UNIT-IV

Schema Refinement AND NORMAL FORMS

TOPICS:-

→ Purpose of Normalization or Schema Refinement.

→ Concept of Functional Dependency

→ Normal forms based on Functional Dependency (1NF, 2NF and 3NF),

→ Concept of Surrogate Key.

→ Boyce-Codd Normal Form (BCNF).

→ Lossless Join and Dependency preserving Decomposition.

→ Fourth Normal Form (4NF).

Database Schema (Definition):-

A database schema is the skeleton structure that represents the logical view of the Entire database. It defines how the data is organized and how the relations

among them are associated.
It formulates all the constraints that are to be applied on the data.

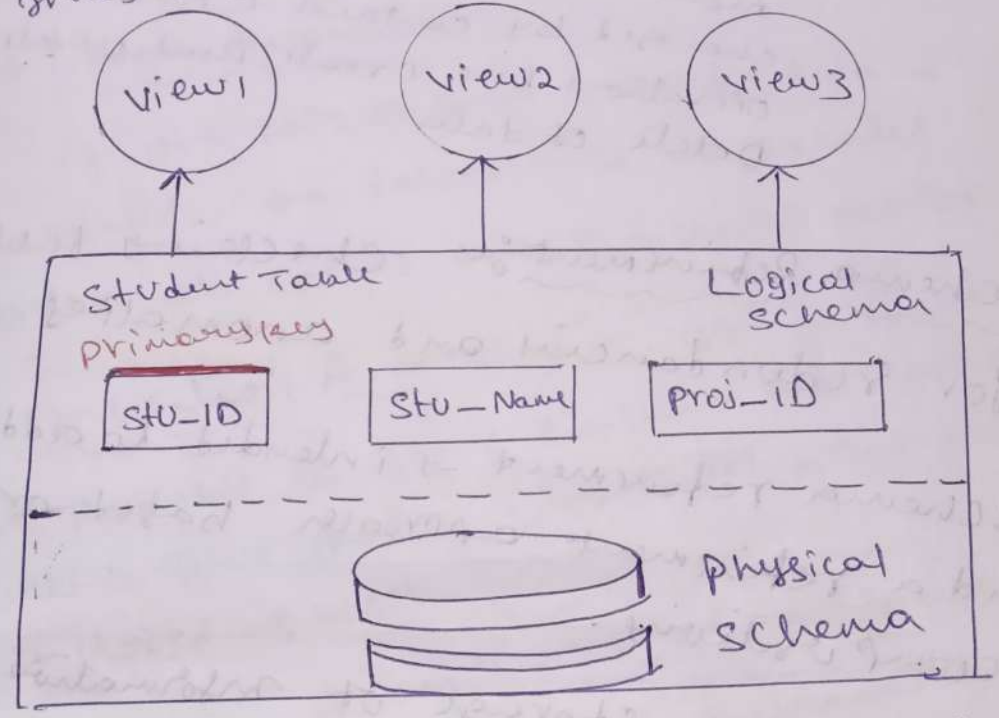
A database schema defines its entities and the relationship among them.
It contains a descriptive detail of the database, which be depicted by means of schema diagrams. The database designers who design the schema to help programmers understand the database and make it useful.

