UNIT V:

Transactions: Transaction Concept, Transaction States, Implementation of Atomicity and Durability, Concurrent Executions, Serializability, Recoverability, Implementation of Isolation.

Concurrency Control: Lock-based protocols, Timestamp-based protocols.

Overview of Storage and Indexing: Data on External Storage, File Organization and Indexing, Index data Structures

Tree Structured Indexing: Indexed Sequential Access Methods (ISAM), B+ Trees: A Dynamic index Structure

……

1. Transaction Concept

Transaction is a set of operations which all are logically related. (OR)

A transaction is a single logical unit of work which accesses and possibly modifies the contents of a database.

Operations in transaction

Transactions access data using read and write operations.

a. Read Operation:

• Read operation reads the data from the database and then stores it in the buffer in main memory.

• For example- **Read(X)** instruction will read the value of X from the database and will store it in the buffer in main memory.

• Steps are:

➢ Find the block that contain data item X.

➢ Copy the block to a buffer in the main memory.

➢ Copy item X from the buffer to the program variable named X.

b. Write Operation:

• Write operation writes the updated data value back to the database from the buffer.

• For example- **Write(X)** will write the updated value of X from the buffer to the database.

• Steps are:

➢ Find address of block which contains data item X.

➢ Copy disk block into a buffer in the main memory

➢ Update data item X in main memory buffer.

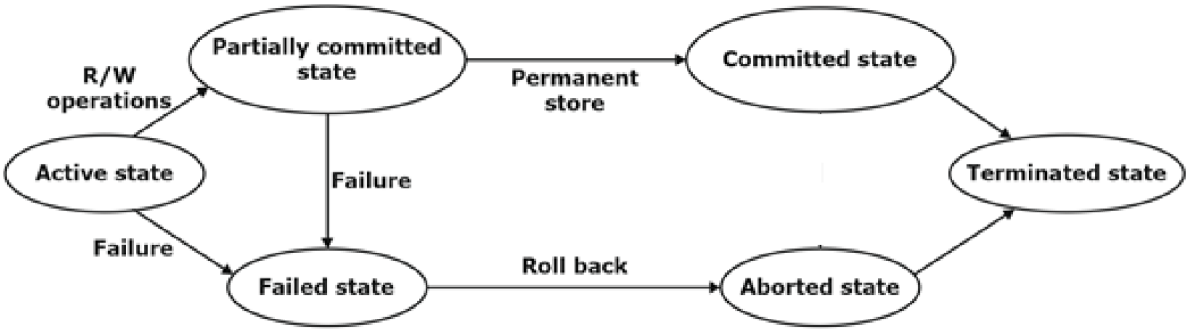
➢ Store the updated block from the buffer back to disk.

Example: Consider a transaction to transfer $50 from account A to account B:

1. **read**(*A*)
2. *A* := *A* − 50
3. **write**(*A*)
4. **read**(*B*)
5. *B* := *B* + 50
6. **write**(*B*)

2. Transaction States

States through which a transaction goes during its lifetime. These are the states which tell about the current state of the Transaction and also tell how we will further do the processing in the transactions. These states govern the rules which decide the fate of the transaction whether it will commit or abort.



Transaction States in DBMS

a) Active state:

- A transaction is said to be in active state as long as its instructions are being executed.
- All the changes made by transaction are now stored on the buffer of main memory.
- If all the 'read and write' operations are performed without any error then it goes to the "partially committed state"; if any instruction fails, it goes to the "failed state".

b) Partially committed state

- When the transaction executed its lost instruction it enters into the partially committed state.
- It is not called fully committed state because changes are stored in main memory.

c) Committed state:

- After all the changes are saved permanently into the database then it goes into the committed state.
- This state is called as fully committed state.

Note:-

- After the transaction has been entered into the committed state, it is not possible to do rollback (undo), because the database has come into a new consistent state.
- The only way to undo the changes is by carrying out a new transaction to reverse the operations of this new transaction is called compensating transactions.

d) Failed state:

- When a transaction executing in active state or partially committed state and there is a failure and it is not possible to go ahead with transaction, then transaction enters into the failed state.
- Now transaction will be roll backed (undo all changes)

e) Abort state:

- When the transaction has roll backed completely then it enters into aborted state.

f) Terminated state

- This is the last state of the life cycle of the transaction.
- After entering into committed state/abort state, transaction enters the terminate state finally.

Additional operations

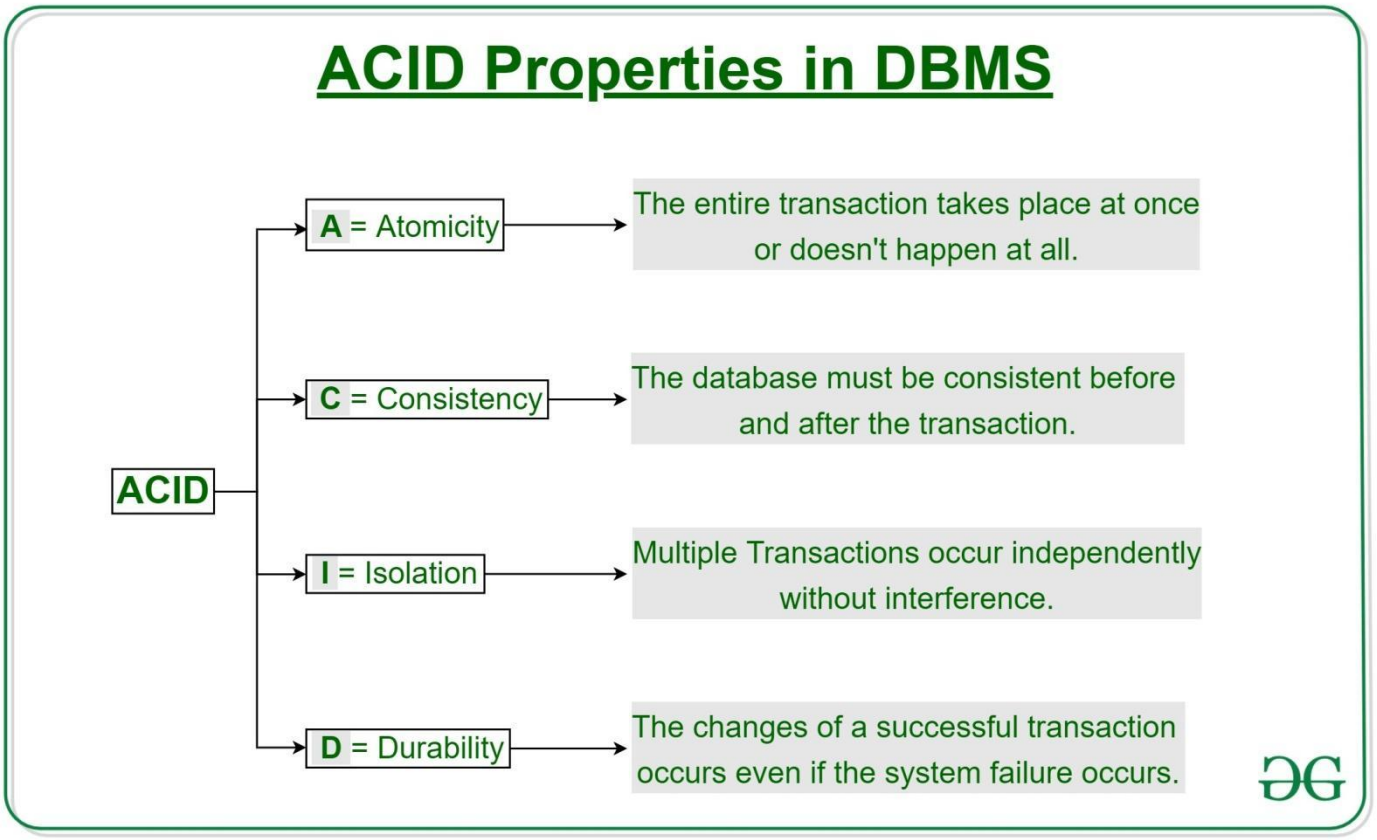
In addition to the preceding operations, some recovery techniques require additional operations that include the following

• UNDO: Similar to rollback, except that it applies to a single operation rather than to a whole transaction.
• REDO: This specifies that certain transaction operations must be redone to ensure that all the operations of a committed transaction have been applied successfully to the database.

3. ACID properties

A transaction is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.

In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called ACID properties.



