Unit – 4 Schema Refinement and Normal Forms

INTRODUCTION TO SCHEMA REFINEMENT

- The Schema Refinement refers to refine the schema by using some technique.
- The best technique of schema refinement is decomposition.

Normalization

- It means "split the tables into small tables which will contain less number of attributes in such a way that table design must not contain any problem of inserting, deleting, updating anomalies and guarantees no redundancy".
- Normalization or Schema Refinement is a technique of organizing the data in the database.
- It is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies.

Redundancy: refers to repetition of same data or duplicate copies of same data stored in different locations.

Anomalies: Anomalies refers to the problems occurred after poorly planned and normalized databases where all the data is stored in one table which is sometimes called a flat file database.

Due to redundancy of data we may get the following problems, those are:

1.insertion anomalies : It may not be possible to store some information unless some other information is stored as well.

2.redundant storage: some information is stored repeatedly

3.update anomalies: If one copy of redundant data is updated, then inconsistency is created unless all redundant copies of data are updated.

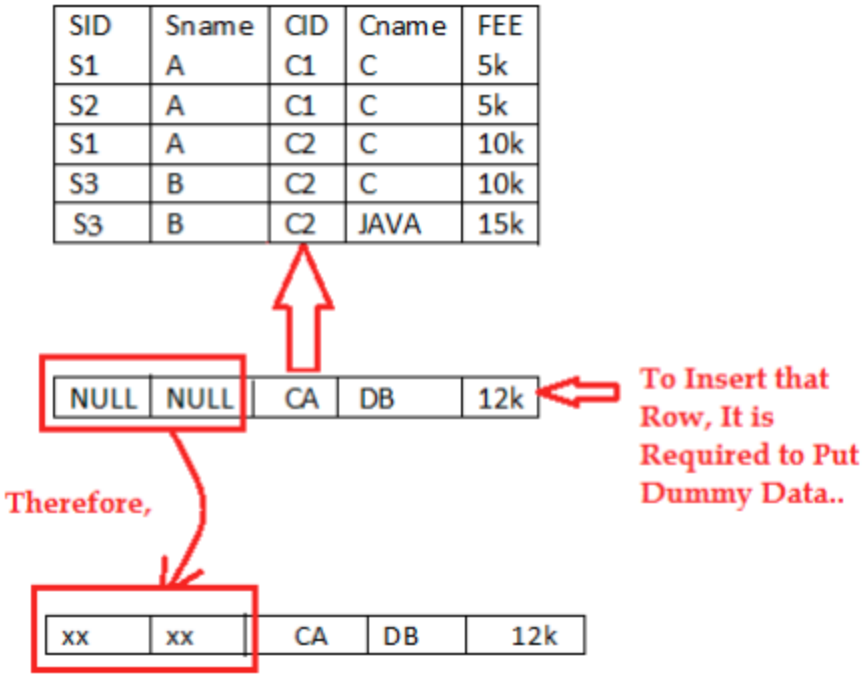
4.deletion anomalies: It may not be possible to delete some information without losing some other information as well

Problem in updation / updation anomaly – If there is updation in the fee from 5000 to 7000, then we have to update FEE column in all the rows, else data will become inconsistent.

SID	Sname	CID	Cname	FEE
S1	A	C1	C	~~5k~~
S2	A	C1	C	~~5k~~
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

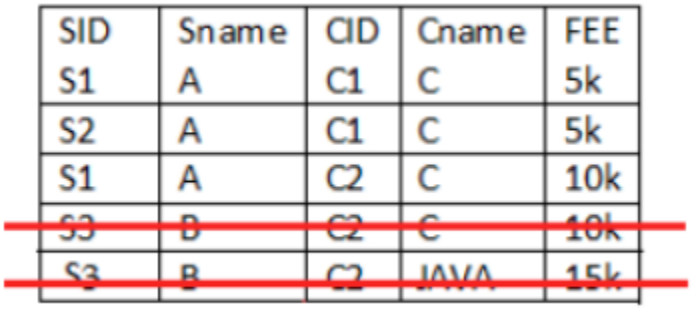
Insertion Anomaly and Deletion Anomaly- These anomalies exist only due to redundancy, otherwise they do not exist.

