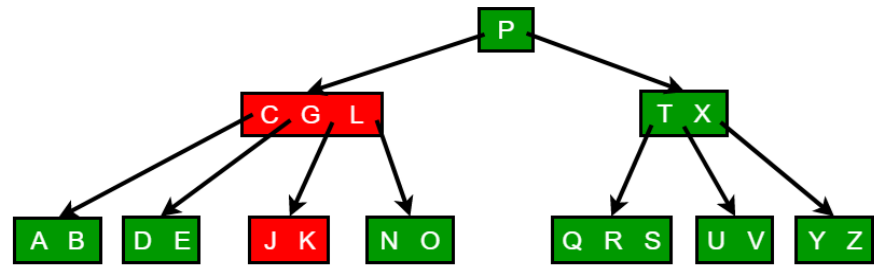
(a) Initial Tree



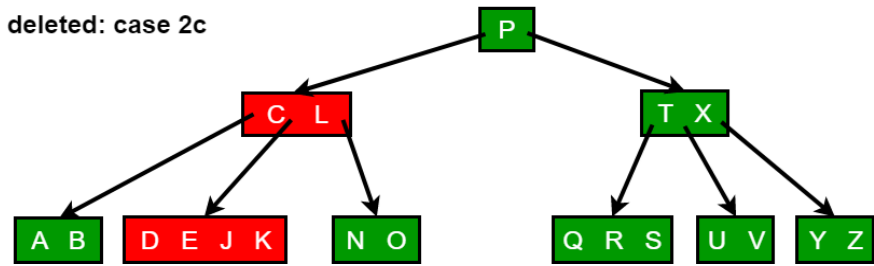
(b) F deleted: case 1



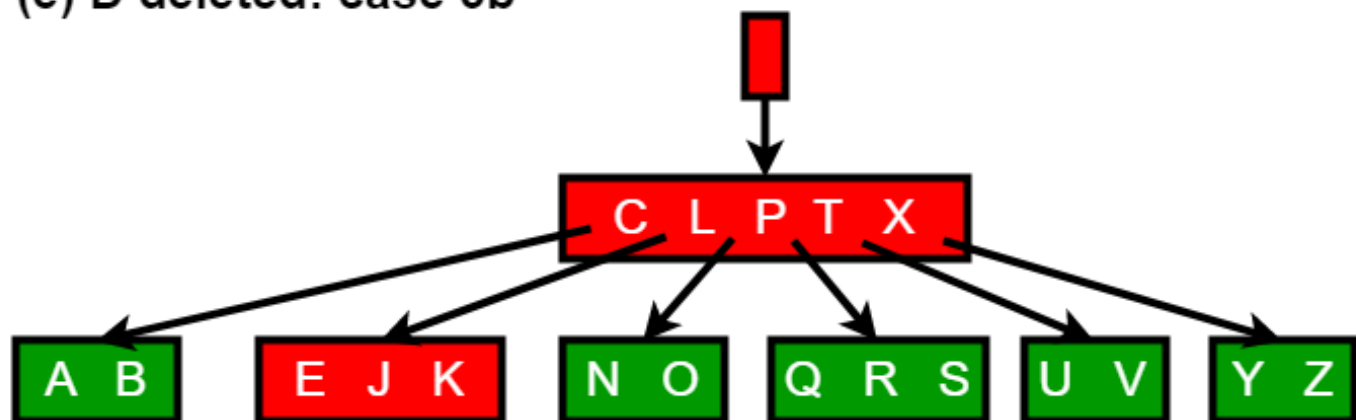
(c) M deleted: case 2a



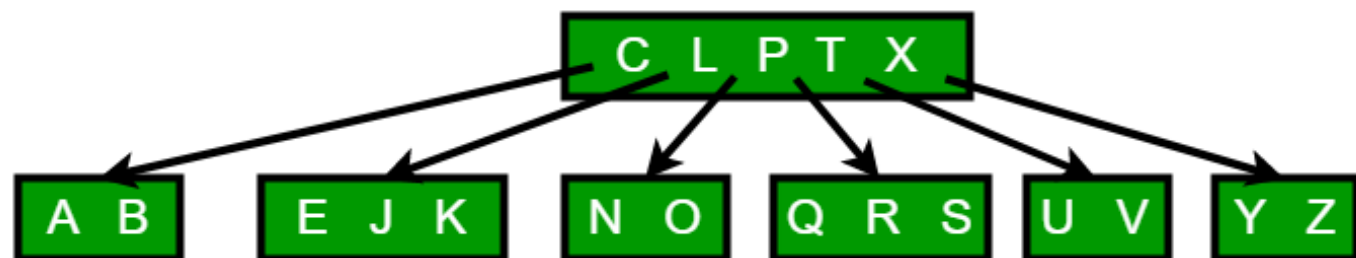