Atomicity

- This property ensures either transaction will occur completely or it does not occur all i.e, no transaction occur partially
- It is also called Null or nothing rule.
- It is responsibility of transaction control manager to ensure atomicity.

Consistency

- Execution of a transaction in isolation (that is with no other transactions executing concurrently) preserves the consistency of the database.
- It is the responsibility of DBMS and application programmer to ensure consistency.

4. Implementation of Atomicity and Durability

The recovery-management component of a database system can support atomicity and durability by a variety of schemes.

Here we are going to learn about one of the simplest scheme called Shadow copy.

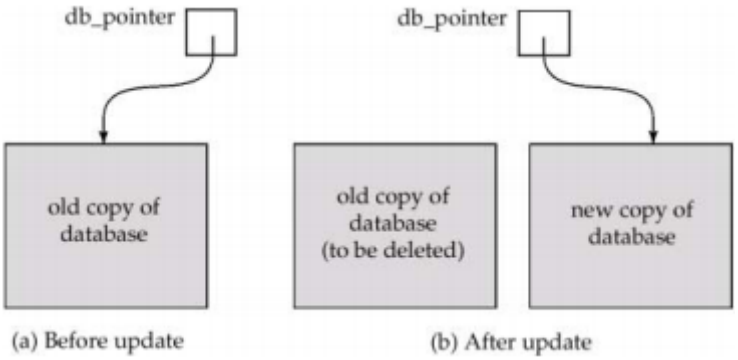
Shadow copy:

- In the shadow-copy scheme, a transaction that wants to update the database first creates a complete copy of the database. All updates are done on the new database copy, leaving the original copy, the shadow copy, untouched. If at any point the transaction has to be aborted, the system merely deletes the new copy. The old copy of the database has not been affected.
- This scheme is based on making copies of the database, called shadow copies, assumes that only one transaction is active at a time. The scheme also assumes that the database is simply a file on disk. A pointer called db-pointer is maintained on disk; it points to the current copy of the database.

If the transaction completes, it is committed as follows:

- First, the operating system is asked to make sure that all pages of the new copy of the database have been written out to disk. (Unix systems use the flush command for this purpose.)
- After the operating system has written all the pages to disk, the database system updates the pointer db-pointer to point to the new copy of the database; the new copy then becomes the current copy of the database. The old copy of the database is then deleted.

Figure below depicts the scheme, showing the database state before and after the update.



The transaction is said to have been committed at the point where the updated db pointer is written to disk.

How the technique handles transaction failures:

- If the transaction fails at any time before db-pointer is updated, the old contents of the database are not affected.
- We can abort the transaction by just deleting the new copy of the database.
- Once the transaction has been committed, all the updates that it performed are in the database pointed to by db pointer.
- Thus, either all updates of the transaction are reflected, or none of the effects are reflected, regardless of transaction failure.

How the technique handles system failures:

- Suppose that the system fails at any time before the updated db-pointer is written to disk. Then, when the system restarts, it will read db-pointer and will thus see the original contents of the database, and none of the effects of the transaction will be visible on the database.

- Next, suppose that the system fails after db-pointer has been updated on disk. Before the pointer is updated, all updated pages of the new copy of the database were written to disk.
- Again, we assume that, once a file is written to disk, its contents will not be damaged even if there is a system failure. Therefore, when the system restarts, it will read db-pointer and will thus see the contents of the database after all the updates performed by the transaction.

Note Points:

The implementation actually depends on the write to db-pointer being atomic; that is, either all its bytes are written or none of its bytes are written.

If some of the bytes of the pointer were updated by the write, but others were not, the pointer is meaningless, and neither old nor new versions of the database may be found when the system restarts.

Luckily, disk systems provide atomic updates to entire blocks, or at least to a disk sector.

In other words, the disk system guarantees that it will update db-pointer atomically, as long as we make sure that db-pointer lies entirely in a single sector, which we can ensure by storing db-pointer at the beginning of a block.

Thus, the atomicity and durability properties of transactions are ensured by the shadow-copy implementation of the recovery-management component.

5. Concurrent Executions

- In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database. It means that the same database is executed simultaneously on a multi-user system by different users.
- While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.
- The thing is that the simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database. Thus, on making the concurrent execution of the transaction operations, there occur several challenging problems that need to be solved.

Problems with Concurrent Execution

In a database transaction, the two main operations are **READ** and **WRITE** operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent. So, the following problems occur with the Concurrent Execution of the operations:

Problem 1: Lost Update Problems (W - W Conflict)

- The problem occurs when multiple transactions execute concurrently and update (changes) from one transaction gets lost.
- It is also called as " Write-Write" conflict

For example:

Consider the below diagram where two transactions T_X and T_Y, are performed on the same account A where the balance of account A is $300.

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A - 50	
t_3	—	READ (A)
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A)	—
t_7		WRITE (A)

LOST UPDATE PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300 (only read).
- At time t2, transaction T_X deducts $50 from account A that becomes $250 (only deducted and not updated/write).
- Alternately, at time t3, transaction T_Y reads the value of account A that will be $300 only because T_X didn't update the value yet.
- At time t4, transaction T_Y adds $100 to account A that becomes $400 (only added but not updated/write).
- At time t6, transaction T_X writes the value of account A that will be updated as $250 only, as T_Y didn't update the value yet.
- Similarly, at time t7, transaction T_Y writes the values of account A, so it will write as done at time t4 that will be $400. It means the value written by T_X is lost, i.e., $250 is lost.

Hence data becomes incorrect, and database sets to inconsistent.

Dirty Read Problems (W-R Conflict)

The dirty read problem occurs when "reading the data written by an uncommitted transaction". It create inconsistency in the database.

For example:

Consider two transactions T_X and T_Y in the below diagram performing read/write operations on account A where the available balance in account A is $300:

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

DIRTY READ PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300.
- At time t2, transaction T_X adds $50 to account A that becomes $350.
- At time t3, transaction T_X writes the updated value in account A, i.e., $350.
- Then at time t4, transaction T_Y reads account A that will be read as $350.
- Then at time t5, transaction T_X rollbacks due to server problem and the value changes back to $300 (as initially).
- But the value for account A remains $350 for transaction T_Y as committed, which is the dirty read and therefore known as the Dirty Read Problem.

Unrepeatable Read Problem (W-R Conflict)

Also known as Inconsistent Retrievals Problem that occurs when in a transaction, two different values are read for the same database item.

For example:

Consider two transactions, T_X and T_Y, performing the read/write operations on account A, having an available balance = $300. The diagram is shown below:

Time	T_x	T_y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	A = A + 100
t_4	—	WRITE (A)
t_5	READ (A)	—

UNREPEATABLE READ PROBLEM

- At time t1, transaction T_X reads the value from account A, i.e., $300.
- At time t2, transaction T_Y reads the value from account A, i.e., $300.
- At time t3, transaction T_Y updates the value of account A by adding $100 to the available balance, and then it becomes $400.
- At time t4, transaction T_Y writes the updated value, i.e., $400.
- After that, at time t5, transaction T_X reads the available value of account A, and that will be read as $400.
- It means that within the same transaction T_X, it reads two different values of account A, i.e., $ 300 initially, and after updation made by transaction T_Y, it reads $400. It is an unrepeatable read and is therefore known as the Unrepeatable read problem.

Thus, in order to maintain consistency in the database and avoid such problems that take place in concurrent execution, management is needed, and that is where the concept of Concurrency Control comes into role.

Phantom read problem

This problem occurs when a transaction reads a variable once but when it tries to read the same variable again, an error occurs saying variable does not exist.

