Normalization or Schema Refinement or Database design

- Normalisation 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.
- The Schema Refinement refers to refine the schema by using some technique. The best technique of schema refinement is decomposition.
- The Basic **Goal of Normalisation** is used to **eliminate redundancy**.
- Redundancy refers to repetition of same data or duplicate copies of same data stored in different locations.

Normalization is used for mainly two purpose :

- Eliminating redundant(useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

Anomalies or Problems Facing without Normalisation :

Anomalies refers to the problems occurred after poorly planned and unnormalised databases where all the data is stored in one table which is sometimes called a flat file database. Let us consider such type of schema -

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
Primary Key(SID,CID)				

Here all the data is stored in a single table which causes redundancy of data or say anomalies as SID and Sname are repeated once for same CID . Let us discuss anomalies one bye one.

Types of Anomalies : (Problems because of Redundancy)

There are three types of Anomalies produced in the database because of redundancy -

- **Updation/Modification Anomaly**
- **Insertion Anomaly**
- **Deletion Anomaly**

1. **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

7k
7k
Costly Operation
More IO Cost

2. **Insertion Anomaly and Deleteion Anomaly-** These anamolies exist only due to redundancy, otherwise they do not exist.
   - **Insertion Anomaly :** 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.

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

NULL	NULL	CA	DB	12k

To Insert that Row, It is Required to Put Dummy Data..

Therefore,

xx	xx	CA	DB	12k

   - **Problem/Disadvantage to Insert Dummy Data -** It results inconsistency. **how?**
     Solution) Suppose if we want to know the number of students, then answer will be, 4 (S1,S2,S3, xx)
     Why we eliminate redundancy or what is the use of eliminating redundancy ?
     Solution) It is not actually the storage problem. The problem is anomalies as shown above as it gives inconsistent answers/ wrong answers.

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

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~~

Deleting student S3 will permanently delete the course B.

Solutions To Anomalies : Decomposition of Tables - Schema Refinement

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

SID	Sname	CID
S1	A	C1
S2	A	C1
S1	A	C2
~~S3~~	~~B~~	~~C2~~
~~S3~~	~~B~~	~~C3~~

PK(SID,CID)

Deletion Anamoly Removed

CID	CNAME	FEE
C1	C	~~5k~~ 7k
C2	C	10k
C3	JAVA	15k
C4	DB	12k

PK(CID)

(Updation Anamoly Removed

Insertion Anamoly Removed

There are some Anomalies in this again -

SID	Sname	CID
S1	~~A~~ (AA)	C1
S2	~~A~~ (AA)	C1
S1	~~A~~ (AA)	C2
S3	B	C2
S3	B	C3
S4	B	xx

Updation Anamoly

A student having no course is enrolled. We have to put dummy data again.

Deletion Anamoly as C2 course is alloted to some students

CID	CNAME	FEE
C1	C	5k
~~C2~~	~~C~~	~~10k~~
C3	JAVA	15k
C4	DB	12k

What is the Solution ?? Solution :

R1

SID	Sname

R2

SID	CID

R3

CID	Cname	Fee

Lossy Join Decomposition :

The best refinement technique for the schema design is Decomposition. Decomposition refers to decompose or break-down of the relational-schema that has many attributes into several schemas with fewer attributes. We should take care some desirable properties while doing decomposition. If we do the careless decomposition, then it will lead to a bad design again.

Careless Decomposition or Lossy Join Decomposition :

Consider a Schema relation which has many attributes and results into redundancy. Let us apply Decomposition

Supplier_Parts :

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10

⇓

Parts :

P#	Qty
301	20
500	50
20	10
400	40
301	10

Supplier :

S#	Sname	City	Qty
3	Smith	London	20
5	Nick	NY	50
2	Steve	Boston	10
5	Nick	NY	40
5	Nick	NY	10

The above decomposition is a careless decomposition or Lossy join Decomposition. Because, Let us apply natural join operation on the decomposed relations.

Parts ⋈ Supplier

Parts ⋈ Supplier :

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
5	Nick	NY	20	10
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10
2	Steve	Boston	301	10

≠ Supplier_Parts

Although, every tuple that appears in the **Supplier_Parts** relation appears in **Parts ⋈ Supplier,** there are tuples in **Parts ⋈ Supplier** that are not in **Supplier_Parts.** The spurious tuples or the extra tuples that are not in the relation are :

- (5, Nick, NY, 20, 10)
- (2, Steve, Boston, 301, 10)

A closer look on the relation **Parts ⋈ Supplier,** will lead to wrong data and therefore, we have less and misleading information.

Definition of Lossy Join Decomposition :

Let **R** be the relational schema with instance **r** is decomposed into $\mathbf{R_{1,R2,\ldots,Rn}}$ with instance $\mathbf{r_1, r_2, \ldots, r_n}$. If $\mathbf{r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n \supset r}$, then it is called Lossy Join Decomposition. i.e. if the original relation is the proper subset of natural joins of all the decompositions, then it is said to be Lossy Join Decomposition. In the above example, we can say that, Supplier_Parts is the subset of natural join of parts and supplier, so we can say that **Parts ⋈ Supplier** ⊃ **Supplier_Parts** and therefore, the decomposition is lossy join decomposition.

Why Lossy Join Decomposition is called Lossy although the relation is getting extra tuples ?

⇒ Because we are loosing original Data.

In short, we design such system such that these undesirable properties do not occur in decomposition.

Normal Forms

The normal forms defined in relational database theory represent guidelines for database design. The normalization rules are designed to prevent update anomalies and data inconsistencies. Normalization rules are divided into following normal form :

1. First Normal Form (1 NF)
2. Second Normal Form (2 NF)
3. Third Normal Form (3 NF)
4. Boyce-Codd Normal Form (BCNF)
5. Multivalued Dependencies and Fourth Normal Form (4 NF)
6. Join Dependencies and Fifth Normal Form (5 NF)

A relation is said to be in a particular normal form if it satisfies a certain specified set of constraints.

First Normal Form (1 NF) :

A relation is in first Normal Form if and only if all underlying domains contain atomic values only. In other words, a relation doesn't have multivalued attributes. For example : Consider a **STUDENT(Sid, Sname, Cname)** relation

Student :

SID	Sname	Cname
S1	A	C,C++
S2	B	C++,DB
S3	A	DB

SID : Primary Key

⇐ Due to occurrence of MVA, the above relation is not in 1 NF.

