Normalization

✓ **What are the problems arises if we can't design a table using Normalization?**

✓ **What is Insertion, Update and Deletion Anomaly?**

Without Normalization if we design a database the following problem arises;

1) **Update Anomaly:** If one copy of such repeated data is update, then inconsistency is created unless all copies are similarly updated.
2) **Insertion Anomaly:** It may not be possible to store some information unless some other information is stored as well.
3) **Deletion Anomaly:** It may not be possible to delete some information without losing some other information.

Example: Consider the suppliers database given below;

S_id	S_name	S_address	Item	Price
S001	Soumya	Haldia	Laptop	30000
S002	Arijit	Contai	Printer	12000
S003	Subhra	Ramnagar	Scanner	3500
S001	Soumya	Haldia	AC	24700

Here if we update the address of the supplier S001 is one item but for other item if we cannot change the address, inconsistency exists – this is update anomaly.

We cannot store the record of a supplier unless the supplier does not currently order at least one item – this is called insertion anomaly.

If we delete all information of supplier, we loss all information of items – this is called deletion anomaly.

✓ **What is Normalization?**

Normalization is a process of decomposing the redundant relation schemas by breaking up their attributes into smaller relation schemas in such a way that both data redundancy and anomalies are minimized. It is a step-by-step decomposition of complex relation into simple relations, for reducing redundancy and anomalies.

✓ **What is the objective of normalization?**

a) To free a collection of relations from undesirable insertion, update and deletion anomalies (difficulty).
b) To reduce the need for restructuring the collection of relations, as new types of data are introduced.
c) To make the relational model more informative to the user.
d) To structure the data, so that any relationship between entities can be represented.
e) To permit simple retrieval of data in response to query.

✓ **Functional Dependency (FD)**

Functional dependency is a relationship that exists when one attribute uniquely determines another attribute.

If R is a relation with attributes X and Y, attribute Y is functional dependent on attribute X, if for each value of X there exists exactly one value of Y. It represented as X→Y, which specifies Y is functionally dependent on X. Here X is a determinant set and Y is a dependent attribute.

Mathematically it can be consider as follows—

Let r be an instance of a relations schema R and x,y ⊆R. The relation r satisfy the functional dependency x → y, if for any two tuples t_1 and t_2 of r as $t_1(x) = t_2(x)$ Imply $t_1(y) = t_2(y)$

✓ **Fully Functional Dependency (FFD)**

A non-key attribute Y is FFD on attribute X, if it is FD on X and not FD on any proper subset of X. For example, if AB → C and B→C, then C is not FFD on AB.

✓ **Transitive Dependency (TD)**

A transitive dependency can occur only in a relation that has three or more attributes. Let A, B, and C are three distinct attributes in the relation. Suppose all three of the following conditions hold:

1. A → B
2. It is not the case that B → A
3. B → C

Then the functional dependency A → C is a transitive dependency.

✓ **Trivial FD**– If a functional dependency (FD) X → Y holds, where Y is a subset of X, then it is called a trivial FD. Trivial FDs always hold.
For Example, AB→A, ABC→BC

✓ **Non-trivial FD**– If an FD X → Y holds, where Y is not a subset of X, then it is called a non-trivial FD. For Example, AB→C, ABC→D

✓ **Completely non-trivial FD** – If an FD X → Y holds, where x intersect Y = Φ, it is said to be a completely non-trivial FD.

✓ **Multivalued Dependency (MVD)**

It occurs when two or more independent multi-valued facts about the same attribute occur within the same table. If in a relation R, having attributes A,B and C, B and C are multi-valued facts about A, which id represented as

A →→ B
A →→ C

and then MVD exists only if B and C are independent of each other.

✓ Closure of functionally dependency (F^+):

Let F be a set of functional dependencies, then the closer of F, denoted by F^+. F^+ is the set of all functional dependencies that are logically implied by F.

Note : If $F^+ = F$, then F is called full family of dependencies.

✓ Armstrong's Axiom for FD. (Rules for finding F^+):

Armstrong's axioms are a set of axioms (or, more precisely, inference rules) used to infer all the functional dependencies on a relational database.

i. **Reflexivity Rule:** If α is a set of attributes and β⊆α, then α →β holds.
ii. **Augmentation Rule:** If α→β holds and γ is a set of attributes, then γα→γβ hold.
iii. **Transitivity Rule:** If α→ β holds and β→ γ holds, then α→γ holds.
iv. **Union Rule:** If α→β hold and α→γ holds and then α→ βγ holds.
v. **Decomposition Rule:** If α→ βγ holds then α→ β holds and α→γ holds.
vi. **Pseudo Transitivity Rule:** If α→β holds and βγ→ δ holds then αγ→δ holds.