Example:

T1	T2
Read(X)	
	Read(X)
Delete(X)	
	Read(X)

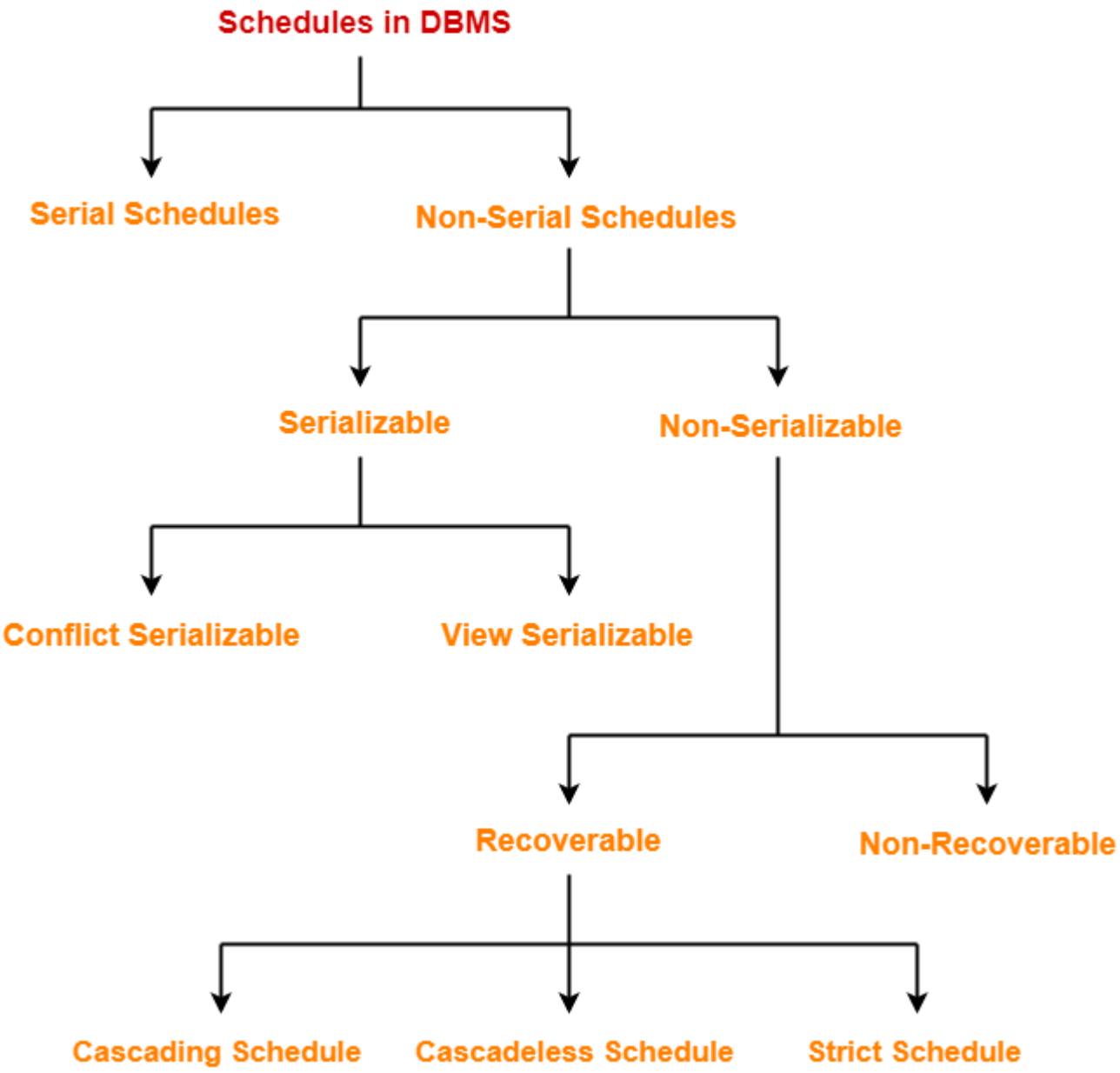
In the above example, once transaction 2 reads the variable X, transaction 1 deletes the variable X without transaction 2's knowledge. Thus, when transaction 2 tries to read X, it is not able to do it.

Incorrect Summary Problem:

Consider a situation, where one transaction is applying the aggregate function on some records while another transaction is updating these records. The aggregate function may calculate some values before the values have been updated and others after they are updated.

6. Serializability

- A schedule is the order in which the operations of multiple transactions appear for execution.
- Serial schedules are always consistent.
- Non-serial schedules are not always consistent.



Serial schedules

- All transactions are executed serially one after the other in serial schedule.
- When one transaction is being executed then no other transactions will be allowed to execute.
- Serial schedules are always consistent, recoverable, cascade less, and strict

Non-serial schedules

- Multiple transactions executed concurrently.
- Operations of all transaction are interleaved (or) mixed with each other.
- Non-Serial schedules are not always consistent, recoverable, cascade less, and strict

Finding Number of Schedules-

Consider there are n numbers of transactions T1, T2, T3 …. , Tn with N1, N2, N3 …. , Nn number of operations respectively.

Total Number of Schedules-

Total number of possible schedules (serial + non-serial) is given by-

$$\frac{(N1 + N2 + N3 + ... + Nn)!}{N1! \times N2! \times N3! \times .... \times Nn!}$$

Total Number of Serial Schedules-

Total number of serial schedules

= Number of different ways of arranging n transactions

= n!

Total Number of Non-Serial Schedules-

Total number of non-serial schedules

= Total number of schedules – Total number of serial schedules

Problem-

Consider there are three transactions with 2, 3, 4 operations respectively, find-

1. How many total numbers of schedules are possible?
2. How many total numbers of serial schedules are possible?
3. How many total numbers of non-serial schedules are possible?

Solution-

Total Number of Schedules-

Using the above formula, we have-

$$\text{Total number of schedules} = \frac{(2 + 3 + 4)!}{2! \times 3! \times 4!}$$

$$= 1260$$

Total Number of Serial Schedules-

Total number of serial schedules

= Number of different ways of arranging 3 transactions

= 3!

= 6

Total Number of Non-Serial Schedules-

Total number of non-serial schedules

= Total number of schedules – Total number of serial schedules

= 1260 – 6

= 1254

Types of serializability

Serializability is mainly of two types 1) conflict serializable schedule 2) view serializable schedule

1) Conflict serializable schedule

If a non-serial schedule can be converted into the serial schedule by swapping the positions of non-conflicting operations then this schedule is known as **conflict serializable schedule**

Conflicting operations

If multiple transactions are working on the same data item and one of the operation is write so these are called as conflicting operations .Conflict pairs for the same data item are:

- Read-Write
- Write-Write
- Write-Read

Checking Whether a Schedule is Conflict Serializable or Not

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

Step-01:

Find and list all the conflicting operations.

Step-02:

Start creating a precedence graph by drawing one node for each transaction.

Step-03:

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

Step-04:

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

Example:

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

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

Solution-

Step-01:

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

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

T1	T2	T3

$R_1(A)$

$R_2(A)$

$R_1(B)$

$R_2(B)$

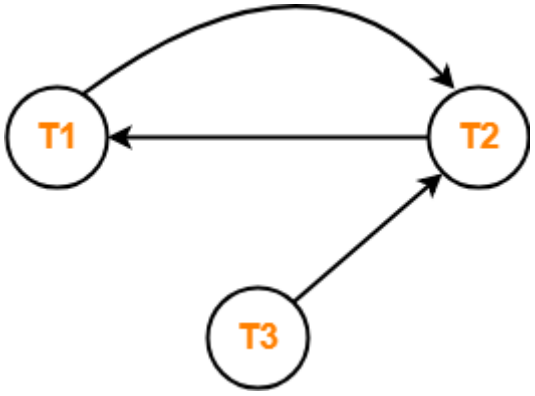
$R_3(B)$

$W_1(A)$

$W_2(B)$

Steps-2,3,4:

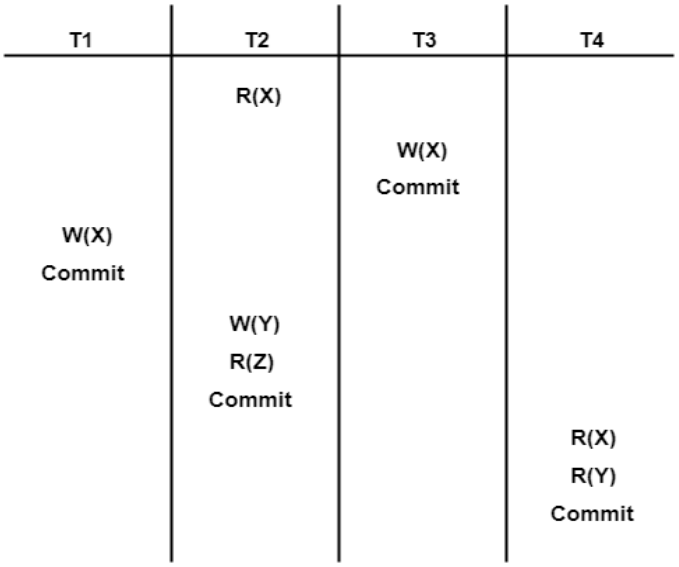
Draw the precedence graph-



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

Problem-02:

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



Solution-

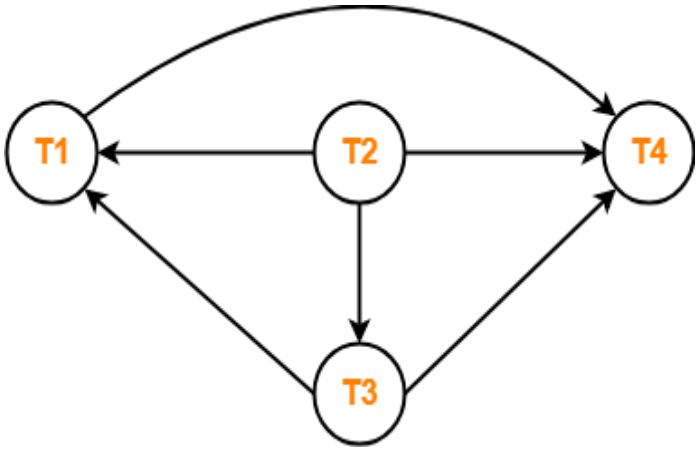
Step-01:

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

- $R_2(X)$, $W_3(X)$ $(T_2 \rightarrow T_3)$
- $R_2(X)$, $W_1(X)$ $(T_2 \rightarrow T_1)$
- $W_3(X)$, $W_1(X)$ $(T_3 \rightarrow T_1)$
- $W_3(X)$, $R_4(X)$ $(T_3 \rightarrow T_4)$
- $W_1(X)$, $R_4(X)$ $(T_1 \rightarrow T_4)$
- $W_2(Y)$, $R_4(Y)$ $(T_2 \rightarrow T_4)$

Steps-2,3,4:

Draw the precedence graph-



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

2) View serializability

If a given schedule is found to be view equivalent to some serial schedule, then it is called as a view serializable schedule.

View Equivalent Schedules-

Consider two schedules S1 and S2 each consisting of two transactions T1 and T2.

Schedules S1 and S2 are called view equivalent if the following three conditions hold true for them-

Initial reads should be same for all data items in the schedule

Write-Read sequence must be same for all data items

Final writers must e same for all data items in both schedules

Checking Whether a Schedule is View Serializable Or Not-

Method-01:

Check whether the given schedule is conflict serializable or not.

- If the given schedule is conflict serializable, then it is surely view serializable. Stop and report your answer.
- If the given schedule is not conflict serializable, then it may or may not be view serializable. Go and check using other methods.

Thumb Rules

- All conflict serializable schedules are view serializable.
- All view serializable schedules may or may not be conflict serializable.

Method-02:

Check if there exists any blind write operation.

(Writing without reading is called as a blind write).

- If there does not exist any blind write, then the schedule is surely not view serializable. Stop and report your answer.
- If there exists any blind write, then the schedule may or may not be view serializable. Go and check using other methods.

Thumb Rule

No blind write means not a view serializable schedule.

Method-03:

In this method, try finding a view equivalent serial schedule.

- By using the above three conditions, write all the dependencies.
- Then, draw a graph using those dependencies.
- If there exists no cycle in the graph, then the schedule is view serializable otherwise not.

Example:

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-

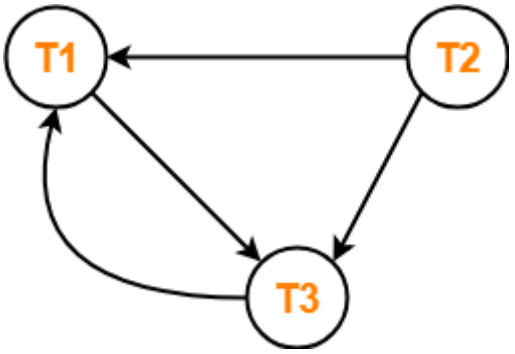
Step-01:

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.

Now,

- Since, the given schedule S is not conflict serializable, so, it may or may not be view serializable.
- To check whether S is view serializable or not, let us use another method.
- Let us check for blind writes.

Checking for Blind Writes-

- There exists a blind write W_3 (A) in the given schedule S.
- Therefore, the given schedule S may or may not be view serializable.

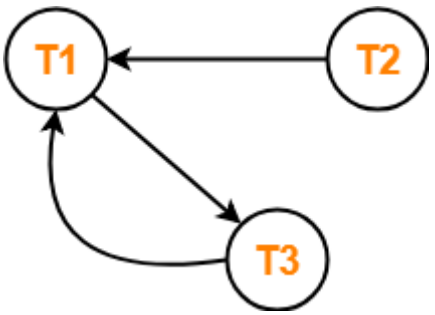
Now,

- To check whether S is view serializable or not, let us use another method.
- Let us derive the dependencies and then draw a dependency graph.

Drawing a Dependency Graph-

- T1 firstly reads A and T3 firstly updates A.
- So, T1 must execute before T3.
- Thus, we get the dependency **T1 → T3**.
- Final updation on A is made by the transaction T1.
- So, T1 must execute after all other transactions.
- Thus, we get the dependency **(T2, T3) → T1**.
- There exists no write-read sequence.

Now, let us draw a dependency graph using these dependencies-



- Clearly, there exists a cycle in the dependency graph.
- Thus, we conclude that the given schedule S is not view serializable.

7. Recoverability

Recovery brings the database from the temporary inconsistent state to a consistent state.

Database recovery can also be defined as mechanisms for restoring a database quickly and accurately after loss or damage.

Different Types of Database Failures

A wide variety of failures can occur in processing a database, ranging from the input of an incorrect data value or complete loss or destruction of the database. Some of the types of failures are listed below:

1. System crashes, resulting in loss of main memory

2. Media failures, resulting in loss of parts of secondary storage

3. Application software errors

4. Natural physical disasters

5. Carelessness or unintentional destruction of data or facilities

Main Recovery Techniques

Three main recovery techniques that are commonly employed are:

1. Deferred update

2. Immediate update

3. Shadow paging

1. Deferred update. / NO-UNDO/REDO

The idea behind deferred update is to defer or postpone any actual updates to the database itself until the transaction completes its execution successfully and reaches its commit point.

During transaction execution, the updates are recorded only in the log and in the transaction workspace. After the transaction reaches its commit point and the log is force-written to disk, the updates are recorded in the database itself.

If a transaction fails before reaching its commit point, there is no need to undo any operations, because the transaction has not affected the database in any way.

Example

Consider the following transaction TRAN1

Transaction TRAN1
read(A)
write(10,B)
write(20,C)
Commit

Using deferred update, the process is:

Time	Action	Log
t1	START	-
t2	read(A)	-
t3	write(10,B)	B = 10
t4	write(20,C)	C = 20
t5	COMMIT	COMMIT

	Before			After		
DISK			B = 6			B = 10
	A = 5	C = 2		A = 5	C = 20	

2. **Immediate update.** / UNDO/REDO

In the immediate update techniques, the database may be updated by the operations of a transaction immediately, before the transaction reaches its commit point. However, these operations are typically recorded in the log on disk by force-writing before they are applied to the database, so that recovery is possible.

When immediate update is allowed, provisions must be made for undoing the effect of update operations on the database, because a transaction can fail after it has applied some updates to the database itself. Hence recovery schemes based on immediate update must include the capability to roll back a transaction by undoing the effect of its write operations

Using immediate update and the transaction TRAN1 again, the process is:

Time	Action	LOG
t1	START	-
t2	read(A)	-
t3	write(10,B)	Was B == 6, now 10
t4	write(20,C)	Was C == 2, now 20
t5	COMMIT	COMMIT

	Before			During			Afte
DISK			B = 6			B = 10	
	A = 5	C = 2		A = 5	C = 2		A =

3. Shadow paging.

Shadow paging maintains two page tables during life of a transaction, current page and shadow page table. When transaction starts, two pages are the same. Shadow page table is never changed thereafter and is used to restore database in the event of failure. During transaction, current page table records all updates to database. When transaction completes, current page table becomes shadow page table.

ARIES Algorithm

ARIES is a recovery algorithm designed to work with a steal, no-force approach. It is more simple and flexible than other algorithms.

ARIES algorithm is used by the recovery manager which is invoked after a crash. Recovery manager will perform restart operations.

Main Key Terms Used in ARIES

Log. These are all file which contain several records. These records contain the information about the state of database at any time. These records are written by the DBMS while any changes done in the database. Normally copy of the log file placed in the different parts of the disk for safety.

LSN. The abbreviation of LSN is log sequence number. It is the ID given to each record in the log file. It will be in monotonically ascending order.

Page LSN. For recovery purpose, every page in the database contains the LSN of the most recent log record that describes a change to this page. This LSN is called the page LSN.

