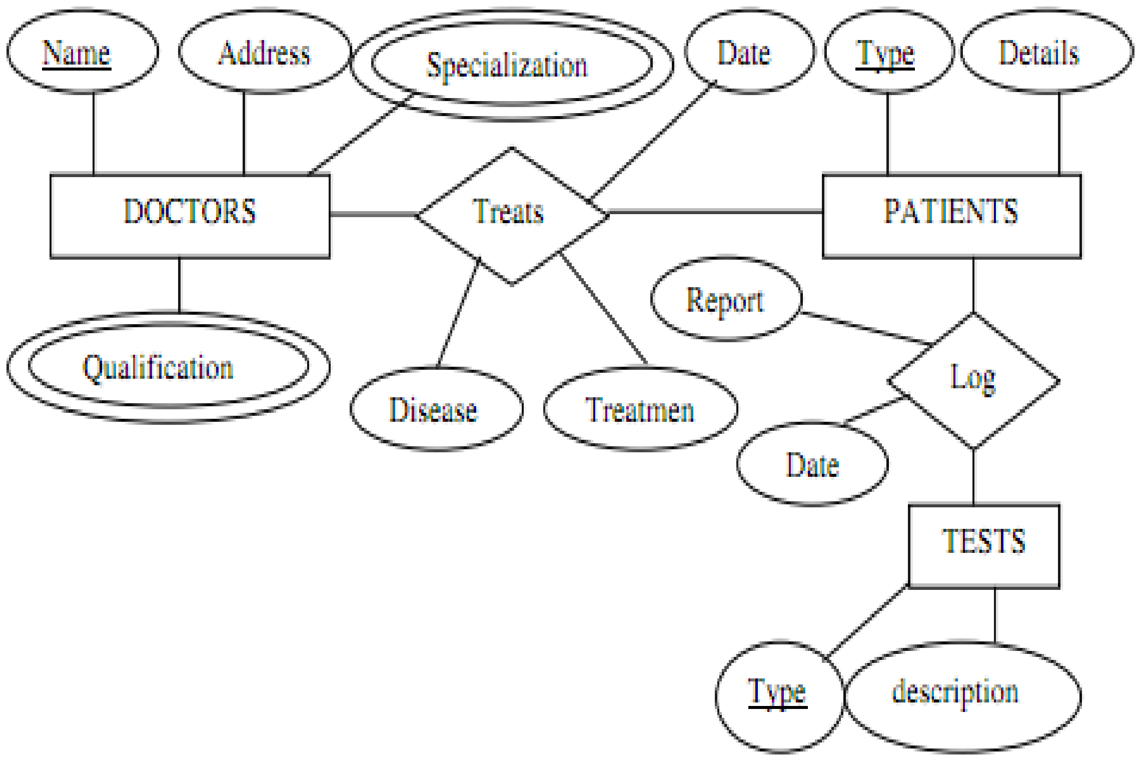
DBMS MID 2

1) Draw an ER diagram for Hospital Management System



Since ERDs are of great use for designing a hospital management system, here we are going to walk you through the steps of designing an ER diagram for your hospital management system. Step 1: The first step is to identify the entity sets. The second step is to map out the attributes of the entities. Identify the type of relationship that exists between the entities. Once you have identified all the relationships, it is time to map out the lines. The last step is to combine all the relationships and draw a complete ER diagram.

2) What are the various attributes in ER model

Attribute Types

- Simple and composite attributes. – Simple Attribute:-They cannot be divided into sub parts. – Composite Attribute:-They can be divided into sub parts. These may appear as a hierarchy. Ex: the attribute "name" can be divided into first name, middle name and last name.

• Single-valued and multi-valued attributes – Single-valued attribute:-Those attributes which can take only one value for a entity. – Multi-valued attribute:-An attribute has a set of values for a specific entity. Ex: phone number attribute

• Derived attributes – Derived attribute:- Value of this attribute can be derived from the value of other related attributes. – Ex: If date_of_birth is an attribute for employee, then age attribute can be derived from the date_of_birth attribute Dr. S Rama Sree, Professor in CSE Dept.

3) Explain weak entity set with an example

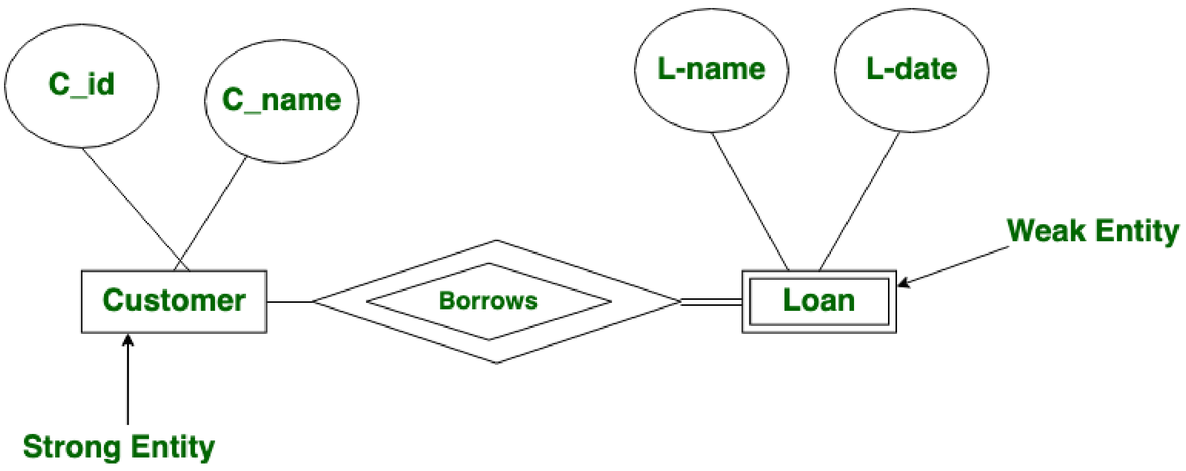
An entity set that does not have a primary key is referred to as a weak entity set.