Problem 1: R= {A, B, C, D} and F= { A→BCD, BC→A, BC→D, D→B } Find F^+. [Minimum 4]
Solution:

BC→A and BC→D	by Union rule BC→AD
BC→D and D→B	by Transitivity Rule BC→B
A→BCD	by Decomposition Rule A→B, A→C and A→D
A→B and BC→D	by Pseudo Transitivity Rule AC→D

SO, F^+ = { BC→AD, BC→B, A→B, A→C, A→D, AC→D }

Problem 2: R = {A, B, C, D} and F = { A→B, A→C, BC→D } . Find F^+. (Minimum 5)
Solution:

A→B and A→C	by Union rule A→BC
A→BC and BC→D	by Transitivity Rule A→D
A→B and A→D	by Union Rule A→BD
A→C and BC→D	by Pseudo Transitivity Rule AB→D
A→B and BC→D	by Pseudo Transitivity Rule AC → D

SO, F^+ = { A→BC, A→D, A→BD, AB→D, AC→D }

Problem 3: R = {A, B, C, G, H, I } and F = {A→B, A→C, CG→H, CG→I, B→H} Find F^+.
Solution:

A→B and A→C	by Union Rule A → BC
CG→H and CG→I	by Union Rule CG→HI
A→B and B→H	by Transitivity Rule A → H
A→BC and A→H	by Union Rule A → BCH
A→C and CG→I	by Pseudo Transitive Rule AG → I
A→C and CG→ H	by Pseudo Transitive Rule AG → H

SO, F^+ = { A→BC, CG→HI, A→H, A→BCH, AG→I, AG→H }

Problem 4: R = { A,B,C,D } F = {A→B, A→C, BC→D} Show that A→D hold in F^+.

Proof.
A→B and A→C	by Union rule A → BC
A→BC and BC→D	by Transitivity Rule A → D

Problem 5: R = { A,B,C,X,Z} F = {A→B, C→X, BX→Z} Find AC→Z hold in F^+.

Proof :
C → X	by Reflectivity rule BC → BX
BC→BX and BX→Z	by Transitivity rule BC→Z
A→B and BC→Z	by Pseudo Transitive Rule AC→Z

✓ Closure of Attribute Set:

Problem 1: R = {A, B, C, G, H, I} and F = {A→B, A→C, CG→H, CG→I, B→H} Find $(AG)^+$.

Let x = AG
So, $x^+ = (AG)^+ = \{AG\}$

For A → B if $w \subseteq x^+$
w z i.e. $A \subseteq \{A,G\}$ yes
so, $X^+ = X^+ \cup z$
$= \{A,G\} \cup \{B\}$
$= \{A,B,G\}$

For A →C if $w \subseteq x^+$
w z i.e. $A \subseteq \{A,B,G\}$ yes
so, $X^+ = X^+ \cup z$
$= \{A,B,G\} \cup \{C\}$
$= \{A,B,C,G\}$

For CG→H if $w \subseteq x^+$
w z i.e. $CG \subseteq \{A,B,C,G\}$ yes
so, $X^+ = X^+ \cup z$
$= \{A,B,C,G\} \cup \{H\}$
$= \{A,B,C,G,H\}$

For CG→I if $w \subseteq x^+$
w z i.e. $CG \subseteq \{A,B,C,G,H\}$ yes
so, $X^+ = X^+ \cup z$
$= \{A,B,C,G,H\} \cup \{I\}$
$= \{A,B,C,G,H,I\}$

For B → H if $w \subseteq x^+$
w z i.e. $B \subseteq \{A,B,C,G,H,I\}$ yes
so, $X^+ = X^+ \cup z$
$= \{A,B,C,G,H,I\} \cup \{H\}$
$= \{A,B,C,G,H,I\}$

SO, $(AG)^+ = \{A,B,C,G,H,I\}$

Problem 2: R = {A, B, C, D, E, H} and
F = {A→BC, CD→E, E→C, D→AEH, ABH→BD, DH→BC}. Find $(BCD)^+$.

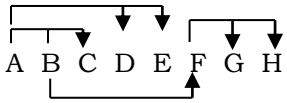
Problem 3: R = {A, B, C, D, E, F, G} and
F = {A→B, BC→DE, AEF→G} Find the closure of AC.

Keys

A key is a set of attributes, with the help of which, we can identify any particular record in a table.

Finding candidate keys

Example 1 : Consider a relation R { A,B,C,D,E,F,G,H } with FD's F = {AB → C, A → DE, B → F, F → GH }. Find the candidate keys.