Insertion Anomalies: New course is introduced C4, But no student is there who is having C4 subject.

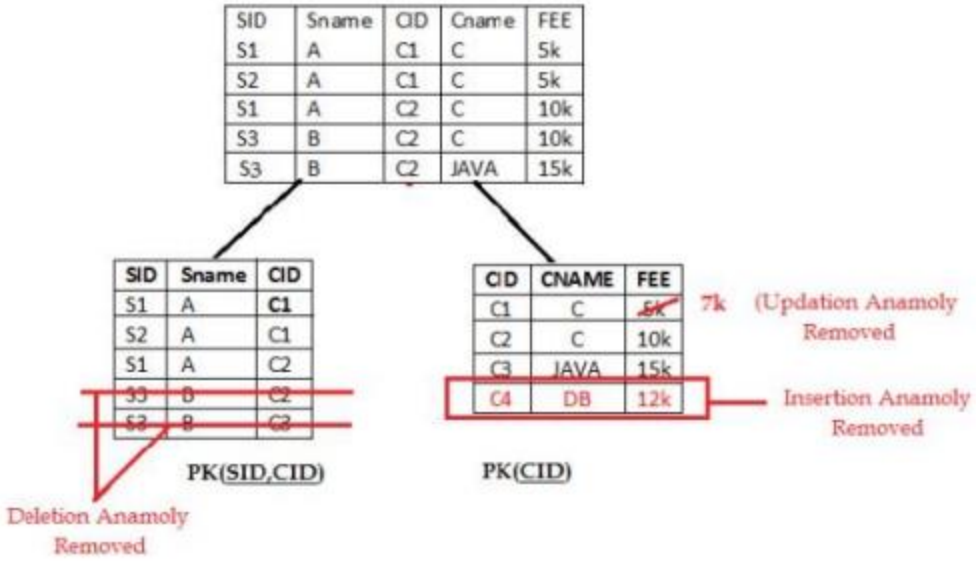


Because of insertion of some data, It is forced to insert some other dummy data.

Deletion Anomaly: Deletion of S3 student cause the deletion of course. Because of deletion of some data forced to delete some other useful data.



Solutions To Anomalies : Decomposition of Tables – Schema Refinement, shown below.



Purpose of Normalization:

➢ Minimize the redundancy in data.

➢ Remove insert, update, and delete anomalies during the database activities.

➢ Reduce the need to organize the data when it is modified or enhanced.

➢ Normalization reduces a complex user view to a set of small and sub groups of fields or relations. This process helps to design a logical data model known as conceptual data model.

Advantages of Normalization:

1. Greater overall database organization will be gained.

2. The amount of unnecessary redundant data reduced.

3. Data integrity is easily maintained within the database.

4. The database & application design processes are much for flexible.

5. Security is easier to maintain or manage.

Disadvantages of Normalization:

1. The disadvantage of normalization is that it produces a lot of tables with a relatively small number of columns. These columns then have to be joined using their primary/foreign key relationship.

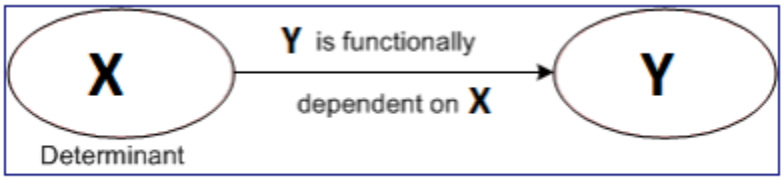
2. This has two disadvantages.

Performance: all the joins required to merge data slow processing & place additional stress on your hardware.

Complex queries: developers have to code complex queries in order to merge data from different tables.