• The discriminator (or partial key) of a weak entity set is the set of attributes that distinguishes among all the entities of a weak entity set.

• The primary key of a weak entity set is formed by the primary key of the strong entity set plus the weak entity set's discriminator

• Identifying relationship is depicted using a double diamond or thick diamond shape

- Weak entity is **depend on strong entity** to ensure the existence of weak entity. Like strong entity, weak entity does not have any primary key, It has partial discriminator key. Weak entity is represented by double rectangle. The relation between one strong and one weak entity is represented by double diamond.



4) Compare Specialization and Generation in ER model

GENERALIZATION	SPECIALIZATION
Generalization works in Bottom-Up approach.	Specialization works in top-down approach.
In Generalization, size of schema gets reduced.	In Specialization, size of schema gets increased.
Generalization is normally applied to group of entities.	We can apply Specialization to a single entity.
Generalization can be defined as a process of creating groupings from various entity sets	Specialization can be defined as process of creating subgrouping within an entity set
In Generalization process, what actually happens is that it takes the union of two or more lower-level entity sets to produce a higher-level entity sets.	Specialization is reverse of Generalization. Specialization is a process of taking a subset of a higher level entity set to form a lower-level entity set.
Generalization process starts with the number of entity sets and it creates high-level entity with the help of some common features.	Specialization process starts from a single entity set and it creates a different entity set by using some different features.
In Generalization, the difference and similarities between lower entities are ignored to form a higher entity.	In Specialization, a higher entity is split to form lower entities.
There is no inheritance in Generalization.	There is inheritance in Specialization.

5) What is functional dependency and explain its types.

A functional dependency is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of other sets. It is denoted as **X → Y**, where X is a set of attributes that is capable of determining the value of Y.

roll_no	name	dept_name	dept_building
42	abc	CO	A4
43	pqr	IT	A3
44	xyz	CO	A4
45	xyz	IT	A3
46	mno	EC	B2
47	jkl	ME	B2

From the above table we can conclude some valid functional dependencies:

- roll_no → { name, dept_name, dept_building },→ Here, roll_no can determine values of fields name, dept_name and dept_building, hence a valid Functional dependency
- roll_no → dept_name , Since, roll_no can determine whole set of {name, dept_name, dept_building}, it can determine its subset dept_name also.
- dept_name → dept_building , Dept_name can identify the dept_building accurately, since departments with different dept_name will also have a different dept_building

- More valid functional dependencies: roll_no → name, {roll_no, name} ⇢ {dept_name, dept_building}, etc.

- **Types of Functional dependencies in DBMS:**

1. Trivial Functional Dependency

In **Trivial Functional Dependency**, a dependent is always a subset of the determinant.
i.e. If **X** → **Y** and **Y is the subset of X**, then it is called trivial functional dependency

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

Here, **{roll_no, name}** → **name** is a trivial functional dependency, since the dependent **name** is a subset of determinant set **{roll_no, name}**
Similarly, **roll_no** → **roll_no** is also an example of trivial functional dependency.

2. Non-trivial Functional Dependency

In **Non-trivial functional dependency**, the dependent is strictly not a subset of the

determinant.
i.e. If **X** → **Y** and **Y is not a subset of X**, then it is called Non-trivial functional dependency.
For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

Here, **roll_no** → **name** is a non-trivial functional dependency, since the dependent **name** is **not a subset of** determinant **roll_no**
Similarly, **{roll_no, name}** → **age** is also a non-trivial functional dependency, since **age** is **not a subset of {roll_no, name}**

3. <u>Multivalued Functional Dependency</u>

In **Multivalued functional dependency**, entities of the dependent set are **not dependent on eachother.**
i.e. If **a** → **{b, c}** and there exists **no functional dependency** between **b and c**, then it is called a **multivalued functional dependency.**
For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18
45	abc	19

Here, **roll_no → {name, age}** is a multivalued functional dependency, since the dependents **name** & **age** are **not dependent** on each other(i.e. **name → age** or **age → name doesn't exist !**)

4. Transitive Functional Dependency

In transitive functional dependency, dependent is indirectly dependent on determinant.
i.e. If **a → b** & **b → c**, then according to axiom of transitivity, **a → c**. This is a **transitive functional dependency**

For example,

enrol_no	name	dept	building_no
42	abc	CO	4
43	pqr	EC	2

enrol_no	name	dept	building_no
44	xyz	IT	1
45	abc	EC	2

Here, **enrol_no** → **dept** and **dept** → **building_no**,
Hence, according to the axiom of transitivity, **enrol_no** → **building_no** is a valid functional dependency. This is an indirect functional dependency, hence called Transitive functional dependency.