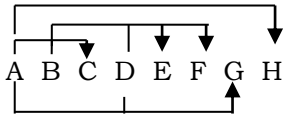


Here A and B attributes do not any incoming edge, means no other attributes can find A and B. So, A and B are the essential attributes and the part of the candidate key. So find $(AB)^+$

$(AB)^+$ = { A,B,D,E,F,G,H,C } using AB → C, A → DE, B → F, F → GH

As combination of AB determines all other attributes of R, AB is the candidate key.

Example 2 : Consider a relation R{A,B,C,D,E,F,G,H } with FD's F ={AB → C, BD → EF, AD →G , A → H}. Find the candidate keys.

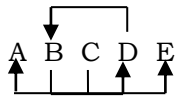


Here A, B and D attributes do not any incoming edge. So, A and B are the essential attributes and the part of the candidate key. So find $(ABD)^+$

$(ABD)^+$ = { A,B,D,C,E,F,G,H } using AB→ C, BD → EF, AD →G , A → H

As combination of ABD determines all other attributes of R, hence ABD is the candidate key.

Example 3 : Consider a relation R{A,B,C,D,E } with FD's F ={ BC →ADE , D → B}. Find the candidate keys.



Here attributes C do not any incoming edge. So, C is the essential attribute and the part of the candidate key. So find $(C)^+$

C^+ = { C } C uniquely can't determines all other attributes, so we try any combination with C.

$(AC)^+$ = { A,C }

$(BC)^+$ = { B,C,A,D,E } using BC →ADE

$(CD)^+$ = { C,D,B,A,E } using BC →ADE , D → B

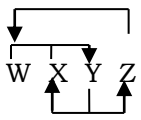
$(CE)^+$ = { C,E }

As combination of BC and CD determines all other attributes of R, hence BC and CD are the candidate keys. Now check for combination of AC and CE is candidate key or not.

$(ACE)^+$ ={A,C,E} ACE not candidate key.

Therefore only BC and CD are candidate key.

Example 4 : Consider a relation R{ W,X,Y,Z } with FD's F ={ Z → W, Y → XZ, WX→Y }. Find the candidate keys.



Here no attribute have, that has no incoming edge. So, no essential attribute have. So we check every possible combination.

W^+ = { W }

X^+ = { X }

Y^+ = { Y,X,Z,W } using Z → W, Y → XZ

Z^+ = { Z,W }

Y uniquely determines all other attributes, so Y is a candidate key.

Now check for combination of W,X and Y,
$(WX)^+$ = { W,X,Y,Z } using Y → XZ, WX→Y
$(XZ)^+$ = { X,Z,W,Y } using BC →ADE
$(WZ)^+$ = { W,Z }

As combination of BC and CD determines all other attributes of R, hence WX and XZ are the candidate keys.
Therefore Y, WX and XZ are candidate keys.

Example 5 : Consider a relation R{ A,B,C,D,E } with FD's F ={ AB →CD, D →A, BC → DE }. Find the candidate keys.
Sol : AB, BC, BD are candidate keys.

Example 6 : Consider a relation R{ A,B,C,D,E,F,G,H} with FD's F ={ CH→G, A→BC, B→CFH, E→A, F →EG }. Find the candidate keys.
Sol : AD, BD, DE and DF are candidate keys.

Example 7 : Consider a relation R{ A,B,C,D,E } with FD's F ={ A→BC, CD→E, B→D, E→A }. Find the candidate keys.
Sol : A, E, BC and CD are candidate keys.

Decomposition

What is decomposition?

Decomposition is the process of breaking down in parts or elements. It replaces a relation with a collection of smaller relations. It breaks the table into multiple tables in a database.

A relation can be decomposed into a collection of relation schemas to eliminate the anomalies and redundancies in original relation.

Types of Decomposition

1. Lossless join Decomposition
2. Lossy join decomposition

Lossless join Decomposition : A relational table is decomposed or factored into two or more smaller tables in such a way that it is possible to get back the original table by joining these decomposed tables. This is called loss less join or non-additive join decomposition.

Let R is a relation and has a set of FD's 'F' over R. The decomposition of R into R_1 and R_2 is loss less w.r.t. F if $R_1 \cup R_2 = R$

Lossy join Decomposition : A relational table is decomposed or factored into two or more smaller tables but it is possible to get back the original table by joining these decomposed tables. This is called lossy join or non-additive join decomposition.

Example 1 : Consider a relation R(A,B,C,D,E) with FD's F = { C→D, AB→CD, A→E }.
Check whether the decomposition of R into R_1(A,B,C), R_2(B,C,D) and R_3(C,D,E) is lossless or lossy.

Solution :

	A	B	C	D	E
R_1	α	α	α		
R_2		α	α	α	
R_3			α	α	α

For, C → D, LHS(C) contains three α and RHS (D) contain two α. So, fill the remaining of D by α.