CLR. The abbreviation of CLR is compensation log record. It is written just before the change recorded in an update log record U is undone.

WAL. The abbreviation of WAL is write-ahead log. Before updating a page to disk, every update log record that describes a change to this page must be forced to stable storage. This is accomplished by forcing all log records up to and including the one with LSN equal to the page LSN to stable storage before writing the page to disk.

There are three phases in restarting process, they are:

1. Analysis

2. Redo

3. Undo

Analysis. In this phase, it will identify whether any page present in the buffer pool is not written into disk and activate the transactions which are in active at the time of crash.

Redo. In this phase, all the operations are restarted and the state of database at the time of crash is obtained. This is done by the help of log files.

Undo. In this phase, the actions of uncommitted transactions are undone. So only committed are taken into account.

Log:

After a crash, we find the following log:

0	BEGIN CHECKPOINT
5	END CHECKPOINT (EMPTY XACT TABLE AND DPT)
10	T1: UPDATE P1 (OLD: YYY NEW: ZZZ)
15	T1: UPDATE P2 (OLD: WWW NEW: XXX)
20	T1: COMMIT

Analysis phase:

Scan forward through the log starting at LSN 0.
LSN 5: Initialize XACT table and DPT to empty.
LSN 10: Add (T1, LSN 10) to XACT table. Add (P1, LSN 10) to DPT.
LSN 15: Set LastLSN=15 for T1 in XACT table. Add (P2, LSN 15) to DPT.
LSN 20: Change T1 status to "Commit" in XACT table

Redo phase:

Scan forward through the log starting at LSN 10.
LSN 10: Read page P1, check PageLSN stored in the page. If PageLSN<10, redo LSN 10 (set value to ZZZ) and set the page's PageLSN=10.
LSN 15: Read page P2, check PageLSN stored in the page. If PageLSN<15, redo LSN 15 (set value to XXX) and set the page's PageLSN=15.

Undo phase:

Do nothing; no transactions to undo.

ARIES algorithm has three main principles:

1. Write-ahead logging

2. Repeating history during redo

3. Logging changes during undo

Write-ahead logging. This states that any change to a database object is first recorded in the log; the record in the log must be written to stable storage before the change to the database object is written to disk.

Repeating history during redo. On restart after a crash, ARIES retraces all actions of the DBMS before the crash and brings the system back to the exact state that it was in at the time of crash. Then, it undoes the actions of transactions still active at the time of crash.

Logging changes during undo. Changes made to the database while undoing a transaction are logged to ensure such an action is not repeated in the event of repeated restarts.

Elements of ARIES

1. The log
2. Tables
3. The write-ahead log protocol
4. Check pointing

8. Implementation of Isolation. / Implementation of Isolation Levels

As we know that, in order to maintain consistency in a database, it follows ACID properties. Among these four properties (Atomicity, Consistency, Isolation, and Durability) Isolation determines how transaction integrity is visible to other users and systems. It means that a transaction should take place in a system in such a way that it is the only transaction that is accessing the resources in a database system. Isolation levels define the degree to which a transaction must be isolated from the data modifications made by any other transaction in the database system. A transaction isolation level is defined by the following phenomena –

- **Dirty Read** – A Dirty read is a situation when a transaction reads data that has not yet been committed. For example, Let's say transaction 1 updates a row and leaves it uncommitted, meanwhile, Transaction 2 reads the updated row. If transaction 1 rolls back the change, transaction 2 will have read data that is considered never to have existed.
- **Non Repeatable read** – Non Repeatable read occurs when a transaction reads the same row

twice and gets a different value each time. For example, suppose transaction T1 reads data. Due to concurrency, another transaction T2 updates the same data and commit, Now if transaction T1 rereads the same data, it will retrieve a different value.

- **Phantom Read** – Phantom Read occurs when two same queries are executed, but the rows retrieved by the two, are different. For example, suppose transaction T1 retrieves a set of rows that satisfy some search criteria. Now, Transaction T2 generates some new rows that match the search criteria for transaction T1. If transaction T1 re-executes the statement that reads the rows, it gets a different set of rows this time.

Based on these phenomena, The SQL standard defines four isolation levels :

1. **Read Uncommitted** – Read Uncommitted is the lowest isolation level. In this level, one transaction may read not yet committed changes made by other transactions, thereby allowing dirty reads. At this level, transactions are not isolated from each other.
2. **Read Committed** – This isolation level guarantees that any data read is committed at the moment it is read. Thus it does not allow dirty read. The transaction holds a read or write lock on the current row, and thus prevents other transactions from reading, updating, or deleting it.
3. **Repeatable Read** – This is the most restrictive isolation level. The transaction holds read locks on all rows it references and writes locks on referenced rows for update and delete actions. Since other transactions cannot read, update or delete these rows, consequently it avoids non-repeatable read.
4. **Serializable** – This is the highest isolation level. A *serializable* execution is guaranteed to be serializable. Serializable execution is defined to be an execution of operations in which concurrently executing transactions appears to be serially executing.

The Table is given below clearly depicts the relationship between isolation levels, read phenomena, and locks :

Isolation Level	Dirty reads	Nor
Read Uncommitted	May occur	Ma
Read Committed	Don't occur	Ma
Repeatable Read	Don't occur	Dor
Serializable	Don't occur	Dor

Anomaly Serializable is not the same as Serializable. That is, it is necessary, but not sufficient that a Serializable schedule should be free of all three phenomena types.

Chapter-II

Concurrency Control: Lock-based protocols, Timestamp-based protocols.

……

Concurrency-control protocols: allow concurrent schedules, but ensure that the schedules are conflict/view serializable, and are recoverable and maybe even cascadeless.

These protocols do not examine the precedence graph as it is being created; instead a protocol imposes a discipline that avoids non-seralizable schedules.

Different concurrency control protocols provide different advantages between the amount of concurrency they allow and the amount of overhead that they impose.

Lock-Based Protocol

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

1. **Shared lock:**

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

2. **Exclusive lock:**

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

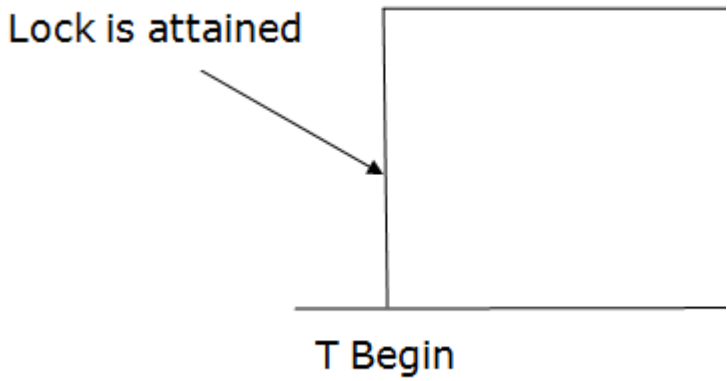
There are four types of lock protocols available:

1. **Simplistic lock protocol**

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

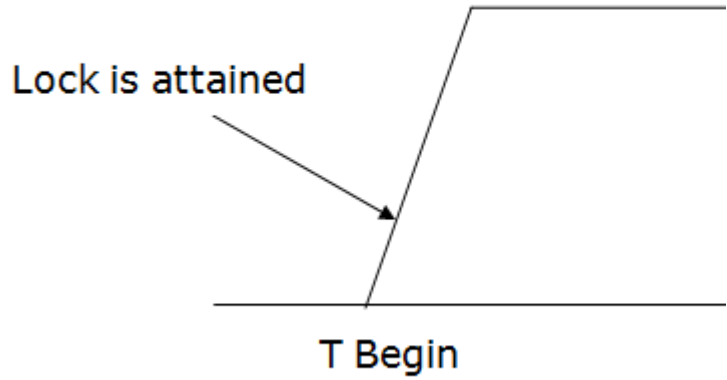
2. **Pre-claiming Lock Protocol**

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



3. **Two-phase locking (2PL)**

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



There are two phases of 2PL:

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

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

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

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

Example:

	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:

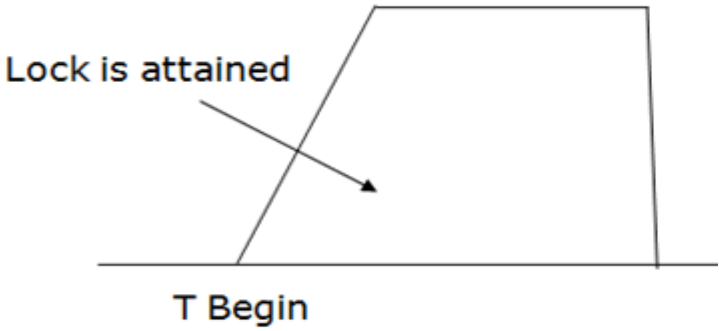
- **Growing phase:** from step 1-3
- **Shrinking phase:** from step 5-7
- **Lock point:** at 3

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.