6) Explain 2NF and 4NF with an example

Fourth normal form (4NF):

Fourth normal form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key. It builds on the first three normal forms (1NF, 2NF and 3NF) and the Boyce-Codd Normal Form (BCNF). It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.

Example – Consider the database table of a class which has two relations R1 contains student ID(SID) and student name (SNAME) and R2 contains course id(CID) and course name (CNAME).
Table – R1(SID, SNAME)

SID	SNAME
S1	A

SID	SNAME
S2	B

Table – R2(CID, CNAME)

CID	CNAME
C1	C
C2	D

When there cross product is done it resulted in multivalued dependencies:

Table – R1 X R2

SID	SNAME	CID	CNAME
S1	A	C1	C
S1	A	C2	D
S2	B	C1	C
S2	B	C2	D

Second Normal Form (2NF):

Second Normal Form (2NF) is based on the concept of full functional dependency. Second Normal Form applies to relations with composite keys, that is, relations with a primary key composed of two or more attributes. A relation with a single-attribute primary key is automatically in at least 2NF. A relation that is not in 2NF may suffer from the update anomalies.

Consider following functional dependencies in relation R (A, B, C, D)

```
AB -> C  [A and B together determine C]
BC -> D  [B and C together determine D]
```

In the above relation, AB is the only candidate key and there is no partial dependency, i.e., any proper subset of AB doesn’t determine any non-prime attribute.

7) What are the properties of decomposition

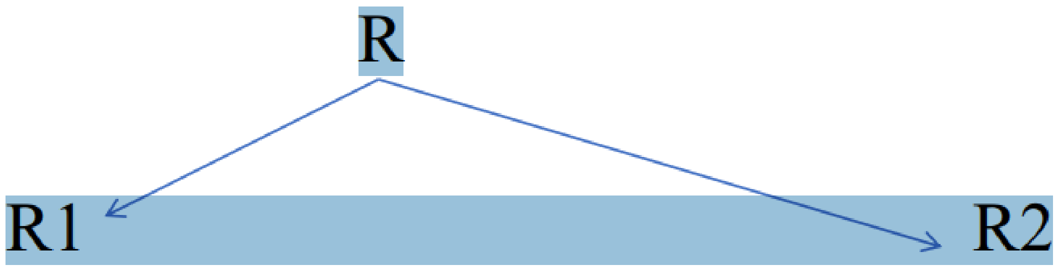
There are 2 types of Decompositions

1) Lossless Decomposition

2) Dependency Preservation

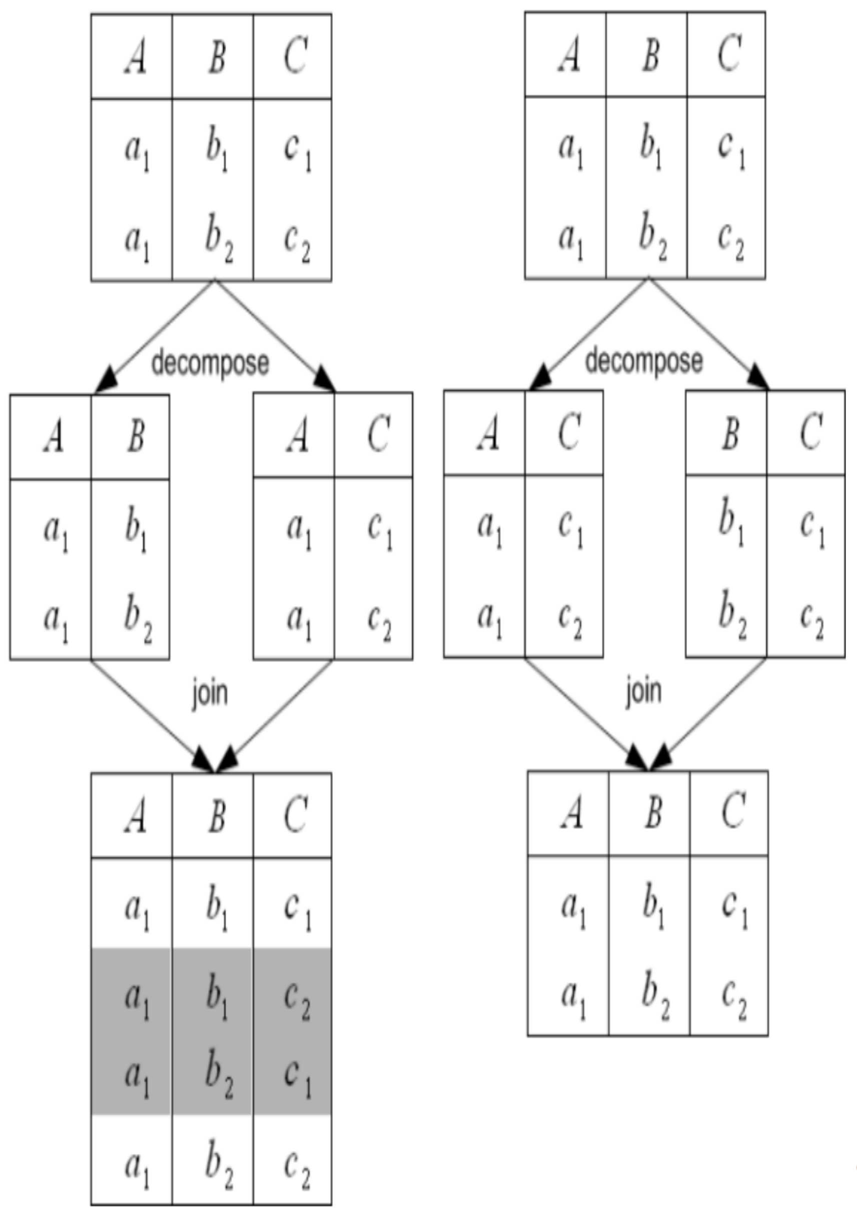
Loss-less join decomposition/ Lossless Decomposition