⇓ **Solution :** Removal of MVA by inserting more rows

Student :

SID	Sname	Cname
S1	A	C
S1	A	C++
S2	B	C++
S2	B	DB
S3	A	DB

SID : Primary Key

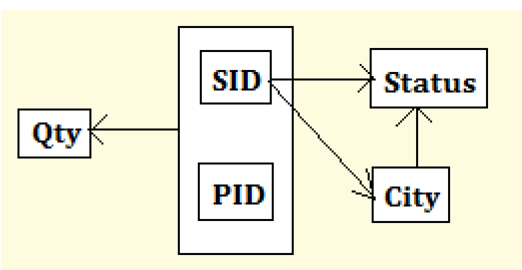
⇐ The relation is in 1nF

Another Example :

Consider another relation **Supplier(SID, Status, City, PID, Qty).**

Supplier :				
SID	**Status**	**City**	**PID**	**Qty**
S1	30	Delhi	P1	100
S1	30	Delhi	P2	125
S1	30	Delhi	P3	200
S1	30	Delhi	P4	130
S2	10	Karnal	P1	115
S2	10	Karnal	P2	250
S3	40	Rohtak	P1	245
S4	30	Delhi	P4	300
S4	30	Delhi	P5	315
Key : (SID, PID)				

Let us assume each supplier has unique SID, and have exactly one Status code and Location(City) and further, Status is functionally dependent on City. A supplier can supply different parts(PID). Key of Supplier relation is the combination of (SID, PID). The **Functional Dependency Diagram** for Supplier relation is
shown as : As there are no multivalued attributes(MVA), hence the Supplier relation is already in 1 NF. But the Supplier relation has anomalies again which are :



Drawback of 1 NF :

1. **Anomalies :**
   - **Deletion Anomaly -** If we delete the tuple <S3,40,Rohtak,P1,245> , then we loose the information about S3 that S3 lives in Rohtak.
   - **Insertion Anomaly -** We cannot insert a Supplier S5 located in Karnal, until S5 supplies atleast one part.
   - **Updation Anomaly -** If Supplier S1 moves from Delhi to Kanpur, then it is difficult to update all the tuples containing (S1, Delhi) as SID and City respectively.

2. Normal Forms are the methods of reducing redundancy. However, Sometimes 1 NF increases redundancy. It does not make any efforts in order to decrease redundancy.

Possibilities of Redundancy in 1 NF :

a) When LHS is not a Superkey :

Let X → Y is a non trivial FD over R with X is not a superkey of R, then redundancy exist between X and Y attribute set. Hence in order to identify the redundancy, we need not to look at the actual data, it can be identified by given functional dependency. Example : X →Y and X is not a Candidate Key ⇒ X can duplicate ⇒ corresponding Y value would duplicate also.

X	Y
1	3
1	3
2	3
2	3
4	6

b) When LHS is a Superkey :

If X → Y is a non trivial FD over R with X is a superkey of R, then redundancy does not exist between X and Y attribute set. Example : X →Y and X is a Candidate Key ⇒ X cannot duplicate ⇒ corresponding Y value may or may not duplicate.

X	Y
1	4
2	6
3	4

2NF - Second Normal Form :

Relation **R** is in Second Normal Form (2NF) only iff :
R should be in 1NF and
R should not contain any Partial Dependency

What is a Partial Dependency ?

Let **R** be a relational Schema and **X,Y,A** be the attribute sets over **R**. **X:** Any Candidate Key **Y:** Proper Subset of Candidate Key **A:** Non Key AttributeIf Y → A exists in R, then R is not in 2 NF. **(Y → A)** is a Partial dependency only if
Y: Proper subset of Candidate Key
A: Non Prime Attribute

Removal of Partial Dependency
If there is any partial dependency, remove partially dependent attributes from original table, place them in a separate table along with a copy of its determinant.

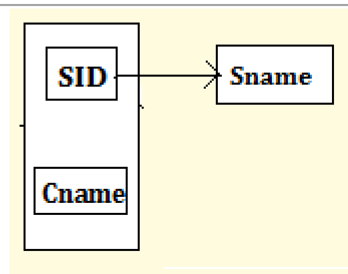
Example 1 :

Consider the relation **Student(SID, Sname, Cname)** which is in 1 NF (No Multi-Valued-Attributes) :

Student :

SID	**Sname**	**Cname**
S1	A	C
S1	A	C++
S2	B	C++
S2	B	DB
S3	A	DB

{SID,Cname} : Primary Key
Functional Dependencies:
{SID,Cname} → Sname
SID → Sname



Partial Dependencies : SID → Sname {as SID is a Proper Subset of Candidate Key {SID,Cname}.

Solution : Removal of Partial Dependency by creating separate table ⇓

R1 :

SID	**Sname**
S1	A
S2	B
S3	A

SID : Primary Key

R2 :

SID	**Cname**
S1	C
S1	C++
S2	C++
S2	DB
S3	DB

{SID,Cname} : Primary Key

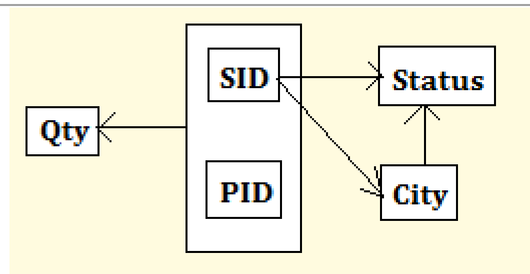
The above two relations R1 and R2
1. Lossless Join
2. 2NF
3. Dependency Preserving

Example 2:
Consider the relation **Supplier(SID, Status, City, PID, Qty)** which is in 1 NF (No Multi-Valued-Attributes) :

Supplier :

SID	**Status**	**City**	**PID**	**Qty**
S1	30	Delhi	P1	100
S1	30	Delhi	P2	125
S1	30	Delhi	P3	200
S1	30	Delhi	P4	130
S2	10	Karnal	P1	115
S2	10	Karnal	P2	250
S3	40	Rohtak	P1	245
S4	30	Delhi	P4	300
S4	30	Delhi	P5	315

Key : (SID, PID)



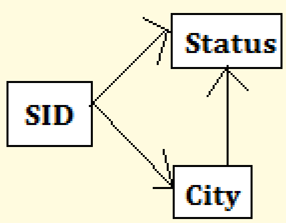
Partial Dependencies : SID → Status SID → City

Solution : Removal of Partial Dependency by creating separate table ⇓

Sup_City :

SID	Status	City

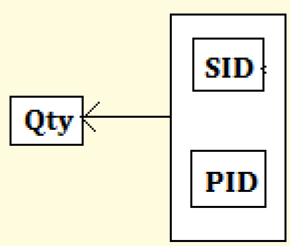
FDD of Sup_City :



Sup_Qty :