A database schema can be divided broadly into two categories:~

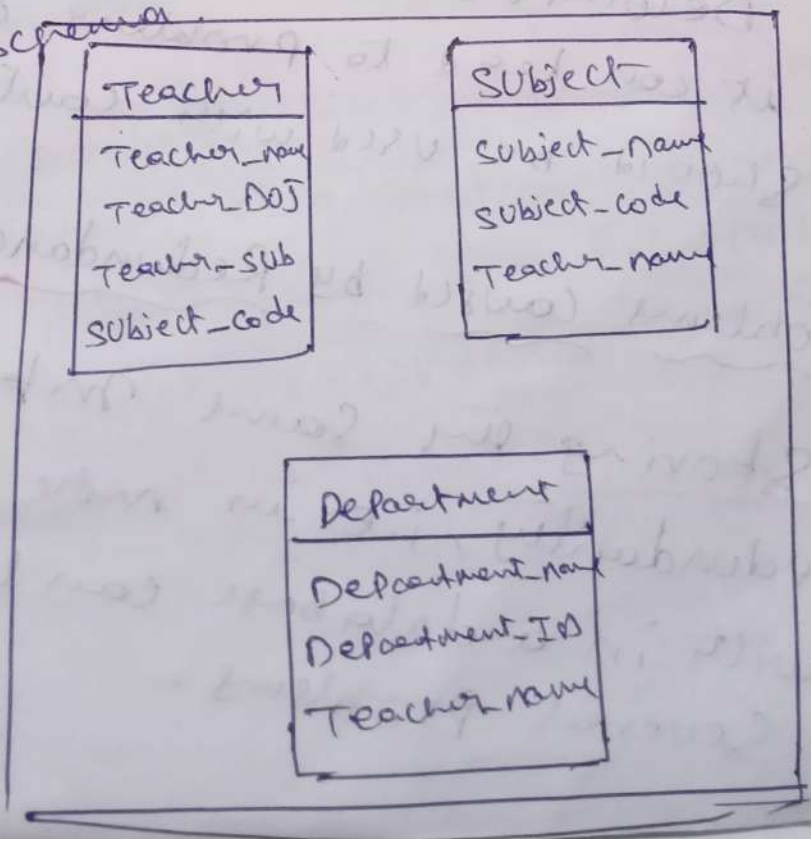
① Physical Database Schema:~
This schema pertains to the actual storage of data and its form of storage like files, indices etc. It defines how the data will be stored in a secondary storage.

② Logical Database Schema:~
This schema all the logical constraint

that need to be applied on the data stored. It defines tables, views and integrity constraints.



* Schema is the over all description of the database. The basic structure of how the data will be stored in the database is called Schema.



Instance :- Instances are the collections of information stored at a particular moment. The instance can be changed by certain CRUD operations like Create, Read, Update, Delete of data.

Schema Refinement :- checking tables for redundancies and anomalies.

→ Schema refinement is intended to address and a refinement approach based on decompositions.

→ Redundant storage of information is the root cause of other problems.

→ Decomposition can Eliminate redundancy it can lead to problems of its own and should be used with caution.

Problems caused by Redundancy :-

Storing the same information redundantly, i.e in more than one place with in a database can lead to several problems.

→ Redundant Storage :-
Some information is stored repeatedly.

→ Update Anomalies :-
if one copy of such repeated data is updated, an inconsistency is created unless all copies are similarly updated.

→ Insertion Anomalies :- It may not be possible to store certain information unless some other, unrelated, information is stored as well.

→ Deletion Anomalies :- It may not be possible to delete certain information without losing some other, unrelated information as well.

Anomali indicates there is some difficulty or a some problem of a constraint in any relational database need to be perform insertion, Deletion, Updation is ~~quite~~ Quite common.

→ Redundant Storage :-
Some information is stored repeatedly.

→ Update Anomalies :-
if one copy of such repeated data is updated, an inconsistency is created unless all copies are similarly updated.

→ Insertion Anomaly :- It may not be possible to store certain information unless some other, unrelated, information is stored as well.

→ Deletion Anomaly :- It may not be possible to delete certain information without losing some other, unrelated information as well.

Anomali indicates there is some difficulty or some problem of a constraint in any relational database need to be perform insertion, Deletion, Updation is ~~quite~~ quite common.

Example:-

Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: EMP_id for storing employee's id, EMP_name for storing employee's name, EMP_address for storing employee's address and EMP_dept for storing the department details in which the employee works. At some point of time the table looks like this:

EMP_Id	EMP_NAME	EMP_Address	EMP_Dept
101	KANTH	VIZAG	D001
101	KANTH	VIZAG	D002
123	MADHU	HYD	D890
166	SATYA	Chennai	D900
166	SATYA	chennai	D004

The above table is not normalized. we will see the problems that we face when a table is not normalized.