Let R be a relation schema and let F be a set of FDs over R. A decomposition of R into two schemas is said to be loss less join, if we recover original relation from decomposed relation. Otherwise it is lossy join. We take projections of a relation and recombine them using natural join, to obtain original relation.



Loss-less join decomposition

Aditya Engineering College (A)



Tuesday, May 31, 2022

Dependency Preservation

- In the dependency preservation, at least one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1

or R2 or must be derivable from the combination of functional dependencies of R1 and R2.

• It allows us to enforce all FDs by examining a single relation on each insertion or modification of a tuple. Ex:-

Let R(SUPPLIER_NO,STATUS,CITY,PART_NO,QUANTITY) be a relation. FDs:- FD1: SUPPLIER_NO→CITY FD2: CITY→STATUS FD3: PART_NO→QUANTITY R is decomposed into R1 & R2 such that R1(SUPPLIER_NO,STATUS,CITY) which satisfies FD1,FD2 R2(SUPPLIER_NO,PART_NO,QUANTITY) which satisfies FD3 We can say that, the decomposition of R into R1 & R2 is dependency preservation.

8) Explain 3NF and BCNF with an example

1. Third Normal Form (3NF) :
A relation is said to be in Third Normal Form (3NF), if it is in 2NF and when no non key attribute is transitively dependent on the primary key i.e., there is no transitive dependency. Also it should satisfy one of the below given conditions. For the function dependency C->D:

- C should be a super key and,
- D should be a prime attribute i.e, D should be a part of the candidate key.

3NF is used to reduce data duplication and to attain data integrity.

Example:
For the relation R(L, M, N, O, P) with functional dependencies as {L->M, MN->P, PO->L}:

```
The candidate keys will be : {LNO, MNO, NOP}

      as the closure of LNO = {L, M, N, O, P}

             closure of MNO = {L, M, N, O, P}

             closure of NOP = {L, M, N, O, P}
```

This relation is in 3NF as it is already in 2NF and has no transitive dependency. Also there is no non prime attribute that is deriving a non prime attribute.

2. Boyce-Codd Normal Form (BCNF) :
BCNF stands for Boyce-Codd normal form and was made by R.F Boyce and E.F Codd in 1974.A functional dependency is said to be in BCNF if these properties hold:

- It should already be in 3NF.
- For a functional dependency say P->Q, P should be a super key.

BCNF is an extension of 3NF and it is has more strict rules than 3NF. Also, it is considered to be more stronger than 3NF.

Example:
for the relation R(A, B, C, D) with functional dependencies as {A->B, A->C, C->D, C->A}:

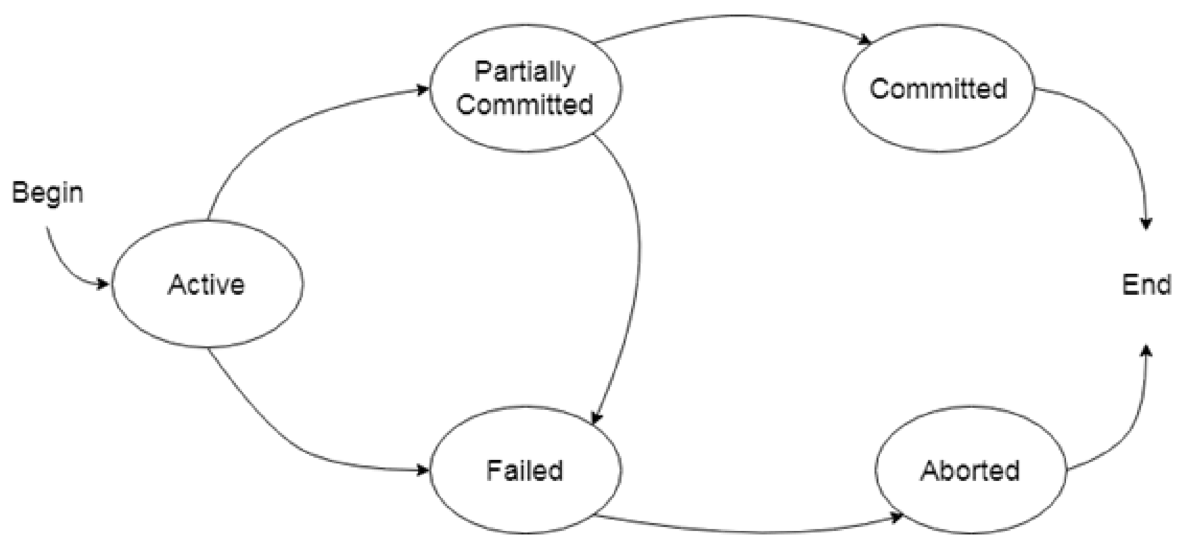
```
The candidate keys will be : {A, C}

      as the closure of A = {A, B, C, D}

             closure of C = {A, B, C, D}
```

This relation is in BCNF as it is already in 3Nf (there is no prime attribute deriving no prime attribute) and on the left hand side of the functional dependency there is a candidate key.