	A	B	C	D	E
R_1	α	α	α	**α**	
R_2		α	α	α	
R_3			α	α	α

For AB → CD, LHS (AB) contain one combination of α, so no change of RHS occur.
For A → E, one α in LHS, so no change.

As there is no row with all α values. Therefore, it is lossy decomposition.

Example 2 : R = {A, B, C, D, E} with FD's F = (A→BC, C→D, D→B, B→E, A→E). The decomposition of this relation into R_1 {A, B, C}, R_2{C, D} and R_3 {B, D, E}. Test the decomposition is either lossless or lousy.

Solution :

	A	B	C	D	E
R_1	α	α	α		
R_2			α	α	
R_3		α		α	α

For A → BC , LHS (A) contain one α, so no change of RHS occur.

For C → D, LHS (C) contain two α and D also contain α, So, fill the corresponding D by α.

	A	B	C	D	E
R_1	α	α	α	α	
R_2			α	α	
R_3		α		α	α

For D → B, LHS (D) contain three α and B also contain two α, So, fill the remaining B by α.

	A	B	C	D	E
R_1	α	α	α	α	
R_2		α	α	α	
R_3		α		α	α

For B → E, LHS (B) contain three α and RHS (E) contain one α, so fill the remaining by α in E.

	A	B	C	D	E
R_1	α	α	α	α	α
R_2		α	α	α	α
R_3		α		α	α

For A → E, LHS (A) contain one α, so no change of RHS occur.

Here one of the row contains all the α. So the decomposition is loss less join decomposition.

Example 3 : Consider a relation R(A,B,C,D,E) with FD's F = { A→B, C→D, A→E }.
Check whether the decomposition of R into R1(A,B,C), R2(B,C,D) and R3(C,D,E) is lossless or lossy.

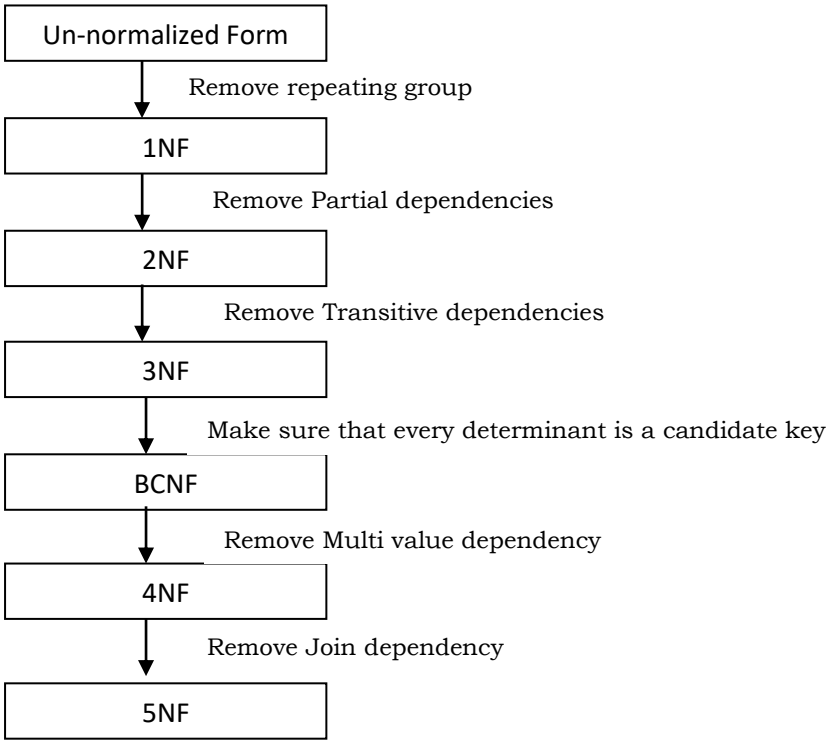
Dependency Preserving Decomposition
Or, What do you mean by Dependency Preservation?

The decomposition of a relation R into R_1, R_2, R_3, …, R_n is dependency preserving decomposition with respect to the set of Functional Dependencies F that hold on R only if the following is hold;

$$(F_1 \cup F_2 \cup F_3 \cup \ldots \cup F_n)^+ = F^+$$

where, F_1, F_2, F_3, …, F_n - Sets of Functional dependencies of relations R_1, R_2, R_3, … R_n.
$(F_1 \cup F_2 \cup F_3 \cup \ldots \cup F_n)^+$ - Closure of Union of all sets of functional dependencies.
F^+ - Closure of set of functional dependency F of R.

Normal Forms



First Normal Form (1NF)

A relation R is in 1NF. if and only if (iff), it holds the following properties –

i) Every field contains a atomic value.
ii) It must have a primary key.
iii) Every non-key attribute is FD on key attribute.