It does not have cascading abort as 2PL does.

Timestamp Ordering Protocol

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

Basic Timestamp ordering protocol works as follows:

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

- If W_TS(X) >TS(Ti) then the operation is rejected.
- If W_TS(X) <= TS(Ti) then the operation is executed.
- Timestamps of all the data items are updated.

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

- If TS(Ti) < R_TS(X) then the operation is rejected.
- If TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

Where,

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

R_TS(X) denotes the Read time-stamp of data-item X.

W_TS(X) denotes the Write time-stamp of data-item X.

Advantages and Disadvantages of TO protocol:

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

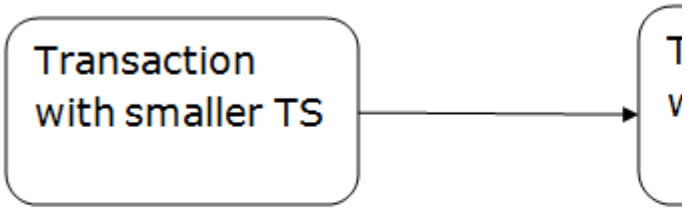


Image: Precedence Graph for TS

- 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.

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	

Figure: 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)	Commit
W(A)	
Commit	

Figure: A Conflict Serializable Schedule

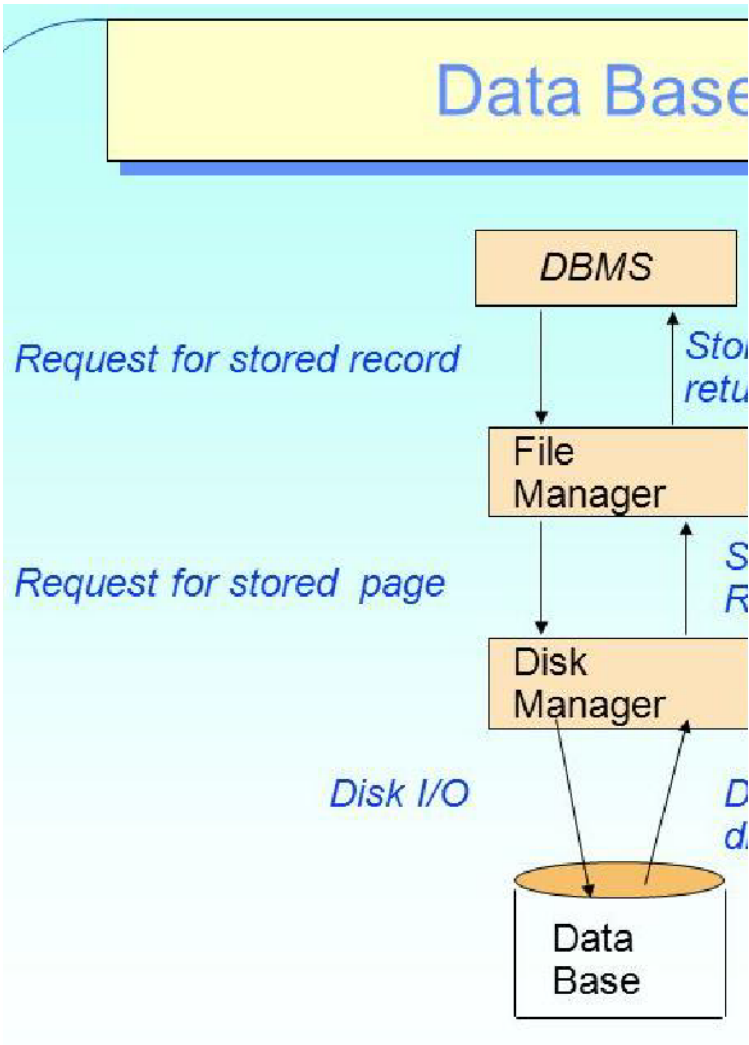
Chapter-III

Overview of Storage and Indexing: Data on External Storage, File Organization and Indexing, Index data Structures

……

Data on external storage:

- Data in a DBMS is stored on storage devices such as disks and tapes
- The disk space manager is responsible for keeping track of available disk space.
- The file manager, which provides the abstraction of a file of records to higher levels of DBMS code, issues requests to the disk space manager to obtain and relinquish space on disk.



File Organizations:

Storing the files in certain order is called file organization. The main objective of file organization is

- Optimal selection of records i.e.; records should be accessed as fast as possible.
- Any insert, update or delete transaction on records should be easy, quick and should not harm other records.
- No duplicate records should be induced as a result of insert, update or delete

- Records should be stored efficiently so that cost of storage is minimal.

Some of the file organizations are

1. Sequential File Organization

2. Heap File Organization

3. Hash/Direct File Organization

4. Indexed Sequential Access Method

5. B+ Tree File Organization

6. Cluster File Organization

Sequential File Organization:

In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key.

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:

1. Pile File Method

2. Sorted File

Pros and Cons of Sequential File Organization – Pros –

- Fast and efficient method for huge amount of data.
- Simple design.
- Files can be easily stored in magnetic tapes i.e cheaper storage mechanism.

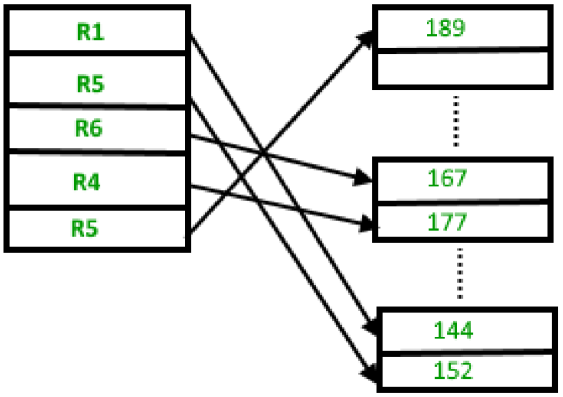
Cons –

- Time wastage as we cannot jump on a particular record that is required, but we have to move in a sequential manner which takes our time.
- Sorted file method is inefficient as it takes time and space for sorting records

*2. **Heap File Organization:***

When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details. File records can be placed anywhere in that memory area.

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.



Pros and Cons of Heap File Organization – Pros –

- Fetching and retrieving records is faster than sequential record but only in case of small databases.
- When there is a huge number of data needs to be loaded into the database at a time, then this method of file Organization is best suited.

Cons –

- Problem of unused memory blocks.
- Inefficient for larger databases.

Hash File Organization:

Hash File Organization uses Hash function computation on some fields of the records. The output of the hash function determines the location of disk block where the records are to be placed.

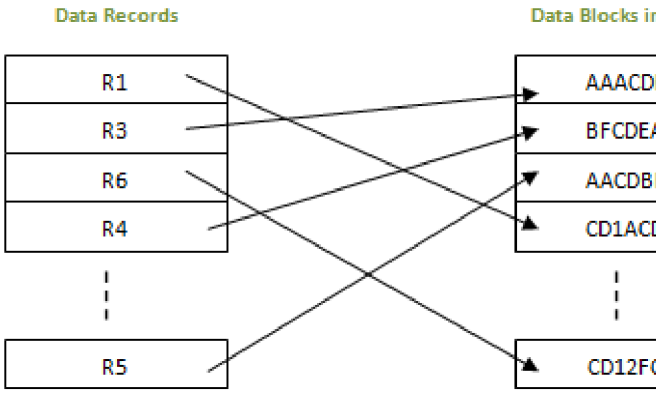
In this method of file organization, hash function is used to calculate the address of the block to store the records.

The hash function can be any simple or complex mathematical function.

The hash function is applied on some columns/attributes – either key or non-key columns to get the block address.

Hence each record is stored randomly irrespective of the order they come. Hence this method is also known as Direct or Random file organization.

If the hash function is generated on key column, then that column is called hash key, and if hash function is generated on non-key column, then the column is hash column.