(d) G deleted: case 2c



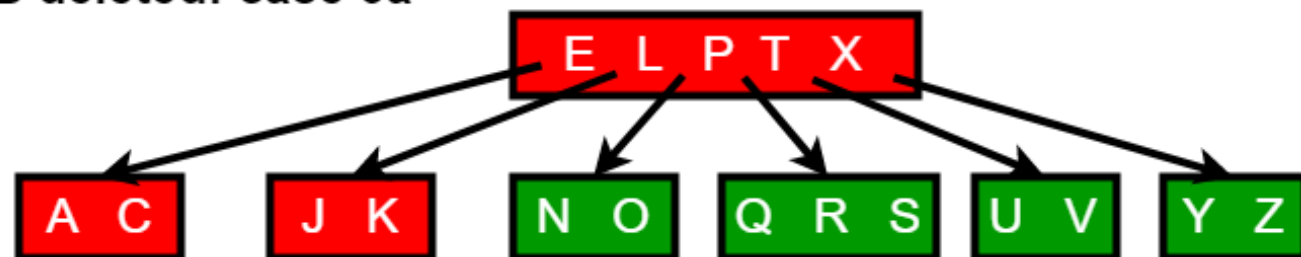
(e) D deleted: case 3b



(e') tree shrinks in height



(f) B deleted: case 3a



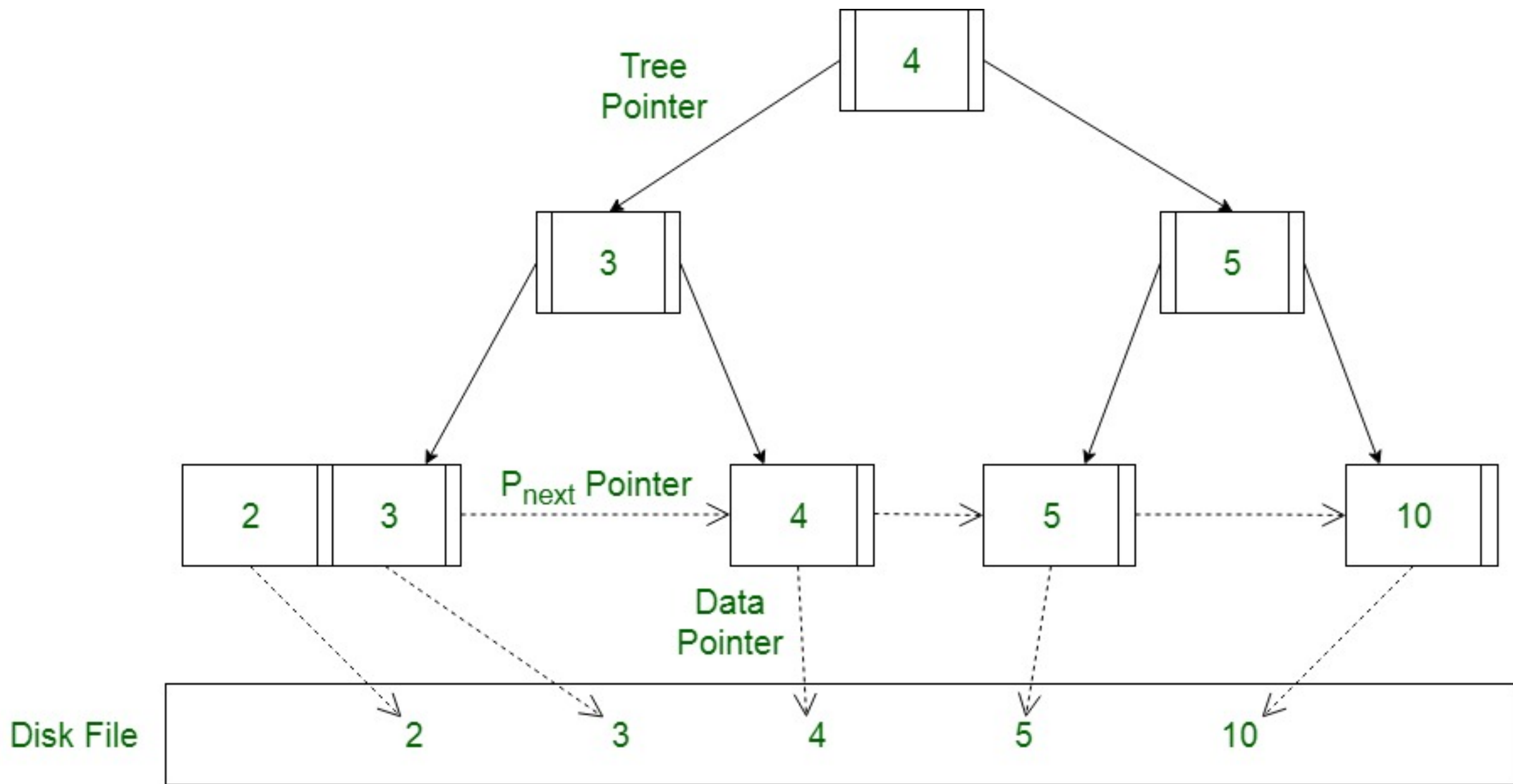
B+ Trees:

B + tree is a variation of B-tree data structure. In a B + tree, data pointers are stored only at the leaf nodes of the tree. In a B+ tree structure of a leaf node differs from the structure of internal nodes.

In a B+ tree structure of a leaf node differs from the structure of internal nodes.

The leaf nodes have an entry for every value of the search field, along with a data pointer to the record (or to the block that contains this record).

The leaf nodes of the B+ tree are linked together to provide ordered access on the search field to the records.



Consider a B+-tree in which the maximum number of keys in a node is 5. What is the minimum number of keys in any non-root node?

A

1



2

C

3

D

4



Transaction and Concurrency Control

Roll no: 106119020

Transaction

- A set of logically related operations
- The main operations of a transaction are:

Read(A): Read operations Read(A) or R(A) reads the value of A from the database and stores it in a buffer in the main memory.

Write (A): Write operation Write(A) or W(A) writes the value back to the database from the buffer

Properties of a Transaction

- **Atomicity:** As a transaction is a set of logically related operations, either all of them should be executed or none
- **Consistency:** If operations of debit and credit transactions on the same account are executed concurrently, it may leave the database in an inconsistent state
- **Isolation:** The result of a transaction should not be visible to others before the transaction is committed
- **Durability:** Once the database has committed a transaction, the changes made by the transaction should be permanent

Isolation levels

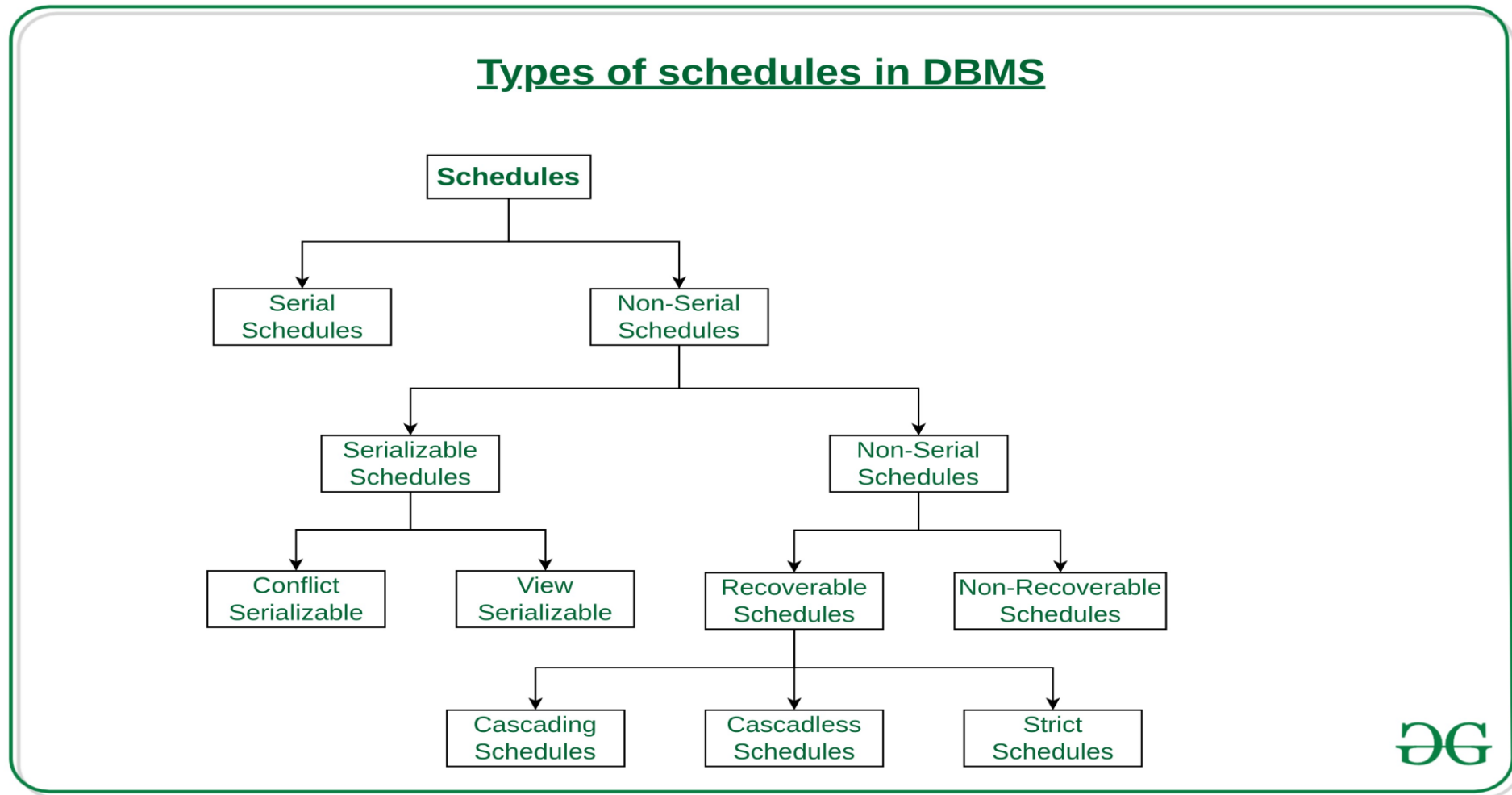
- **Isolation** determines the degree to which a transaction must be isolated from the data modifications made by any other transaction in the database system
- Transaction isolation levels is determined by the following phenomena:
 1. **Dirty Read** – A Dirty read is a situation when a transaction reads data that has not yet been committed.
 2. **Non Repeatable read** – Non Repeatable read occurs when a transaction reads the same row twice and gets a different value each time
 3. **Phantom Read** – Phantom Read occurs when two same queries are executed, but the rows retrieved by the two, are different