SID	PID	Qty

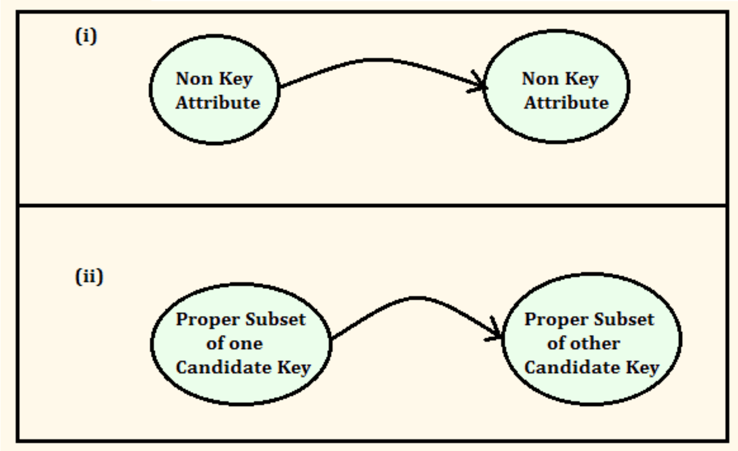
FDD of Sup_qty :



Drawback of 2NF
Anomalies in Relation { Sup_City } :
Deletion Anomaly - If we delete a tuple in Sup_City, then we not only loose the information about a supplier, but also loose the status value of a particular city.
Insertion Anomaly - We cannot insert a City and its status until a supplier supplies atleast one part.
Updation Anomaly - If the status value for a city is changed, then we will face the problem of searching every tuple for that city.

Possibilities of Redundancy in 2NF

However, there is less redundancy in 2NF rather than in 1 NF, but 2NF is not free from redundancy. The possibilities of redundancy in 2NF are :



<u>These two are the possibilities in 2NF which forms redundancy.</u>

The example of (i) is in the Sup_City relation :
City → Status {Non Key Attribute → Non Key Attribute}

The example of (ii) is in the STUDENT relation :
SID → Cname {Proper Subset of 1 CK → Proper Subset of other CK}

<u>Some Points regarding 2NF</u> :

The table is automatically in 2NF if primary key consists of only one attribute or all attributes are part of primary key or table consists of only two attributes.

3NF - Third Normal Form

Let **R** be the relational schema, **R** is in 3NF only if :

- **R** should be in <u>2NF</u>.
- **R** should not contain transitive dependencies.

What is a Transitive Dependency ?
Let **R** be a relational Schema and **X,Y,Z** be the attribute sets over **R.** If **X** is functionally dependent on **Y (X → Y)** and **Y** is functionally dependent on **Z (Y → Z)** then **X** is transitive dependent on **Z (X → Z)**

Removal of Transitive Dependency

If there is any transitive dependency in the relation, then

- Create a separate relation and copy the dependent attribute along with a copy of its determinant. and remove these determinants from the original table.
- Mark dependent attribute as a foreign key in the original relation and Mark dependent attribute as a Primary key in the separate relation

Example of 3NF :

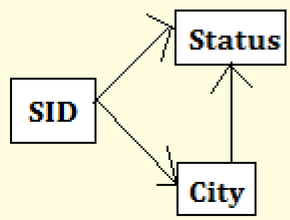
Consider the relation **Sup_City(SID, Status, City)** :

Sup_City :		
SID	Status	City
S1	30	Delhi
S2	10	Karnal
S3	40	Rohtak
S4	30	Delhi
SID : Primary Key		

Sup_City :

SID	Status	City

FDD of Sup_City :



Transitive Dependency : SID → City {As SID → City and City → Status}

Solution : Removal of Transitive Dependency by creating separate table ⇓

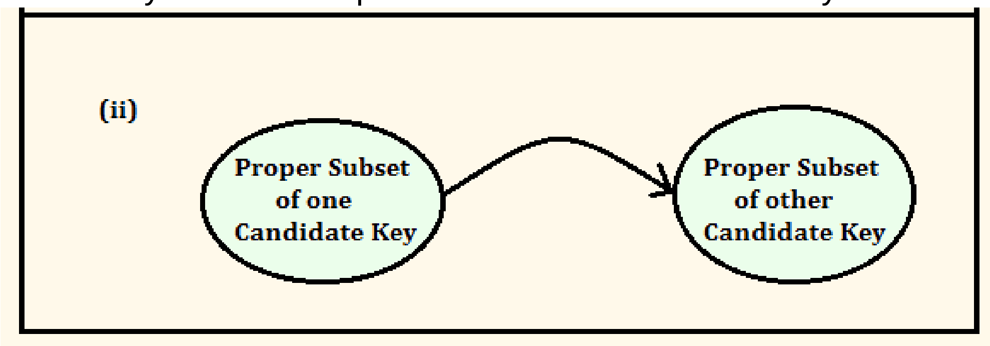
SC :	
SID	**City**
S1	Delhi
S2	Karnal
S3	Rohtak
S4	Delhi
SID : Primary Key	

CS :	
City	**Status**
Delhi	30
Karnal	10
Rohtak	40
City : Primary Key	

The relations SC and CS are in 3NF as they doesn't contain any transitive dependencies.

Possibilities of Redundancy in 3NF

However, there is less redundancy in 3NF than in 2 NF, but again 3NF is not free from redundancy. The possibilities of redundancy in 3NF are :



Some Points regarding 3NF :

1. A table is automatically in 3NF if one of the following hold :
   (i) If relation consists of two attributes.
   (ii) If 2NF table consists of only one non key attributes.

2. If X → A is a dependency, then the table is in the 3NF, if one of the following conditions exists :
   (i) If X is a superkey
   (ii) If X is a part of superkey

3. If X → A is a dependency, then the table is said to be NOT in 3NF if the following :
   (i) If X is a proper subset of some key (partial dependency)
   (ii) If X is not a proper subset of key (non key)

Classification of Dependencies in DBMS -

S.NO	**Classification of Dependencies**	**Which Normal Form Remove these Dependencies**
1	Partial Dependencies	Second Normal Form (2NF)
2	Transitive Dependencies	Third Normal Form (3NF)
3	Multivalued Dependencies	Fourth Normal Form (4NF)
4	Join Dependencies	Fifth Normal Form (5NF)
5	Inclusion Dependency	(Dependencies among the Relations/Tables or Databases)

Partial dependencies and Transitive Dependencies are types of Functional Dependencies.

Functional Dependency

A **Functional dependency** is a relationship between attributes. For example, if we know the value of customer account number, we can obtain customer address, balance etc. By this, we say that customer address and balance is functionally dependent on customer account number. In **general terms**, attribute Y(customer address and balance) is functionally dependent on the attribute X(customer account number), if the value of X determines the value of Y. For More about Functional Dependency - Click Here

Partial Functional Dependency –

A Functional Dependency in which one or more non key attributes are functionally depending on a part of the primary key is called partial functional dependency. or where the determinant consists of key attributes, but not the entire primary key, and the determined consist of non-key attributes.
For example, Consider a Relation R(A,B,C,D,E) having
FD : AB → CDE where PK is AB.