Consider the following relation STUDENT. Normalized it into 1NF.

Course_Name	Roll	Student_Name
DBMS	1	Rahul
	3	Suman
	7	Rohit
COMPILER	2	Rajib
	4	Durlab
	5	Bappa

As Course_Name field not contain atomic value, so the table is not 1NF, so remove the repeating group by filling the missing entries of each incomplete row.

Course_Name	Roll	Student_Name
DBMS	1	Rahul
DBMS	3	Suman
DBMS	7	Rohit
COMPILER	2	Rajib
COMPILER	4	Durlab
COMPILER	5	Bappa

Second Normal Form (2NF)

Before we learn about the second normal form, we need to understand the following −

- **Prime attribute** − An attribute, which is a part of the candidate-key, is known as a prime attribute.
- **Non-prime attribute** − An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

A relation R is in 2NF. if and only if (iff), it holds the following properties –

i) It is in 1NF.
ii) Every non-prime attribute is Full Functionally Depend(FFD) on the prime attribute. That is, if $X \rightarrow A$ holds, then there should not be any proper subset Y of X, for which $Y \rightarrow A$ also holds true.

Note : If every attributes (present in the RHS of the FDs) are prime then relation is in 2NF.

Example1 : Consider a relation R(A,B,C,D) and AB is primary key, so A and B are prime attributes and C and D are non-prime attributes. If AB → C is a FD and A→D, i.e., attribute D is partially dependent on the key attribute. So the relation is not in 2NF.

Example 2 : Consider a relation R(A,B,C,D) and FDs are AB→D, B→C. Is the relation is in 2NF?
Solution : Here AB is a candidate key, as $(AB)^+ = \{A,B,C,D\}$
Since AB is a candidate key, all attribute need to be FFD on AB, but attribute C is only depend on B, so partial dependency holds.
Therefore the relation is not in 2NF.

Third Normal Form(3NF)

For a relation to be in Third Normal Form,

i) it must be in Second Normal form and the following must satisfy −
ii) No non-prime attribute is transitively dependent on prime key attribute.
iii) No non-prime attributes are functionally determines any other non-prime attribute.
iv) For any non-trivial functional dependency, X → A, then either X is a super key or, A is prime attribute.

Example 1 : Consider a relation R(A,B,C,D,E) and FDs are AB→CD, D→A, BC→DE. Is the relation is in 3NF?
Solution : Here AB,BD and BC are the candidate keys.
For, AB→CD, AB is a candidate key, satisfies the 3NF condition.
For, D→A, here D is not candidate key but A is prime attribute, satisfies the 3NF condition.
For, BC→DE, BC is a candidate key, satisfies the 3NF condition.
Therefore the relation is in 3NF.

Boyce-Codd Normal Form(BCNF)

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that −

i) Relation should be in 3NF,and
ii) For any non-trivial functional dependency, X → A, X must be a super-key.

Ex1 : Consider R(A,B,C,D,E,F,G,H) and FDs are AB→C, A→DE, B→F, F→GH. Find highest normal form.
Solution : Here AB is the candidate key.
Checking for BCNF –
For AB→C, As AB is super key.
But for A →DE, A is not a super key, this violate the condition of BCNF.
So the relation is not in BCNF.
Checking for 3NF –
For AB→C, AB is a super key.
For A→DE, A is not super key or D,E are not prime attribute, this violate the condition of 3NF.
So the relation is not in 3NF.
Checking for 2NF –
For AB→C, AB is a candidate key,
For A→DE, Here DE is partially depends on the part of the key, so partial dependency exists.
So the relation is not in 2NF.

Therefore the relation is in 1NF.

Ex2 : Consider R(A,B,C,D,E) and FDs are CE→D, D→B, C→A. Find highest normal form.
Solution : Here CE is the candidate key.
Checking for BCNF –
For CE→D, CE is super key but for D→B and C→A, D and C are not super key For CE→D, CE is super key but for D→B and C→A, D and C are not super key. So violating condition of BCNF.
Checking for 3NF –
For CE→D, CE is super key but for D→B, D is not super key. So this violating condition of 3NF.
So the relation is not in 3NF.
Checking for 2NF –
For CE →D and C→A and CE is the candidate key,
Here partial dependency exists. So this violating condition of 2NF.
So the relation is not in 2NF.
Therefore the relation is in 1NF.