Isolation levels

- The 4 different isolation levels are
 1. **Read Uncommitted** - transaction can see uncommitted changes made by other transactions
 2. **Read Committed** - a transaction can only see changes made by other committed transactions
 3. **Repeatable Read** - guarantees that a transaction will see the same data throughout its duration, even if other transactions commit changes to the data
 4. **Serializable** - a transaction is executed as if it were the only transaction in the system

Isolation Level	Dirty reads	Non-repeatable reads	Phantoms
Read Uncommitted	May occur	May occur	May occur
Read Committed	Don't occur	May occur	May occur
Repeatable Read	Don't occur	Don't occur	May occur
Serializable	Don't occur	Don't occur	Don't occur

Types of Schedules in DBMS



Serializable schedule

- **Conflict Serializable:** A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.
- **Conflicting operations:** Two operations are said to be conflicting if all conditions satisfy:
 - They belong to different transactions
 - They operate on the same data item
 - At Least one of them is a write operation

Non-serial schedule

T1	T2
Read(A) Write(A)	Read(A) Write(A)
Read(B) Write(B)	
	Read(B) Write(B)

Schedule S1

Serial Schedule

T1	T2
Read(A) Write(A) Read(B) Write(B)	Read(A) Write(A)
	Read(B) Write(B)

Schedule S2

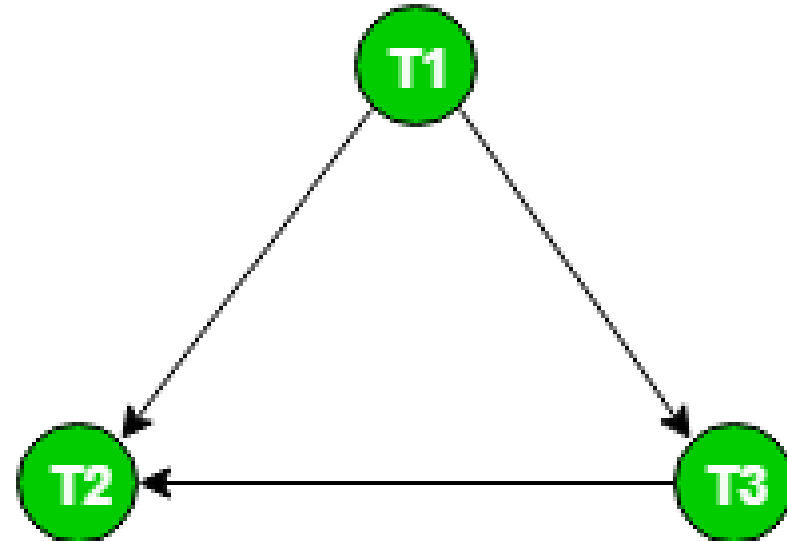
Serializable schedule

Precedency graph:

Check conflict pairs in other transactions and draw the edges. The different conflict pairs are:

1. Read-Write
2. Write-Read
3. Write-Write

T1	T2	T3
R(X)		
		R(Y)
W(X)		
	W(Y)	
		R(X)
	W(X)	



Serializable schedule

- **View Serializable:**

- A Schedule is called **view serializable** if it is view-equivalent to a serial schedule
- Two schedules S1 and S2 are said to be view-equivalent if the below conditions are satisfied :
 - I. **Initial Read:** If a transaction T1 reads data item A from the database in S1 then in S2 also T1 should read A from database
 - II. **Updated Read:** If Ti is reading A which is updated by Tj in S1 then in S2 also Ti should read A which is updated by Tj
 - III. **Final Write operation:** If a transaction T1 updated A at last in S1, then in S2 also T1 should perform final write operations

Serializable schedule

- **View Serializable:**

Example:

The below schedules are not conflict serializable but they are view serializable

S1

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

S2

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

The result of both the schedules will be the same.

Non-Serializable schedule

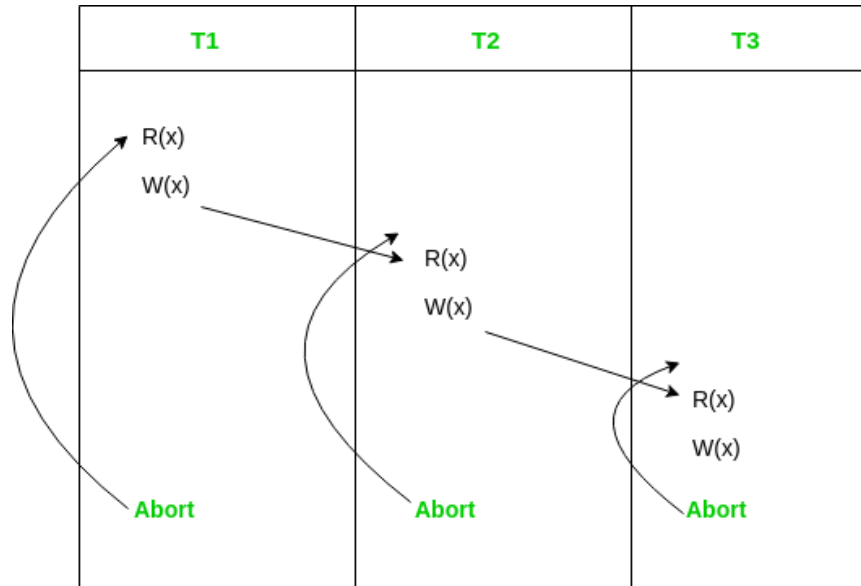
- **Recoverable Schedule:**

If some transaction T_j is reading value updated or written by some other transaction T_i , then the commit of T_j must occur after the commit of T_i

- 3 types of recoverable schedule

1. **Cascading Schedule:**

Also called Avoids cascading aborts/rollbacks (ACA). When there is a failure in one transaction and this leads to the rolling back or aborting other dependent transactions, then such scheduling is referred to as Cascading rollback or cascading abort

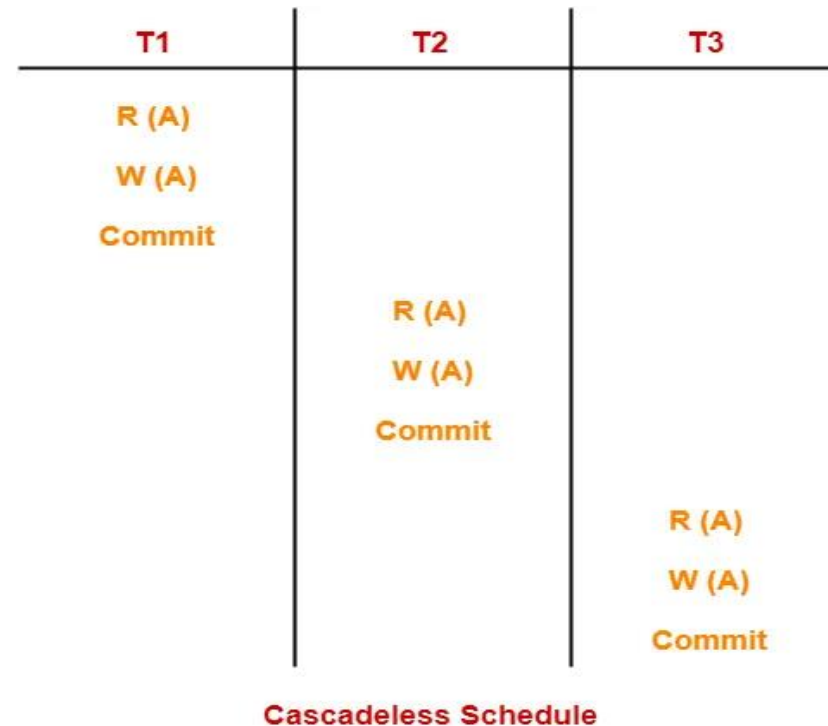


Non-Serializable schedule

2. Cascadeless Schedule:

If in a schedule, a transaction is not allowed to read a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a Cascadeless Schedule.