Concept of Functional Dependency:

- Functional Dependencies are fundamental to the process of Normalization i.e., Functional Dependency plays key role in differentiating good database design from bad database designs.
- A functional dependency is a "type of constraint that is a generalization of the notation of the key".
- Functional Dependency describes the relationship between attributes (columns) in a table. Functional dependency is represented by an arrow sign (→).
- In other words, a dependency FD: "X → Y" means that the values of Y are determined by the values of X.
- Two tuples sharing the same values of X will necessarily have the same values of Y.
- An attribute on left hand side is known as "Determinant". Here X is a Determinant.



Functional dependency between X and Y

Reasoning about functional dependencies:
Armstrong Axioms (Inference Rules) :

- The term Armstrong axioms refers to the sound and complete set of inference rules or axioms, introduced by William W. Armstrong, that is used to test logical implication of functional dependencies.
- Armstrong axioms define the set of rules for reasoning about functional dependencies and also to infer all the functional dependencies on a relational database.

Various axioms rules or inference rules:
Primary axioms:

Rule 1	**Reflexivity** If A is a set of attributes and B is a subset of A, then A holds B. { A → B }
Rule 2	**Augmentation** If A hold B and C is a set of attributes, then AC holds BC. {AC → BC} It means that attribute in dependencies does not change the basic dependencies.
Rule 3	**Transitivity** If A holds B and B holds C, then A holds C. If {A → B} and {B → C}, then {A → C} A holds B {A → B} means that A functionally determines B.

Secondary or derived axioms:

Rule 1	**Union** If A holds B and A holds C, then A holds BC. If{A → B} and {A → C}, then {A → BC}
Rule 2	**Decomposition** If A holds BC and A holds B, then A holds C. If{A → BC} and {A → B}, then {A → C}
Rule 3	**Pseudo Transitivity** If A holds B and BC holds D, then AC holds D. If{A → B} and {BC → D}, then {AC → D}

Closure of a Set of Attributes:

- Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.
- The set of FD's that is logically implied by F is called the closure of F and written as F+. And it is defined as "If F is a set FD's on a relation R, the F+, the closure of F by using the inferences axioms that are not contained in F+.
- **Example:** ***R (A, B, C, D) and set of Functional Dependencies are A→B, B→D, C→B then what is the Closure of A, B, C, D?***
  Solution: A+ is A+→ {A, B, D} i.e., A→B, B→D is exists and C is not FD on A. So it is eliminated. B+→ { B, D} i.e., B→D is exists and A, C is not FD on A. So it is eliminated.
  C+→ {C, B, D} i.e., C→B, B→D is exists and A is not FD on C. So it is eliminated.

Types of functional dependencies:

1. **Fully Functional Dependency:**
   A functional dependency is said to be full dependency "if and only if the determinant of the functional dependency if either candidate key or super key, and the dependent can be either prime or non-prime attribute".
   (OR)
   Let's take the functional dependency X → Y (i.e., X determines y). Here Y is said to be fully determinant, if it cannot determine any subset of X.
   Example: Consider the following determinant ABC → D i.e., ABC determines D but D is not determined by any subset of A/ BC/C/B/AB i.e., BC→D, C→D, A→D Functional dependencies are not exists. So D is Fully Functional Dependent.

2. **Partial Functional Dependency:**
   If a non-prime attribute of the relation is getting derived by only a part of the candidate key, then such dependency is known as Partial Dependency.
   (OR)
   In a relation having more than one key field, a subset of non key fields may depend on all key fields but another subset or a particular non-key field may depend on only one of the key fields. Such dependency is defined as Partial Dependency.
   Example: Consider the following determinants AC→P, A→D, D→P. From these determinants P is not fully FD on AC. Because, If we find A+ (means A's Closure) A→D, D→P i.e., A→P. But we don't have any requirement of C. C attribute is removed completely. So P is Partially

Dependent on AC. Under the following conditions a table cannot have partial F.D (1) If primary key consists a single attribute (2) If table consists only two attributes (3) If all the attributes in the table are part of the primary key.

3. **Transitive Functional Dependency:**
   If a non-prime attribute of a relation is getting derived by either another non-prime attribute or the combination of the part of the candidate key along with non-prime attribute, then such dependency is defined as Transitive dependency. i.e., in a relation, there may be dependency among non-key fields. Such dependency is called Transitive Functional Dependency.
   Example: X→Y, and Y→Z then we can determine X→Z holds. Under the following Circumstances, a table cannot have transitive F.D
   (1) If table consists only two attributes
   (2) If all the attributes in the table are part of the primary key.