Then, { A → C; A → D; A → E; B → C; B → D; B → E }
all are Partial Dependencies.
To know more about Partial Dependency - Click Here

Transitive Dependency –

Given a relation R(A,B,C) then dependency like A–>B, B–>C is a transitive dependency, since A–>C is implied .

In the above Fig 1,
SSN --> DMGRSSN is a transitive FD
{since SSN --> DNUMBER and DNUMBER --> DMGRSSN hold}

SSN --> ENAME is non-transitive FD since there is no set of attributes X
where SSN --> X and X --> ENAME.
To know more about Transitive Dependency - Click Here

Multivalued Dependency

Consider a relation Faculty (FID, Course, Book) which consists of two multivalued attributes (Course and Book). The two multivalued attributes are independent of each other.

FID	Course	Book
1	C1/C2	B1/B2
2	C1	B1

⇒

FID	Course	Book
1	C1	B1
1	C1	B2
1	C2	B1
1	C2	B2
2	C1	B1

it is clear that there are multiple copies of the information about Course and Book. This is an example of a multivalued dependency which occurs when a relation has more than one independent, multivalued attribute. A multivalued dependency occurs when a relation R has attributes A(FID), B(Course), and C(Book) such that

- A determines a set of values for B
- A determines a set of values for C and
- B and C are independent of each other. (No relation between Course and Book)

These multivalued dependencies can be indicated as follows :

- (FID → → Course)
- (FID → → Book)

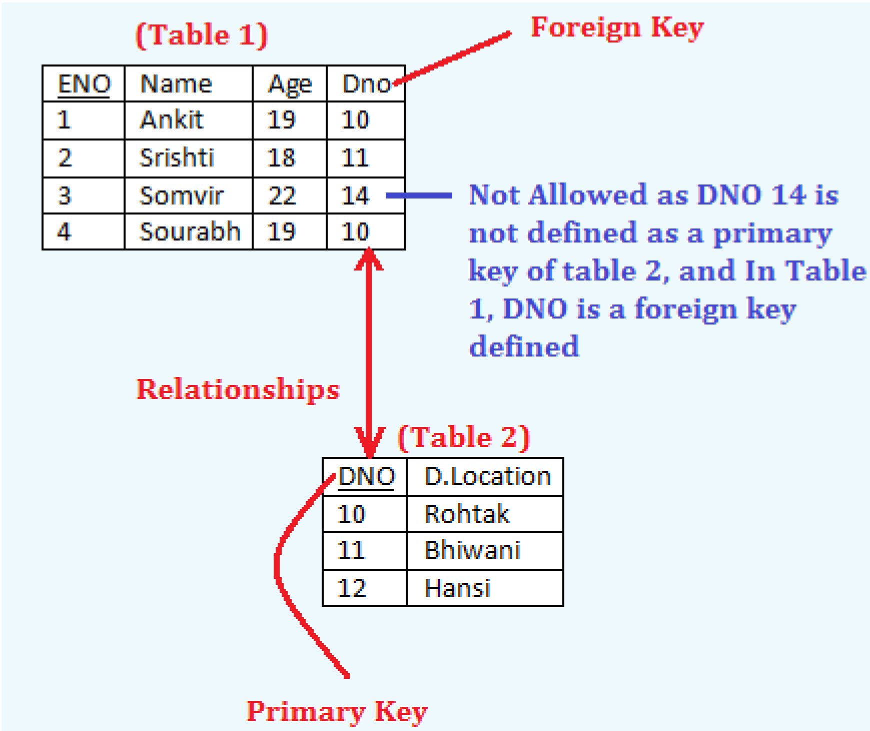
To know more about Multivalued Dependency - Click Here

Join Dependency

Let **R** be a relation. Let **A, B, ..., Z** be arbitrary subsets of **R's** attributes. **R** satisfies the **JD * (A, B, ..., Z)** if and only if **R** is equal to the join of its projections on **A, B, ..., Z.** A join dependency **JD(R1, R2, ..., Rn)** specified on relation schema **R,** is a trivial JD, if one of the relation schemas **Ri** in **JD(R1, R2, ...,Rn)** is equal to R.

Inclusion Dependencies

An inclusion dependency (shortly called as INDs) are the dependencies which exists when some columns of a relation are contained in other columns (usually of a second relation). The example of Inclusion dependency is a foreign key constraint or Referential Integrity Constraint as it states that the referring column(s) in one relation must be contained in the primary key column(s) of the referenced relation.



Objective of Inclusion Dependencies:

To formalize two types of interrelational constraints which cannot be expressed using F.D.s or MVDs:

- Referential integrity constraints
- Class/subclass relationships

Inclusion dependencies are mostly key-based, i.e. which involve only keys. Referential Integrity or Foreign key constraints are a good example of key-based inclusion dependencies. An ERD(er diagram) that involves ISA hierarchies also leads to key-based inclusion dependencies. If all inclusion dependencies are key-based then we rarely have to worry about splitting attribute groups that participate in inclusions, since decompositions usually do not split the primary key. Note that going from 3NF to BCNF always involves splitting some key, hopefully not the primary key, since the dependency guiding the split is of the form $X \rightarrow A$ where A is part of a key.

BCNF - Boyce Codd Normal Form in DBMS

Let **R** be the relational schema, **R** is in BCNF only if :
R should be in 3NF.
Every Functional Dependency will have a Superkey on the LHS or all determinants are the superkeys.

Example :

Consider the following relationship **R(ABCD)** having following functional dependencies

F = {A → BCD, BC → AD, D → B}		
Candidate Keys are : $(A)^+$ = {ABCD} $(BC)^+$ = {BCAD} $(DC)^+$ = {DCBA}		
Functional Dependency	**Is FD in BCNF or not ?**	**Reason ?**
A → BCD	Yes	A is a super key
BC → AD	Yes	BC is also a super key
D → A	No	D is not super key, it is part of key

Solution : Decomposition in BCNF

The relation **R(ABCD)** is decomposed into two relations **R1** and **R2** such that :
R1(A,D,C) R2(D,B)

The above two relations **R1** and **R2**
Lossless Join
BCNF Decomposition
But Not Dependency Preserving
Redundancy in BCNF
0% redundancy, Because of Single Valued Functional Dependency. Redundancy may exist because of Multivalued Dependency.