Update Anomaly:- If we want to update the address of KANTH then we have to update the same in two rows or the data will become inconsistent. The correct address gets updated in one department but not in other as per the database.

KANTH would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert Anomaly :- Suppose a new employee joins the company, who is under training and currently not assigned to any Department then we could not be able to insert the data into the table if Emp_dept field doesn't allow nulls.

Delete Anomaly :- Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Madhu since she is assigned only to this department.

TO overcome these anomalies we need to normalize the data.

1) Dependency Preservation :- It ensures that each FD is represented in some individual resulting relation after decomposition.

2) Lossless join property :- It guarantees that the spurious tuple generation problem does not occur with respect to the relation schema created after decomposition.

Concept of Functional Dependency

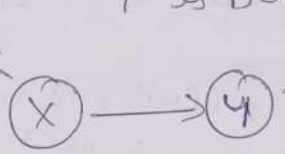
It is a method that describe the relationship between attributes.

Example:- $X \rightarrow Y$ means

X Determines Y

or

Y is Determined by X



Example:-

Sid	Sname
101	KANTH (Laxmikanth)
102	KANTH (Rajinikanth)

(or)

sid	Sname
1	KANTH
1	KANTH

Example 2:-

① Valid (one student)

Sid	Sname
1	KANTH
1	KANTH

$X \rightarrow Y$ Valid

② 2 students

Sid	Sname
1	KANTH
2	KANTH

③ Valid

Sid	Sname
1	KANTH
2	Madhu

2 different students

④ Invalid case

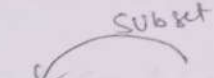
Sid	Sname
1	KANTH
1	Madhu

one sid - two different names

Functional Dependency are of two types :-

(1) Trivial (2) Non-Trivial.

(1) Trivial FD :- if X Determines Y then Y is subset of X. Trivial FD always true.



(Reflexive property)

method ⇒ II

$X \rightarrow Y$

L.H.S ∩ R.H.S ≠ ∅.

sid, sname → sid ⇒ sid.

(2) Non Trivial FD :- when X → Y holds true where Y is not a subset of X. ina relation sup, if attribute Y is not a subset of attribute A, then it is considered as a non-trivial dependency $X \rightarrow Y$ then $X \cap Y = \emptyset$.

(complete non-trivial)

Eg: sid → sname

sid →

If an FD X → Y holds, where Y is not a subset of X, then it is called a non-trivial FD.

Properties of Functional Dependency:-

(1) **(Trivial) **Reflexivity**:- if y is subset of x then
$x \to y$.
if x is set of attributes and y is subset of x, then x holds y.

(2) **Augmentation**:- If $x \to y$ then $xz \to yz$
Eg:- Sid → Sname
Sid, Abone → Sname, Abone
i.e adding dependencies, does not change its basic dependencies.

(3) * **Transitive**:- if $x \to y$ and $y \to z$ then
$x \to z$
Eg:- Sid → Sname and Sname → City
Sid → City.
$x \to y$ is functionally that determines b.

(4) **Union**:- if $x \to y$ and $x \to z$ then
$x \to yz$ (yz having union)

(5) **Decomposition**:- if $x \to yz$ then
$x \to y$ and $x \to z$

Check condition:
$xy \to z$, $x \to z$, $y \to z$ Invalid.
we cannot decompose left hand side.

⑥ Pseudo transitivity:-

if $x \to y$ and $wy \to z$ then

$wx \to z$

⑦ composition :-

if $x \to y$ and $z \to w$ then

$xz \to yw$

Functional Dependency :- The value of one attribute determine the value of another attribute.

candidate key :-

Attribute that uniquely identifies a row in a relation could be a combination of (non-redundant) attributes

- Each non-key field is functionally dependent on every candidate key.