4. Trivial Functional Dependency:

It is basically related to Reflexive rule. i.e., if X is a set of attributes, and Y is subset of X then X→Y holds. Example: ABC→BC is a Trivial Dependency.

5. **Multi-Valued Dependency:**
   Consider 3 fields X, Y, and Z in a relation. If for each value of X, there is a well-defined set of values Y and Well-defined set of values of Z and set of values of Y is independent of the set values of Z. This dependency is Multi-valued Dependency. i.e., X →Y / Z.

Operations performed functional dependencies (applications of closure set of attributes):

(1) To identify the additional F.D's.

(2) To identify the keys.

(3) To identify the equivalences of the F.D's

(4) To identify irreducible set (minimal set) of F.D's or canonical forms of F.D's or standard form of F.D's.

(1) To identify the additional F.D's :

To check any F.D's like A→B can be determined from F1 or not. Complete A+ from F1 is A+ includes B also then; A→B can be derived as a F.D in F1.

Example: In a schema with attributes A,B,C,D and E the following set of attributes are given A→B, A→C, CD→E, B→D, E→A. Find CD→AC determines from the given FDs or not.
Sol: Given FD is CD→AC find the closure set of CD. CD+ = CDE (∵ CD→ E) = CDEA (∵ E→ A) = CDEAB (∵ A→ B) From the closure set the attributes AC are determined by CD so CD→ AC.

To practice: Check D→A can be derived from the following FDs or not AB→C, BC→AD, D→E, CF→B.

(2) Identification of key by using closure set as attributes:
A key attribute: An attribute that is capable of identifying all other attributes in a given table.

(i) Primary key: It is an unique value attribute in a table to enforce entity integrity and ti identify rows in the table uniquely.

(ii) Composite Primary Key: Sometimes single attribute is not sufficient to identify uniquely the rows in the table so, we combine 2 or more attributes to identify the rows uniquely.

(iii) Candidate keys: Sometimes 2 or more independent attribute or attributes can be used to identify the rows uniquely Eg :(vech no,veng no,purchase date)

Either vehicle no or vehicle engine no can be used as a key attribute then they are called as candidate keys one of the candidate key can be elected as primary key.

Example 1: Find candidate keys for the relation R(ABCD) having following FD's AB→CD, C→A, D→A.

Sol: From the given FD's, the attribute B is key attribute because it is not in RHS of functional dependency.

$B^+ = B$ (not a candidate key, find the combinations of B)

$AB^+ = ABCD$ ($\because AB \rightarrow CD$)

$BC^+ = BCAD$ ($\because C \rightarrow A, AB \rightarrow CD$)

$BD^+ = BDA$ ($\because D \rightarrow A$)

$CD^+ = CDA$ ($\because D \rightarrow A$)

$AC^+ = AC$

From the above attributes AB and BC determines all attributes. AB, BC are candidate keys.

Example 2: Find candidate keys for the relation R(ABCDE) having following FD's A→BC, CD→E, B→D, E→A.

Sol: From the given FD's, no attribute is key attribute because all are in RHS of functional dependency. So check for all attributes of LHS

A^+ = ABC ($\because A \rightarrow BC$)
= ABCD ($\because B \rightarrow D$)
= ABCDE ($\because CD \rightarrow E$)

B^+ = BD ($\because B \rightarrow D$)

E^+ = EA ($\because E \rightarrow A$)
= EABC ($\because A \rightarrow BC$)
= EABCD ($\because B \rightarrow D$)

C^+ = C

D^+ = D

CD^+ = CDE ($\because CD \rightarrow E$)
= CDEA ($\because E \rightarrow A$)
= CDEAB ($\because A \rightarrow BC$)

BC^+ = BCD ($\because B \rightarrow D$)
= BCDE ($\because CD \rightarrow E$)
= BCDEA ($\because E \rightarrow A$)

From the above attributes A, E, CD and BC determines all attributes.
A, E, CD, BC are candidate keys.