Some Notes Regarding BCNF

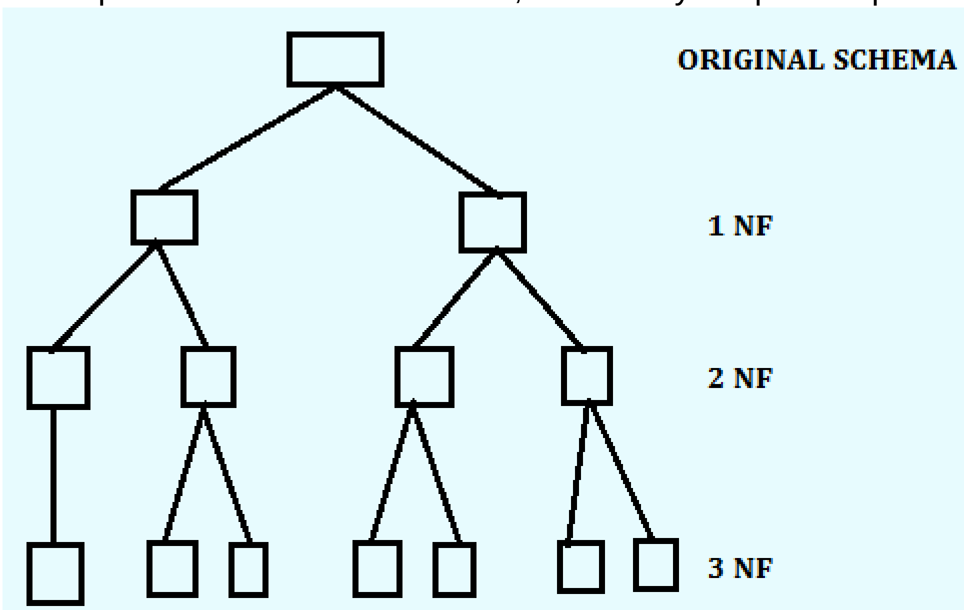
There is sometimes more than one BCNF decomposition of a given schema.
Some of the BCNF decompositions may also yield dependency preservation, while others may not.

Difference between 3NF and BCNF –

S.NO	**3NF**	**BCNF**
1.	It concentrates on Primary Key	It concentrates on Candidate Key.
2.	Redundancy is high as compared to BCNF	0% redundancy
3.	It may preserve all the dependencies	It may not preserve the dependencies.
4.	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

Desirable Properties of Decomposition -

If we apply the normal forms or normalization or schema refinement technique - Decomposition to the universal table, then it may be splitted up into different fragments.



At any stage, if we combine the fragments (denormalization), it should give the original table in terms of columns and rows and it will be described as the following properties :

- Losless Join Decomposition Property - Click
- Dependency Preserving Property - Click

Lossless Join Decomposition

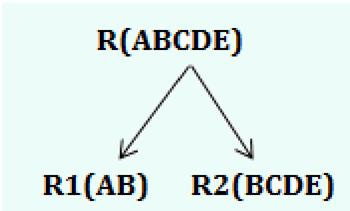
Definition 1 :

Let **R** be the relational schema with instance **r** is decomposed into $R_1, R_2, ..., R_n$ with instance $r_1, r_2, ..., r_n$. If $r_1 \bowtie r_2 \bowtie .... \bowtie r_n = r$, then it is called Lossless Join Decomposition. i.e. if natural joins of all the decompositions gives the original relation, then it is said to be Lossless Join Decomposition.

Definition 2: Another Definition or To check whether a Decomposition is a lossless or lossy decomposition -

Let **R** be a relation schema, **F** be a set of functional dependencies on **R.** Let **R** is decomposed in $R_1, R_2, ..., R_n$. The decomposition is a lossless-join decomposition of R if

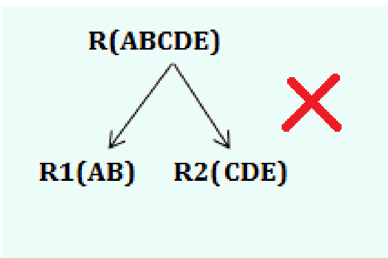
(a) $R_1 \cup R_2 \cup ...... \cup R_n \equiv R$



and

(b) Let R_i and R_j be the any two subrelations, R_i and R_j can be merge into single relation R_{ij} with attribute set $R_i \cup R_j$ only if

(i) $R_i \cap R_j \neq \Phi$



(ii) $R_i \cap R_j \rightarrow R_j$ {R_i and R_j should be super key of R_i} and
$R_i \cap R_j \rightarrow R_i$ {R_i and R_j should be super key of R_j}

(c) Repeat (a) untin N relations become single relation. If is possible to merge into single relation, then decomposition is losless, otherwise lossy.

Example : How to Find Lossless Join Decomposition -

Method 1 : (Not Useful for Gate Students)

Consider the previous example Supplier_Parts which is decomposed into supplier and parts relation but doing the decomposition in a different way :

Supplier_Parts :

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10

$\Downarrow$

Parts :

S#	P#	Qty
3	301	20
5	500	50
2	20	10
5	400	40
5	301	10

Supplier :

S#	Sname	City	Qty
3	Smith	London	20
5	Nick	NY	50
2	Steve	Boston	10
5	Nick	NY	40
5	Nick	NY	10

The above decomposition is a Lossless join Decomposition. Because, Let us apply natural join operation on the decomposed relations.

Parts ⋈ Supplier

Parts ⋈ Supplier :

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10

= Supplier_Parts

Hence the Decomposition is lossless join decomposition.

Method 2 : (Useful for Gate Students)

The above method is very time consuming for the **gate students**. The method 2 is very simple and fast. → Consider again the relation **Supplier_Parts**. Try to decompose the relation so that the common attribute in the tables is a key for atleast one table. Here,

In Supplier relation Supplier(S#,Sname,City):

<u>S#</u> → Sname

<u>S#</u> → City

In parts relation Parts(S#,P#,Qty) :

(<u>S#,P#</u>) → Qty

Let Supplier_Parts(S#,Sname,City,P#,Qty) = R
Supplier(<u>S#</u>,Sname,City) = R1
Parts(<u>S#,P#</u>,Qty) = R2

a) $R1 \cup R2 = R$ √ Satisfied
b)
(i) $R1 \cap R2 = S\# \neq \Phi$ √ Satisfied

(ii) $(S\#)^+ = \{S\#, Sname, City\} = R1 \rightarrow$ (Superkey)
$R1 \cap R2 \rightarrow R1$ √ Satisfied
{where R1 and R2 is superkey of R1}

(c) $\Rightarrow R_{12}$(S#,Sname,City,P#,Qty) = R √ Satisfied
$\Rightarrow$ becomes a single relation.