When a record has to be retrieved, based on the hash key column, the address is generated and directly from that address whole record is retrieved. Here no effort to traverse through whole file. Similarly when a new record has to be inserted, the address is generated by hash key and record is directly inserted. Same is the case with update and delete

Clustered file organization:

Clustered file organization is not considered good for large databases. In this mechanism, related records from one or more relations are kept in the same disk block, that is, the ordering of records is not based on primary key or search key.

In this method two or more table which are frequently used to join and get the results are stored in the same file called clusters. These files will have two or more tables in the same data block and the key columns which map these tables are stored only once. This method hence reduces the cost of searching for various records in different files. All the records are found at one place and hence making search efficient.

STUDENT			
STUDENT_ID	STUDENT_NAME	ADDRESS	COURSE_ID
100	Kathy	Troy	230
101	Patricia	Clinton Township	240
102	James	Fraser Town	230
103	Antony	Novi	250
104	Charles	Novi	250

COURSE	
COURSE_ID	COURSE_NAME
230	Database
240	Java
250	Perl

Comparison of file organizations:

The operations to be considered for comparisons of file organizations are below

- **Scan:** Fetch all records in the file. T disk into the buffer pool. There is al the record on the page (in the pool).
- **Search with equality selection:** Fe tion, for example, "Find the Students that contain qualifying records must b must be located within retrieved page
- **Search with range selection:** Fetc for example, "Find all Students record
- **Insert:** Insert a given record into the into which the new record must be in to include the new record, and then w the file organization, we may have to f well.
- **Delete:** Delete a record that is spe page that contains the record, fetch Depending on the file organization, we other pages as well.

File Type	*Scan*	*Equality Search*	*Rang Sear*
Heap	BD	$0.5BD$	BD
Sorted	BD	$Dlog_2B$	$Dlog$ *matc*
Hashed	$1.25BD$	D	1.25

Figure 8.1 A Cor

Where

B - Number of data pages

R records per page

D- Average time to read or write a disk page

C- Average time to process a record

Indexing and indexed data structures

We know that data is stored in the form of records. Every record has a key field, which helps it to be recognized uniquely.

Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.

Indexing is defined based on its indexing attributes. Indexing can be of the following types –

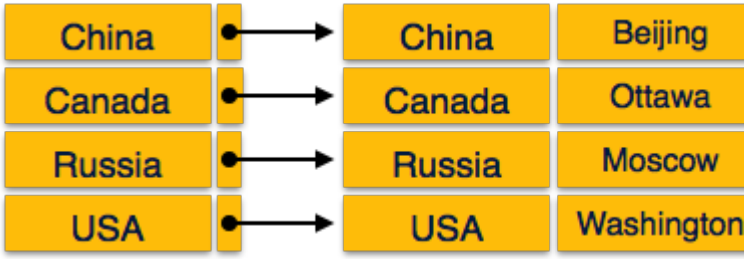
- **Primary Index** – Primary index is defined on an ordered data file. The data file is ordered on a **key field**. The key field is generally the primary key of the relation.
- **Secondary Index** – Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
- **Clustering Index** – Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

Ordered Indexing is of two types –

- Dense Index
- Sparse Index

Dense Index

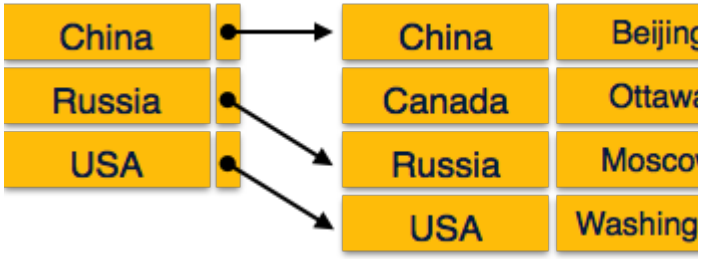
In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.



Sparse Index

In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search

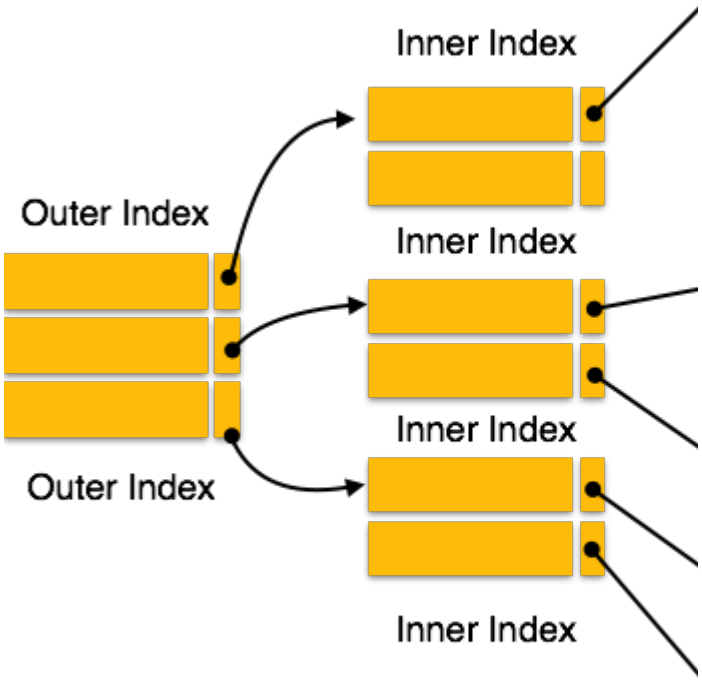
a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.



Multilevel Index

Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.

Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.



Chapter-III

Tree Structured Indexing: Indexed Sequential Access Methods (ISAM), B+ Trees: A Dynamic index Structure

……

1. Indexed Sequential Access Methods (ISAM)

ISAM stands for indexed sequential access method. ISAM is a static index structure that is effective when the file is not frequently updated. In ISAM new entries are inserted in overflow pages.

In ISAM data structure the number of leaf pages in a tree is fixed at the time of file creation. The figure below illustrates the ISAM structure.

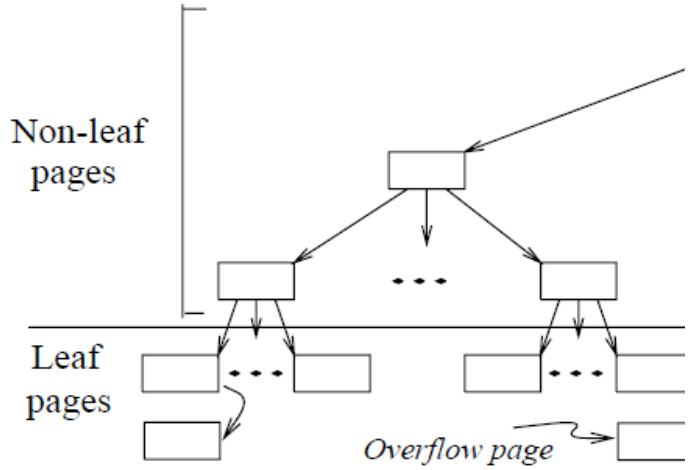


Figure 9.3 ISAI

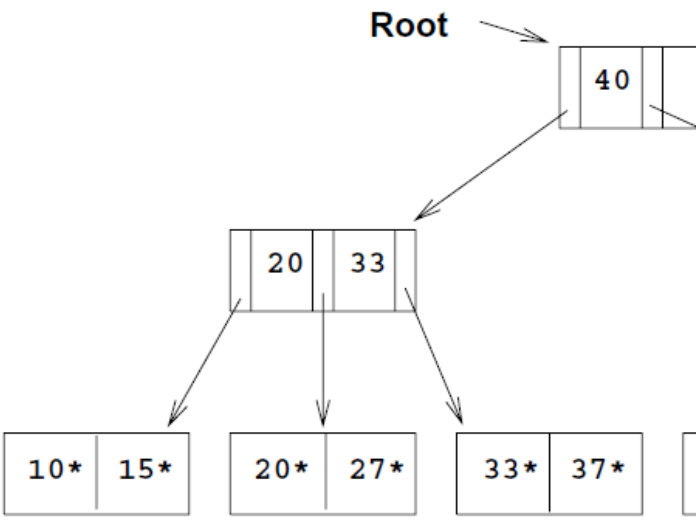


Figure 9.5 Samp

The ISAM index can be created in one of the following alternatives

1. A data entry K* in index is actually a data record with the search key 'k'

2. A data entry is a pair of (k,rid) where k is the search key value and rid is the id of a data record.

3. A data entry is a pair of (k, rid_list) where k is the search key value and rid_list is a list of data record ids.

The figure below shows page's allocation

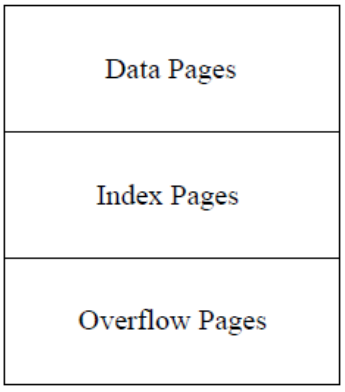


Figure 9.4 Page Allocation in ISAM

The basic operations supported by ISAM are insertion, deletion, and search.