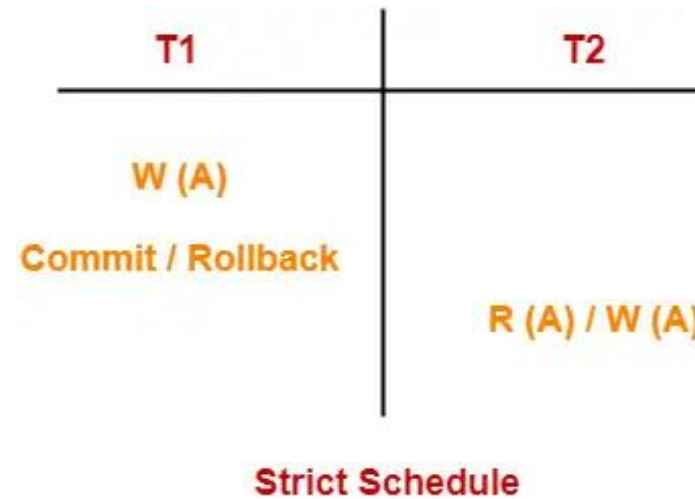
Therefore, it avoids cascading roll back and thus saves CPU time



Non-Serializable schedule

2. Strict Schedule:

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



Non-Serializable schedule

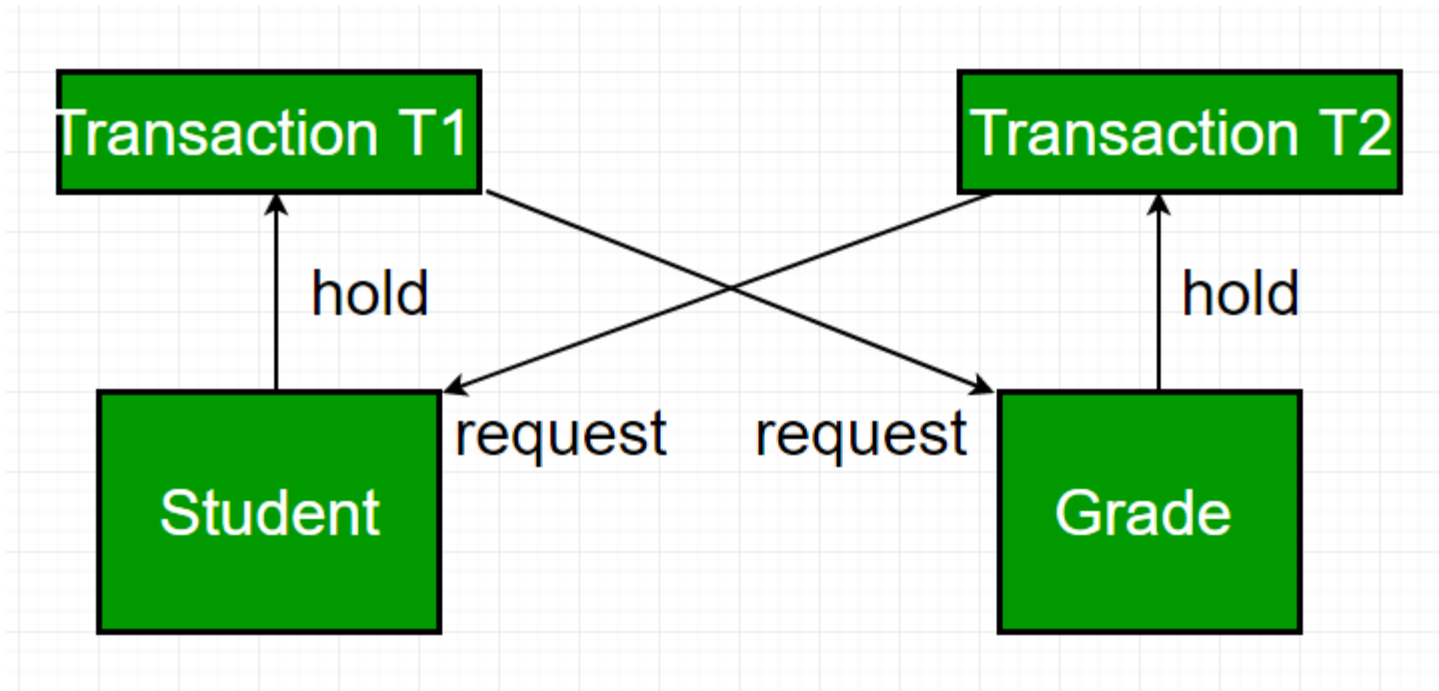
- **Non recoverable schedule**

If some transaction T_j is reading value updated or written by some other transaction T_i and the commit operation of T_i doesn't occur before the commit operation of T_j , the schedule is non recoverable.