Normalization :- TO Remove duplicate Values. It is the process of organising the attributes and tables of a relational database to minimise data redundancy with the help of functional dependencies and ① Dependency preservation ② Lossless join property.

NORMAL FORMS BASED ON FUNCTIONAL Dependency

~~Assign~~.

TO overcome the above Anomalies we need to normalize the data.

Normalization :- The most commonly used Normal forms are given below:

1) FIRST NORMAL FORM (1NF)
2) Second NORMAL FORM (2NF)
3) Third Normal form (3NF)

1) FIRST NORMAL FORM (1NF) :-

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Example: SUPPOSE a company wants to store the names and contact-details of its employees. It creates a table that looks like this :-

EMP_Id	EMP_Name	EMP_Address	EMP_Mobile
101	KANTH	VIZAG	8912312390
102	madhu	HYD	8812121212 9900012222
103	KIRAN	CHENNAI	7778881212
104	RAM	Bangalore	9990000123 8123450987

Two Employees (madhu & RAM) are having two mobile numbers so the company stored them in the same field as you can see in the above table.

This table is not in 1NF as the rule says "each attribute of a table must have atomic (single) values", the EMP_Mobile values for employees madhu & RAM violates that rule.

TO make the table complies with 1NF we should have the data like this.

EMP_Id	EMP_Name	EMP_Address	EMP_Mobile
101	KANTH	VIZAG	8912312390
102	madhu	HYD	8812121212
102	madhu	HYD	9900012222
103	KIRAN	CHENNAI	7778881212
104	RAM	Bangalore	9990000123
104	RAM	Bangalore	8123450987

Second Normal Form

A table is said to be in 2NF if both the following conditions hold:

Rule 1:

The Table or Relation must be in 1st Normal form.

Rule 2: Every NON-key attribute is fully functionally dependent on the ENTIRE primary key.

(or)

All the non-prime attributes should be fully functional dependent on candidate key

(or)

NO non-prime attribute is dependent on a proper subset of any candidate key of Table.

(or)

- Every non-key attribute must be defined by the Entire key, not by only part of the key.

→ NO partial functional dependencies

No non-prime attribute Definition: if an attribute is not the part of a key, every that attribute is called non-primary attribute.

(or)

An attribute that is not part of any candidate key is known as non-prime attribute.

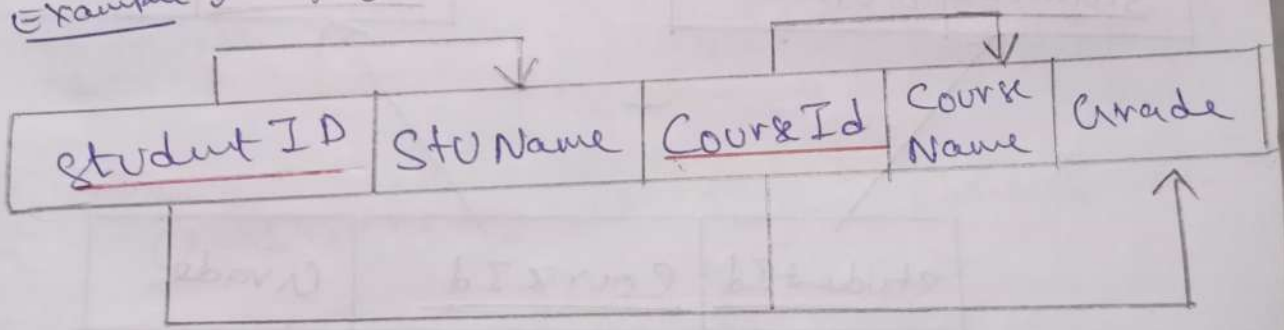
Primary attribute Definition:- If an attribute is the part of a key, then that attribute is called primary attribute.

PROPER SUBSET:-

A PROPER subset of a set A is subset of A that is not equal to A. In other words, if B is a proper subset of A, then all elements of B are in A but A contains at least one element that is not in B.

For Example:- if $A = \{1,3,5\}$ then $B = \{1,5\}$ is a proper subset of A.