Hence all conditions are satisfied. So R is a Lossless Join Decomposition

Dependency Preserving Decomposition with Example

The second property of decomposition is **Dependency Preserving Decomposition**. If the original table is decomposed into multiple fragments, then somehow, we suppose to get all original FDs from these fragments. In other words, every dependency in original table must be preserved or say, every dependency must be satisfied by at least one decomposed table. Let R be the original relational schema having FD set F. Let R_1 and R_2 having FD set F_1 and F_2 respectively, are the decomposed sub-relations of R. The decomposition of R is said to be preserving if

$F_1 \cup F_2 \equiv F$ {Decomposition Preserving Dependency}
If $F_1 \cup F_2 \subset F$ {Decomposition NOT Preserving Dependency}
and $F_1 \cup F_2 \supset F$ {this is not possible}

How to find whether a decomposition is preserving dependency or not ?

Method 1 : (Not useful for gate students)

The Method 1 is an algorithm to find the preserving dependency.
<u>Algorithm</u> :
Input: $X \rightarrow Y$ in F and a decomposition of R $\{R_1, R_2, \ldots, R_n\}$
Output: return true if $X \rightarrow Y$ is in G^+, i.e., Y is a subset of Z else return false

```
begin
  Z := X;
  while changes to Z occur do
  for  i := 1 to n do
         Z := Z ∪ ((Z ∩ Ri)+ ∩ Ri) w.r.t. F;
  if Y is a subset of Z then return true
  else return false;
end;
```

R(ABCDEF) has following FD's
F = {A→BCD, A→EF, BC→AD, BC→E, BC→F, B→F, D→E}
D = {ABCD, BF, DE}
check whether decomposition is dependency preserving or not
Solution :
The following dependencies can be projected into the following decomposition

ABCD (R1)	BF (R2)	DE (R3)
A → BCD BC → AD	B → F	D → E

The FDs in the table are preserved also as they are projected in their corresponding decomposition.
Check whether BC → E and A → EF preserved in the decomposition or not
Apply the algorithm above :
<u>For BC → E :</u>
Let Z = BC

$(Z \cap R_1)$ = BC ∩ ABCD = BC // Intersection
$(Z \cap R_1)^+$ = $(BC)^+$ = BCADEF // Closure
$\{(Z \cap R_1)^+ \cap R_1\}$ = BCADEF ∩ ABCD = ABCD // Intersection
Z= $[\{(Z \cap R_1)^+ \cap R_1\} \cup Z]$ = ABCD ∪ BC = ABCD // Updating Z (Union)

Z does not contain E(RHS of FD- BC → E), Repeating procedure with R_2

Now, z = ABCD
$(Z \cap R_2)$ = ABCD ∩ BF = B // Intersection
$(Z \cap R_2)^+$ = B^+ = BF // Closure
$\{(Z \cap R_2)^+ \cap R_2\}$ = BF ∩ BF = BF // Intersection
Z= $[\{(Z \cap R_2)^+ \cap R_2\} \cup Z]$ = ABCD ∪ BF = ABCDF // Updating Z (Union)

Z does not contain E(RHS of FD- BC → E), Repeating procedure with R_3

Now Z = ABCDF
$(Z \cap R_3)$ = ABCDF ∩ DE = B // Intersection
$(Z \cap R_3)^+$ = D^+ = DE // Closure
$\{(Z \cap R_3)^+ \cap R_3\}$ = DE ∩ DE = BF // Intersection
Z= $[\{(Z \cap R_3)^+ \cap R_3\} \cup Z]$ = DE ∪ ABCDF = ABCDEF // Updating Z (Union)

Stopping the algorithm as Z contains E, So BC→E preserves dependency

<u>For A → EF :</u>
Let Z = A
$(Z \cap R_1)$ = A ∩ ABCD = A // Intersection

$(Z \cap R_1)^+$ = A^+ = ABCDEF // Closure
$\{(Z \cap R_1)^+ \cap R_1\}$ = ABCDEF ∩ ABCD = ABCD // Intersection
Z= $[\{(Z \cap R_1)^+ \cap R_1\} \cup Z]$ = ABCD ∪ A = ABCD // Updating Z (Union)

Z does not contain E and F(RHS of FD- A→EF), Repeating procedure with R2

Now, z = ABCD
$(Z \cap R_2)$ = ABCD ∩ BF = B // Intersection
$(Z \cap R_2)^+$ = B^+ = BF // Closure
$\{(Z \cap R_2)^+ \cap R_2\}$ = BF ∩ BF = BF // Intersection
Z= $[\{(Z \cap R_2)^+ \cap R_2\} \cup Z]$ = ABCD ∪ BF = ABCDF // Updating Z (Union)

A→F preserves dependency. Checking for A→E, Repeating procedure with R3

Now Z = ABCDF
$(Z \cap R_3)$ = ABCDF ∩ DE = B // Intersection
$(Z \cap R_3)^+$ = D^+ = DE // Closure
$\{(Z \cap R_3)^+ \cap R_3\}$ = DE ∩ DE = BF // Intersection
Z= $[\{(Z \cap R_3)^+ \cap R_3\} \cup Z]$ = DE ∪ ABCDF = ABCDEF // Updating Z (Union)

Stopping the algorithm as Z contains F, So A→F also preserves
dependency

Method 2 : (Useful for gate students)

R(ABCDEF) has following FD's
F = {A→BCD, A→EF, BC→AD, BC→E, BC→F, B→F, D→E}
D = {ABCD, BF, DE}
check whether decomposition is dependency preserving or not
Solution :
The following dependencies can be projected into the following decomposition

ABCD (R1)	BF (R2)	DE (R3)
A → B A → C A → D BC → A BC → D	B → F	D → E

Try to infer the reverse FDs which are present in the table by taking closure w.r.t F.
Look at the RHS of FDs and take closure (B,C,D,A,F,E)
Infer Reverse FDs -
B+ w.r.t F = {BF} //doesn't contain A. So, B → A cannot be inferred

C+ w.r.t F = {C} //doesn't contain A. So, C → A cannot be inferred

D+ w.r.t F = {DF} //doesn't contain A and BC. So, D → A and D → BC
cannot be inferred

A+ w.r.t F = {ABCDEF} // A → BC can be inferred, but it is equal to
A → B and A → C

F+ w.r.t F = {F} //doesn't contain B. So, F → B cannot be inferred

E+ w.r.t F = {E} //doesn't contain D. So, E → D cannot be inferred

No reverse FDs can be inferred from the closure.
<u>Checking BC → E preserves dependency or not -</u>
Compute (BC)+ w.r.t Table FDs
(BC)+ = {BCDAFEE}
as closure of BC w.r.t table FDs contains EF. So BC→EF preserves dependency.