(3) To identify equivalence of F.D's :

Different database designers may define different F.D's sets from the same requirements. To evaluate whether they are equivalent if we are able to derive all F.D's in G from F and vice versa

Q1 Consider the following two sets of FDs

$$F = \begin{matrix} A \rightarrow C \\ AC \rightarrow D \\ E \rightarrow AD \\ E \rightarrow H \end{matrix}$$

$$G = \begin{matrix} A \rightarrow CD \\ E \rightarrow AH \end{matrix}$$

Find the equivalence of two sets of FDs.

Sol:

Step 1: Take set F and enclose all FD's in G that can be derived from F.

$A \rightarrow CD$

A^+ from F

$=A$

$=AC$ ($\because A \rightarrow C$)

$=ACD$ ($\because AC \rightarrow D$)

$\therefore A \rightarrow CD$ can be derived from F

$E \rightarrow AH$

E+ from F

$=E$

$=EAD$

$=EADH$

$\therefore E \rightarrow AH$ can be derived from F

Step 2: Take set G and enclose all F.D's in F that can be derived from G. $A \rightarrow C$

A+ from G

$=A$

$=ACD$

$A \rightarrow C$ can be derived from G

$E \rightarrow AD$

E^+ from G

$=E$

$=EAH$

$=EAHCD$

$E \rightarrow AH$ & $E \rightarrow AD$ can be derived from G

$\therefore$ G and F are equivalent.

Q2 Consider two sets of FDs on the attributes ABCDE

$$F= \begin{matrix} B \rightarrow CD \\ AD \rightarrow E \\ B \rightarrow A \end{matrix}$$

$$G= \begin{matrix} B \rightarrow CDE \\ B \rightarrow ABC \\ AD \rightarrow E \end{matrix}$$

Find whether they are equivalent or not

(4) To identify the irreducible form of FD's /canonical Form (minimal cover):

We try to minimize the functional dependency. The minimize FD should be equivalent to original FD, Procedure to find minimal set:

Step 1: Have single attributes on the RHS for every FD.

Step 2: Evaluate all F.D's in step 1 for their necessity. If they are not necessary, remove them from the list.

Step 3: Evaluate the necessity of the LHS attributes in FD's obtained from step 2.If they are not necessary remove from FD.

Step 4: Apply the union rule for common to LHS attribute in the FD's obtained from step 3.Then we will get irreducible set.

Q1 Find the irreducible set from the following FDs

F=

$A \rightarrow B$
$C \rightarrow B$
$D \rightarrow ABC$
$AC \rightarrow D$

Sol:

Step 1:

(1) $A \rightarrow B$
(2) $C \rightarrow B$
(3) $D \rightarrow A$
(4) $D \rightarrow B$
(5) $D \rightarrow C$
(6) $AC \rightarrow D$

Step 2:

Remove 1 & compute A^+ from2, 3,4,5,6

$A^+=A$

∴We need 1

Remove 2 and compute 1, 3, 4, 5&6

$C^+=C$

∴We need 2.

Remove 3 and compute D^+ from 1, 2, 4, 5&6

$D^+=DBC$

∴We need 3.

Remove 3 and compute D^+ from 1, 2, 4, 5&6

Remove 4 and compute D+ from 1, 2, 4, 5&6

$D^+=ADCB$

∴D→B can be removed.

Remove 5 and compute D^+ from 1, 2, 3,4&6

$D^+=ABD$

∴We need 5.

Remove 6 and compute D^+ from 1, 2,3, 4, 5

$AC^+=ACB$

∴We need 6.