Ex3 : Consider R(A,B,C,D,E,F) and FDs are AB→C, DC→AE, E→F. Find highest normal form.
Solution : 1NF

Ex4 : Consider R(A,B,C,D,E,F,G,H,I) and FDs are AB→C, BD→EF, AD→GH, A→I. Find highest normal form.
Solution : 1NF

Ex5 : Consider R(A,B,C,D,E) and FDs are AC→ADE, D→B. Find highest normal form.
Solution : 3NF

Ex6 : Consider R(A,B,C) and FDs are A→B, B→C, C→A. Find highest normal form.
Solution : BCNF

✓ **"Any relation schema with two attributes is in BCNF" – prove.**

✓ **"Every Binary relation is in BCNF" – justify.**

Consider a relation schema R(A,B) with two attribute. The only possible non trivial functional dependencies (FDs) are A→B and B→A.

There are four possible cases :

i) No FD holds in R; In this case, the key is {A,B} and the relation satisfies BCNF.
ii) Only A→B holds; In this case, the key is {A} and the relation satisfies BCNF.
iii) Only B→A holds; In this case, the key is {B} and the relation satisfies BCNF.
iv) Both A→B and B→A holds; In this case there are two keys A and B and the relation satisfies BCNF.

✓ **List the three design goals for relational databases, and explain why each is desirable.**

The three design goals are lossless-join decompositions, dependency preserving decompositions, and minimization of repetition of information. They are desirable so we can maintain an accurate database, check correctness of updates quickly, and use the smallest amount of space possible.

✓ **Difference Between 3NF and BCNF**

3NF	BCNF
A dependency X → Y is allowed in 3NF if X is a super key or Y is a part of some key.	A dependency X → Y is allowed if X is a super key
No non-prime attribute must be transitively dependent on the Candidate key.	For any trivial dependency in a relation R say X→Y, X should be a super key of relation R.
Lossless decomposition can be achieved in 3NF.	Sometime Lossless decomposition is hard to achieve in BCNF.
3NF can always be obtained.	BNCF **cannot** always be obtained

✓ **" All BCNF is in 3NF, but all 3NF is not in BCNF "— justify.**

Relations that are in 3NF are also in BCNF. But, a 3NF relation is not in BCNF.

A schema R is in BCNF with respect to a set FDs, if for all FDs in F of the form α→β ,where α, β ⊆ R holds following condition :

α is a super key for R.

A relational schema R is in 3NF if for every FD α →β either α is a super key or β is the part of some key for R.

Thus the definition of 3NF permits a few additional functional dependencies involving key attributes that are prohibited by BCNF. So, we can say that All BCNF is in 3NF, but all 3NF is not in BCNF.

What is Multivalued Dependency (MVD)?

Let **R** be the relational schema, **X,Y** be the attribute sets over **R**. A MVD **(X→→Y)** exists on a relation **R** ,If two tuples $\mathbf{t_1}$ and $\mathbf{t_2}$ exists in **R**, such that $\mathbf{t_1[X] = t_2[X]}$ then two tuples $\mathbf{t_3}$ and $\mathbf{t_4}$should also exist in **R** with the following properties where **Z = R – {X ∪ Y}:**

$t_3[X] = t_4[X] = t_1[X] = t_2[X]$
$t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$
$t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$

The tuples t_1, t_2, t_3, t_4 are not necessarily distinct.

File Organization

Data: Raw materials of information.
Information: Process data is known as information.
Record: Record is a collection of fields.
File: File is a collection of same type records.

File organization: The arrangement of records in a file is defined as file organization. A file is organized logically as a sequence of records.

✓ Goal of file organization:

The major goal of file organization are—

1) To provide an efficient method to locate record for processing.
2) To facilitate file creation for its maintenance in future.

✓ What are the operation s related with the file?

1) Scanning or fetching records from the file.
2) Searching a specific records.
3) Searching records between a particular range.
4) Insertion of new record into a file.
5) Deleting a record from the file. etc.

✓ Types of File Organization.

A file may be organized as –

i) Heap File Organization
ii) Sequential File Organization
iii) Indexed Sequential File Organization
iv) Hash File Organization
v) Cluster File Organization

✓ Heap File Organization :

This is the simplest form of file organization. In heap file organization –

- File records can be placed anywhere in that memory area.
- It is the responsibility of the software to manage the records.
- Heap File does not support any ordering, sequencing, or indexing on its own.

Advantages of Heap File Organization

i) Simple file organization.
ii) Insertion of new record is less time consuming.
iii) Space utilization is very high.
iv) Best method for bulk loading data into a relation.

Disadvantages of Heap File Organization

i) This method is inefficient for larger databases as it takes time to search/modify the record.
ii) Proper memory management is required to boost the performance.
iii) Slow retrieval of records.
iv) Deletion operation can take more time.

✓ Sequential file organization:

Every file record contains a data field (attribute) to uniquely identify that record. In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key. A sequential file may be stored on a sequential storage device such as – magnetic tape.