9) What is transaction? What are the different states of a transaction

Transaction

- The transaction is a set of logically related operation. It contains a group of tasks.
- A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.



Active state

- The active state is the first state of every transaction. In this state, the transaction is being executed.
- For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

Partially committed

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.
- In the total mark calculation example, a final display of the total marks step is executed in this state.

Committed

A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

Failed state

- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

Aborted

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
  0. Re-start the transaction
  1. Kill the transaction

10) Explain Conflict serializability with an example

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

Example: –

- **Conflicting** operations pair ($R_1(A)$, $W_2(A)$) because they belong to two different transactions on same data item A and one of them is write operation.
- Similarly, ($W_1(A)$, $W_2(A)$) and ($W_1(A)$, $R_2(A)$) pairs are also **conflicting**.
- On the other hand, ($R_1(A)$, $W_2(B)$) pair is **non-conflicting** because they operate on different data item.
- Similarly, (($W_1(A)$, $W_2(B)$) pair is **non-conflicting.**

Consider the following schedule:

S1: $R_1(A)$, $W_1(A)$, $R_2(A)$, $W_2(A)$, $R_1(B)$, $W_1(B)$, $R_2(B)$, $W_2(B)$

If O_i and O_j are two operations in a transaction and $O_i < O_j$ (O_i is executed before O_j), same order will follow in the schedule as well. Using this property, we can get two transactions of schedule S1 as:

T1: $R_1(A)$, $W_1(A)$, $R_1(B)$, $W_1(B)$

```
T2: R₂(A), W₂(A), R₂(B), W₂(B)
```

Possible Serial Schedules are: T1->T2 or T2->T1
-> **Swapping non-conflicting operation**s R_2(A) and R_1(B) in S1, the schedule becomes,
S11: R_1(A), W_1(A), R_1(B), **W_2(A)**, R_2(A), **W_1(B)**, R_2(B), W_2(B)

-> Similarly, s**wapping non-conflicting operations** W_2(A) and W_1(B) in S11, the schedule becomes,
S12: R_1(A), W_1(A), R_1(B), W_1(B), R_2(A), W_2(A), R_2(B), W_2(B)

S12 is a serial schedule in which all operations of T1 are performed before starting any operation of T2. Since S has been transformed into a serial schedule S12 by swapping non-conflicting operations of S1, S1 is conflict serializable.

Let us take another Schedule:

```
S2: R₂(A), W₂(A), R₁(A), W₁(A), R₁(B), W₁(B), R₂(B), W₂(B)
```

Two transactions will be:

```
T1: R₁(A), W₁(A), R₁(B), W₁(B)
T2: R₂(A), W₂(A), R₂(B), W₂(B)
```

Possible Serial Schedules are: T1->T2 or T2->T1
Original Schedule is:

S2: R_2(A), W_2(A), **R_1(A)**, W_1(A), R_1(B), W_1(B), **R_2(B)**, W_2(B)

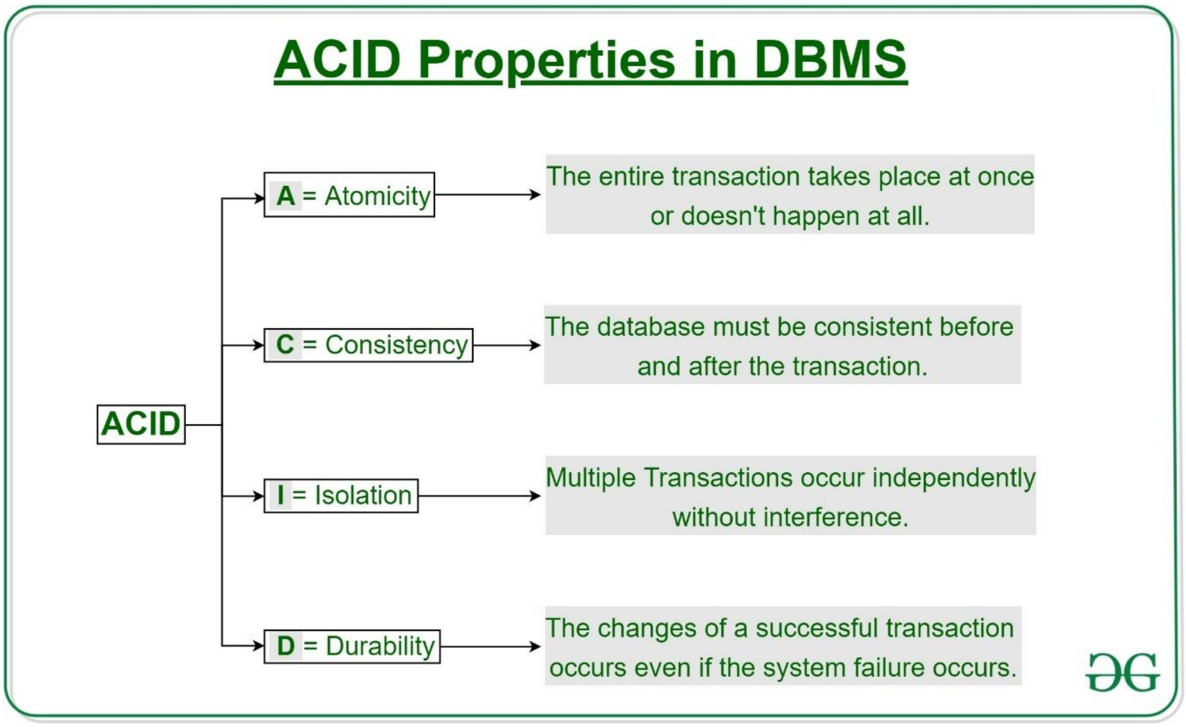
Swapping non-conflicting operations R_1(A) and R_2(B) in S2, the schedule becomes,
S21: R_2(A), W_2(A), R_2(B), **W_1(A)**, R_1(B), W_1(B), R_1(A), **W_2(B)**