Example:- Functional Dependencies in Student



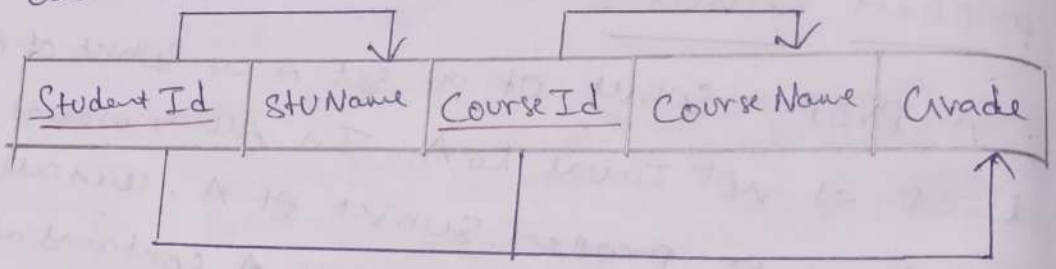
Can represent FD's with arrows as above, or

- StudentId → StuName.
- CourseId → CourseName.
- StudentId, CourseId → Grade

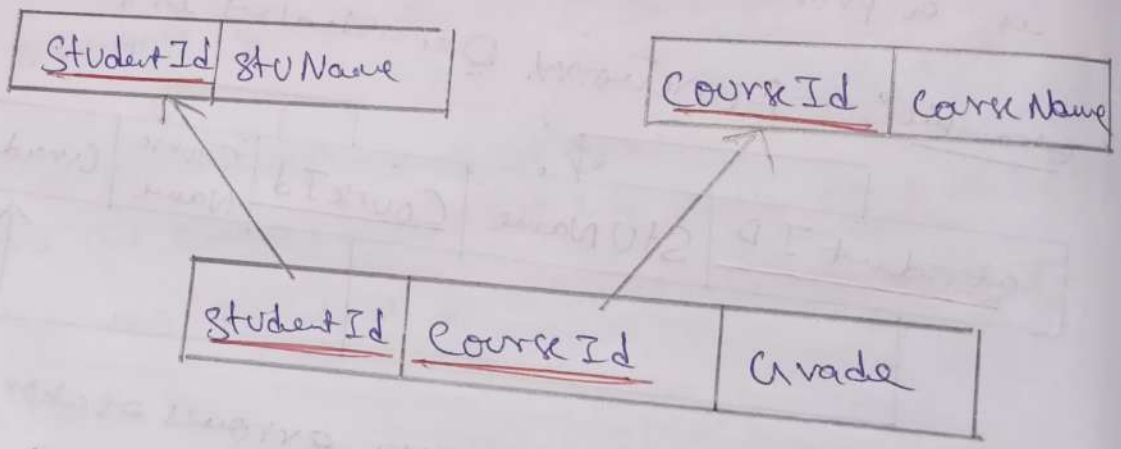
Therefore, NOT in 2nd Normal Form!!

2 Normal form : Normalizing

How do we convert the partial dependencies into normal ones? By decomposing into more tables.



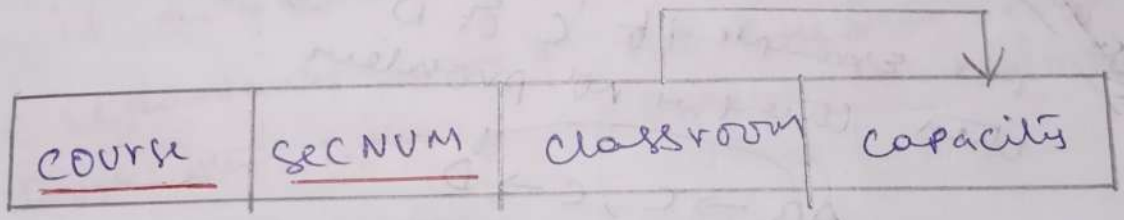
The above arrows represents functional Dependency, below they mean Foreign key constraints.