<u>Checking A → EF preserves dependency or not -</u>
Compute A+ w.r.t Table FDs
(A)+ = {ABCDFE}
as closure of A w.r.t table FDs contains EF, So, A → EF preserves dependency.

Decompose the Relation R till BCNF – Question

Question 2 :
R(ABDLPT)
FD : {B → PT, T → L, A → D}
Decompose the Relation R till BCNF.
Solution :
Step 1 : Find all the candidate keys of R.
Candidate Key : {AB}

Step 2 : <u>Checking For 2NF</u> :
(a) FD which violates 2NF :
B → PT
A → D

(b)

Applying Decomposition Algorithm to FD: B → PT $\boxed{\text{ABDLPT}}$		
Compute Closure of LHS	**Relation$_i$** = All Attributes in Closure	**Relation$_j$** = All attributes on LHS of FD ∪ All attributes of R not in Closure
$(B)^+$ = {BPTL}	BPTL	BAD
	B → PT √ T → L √	A → D ×

Applying Decomposition Algorithm to FD: A → D $\boxed{\text{BAD}}$		
Compute Closure of LHS	**Relation$_i$** = All Attributes in Closure	**Relation$_j$** = All attributes on LHS of FD ∪ All attributes of R not in Closure
$(A)^+$ = {AD}	AD	AB
	A → D √	There is no FD for this relation. But {AB} is a CK and is missing in other's decomposed relation, So, the relation "AB" is added.

Check the CK of R is preserved in the decomposed relations- yes,
it is preserved in "AB" relation.
Hence the decomposition in 2NF :

CK : B	CK : A	CK : AB
BPTL	AD	AB

B → PT √ T → L √ A → D √ AB : CK

Step 3 : Checking For 3NF :

(a) FD which violates 3NF :

T → L

(b)

Applying Decomposition Algorithm to FD: T → L $\boxed{\text{BPTL}}$		
Compute Closure of LHS	**Relation$_i$** = All Attributes in Closure	**Relation$_j$** = All attributes on LHS of FD ∪ All attributes of R not in Closure
$(T)^+$ = {TL}	TL	TBP
	T → L √	B → PT √

Check the CK of R is preserved in all the decomposed relations- yes it is preserved in "AB" relation.
Hence the decomposition in 3NF :

CK : A	CK : AB	CK : T	CK : B
AD	AB	TL	TBP

A → D √ CK relation T → L √ B → PT √

Step 4 : <u>Checking For BCNF</u> :

FD which violates BCNF : None

Hence the decomposition is already in BCNF also.

Decompose the Relation R till BCNF – Question3

Question 2 :
R(ABDLPT)
FD : {B → PT, T → L, A → D}
Decompose the Relation R till BCNF.
Solution :
Step 1 : Find all the candidate keys of R.

Candidate Key : {AB}

Step 2 : <u>Checking For 2NF</u> :

(a) FD which violates 2NF :

B → PT
A → D

(b)

Applying Decomposition Algorithm to FD: B → PT $\boxed{\text{ABDLPT}}$		
Compute Closure of LHS	**Relation$_i$** = All Attributes in Closure	**Relation$_j$** = All attributes on LHS of FD ∪ All attributes of R not in Closure
$(B)^+$ = {BPTL}	BPTL	BAD
	B → PT √ T → L √	A → D ×

Applying Decomposition Algorithm to FD: A → D $\boxed{\text{BAD}}$		
Compute Closure of LHS	**Relation$_i$** = All Attributes in Closure	**Relation$_j$** = All attributes on LHS of FD ∪ All attributes of R not in Closure
$(A)^+$ = {AD}	AD	AB
	A → D √	There is no FD for this relation. But {AB} is a CK and is missing in other's decomposed relation, So, the relation "AB" is added.

Check the CK of R is preserved in the decomposed relations- yes, it is preserved in "AB" relation.

Hence the decomposition in 2NF :

CK : B	CK : A	CK : AB
BPTL	AD	AB

B → PT √ T → L √ A → D √ AB : CK

Step 3 : <u>Checking For 3NF</u> :

(a) FD which violates 3NF :

T → L

(b)

Applying Decomposition Algorithm to FD: T → L $\boxed{\text{BPTL}}$		
Compute Closure of LHS	**Relation$_i$** = All Attributes in Closure	**Relation$_j$** = All attributes on LHS of FD ∪ All attributes of R not in Closure
$(T)^+$ = {TL}	TL	TBP
	T → L √	B → PT √

Check the CK of R is preserved in all the decomposed relations- yes it is preserved in "AB" relation.
Hence the decomposition in 3NF :

CK : A	CK : AB	CK : T	CK : B
AD	AB	TL	TBP

A → D √ CK relation T → L √ B → PT √

Step 4 : <u>Checking For BCNF</u> :

FD which violates BCNF : None

Hence the decomposition is already in BCNF also.

Normal Forms Questions - Part 2

Decompose the relation R, till BCNF.

Question 1 :
R(ABCDEFGHIJ)
FD : {AB → C, A → DE, B → F, F → GH, D → IJ}
Solution :
Step 1 : Find all the candidate keys of R.

Candidate Key : {AB}

Step 2 : <u>Checking For 2NF</u> :

(a) FD which violates 2NF :

A → DE
B → F

(b)

Applying Decomposition Algorithm to FD: A → DE ABCDEFGHIJ		
Compute Closure of LHS	**Relation$_i$ = All Attributes in Closure**	**Relation$_j$ = All attributes on LHS of FD ∪ All attributes of R not in Closure**
$(A)^+$ = {ADEIJ}	ADEIJ	ABCFGH
	A → DE √ D → IJ √	AB → C √ B → F × F → GH √

Applying Decomposition Algorithm to FD: B → F ABCFGH		
Compute Closure of LHS	**Relation$_i$ = All Attributes in Closure**	**Relation$_j$ = All attributes on LHS of FD ∪ All attributes of R not in Closure**
$(B)^+$ = {BFGH}	BFGH	BAC
	B → F √ F → GH √	AB → C √

Check the CK of R is preserved in the decomposed relations- yes, it is preserved in "BAC" relation.
Hence the decomposition in 2NF :

CK : A	CK : B	CK : AB
ADEIJ	BFGH	BAC

A → DE √ D → IJ √ B → F √ F → GH √ AB → C √

Step 3 : <u>Checking For 3NF</u> :

(a) FD which violates 3NF :

D → IJ

F → GH

(b)

Applying Decomposition Algorithm to FD: D → IJ ADEIJ		
Compute Closure of LHS	**Relation$_i$ = All Attributes in Closure**	**Relation$_j$ = All attributes on LHS of FD ∪ All attributes of R not in Closure**
$(D)^+$ = {DIJ}	DIJ	DAE
	D → IJ √	A → DE √

Applying Decomposition Algorithm to FD: F → GH BFGH		
Compute Closure of LHS	**Relation$_i$ = All Attributes in Closure**	**Relation$_j$ = All attributes on LHS of FD ∪ All attributes of R not in Closure**
$(F)^+$ = {FGH}	FGH	FB
	F → GH √	B → F √

Check the CK of R is preserved in the decomposed relations- yes it is preserved in "BAC" relation.
Hence the decomposition in 3NF :

CK : D	CK : A	CK : F	CK : B	CK : AB
DIJ	DAE	FGH	BF	BAC

D → IJ √ A → DE √ F → GH √ B → F √ AB → C √

Step 4 : <u>Checking For BCNF</u> :

FD which violates BCNF : None

Hence the decomposition is already in BCNF also.