Similarly, swapping non-conflicting operations W_1(A) and W_2(B) in S21, the schedule becomes,
S22: R_2(A), W_2(A), R_2(B), W_2(B), R_1(B), W_1(B), R_1(A), W_1(A)

In schedule S22, all operations of T2 are performed first, but operations of T1 are not in order (order should be R_1(A), W_1(A), R_1(B), W_1(B)). So S2 is not conflict serializable.

11) Explain the ACID properties of transactions

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


ACID Properties in DBMS
ACID
A = Atomicity
The entire transaction takes place at once or doesn't happen at all.
C = Consistency
The database must be consistent before and after the transaction.
I = Isolation
Multiple Transactions occur independently without interference.
D = Durability
The changes of a successful transaction occurs even if the system failure occurs.

Atomicity

All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are.
For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.

Consistency

Data is in a consistent state when a transaction starts and when it ends.
For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.

Isolation

The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized.
For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.

Durability

After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure.
For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.

12) Illustrate Hash based and Tree based indexing

Hash Based Indexing

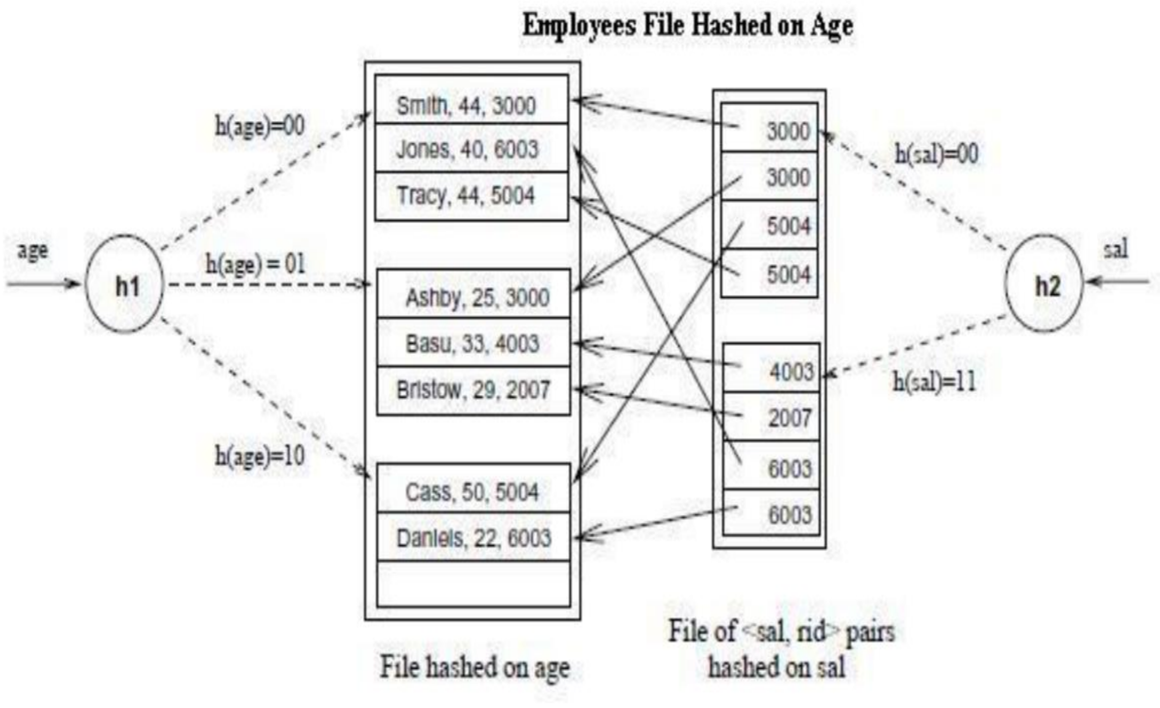
• We can organize records using a technique called hashing to quickly find records when given a search key value.

• Ex:- we can retrieve employee record based on the name field

• The records in a file are grouped in buckets, where a bucket consists of a primary page and zero or more overflow pages linked in a chain.

• Hashing function h: h(r) = bucket in which (data entry for) record r belongs. h looks at the search key fields of r.