THIRD NORMAL FORM

The Table must be in the second Normal form. and there should be no transitive dependency in table.

A transitive dependency is when a non key attribute depends on another non-key attribute. This is called a transitive because the primary key is a determinant for another attribute, which in turn is a determinant for a third attribute.

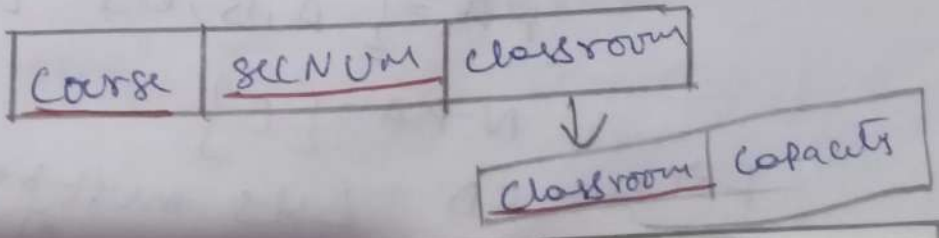


classroom → capacity — Transitive

Any partial FDs? - NO

Any Transitive FDs? → yes!

To eliminate the Transitive Dependency by decomposing the table in to two.



Roll	state	city

Candidate key = {Rollno}

P.A = {Rollno}
N-P.A = {state, city}

FD → Rollno → state → valid
state → city → (Non prime Attribute determining another Non prime)

According to transitivity Now Rollno → city which is not in FD.

R(ABCD)

Candidate Key : AB

FD: AB → C,
C → D ('D' Non prime attribute determine by 'C' Non prime attribute)

prime attribute : A, B
Non prime attribute : C, D

Closure of AB+ = ABCD

For example if 'C' or 'D' is Prime attribute then there is no problem.

AB → C, C → D

Example-II
condition-II

R(ABCD)

Closure of AB

FD: AB → CD ✓
D → A ✓

AB+ = ABCD

CK {AB} {DB}

DB+ = DBAC

PA = {A, B, D} {To make = Candidate key}

N-P.A = {C}

For each FD LHS must be candidate key or SK. or RHS must be prime attribute

Boyce - Codd Normal Form (BCNF)

Boyce - Codd normal form (BCNF) is a normal form used in database normalization. It is a slightly stronger version of the third normal form.

A relational schema R is in Boyce - Codd normal form if and only if for every one of its dependencies $X \to Y$, at least one of the following conditions hold:

(1) $X \to Y$ is a trivial functional dependency ($Y \subseteq X$)

(2) X is a superkey for schema R.

if a relational schema is in BCNF then all redundancy based on FD has been removed, although other types of redundancy may still exist.

(or)

A database table is in BCNF if and only if there are no non-trivial FD of attribute on any thing other than a superset of a candidate key.

BCNF can also be called as BCNF special case of third Normal form.

In third Normal form (3NF) i) L.H.S may be candidate key (or) super key.

ii) R.H.S may be a prime attribute

In BCNF for all the F.D's, all attributes must be dependent on candidate key. Restricted to L.H.S.

Example:-

ROLLNO	Name	Votyid	age
1	KANTH	P123	35
2	MOHAN	M123	30
3	Ravi	R123	25
4	Krishna	K123	20

Candidate Key = { ROLLNO, Votyid }

Functional Dependency = { ROLLNO → Name, ROLLNO → Votyid, Votyid → age, Votyid → ROLLNO } — F.D