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

Deadlocks

- In a database, a deadlock is an unwanted situation in which two or more transactions are waiting indefinitely for one another to give up locks



Deadlock prevention mechanism

- **Wait-Die scheme:**

In this scheme, if a transaction requests to lock a resource (data item), which is already held with a conflicting lock by another transaction, then one of the two possibilities may occur –

- If $TS(T_i) < TS(T_j)$ – that is T_i , which is requesting a conflicting lock, is older than T_j – then T_i is allowed to wait until the data-item is available.
- If $TS(T_i) > TS(t_j)$ – that is T_i is younger than T_j – then T_i dies. T_i is restarted later with a random delay but with the same timestamp.

Deadlock prevention mechanism

- **Wound-Wait scheme:**

In this scheme, if a transaction requests to lock a resource (data item), which is already held with conflicting lock by some another transaction, one of the two possibilities may occur –

- If $TS(T_i) < TS(T_j)$, then T_i forces T_j to be rolled back – that is T_i wounds T_j . T_j is restarted later with a random delay but with the same timestamp.
- If $TS(T_i) > TS(T_j)$, then T_i is forced to wait until the resource is available

Concurrency control protocols

- allow concurrent schedules, but ensure that the schedules are conflict/view serializable, and are recoverable and maybe even cascadeless
- The different types of concurrency control protocols are:
 1. Lock Based Protocol
 2. Graph Based Protocol
 3. Time-Stamp Ordering Protocol
 4. Multiple Granularity Protocol
 5. Multi Version Protocol

Lock Based Protocol

- Shared Lock(S)
it can be shared between transactions because while holding this lock the transaction does not have the permission to update data on the data item
- Exclusive Lock(X)
Data item can be both read as well as written

Compatibility Matrix

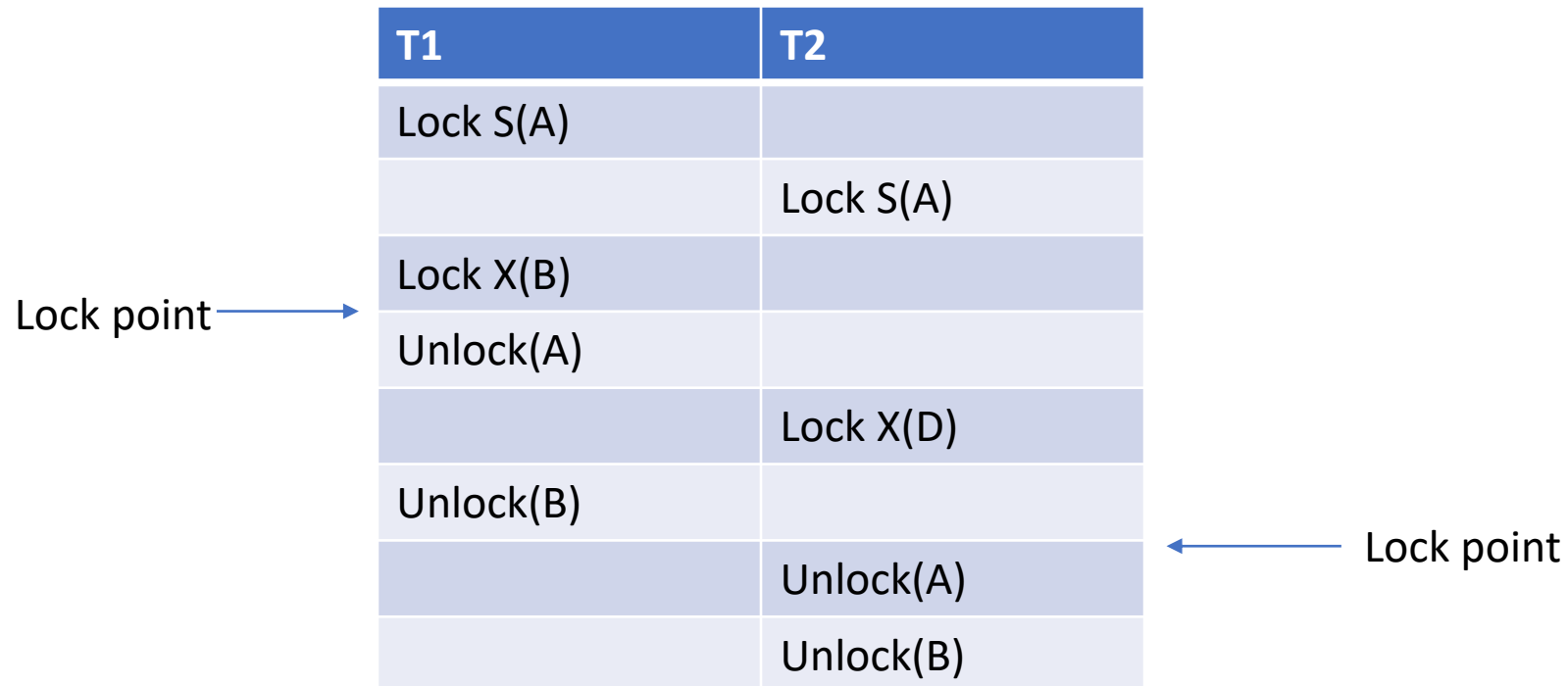
Already present on item	Currently requested for item	
	S	X
	S	X
S	Yes	No
X	No	No