Hash Based Indexing



Tree Based Indexing

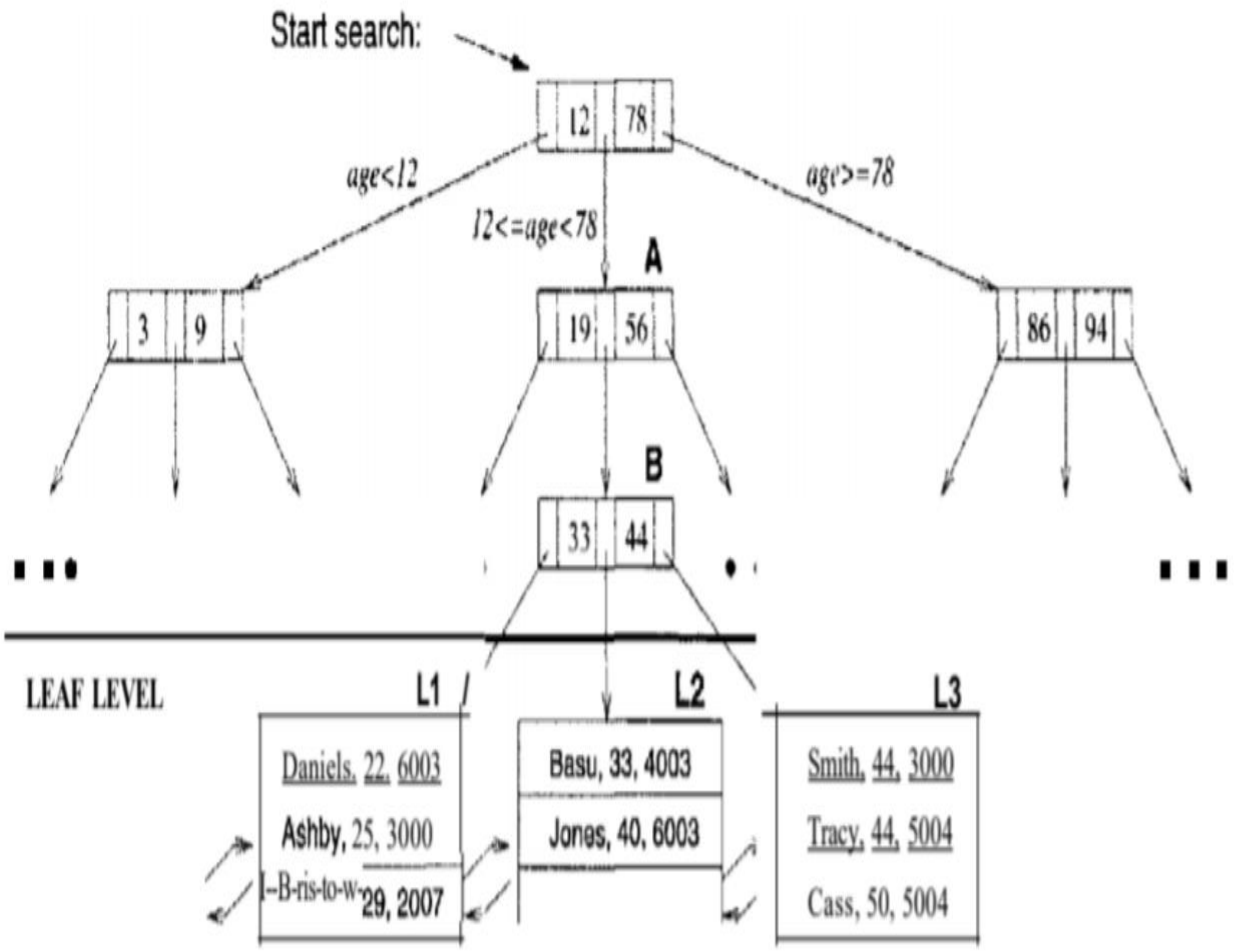
The data entries are arranged in sorted order by search key value & hierarchical search data structure is maintained that directs searches to the correct page of data entries.

Ex:- Employee records are organized in a tree-structured index with search key age.

Each node in this diagram(A,B,L1,L2) is a physical page & retrieving a node involves a disk I/O.

The lowest level of the tree called leaf level contains the data entries.

Tree Based Indexing



13) What are the operations of B+ tree? Explain with a neat diagram the operations

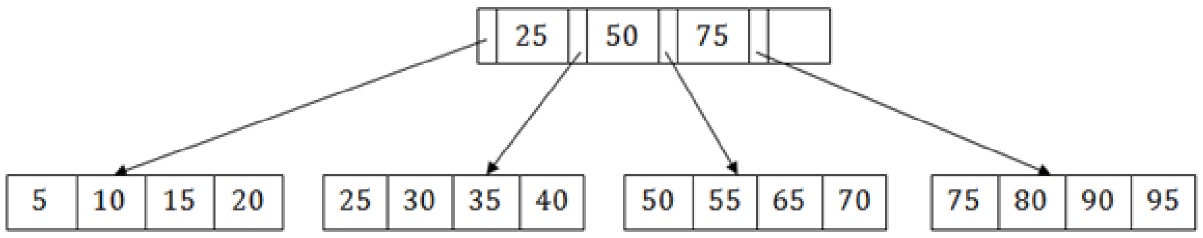
B+ Tree

- The B+ tree is a balanced binary search tree. It follows a multi-level index format.

Searching a record in B+ Tree

Suppose we have to search 55 in the below B+ tree structure. First, we will fetch for the intermediary node which will direct to the leaf node that can contain a record for 55.