1. FD i.e ROLLNO → Name (FD is valid)

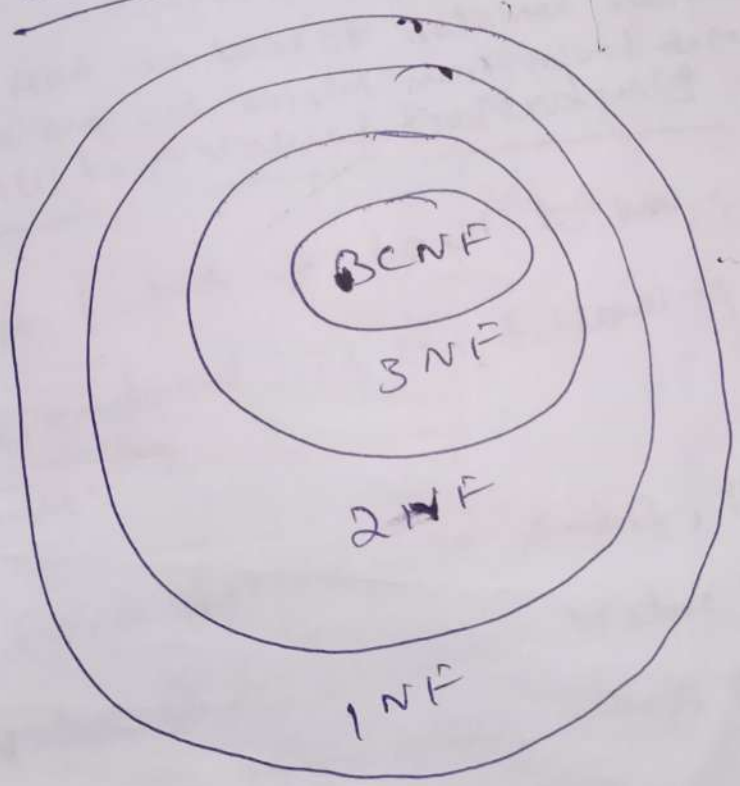
L.H.S is candidate key (or) super key

2. FD. i.e RollNO → Voterid (valid)
L.H.S is candidate key

3. Voterid → age (valid.)

4. Voterid → RollNO (valid)

L.H.S of each FD should be candidate key or superkey.



FOURTH NORMAL FORM (4NF)

For a table to satisfy the Fourth Normal form, it should satisfy the following two conditions:

1) It should be in the Boyce-Codd Normal form

(2) And, the table should not have any multi-valued Dependency.

→ 4NF is a level of database normalization where there are non-trivial multivalued dependencies other than C.K.

Multi-valued Dependency

A table is said to have multi-valued dependency, if the following conditions are true:

(1) For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.

2. Also, a table should have at least 3 columns for it to have a multi-valued Dependency.

3. And, for a relation R(A, B, C),

if there is a multi-valued dependency between A and B, then B and C should be independent of each other.

if all these conditions are true for any relation (table), it is said to have multi-valued dependency.

A	B	C
	B1	C1
A1	B2	C2

multi-valued dependency between $A \rightarrow B$ and $A \rightarrow C$.

Course	Instructor	Textbook
CSE	RAM	C
	SAM	C++
	KIM	
CST	JIM	DBMS
		JAVA

As in this table multivalued Dependency exists so this table is suffering from all anomaly so to solve this splitting of table is reauried.

course → Instructor
course → Text book

course	Instructor
CSE	RAM
CSE	SAM
CSE	KIM
CST	RIM

Instructor Relation

course	~~Instructor~~ Text Book
CSE	C
CSE	C++
CST	DBMS
CST	JAVA

Text Book

Lossless / Lossy Join Decomposition

when we want to Normalize we need to decompose the table.
when we do the decomposition we need to follow two Rules

1. Lossless vs lossy Decomposition
2. Dependency preserving Decomposition

1. Lossless versus Lossy Decomposition :-

R.

A	B	C
1	2	1
2	2	2
3	3	2

R1 (AB)

A	B
1	2
2	2
3	3

R2 (BC)

B	C
2	1
2	2
3	2

Query :- find the value of C if the value of A = '1'

```
select R2.C from R2 Natural Join R1 where
R1.A = "1";
```

R1		R2	
A	B	B	C
1	2	2	1 ✓
1	2	3	2 ✓
1	2	3	2 ✗
2	2	2	1 ✓
2	2	2	2 ✓
2	2	3	2 ✗
3	3	2	1 ✗
3	3	2	2 ✗
3	3	3	2 ✓

Join means
Cross product + condition
(equivalence)

R'

A	B	C	
1	2	1	✓
1	2	2	
2	2	1	
2	2	2	✓
3	3	2	✓

Spurious tuples.