Example:

Consider the tree shown in following figure to illustrate the ISAM index structure

1. Searching

For example, to locate a record with the key value 27, we start at the root and follow the left pointer, since 27 < 40. We then follow the middle pointer, since 20 <= 27 < 33. For a range search, we find the first qualifying data entry as for an equality selection and then retrieve primary leaf pages sequentially.

2. Insertion

If we now insert with key value 23, the entry 23* belongs in the second data page, which already contains

20* and 27* and has no more space. We deal with this situation by adding an *overflow* page and putting 23* in the overflow page. Chains of overflow pages can easily develop. For instance, inserting 48*, 41*, and 42* leads to an overflow chain of two pages.

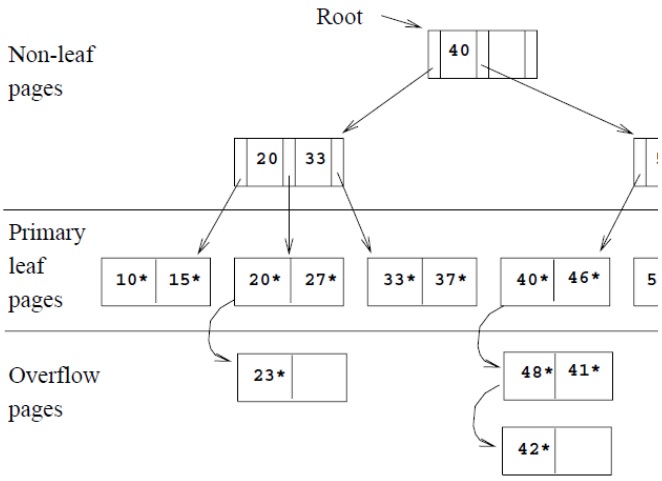


Figure 9.6 ISAM Tree after Inserts

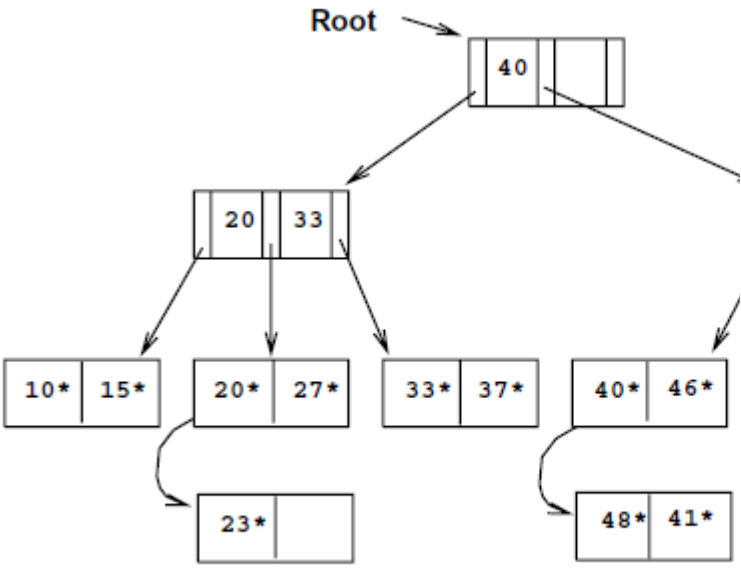


Figure 9.7 ISAM Tree after Dele

3. Deletion

The deletion of an entry *k_* is handled by simply removing the entry. If this entry is on an overflow page and the overflow page becomes empty, the page can be removed.

If the entry is on a primary page and deletion makes the primary page empty, the simplest approach is to simply leave the empty primary page as it is.

Below ISAM structure shows after deletion of the entries 42*, 51*, and 97*. Note that after deleting 51*, the key value 51 continues to appear in the index level. A subsequent search for 51* would go to the correct leaf page and determine that the entry is not in the tree.

Pros of ISAM:

- In this method, each record has the address of its data block, searching a record in a huge database is quick and easy.
- This method supports range retrieval and partial retrieval of records. Since the index is based on the primary key values, we can retrieve the data for the given range of value. In the same way, the partial value can also be easily searched, i.e., the student name starting with 'JA' can be easily searched.

Cons of ISAM

- This method requires extra space in the disk to store the index value.
- When the new records are inserted, then these files have to be reconstructed to maintain the sequence.
- When the record is deleted, then the space used by it needs to be released. Otherwise, the performance of the database will slow down.

2. B+ Trees: A Dynamic index Structure

• It is a balanced search tree or height restricted search tree.

• Maximum height of the tree should not exceed O(logn).

• A B+ tree's leaf nodes represent real data pointers.

• The B+ tree maintains equilibrium by keeping all leaf nodes at the same height.

• All the leaf nodes are connected via link, just like in the linked list.

Structure:

• The leaf nodes are at equal distance from the root.

• The structure consists of an internal and leaf node.

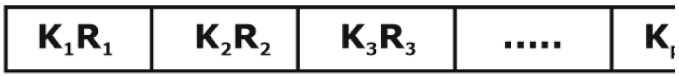
• Order P of a tree is the maximum block pointers that can be stored in a B+ Tree

Let, K is the key, B is the block pointer, and R is the record pointer.

Then, the node structure will be:

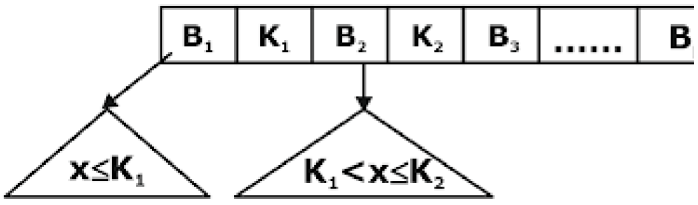
(a) Leaf Node:

It consists of a set of (Key, record pointer) pairs and one block pointer which points to next leaf node.



(b) Internal Node:

It contains only keys and Block pointers



Balancing Conditions:

• Every internal node except root node contains at least [p/2] block pointers & [p/2]-1 keys.

• Internal node can contain at most p pointers and (p – 1) keys.

• Root nodes can be with at least 2 block pointers and 1 key. And atmost p blocks pointer (p – 1) keys.

• Root nodes can be with at least 2 block pointer and 1 key and atmost p block pointer & (p – 1) keys.

• The leaf node can contain at least [p/2]-1 keys and atmost P-1 keys

Note: Here 'p' is the order of the tree.

B⁺ Tree Insertion

- B⁺ trees are filled from bottom and each entry is done at the leaf node.

<u>If insertion is at leaf node</u>

1. Insert the key/reference pair into the node if there is a space. If there is no space i.e overflows

2. Insertion of new node if overflow occurs split the now into two nodes

a) First node contain [m-1/2] keys

b) Second node contains remaining nodes

c) Copy the smallest element from the second node to parent node.

<u>If insertion is at non-leaf node (internal node)</u>

Split the node into two nodes

a) The first node contains [m/2]-1 keys

b) Move the smallest element from the remaining keys to parent node

c) Second node contains the remaining keys.

B⁺ Tree Deletion

- B⁺ tree entries are deleted at the leaf nodes.
- The target entry is searched and deleted.
    - If it is an internal node, delete and replace with the entry from the left position.
- After deletion, underflow is tested,
    - If underflow occurs, distribute the entries from the nodes left to it.
- If distribution is not possible from left, then

- Distribute from the nodes right to it.

- If distribution is not possible from left or from right, then
  - Merge the node with left and right to it.

Important formula for B+ Tree:

• Order of non-leaf Node:

[p × size of block pointer] + [(p-1) ×size of key field] ≤ Block Size.

• Order of Leaf Node:

$[(p_{leaf} - 1) \times$ (size of key field + size of record pointer] + [p × size of block pointer] ≤ Block Size.

Example:

Construct a B+ tree of order 4 to insert elements 1, 3, 5, 7, 9, 2, 4, 6, 8, and 10 in the given order. Also show the tree after deletion of elements 9, 7, and 8 in the same order.

Solution:

