Suppose a new student, Charlie wants to enroll in a course, but the course has not been taught by any instructor yet.

In this case we cannot add Charlie's enrollment record without also specifying an instructor, even though its not relevant to the enrollment.

2. Deletion Anomalies

It occurs when deleting data from the DB results in the unintentional loss of other, unrelated data.

This typically happens when deleting a record also removes data that is still relevant to other records.

Example Using the same denormalized table as before

Suppose Charlie decide to drop the Math course (CourseId = C101).

If we delete Charlie's enrollment record from the table, the information about the Math course (CourseID = C101) and its instructor (Smith) will also be lost, even though other students might still be enrolled in the course.

3. Modification Anomalies :

↳ It occurs when updating data in the DB leads to inconsistencies b/w related records.

This typically happens when the same data is stored redundantly in multiple places, and updates are not applied consistently across all instances.

Example. → Continuing with the same denormalized table, suppose Smith changes his name to Dr. Smith. in one record (eg: for the Math course) but forget to update it in all other records where Prof. Smith is listed as the instructor, inconsistency arises.

Now, the DB contains conflicting information about the instructor's name, leading to a modification to anomaly. [update krna hai Name Jane se Smith ko and our Jane table ka data same teacher hai lekin Smith name ka teacher hai, toh DB me two entries pdha pda hai].

Q20. When a transaction is rolled back under timestamp ordering it is assigned a new timestamp & why can it not simply keep its old timestamp.?

Ans. In timestamp ordering concurrency control, transactions are assigned unique timestamps that represent their start time.

These timestamps are used to determine the order in which transactions are executed and to resolve conflicts b/w transactions.

When a transaction is rolled back and needs to be restarted, assigning it a new timestamp rather than keeping its old timestamp is essential for maintaining the integrity and correctness of concurrency control mechanism.

Here's Why:

1. Preserving Transaction ordering
   ↳ Timestamp ordering relies on the temporal order of transaction to ensure serializability. Each transaction's timestamp indicates its position in the global order of transaction. If a rolled-back transaction were allowed to keep its old timestamp, it would violate the temporal order, potentially leading to incorrect execution sequence and violating the serializability property.

2. Preventing Aborted Transactions from Retaining Priority:
   ↳ Allowing a rolled-back transaction to keep its old timestamp could give it priority over newer transactions which is unfair to other transactions.

   Transaction that have been rolled back due to conflicts or errors should not be given preferential treatment over transaction that have not yet been executed or committed.

3. Avoiding Rescheduling Conflicts.

↳ Keeping a rolled-back transaction's old timestamp could lead to conflicts with other transactions that you have been assigned timestamps after the rolled back occurred.

This could result in scheduling conflicts, where transactions are re-executed in an order that does not reflect their actual temporal order, potentially leading to incorrect outcomes or violating isolation levels.

4. Ensuring correctness and Consistency

↳ Timestamp ordering aims to ensure correctness and consistency in transaction execution by enforcing a strict order based on transaction start time.

Allowing rolled back transaction to retain their old time stamps could compromise these objectives by introducing inconsistencies or violating the principles of concurrency control.

Concurrency control.

This technique is used for performing multiple transaction on the database at the same time.

It can generate the intermediate value of a single data item that is stored in the permanent memory in different areas called as version of the data item.

In this technique - A lot of version of data item are store so that each version of data item are used for different categories of the transaction.

According to this the various version of control are used for the different categories of the transaction.

It is divided into types.

1) Multi version Concurrency control using time stamp order.
2) " " " using lock.

File organisation ...

- It is a collection of records.
- Using PK we can access the records.
- File organisation is a logical relationship among various records.
- This method define how file records are mapped onto disk blocks.
- It is used to describe the way in which the records are stored in terms of blocks and the block are placed on the storage medium.
- Files of fixed length records are easier to implement than the files of variable length records.

Objective of file organisation ...

- It contains an optimal selection of records, i.e. records can be selected as fast as possible.
- To perform insert, delete, or update transaction on the record should be quick and easy.
- The duplicate records cannot be induced as a result of insert, update and delete.
- For the minimal cost of storage, records should be stored efficiently.

Type of file organisation

(i) Sequential file organisation → (I) File File Method (Random)
→ (II) Sorted File Method. (Sorted manner).

(ii) Heap " "

(iii) Hash " "

(iv) B+ " "

(v) Indexed sequencial access method (ISAM)

(vi) Cluster file organisation.

B^+ Tree :-

↳ The B^+ Tree is a balanced binary search tree.

↳ It follows a multi level index format.

↳ In B^+ Tree, the leaf nodes denote actual data pointers.

↳ Also, ensure that all leaf nodes remain at the same height.

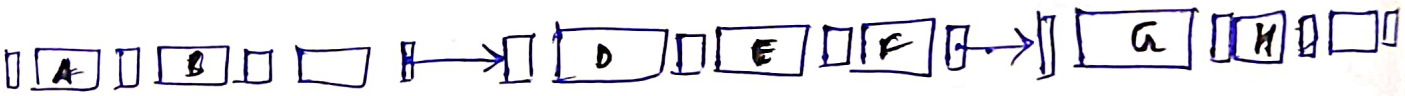
↳ The leaf nodes are linked using a linked list. Therefore, it can support random access as well as sequential access.

Structure of B^+ Tree :-

↳ every leaf node is at equal distance from the root node.

↳ B^+ Tree is of the order n where n is fixed for every B^+ Tree.

↳ It contains an internal node and leaf node.



Internal Node :-

→ An internal node of B^+ Tree can contain at least n/2 record pointers except the root node.

→ At most, an internal node of the tree contains n pointers.

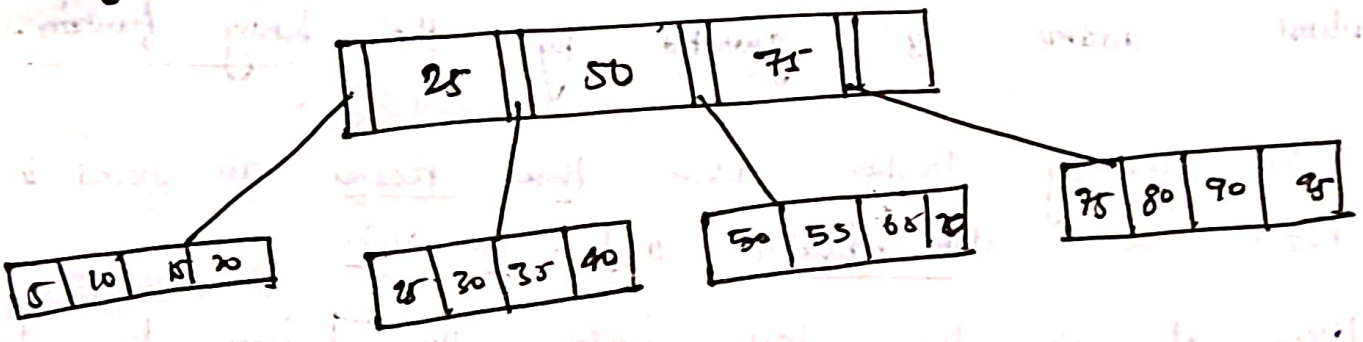
Leaf Node.

↳ The leaf of B+ Tree can contain at least n/2 record pointers and n/2 key values.

↳ At most, a leaf node contains n record pointer and n key values.

↳ Every leaf node of the B+ Tree contains one block pointer P to point to next leaf node.

Searching a records in B+ Tree.



→ First we will fetch for intermediary node which will direct to the leaf node that contain a record for 55.

* Hashing

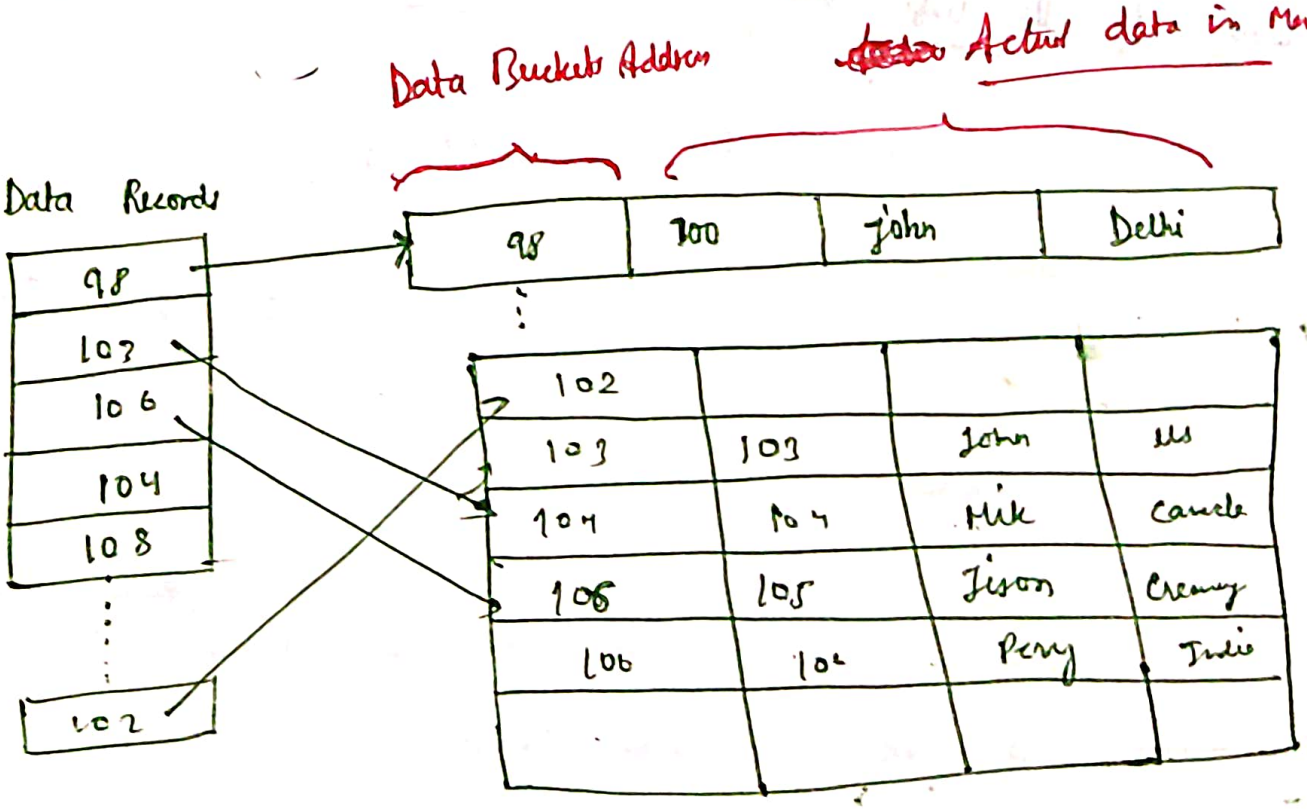
In a huge database structure, it is very inefficient to search all the index values and read the desired data.

Hashing technique is used to calculate the direct location of a data records on the disk without using index structure.

↳ In this technique, data is stored at the data blocks whose address is generated by the hasing function.

↳ The memory location where these records are stored is known as data bucket and data block.

↳ Most of the time Hash function use primary key to generate the address of the data block.



Two types of hashing ...

(i) Static Hashing.

(ii) Dynamic Hashing.

Static Hashing

↳ The resultant data bucket always be the same.

↳ the no. of data buckets in memory remains constant throughout.

- Seaching a record
- Insert a record.
- Delete a record
- Update a record.

1. **Open Hashing** :- When a function generate an address at which data is already stored, then the next bucket will be allocated to it. this mechanism is called as Linear probing.

eg:- R_3 ek naya address hai usko insert krna hai → Hash function generate. kiya address 112.

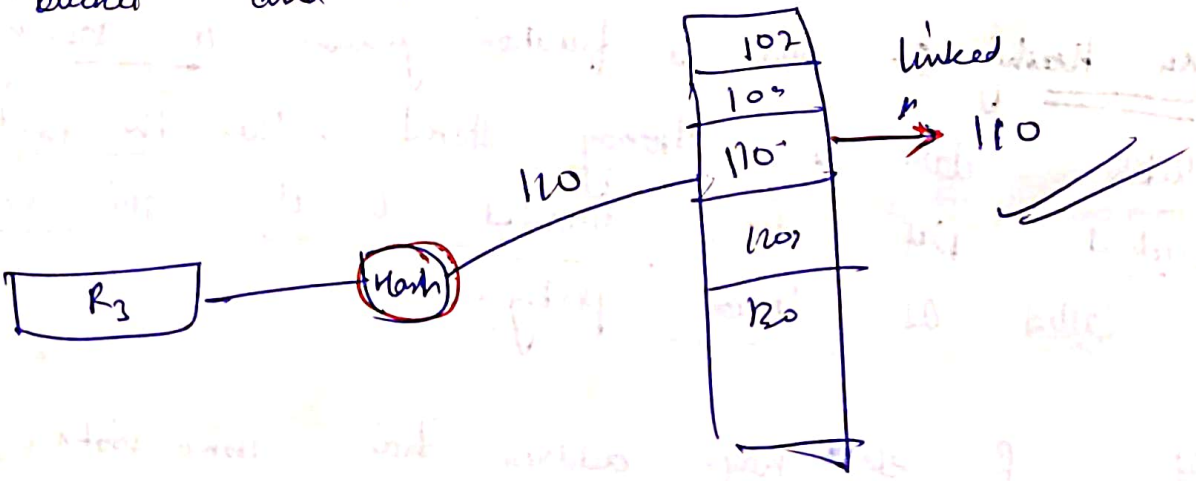
for R_3 → Agar 112 is already taken hai to wo next khali space mein assign kr dega.

(ii) Close Hashing

When buckets are full, then a new data bucket is allocated for the same hash result and is linked after the previous one. This mechanism is known as overflow chaining.

Eg:- R_3 ek naya address jisko insert karna hai

Agar Data Buckets bhara hua and uss address bhi bhara hua hai jo R_3 ka liye bana tha to ek a new bucket is inserted at the end of 110 bucket and is linked to it



Dynamic Hashing *.

↳ It is used to overcome the problem of Static hashing like bucket overflow

↳ Data buckets grow or shrink as the record increases or decrease. Extendable.
This is known as Extendable hashing Method.

↳ It allows insertion and deletion without resulting in poor performance...

How to search a key.

- First calculate the hash address of the key.
- Check how many bits are used in the dictionary and these bits are called as i.
- Take the least significant i bits of the hash address. This gives an index of the directory
- Now using the index, go to the directory and find bucket address where the record might be.

X. File system vs. DBMS.

File system	DBMS
Stores and organizes data in file on storage.	Manages data in database using structured formats.
High redundancy due to separate file	low redundancy using normalization
Hard to maintain consistency accross file	low redundancy using normalizatio
Limited or no current accss	Allows multiple users with proper controls.
Basic file permissions for security.	Advanced security like authentic and roles.
No Acid property	Full transaction suffort with Acid property
No way to represent relationships between data	suppots relationship using key.
Slower for complex Queries	Optimized Query execution
Limited scalability for large data	High scalability for growing data needs.
Difficult to share and update data simultaneously	Enables efficient sharing across users.

Attribute closure. / Closure set. Jenny's lecture.

R (A, B, C, D, E)

FD {A → B, B → C, C → D, D → E}

A → C (Transitive)
A → A → Reflexive
A → D
A → E

B → D
B → E
B → B

C → D
C → E
C → C

D → E

A → ABCDE   B → BCDE   C → CDE

X^+ → set of attribute — all attributes determined by X.

$A^+ = \{A, B, C, D, E\}$ (SK)

$AD^+ = \{A, D, B, C, E\}$.

$B^+ = \{B, C, D, E\}$.

$C^+ = \{C, D, E\}$.

short trick agar AD^+ ko nikalna hai dependency ko to jo hai relation AD jo jo set mein defined hoga and phir ke set mein add karne ke baad usse jo jo define hoga.

All the attributes that is connected to A^+ that is a super key is also a super key in this.

Super key for this → set of attributes whose closure contains all attribute of a given relation.

Q. Find all the candidate key of a relation

R (A, B, C, D, E)

FD: {A → B, D → E}.

$ABCDE^+ = \{A, B, C, D, E\}$ → must be a super key.

ACDE → super key. {A, B, C, D, E} ✓ SK.

$ACD^+ = \{A, B, C, D, E\}$ ✓ SK → Candidate key.

SK: A ✗, C ✗, D ✗, AC ✗, CD ✗, AD ✗

Prime attributes :- prime attributes are part of candidate key.

(A, C, D) are prime key. Attributes.

If no prime attribute is present in the right side of FD then there is only one candidate key.

Q. R (A, B, C, D)

FD — { A → B, B → C, C → A }.

A → C.

ABCD$^+$ = { A, B, C, D } → SK (must).

ACD$^+$ = { A, B, C, D }

CD$^+$ = { C, A, D, B } → candidate key. ①

C, D, → prim attribute

AD

A

ACD$^+$ = { A, B, C, D }

AD$^+$ = { A, D, B, C } → candidate key ②.

A, D — prim attribute

BD = { A, D, B, C }. → candidate key ③.

R(A, B, C, D)

FD = { AB → CD, D → B, C → A }

AB CD$^+$ → { A, B, C, D } → SK (must).

AB$^+$ → { A, B, C, D } → candidate key. ①

AB CD → ~~{ A, B, C, D }~~

~~ABCD~~ ACD ⇒ C → A

CD$^+$ → { A, B, C, D } → candidate key ②.

ABCD ~~ACD~~ If $AB \rightarrow CD$ then $A \rightarrow C$ and $B \rightarrow D$.

$\therefore$ ABCD

AD+ $\rightarrow$ candidate key.

and ABCD $\rightarrow$ ABC $\rightarrow$ BC $\rightarrow$ candidate key.

Q. R{A, B, C, D, E, F}

F = {AB → C, C → DE, E → F, D → A, C → B}.

ABCDEF+ = {A, B, C, D, E, F} → SK ~~(not)~~

ABDEF+ = U.

ABF+ = {U} → (AB → C and C → DE)

$\therefore$ AB → DE.

AB+ = {U} → AB → C and C → DE mean C → E and E → F we remove F.

AB+

candidate key ~~AB and B~~

~~AB~~ Prime Attribute → {A, B}

{D, B} → D = {DA} Not super key.

B = {B}

{C} → candidate key {D, C} ✗ ✗

C = {DE, C, BA} (↓F) All Attributes are there super key

{CB} ✗

Candidate key {A, B} {D, B} {C}.

* (i) Insertion Anamaly.
(ii) Update Anamaly
(iii) Delete Anamaly.

2NF. → candidate key
→ proper subset of candidate key ≠ non-prim attribute

R(A, B, C, D)
FD: { AB → CD, C → A, D → B }
↓
A → C and, C → A, D → B
B → D

{ AB, BC, CD, AD } Candidate Key → proper subset of candidate key → NPA ✗

A ≠ Non-prim attribute
B ≠ "
C ≠ "
D ≠ "

} There fore it is in 2NF.

3NF It should be in 2NF.
No - Transitive dependency for non prim Attributes
A → B and B → C
↓
A → C — Non prim
Non Prime Attribute.

NPP → NPA
Non prim attribute

Q. R(ABCD)

FD :- {A → B, B → C, C → D}

ABCD = {A, B, C, D}

A ⊄ D

AD

(A+) → Sh.

and → Candidate key.

2NF.

A → C

C → D

Now

NPA → NPA

A → B ✓

PA — ✓

(PA)

NPA NPA

B → C

These for it is non prim Attribut.

2NF and but Not 3NF.

BCNF.

It is in 3NF.

for each non-trivial FD (X → Y) X must be a Super key.

Q R(A, B, C)

FD = {A → B, B → C, C → A}

ABC+ = {A, B, C}

(A+) → super key.

→ candidate key

Question main check karna hai left side wale super key hai ki nahi

PA → A :.

A → C No prop subset.

∴ C → Candidate key.

B → Candidate key.

A, B, C → Candidate key.

Each functional dependency satisfy that all these are super key

Proper subset of $ABC \to \{AB, BC, AC, A, B, C\}$

2NF.

$R\{A\ B\ C\ D\ E\ F\}$

$FD = \{A \to B,\ B \to C,\ C \to D,\ D \to E\}$

$A \to E$ $A \to C$ $A \to D$ $C \to E$

$A^+ = \{A\ B\ C\ D\ E\ F\}$

$A^+ = \{A\ B\ C\ D\ E\ F\}$ → Candidate key

AF → only candidate key.

A, F Prime Attribute

Non prim attrib = B C D E

$A \to B$ → that is non prim Attribute

∴ Not in 2NF

$A \to B$ → partial dependency.

Q. $R(A\ B\ C\ D)$

$F = \{AB \to CD,\ AC \to A,\ D \to B$

$AB \to C,\ CB \to D\}$

Yes 2nd Normal form

AD^+

$AB = \{A\ B\ C\ D\}$ → candidate key

$A^+ = \{A, C\}$

$B^+ = \{B\ D\}$

AB → AD

CD

ED

$\{AB, AD, CB, CD\}$ → Candidate key.

$\{A\ B\ C\ D\}$ Prim Attribute

No Non prim attribute