We got 5 rows
It is called flaw
we can call
lossy join
Decomposition.
Inconsistency.
This

we keep B as common. But B is
→ Even if we keep AC, BC also we get (loss)
→ common Attribute should be candidate key
of superkey of either R_1 or R_2 or Both.
→ so finally $R_1(AB)$
$R_2(AC)$.

Rules :-

1) $R_1 \cup R_2 = R$
$AB \cup AC$
$ABC = AB \cup$

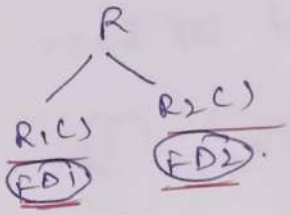
$R_1 \cap R_2 = \phi$
$AB \cap AC$
$A = \phi$

5. $R_1 \subset R$ or $R_2 \subset R$
or Both

Dependency Preserving Decomposition

In a Table where are attributes.
And given Functional Dependencies. like
$A \to B$, $B \to C$. Hidden F.D$^+$ $A \to B$, $B \to C$, $A \to C$

$FD_1 \cup FD_2 = FD^+$



The Union of attributes of R_1 and R_2 should be equal to R.

we need to check the dependency preserve.
Let R(ABCD) with functional Dependencies
$\{ A \to B, B \to C, C \to D, D \to B \}$
if R is decomposed into $R_1(A,B)$, $R_2(B,C)$
$R_3(B,D)$

Trivial $\to$ $A \cap A = A$

Dependencies

$R_1(AB)$	$R_2(BC)$	$R_3(BD)$
$A \to A$ ✗	$B \to C$ ✓	$R(BD)$
$B \to B$ ✗	$C \to B$ ✓	$B \to D$ ✓
$A \to B$ ✓		$D \to B$ ✓
$B \to A$ ✗		

$B^+ = BCD$

A Determine A

$A \cap A = A$ But not ∅ Leave Trivial.
Take Non-Trivial
$B^+ = BCD$
In this table Nontrivial & closure

The Possible Preserved

$A \rightarrow B$ ✓
$B \rightarrow C$ ✓
$C \rightarrow B$ ✓
$B \rightarrow D$ ✓
$D \rightarrow B$ ✓

$C^+ = CBD$

Uniave key: The values of the attribute in table is uniave, NOT updatable and may be NULL

Surrogate key: It is uniave, NOT NULL and updatable.

An Artificial key which aims to uniavely identify each record is called a surrogate key. These kind of keys are uniave because they are created when you don't have any natural primary key.

The usual surrogate key use is a uniave sequential number.

A surrogate key in a database is a unique identifier for either an entity in the modeled world or an object in the database.
The surrogate key is not derived from application data, unlike a natural key which is derived from application data.

or

A surrogate represents an entity in the outside world. The surrogate is internally generated by the system but is nevertheless visible to the user or application.

Approaches to generating surrogate include

Example :-

oracle sequence, or generated as identity

Dependency-preserving Decomposition:-

The dependency preservation decomposition is another property of decomposed relational database schema D in which each functional dependency $X \rightarrow Y$ specified in F either appeared directly in one of the Relation schemas R_i in the decomposed D or could be inferred from the dependencies that appeared in some R_i.

Decomposition $D = \{R_1, R_2, R_3 \ldots R_m\}$ of R is said to be dependency preserving with respect to F if the union of the projection of F on each R_i, in D is equivalent to F.

In other words, R C Join of
R_1, R_1 over X.

The dependencies are preserved because every dependency in F represents a constraint on the database. if decomposition is not dependency – preserving Some dependency is lost in the decomposition.