Advantage:

i) Retrieval of records on a primary key is very fast.
ii) When there are large volumes of data, this method is very fast and efficient.
iii) The design is very simple compared other file organization..
iv) Absence of auxiliary data structure.

Disadvantage:

i) Sorted file method always involves the effort for sorting the record. Each time any insert/update/ delete transaction is performed, file is sorted.
ii) Replay for the simple query is time consuming.
iii) Creation and updating of a new file is costly.

✓ Indexed Sequential File organization:

This is an advanced sequential file organization method. Here records are stored in order of primary key in the file. Using the primary key, the records are sorted. For each primary key, an index value is generated and mapped with the record. This index is nothing but the address of record in the file.

Advantages:

i) Since each record has its data block address, searching for a record in larger database is easy and quick.
ii) Allows record to be processed efficiently in both sequentially a number order, depending on the processing operations.
iii) Data can be accessed directly and quickly.

Disadvantages:

i) An extra cost to maintain index has to be afforded. i.e.; we need to have extra space in the disk to store this index value.
ii) As file grows, performance rapidly decreases because of overflow and consequently re-organization is required.

✓ Hash/Direct 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.

Advantages:

i) Records need not be sorted after any of the transaction.
ii) Since block address is known by hash function, accessing any record is very faster.
iii) It is suitable for online transaction systems like online banking, ticket booking system etc.

Disadvantages:

i) Since all the records are randomly stored, they are scattered in the memory. Hence memory is not efficiently used.
ii) If we are searching for range of data, then this method is not suitable. Because, each record will be stored at random address.
iii) There may not be any backup in case a file is destroyed.

✓ 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.

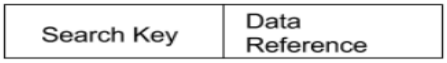
DBMS - Indexing

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.

Indexes are created using some database columns.

- The first column is the Search key that contains a copy of the primary key or candidate key of the table.
- The second column is the Data Reference which contains a set of pointers holding the address of the disk block where that particular key value can be found.



Structure of an index

Indexing Methods

Ordered Indices

The indices are usually sorted so that the searching is faster. The indices which are sorted are known as ordered indices.

- If the search key of any index specifies same order as the sequential order of the file, it is known as primary index or clustering index.
  Note: The search key of a primary index is usually the primary key, but it is not necessarily so.
- If the search key of any index specifies an order different from the sequential order of the file, it is called the secondary index or non-clustering index.

Clustered Indexing

Clustering index is defined on an ordered data file. The data file is ordered on a non-key field. In some cases, the index is created on non-primary key columns which may not be unique for each record. In such cases, in order to identify the records faster, we will group two or more columns together to get the unique values and create index out of them. This method is known as clustering index. Basically, records with similar characteristics are grouped together and indexes are created for these groups.

For example, students studying in each semester are grouped together. i.e. 1st Semester students, 2nd semester students, 3rd semester students etc are grouped.

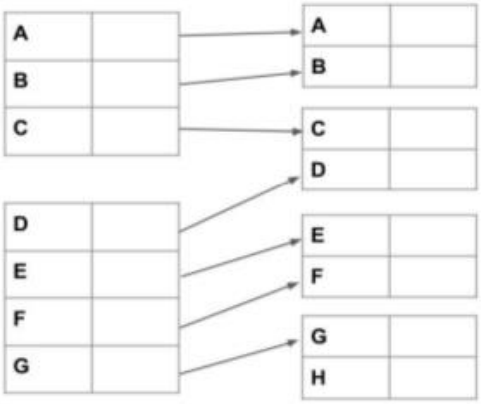
Primary Index

In this case, the data is sorted according to the search key. It induces sequential file organization.

In this case, the primary key of the database table is used to create the index. As primary keys are unique and are stored in sorted manner, the performance of searching operation is quite efficient. The primary index is classified into two types : **Dense Index** and **Sparse Index**.

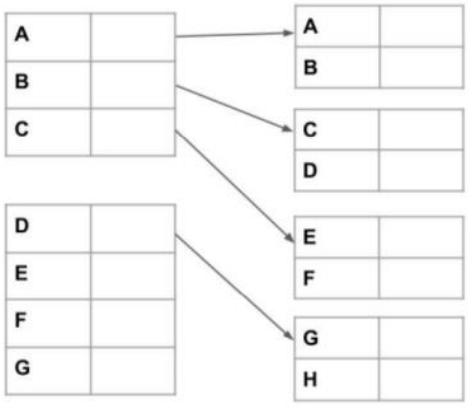
Dense Index

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

- The index record appears only for a few items in the data file. Each item points to a block as shown.
- To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.
- We start at that record pointed to by the index record, and proceed along the pointers in the file (that is, sequentially) until we find the desired record.