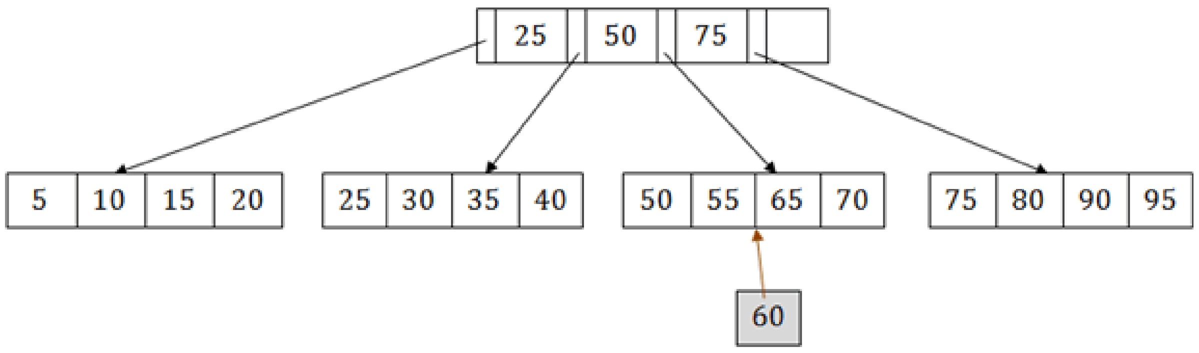
So, in the intermediary node, we will find a branch between 50 and 75 nodes. Then at the end, we will be redirected to the third leaf node. Here DBMS will perform a sequential search to find 55.



B+ Tree Insertion

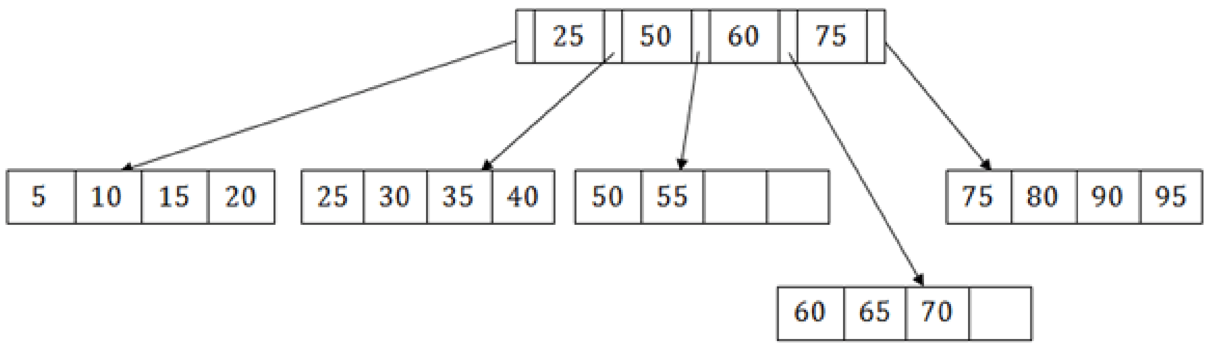
Suppose we want to insert a record 60 in the below structure. It will go to the 3rd leaf node after 55. It is a balanced tree, and a leaf node of this tree is already full, so we cannot insert 60 there.

In this case, we have to split the leaf node, so that it can be inserted into tree without affecting the fill factor, balance and order.



The 3rd leaf node has the values (50, 55, 60, 65, 70) and its current root node is 50. We will split the leaf node of the tree in the middle so that its balance is not altered. So we can group (50, 55) and (60, 65, 70) into 2 leaf nodes.

If these two has to be leaf nodes, the intermediate node cannot branch from 50. It should have 60 added to it, and then we can have pointers to a new leaf node.

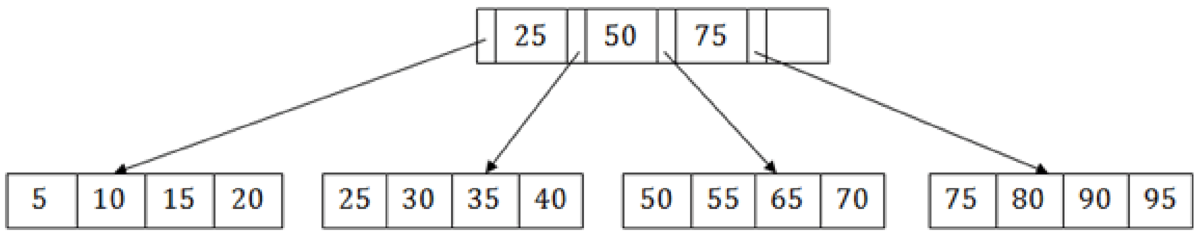


This is how we can insert an entry when there is overflow. In a normal scenario, it is very easy to find the node where it fits and then place it in that leaf node.

B+ Tree Deletion

Suppose we want to delete 60 from the above example. In this case, we have to remove 60 from the intermediate node as well as from the 4th leaf node too. If we remove it from the intermediate node, then the tree will not satisfy the rule of the B+ tree. So we need to modify it to have a balanced tree.

After deleting node 60 from above B+ tree and re-arranging the nodes, it will show as follows:


25 50 75
5 10 15 20
25 30 35 40
50 55 65 70
75 80 90 95