Disadvantages:

- May not be sufficient to ensure serializability
- May not be free from irrecoverability
- May not be free from deadlock
- May not be free from starvation

2 Phase Locking

- Growing Phase: Locks are acquired and no locks are released
- Shrinking Phase: Locks are released and no locks are acquired
- Each Transaction has a Growing and a Shrinking Phase
- This Locking technique always ensures Serializability, but isn't free from deadlock

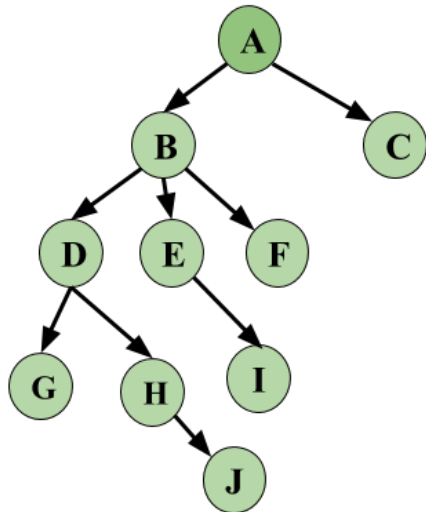


Graph Based Locking

- **Prerequisite:** we know the order to access a Database Item
- The protocol following the implementation of Partial Ordering is stated as-
 - If $d_i \rightarrow d_j$ then any transaction accessing both d_i and d_j must access d_i before accessing d_j .
 - Implies that the set **D** may now be viewed as a directed acyclic graph (DAG), called a *database graph*
 - Only exclusive locks are allowed
 - The first lock by T_i may be on any data item. Subsequently, a data Q can be locked by T_i only if the parent of Q is currently locked by T_i
 - Data items can be unlocked at any time

Graph Based Locking

Database graph



Schedule

T1	T2
Lock-X(B)	
	Lock-X(D)
	Lock-X(H)
	Unlock(D)
Lock-X(E)	

Advantages:

- Ensures Conflict Serializable Schedule.
- Ensures Deadlock Free Schedule
- Unlocking can be done anytime

Question 1

1.

```
T1: read (P) ;  
    read (Q) ;  
    if P = 0 then Q := Q + 1 ;  
    write (Q) ;
```

```
T2: read (Q) ;  
    read (P) ;  
    if Q = 0 then P := P + 1 ;  
    write (P) ;
```

Any non-serial interleaving of T1 and T2 for concurrent execution leads to

- (A) A serializable schedule
- (B) A schedule that is not conflict serializable
- (C) A conflict serializable schedule
- (D) A schedule for which a precedence graph cannot be drawn

Question 1

1. Answer: (B) A schedule that is not conflict serializable

If we want to interleave S1 and S2 so that S2 follows S1, the last operation of S1 should be after the first operation of S2(atleast).

But if the last operation of S1 and first operation of S2 are conflicting operations. Therefore, this will result in a schedule that is not Serializable.

Question 2

2. Match the following:

P. Recoverable	1. T_j reads data items written by T_i , the T_j commits after T_i commits
Q. Cascadeless	2. Reading uncommitted data
R. Dirty read	3. T_j reads data items written by T_i , the T_i commits after T_j commits
S. Non recoverable	4. T_j reads data items written by T_i , the commit operation of T_i appears before the read operation of T_j

- I. P-2, Q-1, R-4, S-3
- II. P-2, Q-3, R-4, S-1
- III. P-3, Q-4, R-2, S-1
- IV. P-1, Q-4, R-2, S-3