Step 3:

$A \rightarrow B$
$C \rightarrow B$
$D \rightarrow A$
$D \rightarrow C$
$AC \rightarrow D$

Remove A

$A \rightarrow B$	$A \rightarrow B$
$C \rightarrow B$	$C \rightarrow B$
$D \rightarrow A$	$D \rightarrow A$
$D \rightarrow C$	$D \rightarrow C$
$C \rightarrow D$	$AC \rightarrow D$
$C^+=CDAB$	$C^+=CB$

$\therefore C^+ \not\equiv C^+$

Remove C

$A \rightarrow B$	$A \rightarrow B$
$C \rightarrow B$	$C \rightarrow B$
$D \rightarrow A$	$D \rightarrow A$
$D \rightarrow C$	$D \rightarrow C$
$A \rightarrow D$	$AC \rightarrow D$
$A^+=ADCB$	$A^+=AB$

$A^+ \not\equiv A^+$

Step 4:

$A \rightarrow B$
$C \rightarrow B$
$D \rightarrow A$
$D \rightarrow C$
$AC \rightarrow D$

$A \rightarrow B$
$C \rightarrow B$
$D \rightarrow AC$
$AC \rightarrow D$

Therefore, it is an irreducible F.D.

To Practice:

Q2 Consider Universal relation with attributes ABC and FDs

AB→C

C→B

A→B

Find the Irreducible set

Normal forms based on functional dependency (1NF, 2NF and 3 NF, Boyce Codd normal form (BCNF), 4NF)

Normalization means "split the tables into small tables which will contain less number of attributes in such a way that table design must not contain any problem of inserting, deleting, updating anomalies and guarantees no redundancy".

The evolution of Normalization theories / Steps of Normalization / Different Normal Forms is illustrated below

1. First Normal Form (1NF)

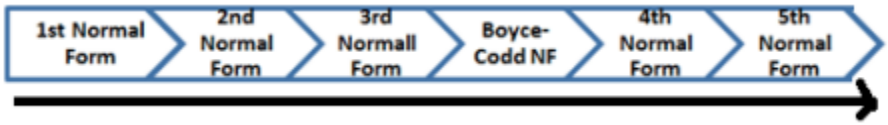
2. Second Normal Form (2NF)

3. Third Normal Form (3NF)

4. Boyce-Codd Normal Form (BCNF)

5. Fourth Normal Form (4NF)

6. Fifth Normal Form (5NF).



Points to be Remember

➢ 1 NF is a mandatory NF and remaining are the optional

➢ If you construct E-R diagrams in to the tables, then 4 NF and 5 NF need not be applied on the table.

➢ Practically applied normalization is upto 3NF and very rarely we will go beyond that.

➢ 2 NF dealing with the partial dependencies and 3NF is dealing with transitive dependencies.

First Normal Form (1NF):

A relation is said to in the 1NF if it is already in un-normalized form and it satisfies the following conditions or rules or qualifications are:

1. Each attribute name must be unique.

2. Each attribute value must be single or atomic i.e., Single Valued Attributes.

3. Each row / record must be unique.

4. There is no repeating group's.

Example: How do we bring an un-normalized table into first normal form? Consider the following relation:

TABLE_PRODUCT

Product ID	Color	Price
1	red, green	15.99
2	yellow	23.99
3	green	17.50
4	yellow, blue	9.99
5	red	29.99

Solution: This table is not in first normal form because the [Color] column can contain multiple values. For example, the first row includes values "red" and "green." To bring this table to first normal form, we split the table into two tables and now we have the resulting tables:

TABLE_PRODUCT_PRICE

Product ID	Price
1	15.99
2	23.99
3	17.50
4	9.99
5	29.99

TABLE_PRODUCT_COLOR

Product ID	Color
1	red
1	green
2	yellow
3	green
4	yellow
4	blue
5	red

Second Normal Form (2NF):

A relation is said to be in 2NF, if it is already in 1st NF and it has no Partial Dependency i.e., no non-prime attribute is dependent on the only a part of the candidate key.

(OR)

A relation is in second normal form if it satisfies the following conditions:

• It is in first normal form

• All non-key attributes are fully functional dependent on the primary key.

Partial Functional Dependency: If a non-prime attribute of the relation is getting derived by only a part of the candidate key, then such dependency is known as Partial Dependency.