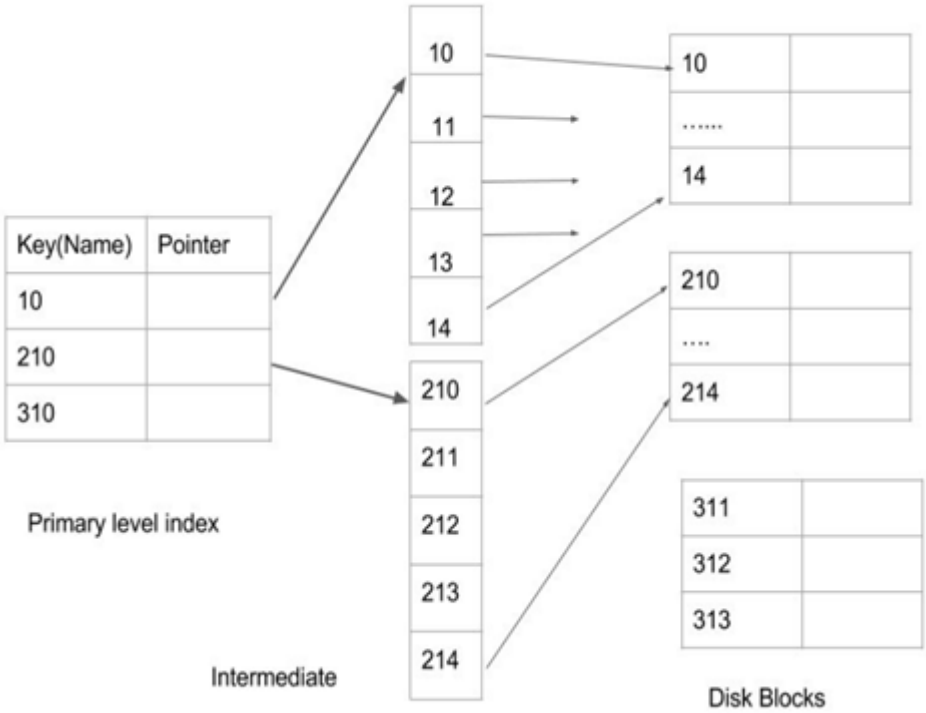


Secondary Index

In the sparse indexing, as the table size grows, the (index, address) mapping file size also grows. In the memory, usually these mappings are kept in the primary memory so that address fetch should be faster. And latter the actual data is searched from the secondary memory based on the address got from mapping. If the size of this mapping grows, fetching the address itself becomes slower. Hence sparse index will not be efficient.

It is used to optimize query processing and access records in a database with some information other than the usual search key (primary key). In this two levels of indexing are used in order to reduce the mapping size of the first level and in general. Initially, for the first level, a large range of numbers is selected so that the mapping size is small. Further, each range is divided into further sub ranges.

In order for quick memory access, first level is stored in the primary memory. Actual physical location of the data is determined by the second mapping level.



What is Transaction?

Transaction is an action, or series of actions that are being performed by a single user or application program, which reads or updates the contents of the database.

A transaction can be defined as a logical unit of work on the database. This may be an entire program, a piece of a program or a single command (like the SQL commands such as INSERT or UPDATE) and it may engage in any number of operations on the database.

ACID Properties

A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

- **Atomicity** − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed.
- **Consistency** − The database must remain in a consistent state after any transaction. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
- **Isolation** − In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.
- **Durability** − The effects of a successfully accomplished transaction are permanently recorded in the database. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.

Difference Between DELETE and TRUNCATE in SQL

Basis for comparison	**DELETE**	**TRUNCATE**
Basic	You can specify the tuple that you want to delete.	It deletes all the tuples from a relation.
Language	DELETE is a Data Manipulation Language command.	TRUNCATE is a Data Definition Language command.
WHERE	DELETE command can have WHERE clause.	TRUNCATE command do not have WHERE clause.
Deletion	DELETE command eliminate the tuples one-by-one.	TRUNCATE delete the entire data page containing the tuples.
Restore	DELETE command can be followed either by COMMIT or ROLLBACK.	TRUNCATE command can't be ROLLBACK.

Difference between Generalization and Specialization in DBMS

GENERALIZATION	**SPECIALIZATION**
It proceeds in a bottom-up manner.	It proceeds in a top-down manner.
Generalization extracts the common features of multiple entities to form a new entity.	Specialization splits an entity to form multiple new entities that inherit some feature of the splitting entity.
Generalization reduces the size of a schema.	Specialization increases the size of a schema.
Generalization results in forming a single entity from multiple entities.	Specialization results in forming the multiple entity from a single entity.