Example: Consider the following relation

TABLE_PURCHASE_DETAIL

Customer ID	Store ID	Purchase Location
1	1	Los Angeles
1	3	San Francisco
2	1	Los Angeles
3	2	New York
4	3	San Francisco

➔This table has a composite primary key [Customer ID, Store ID]. The non-key attribute is [Purchase Location]. In this case, [Purchase Location] only depends on [Store ID], which is only part of the primary key. Therefore, this table does not satisfy second normal form.

➔ To bring this table to second normal form, we break the table into two tables, and now we have the following:

TABLE_PURCHASE

Customer ID	Store ID
1	1
1	3
2	1
3	2
4	3

TABLE_STORE

Store ID	Purchase Location
1	Los Angeles
2	New York
3	San Francisco

Third Normal Form (3NF):

A database is in third normal form if it satisfies the following conditions:

• It is in 2NF.

• There is no transitive functional dependency

➢ By transitive functional dependency, we mean we have the following relationships in the table: A is functionally dependent on B, and B is functionally dependent on C. In this case, C is transitively dependent on A via B. and A non-key attribute is depending on a non-key attribute.

Example: Consider the following relation

TABLE_BOOK_DETAIL

Book ID	Genre ID	Genre Type	Price
1	1	Gardening	25.99
2	2	Sports	14.99
3	1	Gardening	10.00
4	3	Travel	12.99
5	2	Sports	17.99

➔ In the table able, [Book ID] determines [Genre ID], and [Genre ID] determines [Genre Type]. Therefore, [Book ID] determines [Genre Type] via [Genre ID] and we have transitive functional dependency, and this structure does not satisfy third normal form.

➔ To bring this table to third normal form, we split the table into two as follows:

TABLE_BOOK

Book ID	Genre ID	Price
1	1	25.99
2	2	14.99
3	1	10.00
4	3	12.99
5	2	17.99

TABLE_GENRE

Genre ID	Genre Type
1	Gardening
2	Sports
3	Travel

Boyce-Codd normal form (BCNF):

A relation is said to be in BCNF, if and only if every determinant should be a candidate key.

✓ BCNF is the advance version of 3NF. It is stricter than 3NF.

✓ A table is in 3NF if for every functional dependency X → Y, X is the super key of the table.

✓ For BCNF, the table should be in 3NF and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department.
EMPLOYEE table:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

➔ In the above table Functional dependencies are as follows: EMP_ID → EMP_COUNTRY and EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO} Candidate key: {EMP-ID, EMP-DEPT}

➔ The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys. To convert the given table into BCNF, we decompose it into three tables

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
283	D394	300
Stores	D283	232
549	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Fourth Normal Form (4NF):

A relation said to be in 4NF if it is in Boyce Codd normal form and should have no multi-valued dependency.

✓ For a dependency A→ B, if for a single value of A, multiple value of B exists then the relation will be multi-valued dependency.

✓ Note: **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.

Example

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

➢ The given STUDENT table is in 3NF but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY. In the STUDENT relation, student with STU_ID, 21 contains two courses, Computer and Math and two hobbies, Dancing and Singing. So there is a Multi-valued dependency on STU_ID, which leads to un-necessary repetition of data.

➢ So to make the above table into 4NF, we can decompose it into two tables:

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Fifth normal form (5NF)

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

Example:

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

Lossless join and Dependency preserving decomposition:

Decomposition: Decomposition of a relation is done when a relation in relational model is not in appropriate normal form. Relation R is decomposed into two or more relations if decomposition is lossless join as well as dependency preserving.

➔ If we decompose a relation R into relations R1 and R2, There are 2 types of decomposition:

1. **Decomposition is lossy**, if **R1 ⋈ R2 ⊃ R.**
   - ✓ The decomposition of relation R into R1 and R2 is **lossy** when the join of R1 and R2 does not yield the same relation as in R." One of the disadvantages of decomposition into two or more relational schemes (or tables) is that some information is lost during retrieval of original relation or table.
   - ✓ Consider that we have table STUDENT with three attribute roll_no , sname and department.

Roll_no	Sname	Dept
111	parimal	COMPUTER
222	parimal	ELECTRICAL

- ✓ This relation is decomposed into two relation no_name and name_dept:

Roll_no	Sname
111	parimal
222	parimal

Sname	Dept
parimal	COMPUTER
parimal	ELECTRICAL

- ✓ In lossy decomposition, spurious tuples are generated when a natural join is applied to the relations in the decomposition.

stu_joined :

Roll_no	Sname	Dept
111	parimal	COMPUTER
111	parimal	ELECTRICAL
222	parimal	COMPUTER
222	parimal	ELECTRICAL

- ✓ The above decomposition is a bad decomposition or Lossy decomposition.

2. **Lossless Join Decomposition:** Decomposition is lossless if **R1 ⋈ R2 = R**
   - ✓ This is also referred as **non-additive decomposition**.
   - ✓ The lossless-join decomposition is always defined with respect to a specific set F of dependencies
   - ✓ **To check for lossless join decomposition using FD set, following conditions must hold:**
     1. Union of Attributes of R1 and R2 must be equal to attribute of R. Each attribute of R must be either in R1 or in R2.

        Att(R1) U Att(R2) = Att(R).
     2. Intersection of Attributes of R1 and R2 must not be NULL.

        Att(R1) ∩ Att(R2) ≠ Φ.
     3. Common attribute must be a key for at least one relation (R1 or R2)

        Att(R1) ∩ Att(R2) -> Att(R1) or Att(R1) ∩ Att(R2) -> Att(R2).

   - ✓ **Example 1:,** A relation R (A, B, C, D) with FD set{A->BC} is decomposed into R1(ABC) and R2(AD) which is a lossless join decomposition as:
     1. First condition holds true as Att(R1) U Att(R2) = (ABC) U (AD) = (ABCD) = Att(R).
     2. Second condition holds true as Att(R1) ∩ Att(R2) = (ABC) ∩ (AD) ≠ Φ
     3. Third condition holds true as Att(R1) ∩ Att(R2) = A is a key of R1(ABC) because A->BC is given.

✓ **Example 2:** Consider that we have table STUDENT with three attribute roll_no, sname and department.
STUDENT:

Roll_no	Sname	Dept
111	parimal	COMPUTER
222	parimal	ELECTRICAL

This relation is decomposed into two relation Stu_name and Stu_dept:

Roll_no	Sname
111	parimal
222	parimal

Roll_no	Dept
111	COMPUTER
222	ELECTRICAL

Now, when these two relations are joined on the comman column 'roll_no' ,the resultant relation will look like stu_joined.

Roll_no	Sname	Dept
111	parimal	COMPUTER
222	parimal	ELECTRICAL

In lossless decomposition, no any spurious tuples are generated when a natural joined is applied to the relations in the decomposition.

➢ **Dependency Preserving Decomposition:**

- ✓ If we decompose a relation R into relations R1 and R2, All dependencies of R either must be a part of R1 or R2 or must be derivable from combination of FD's of R1 and R2.
- ✓ The dependency preservation decomposition is another property of decomposed relational database schema D in which each functional dependency X -> 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 appear in some R_i.
- ✓ Decomposition D = {R_1, R_2, R_3,.., ,R_n} of R is said to be dependency-preserving with respect to F if the union of the projections of F on each R_i , in D is equivalent to F. In other words, R ⊂ join of R_1, R_i over X. The dependencies are preserved because each dependency in F represents a constraint on the database. If decomposition is not dependency-preserving, some dependency is lost in the decomposition.
- ✓ **Example 1:** A relation R (A, B, C, D) with FD set {A->BC} is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of R1(ABC).
- ✓ **Example 2:** Let a relation R(A,B,C,D) and set a FDs F = { A -> B , A -> C , C -> D} are given. A relation R is decomposed into –

```
R1 = (A, B, C) with FDs F1 = {A -> B, A -> C}, and
R2 = (C, D) with FDs F2 = {C -> D}.
     F' = F1 ∪ F2 = {A -> B, A -> C, C -> D}
      So, F' = F.
     And so, F'+ = F+.
```

Thus, the decomposition is dependency preserving decomposition.