What is a Multivalued Dependency ?

To understand the concept of MVD, let us consider a schema denoted as **MPD (Man, Phones, Dog_Like)**,

Person :				Meaning of the tuples
Man(M)	**Phones(P)**	**Dogs_Like(D)**	⇒	Man M have phones P, and likes the dogs D.
M1	P1/P2	D1/D2	⇒	M1 have phones P1 and P2, and likes the dogs D1 and D2.
M2	P3	D2	⇒	M2 have phones P3, and likes the dog D2.
Key : MPD				

There are no non trivial FDs because all attributes are combined forming Candidate Key i.e. MDP. The multivalued dependency is shown by "→→". So, in the above relation, two multivalued dependencies exists -

1. **Man →→ Phones**
2. **Man →→ Dogs_Like**

A man's phone are independent of the dogs they like. But after converting the above relation in Single Valued Attribute, Each of a man's phones appears with each of the dogs they like in all combinations.

Man(M)	Phones(P)	Dogs_Likes(D)
M1	P1	D1
M1	P2	D2
M2	P3	D2
M1	P1	D2
M1	P2	D1

Some Points to note here about relation Person :

- Some unwanted(shaded) tuples will also exist in the relation while converting it into single valued attributes.
- However, We can see that the relation is in BCNF, and thus we would not consider decomposing if further if we looked only at the FDs that hold over the relation Person.
- The **redundancy exists in BCNF relation because of MVD**.

Where MVD occurs ?

- If two or more independent relations are kept in a single relation, then Multivalued Dependency is possible. For example, Let there are two relations :
    - **Student(SID, Sname)** where **(SID → Sname)**
    - **Course(CID, Cname)** where **(CID → Cname)**, There is no relation defined between Student and Course. If we kept them in a single relation named **Student_Course**, then MVD will exists because of m:n Cardinality.

Student :

SID	Sname
S1	A
S2	B

Course :

CID	Cname
C1	C
C2	B

Merging using Cross Product – As Student and Course do not have any relation, So taking all the possible combinations by using Cross product

SID	Sname	CID	Cname
S1	A	C1	C
S1	A	C2	B
S2	B	C1	C
S2	B	C2	B
2 Multivalued Dependency exists : 1. SID →→ CID 2. SID →→ Cname			

- If two or more multivalued attributes exists in a relation, then while converting into single valued attributes, MVD exists. The relation **"Person"** is such type of example.

Definition of 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 $\mathbf{t_1[X] = t_2[Y]}$ 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 $\mathbf{t_1}$**,** $\mathbf{t_2}$**,** $\mathbf{t_3}$**,** $\mathbf{t_4}$ are not necessarily distinct.

Inference Rules of MVD (Five Rules)

Three of the additional rules involve only MVDs :

C-	**Complementation**	:	If $X \rightarrow\rightarrow Y$, then $X \rightarrow\rightarrow \{R - (X \cup Y)\}$.
A-	**Augmentation**	:	If $X \rightarrow\rightarrow Y$ and $W \supseteq Z$, then $WX \rightarrow\rightarrow YZ$.
T-	**Transitivity**	:	If $X \rightarrow\rightarrow Y$ and $Y \rightarrow\rightarrow Z$, then $X \rightarrow\rightarrow (Z - Y)$.

The remaining two rules relate FDs and MVDs :

- **Replication :** If $X \rightarrow Y$, then $X \rightarrow\rightarrow Y$ but the reverse is not true.
- **Coalescence :** If $X \rightarrow\rightarrow Y$ and there is a W such that $W \cap Y$ is empty, $W \rightarrow Z$, and $Y \supseteq Z$, then $X \rightarrow Z$.

Trivial and Non Trivial MVD :

A MVD **X** $\rightarrow\rightarrow$ **Y** in **R** is called a trivial MVD is

a. **Y** is a subset of **X (X ⊇ Y)** or
b. **X ∪ Y = R**. Otherwise, it is a non trivial MVD and we have to repeat values redundantly in the tuples.

Removal of MVD :

Solution : Fourth Normal Form (4NF)

Join Dependencies and Fifth Normal Form (5NF)

Fifth Normal Form (5NF)

Definition 1 : A relation R is in 5NF if and only if every join dependency in R is implied by the candidate keys of R. **Definition 2 :** A relation decomposed into two relations must have loss-less join Property, which ensures that no spurious or extra tuples are generated, when relations are reunited through a natural join.

What is a Join Dependency(JD) ??

Let **R** be a relation. Let **A, B, ..., Z** be arbitrary subsets of **R's** attributes. **R** satisfies the **JD * (A, B, ..., Z)** if and only if **R** is equal to the join of its projections on **A, B, ..., Z.** A join dependency **JD(R1, R2, ..., Rn)** specified on relation schema **R,** is a trivial JD, if one of the relation schemas **Ri** in **JD(R1, R2, ...,Rn)** is equal to R.

Join dependency is used in the following case :

When there is no lossless join decomposition of **R** into two relation schemas, but there is a lossless join decompositions of **R** into more than two relation schemas. **Point :** A join dependency is very difficult in a database, hence normally not used.

Negative Example :

Consider a relation **ACP(Agent, Company, Product)**

ACP :				**Meaning of the tuples**
Agent(A)	**Company(C)**	**Product(P)**	⇒	**Agent sells Company's Products.**
A1	PQR	Nut	⇒	A1 sells PQR's Nuts and Screw.
A1	PQR	Screw		
A1	XYZ	Bolt	⇒	A1 sells XYZ's Bolts.
A2	PQR	Bolt	⇒	A2 sells PQR's Bolts.

The table is in 4 NF as it does not contain multivalued dependency. But the relation **contains redundancy** as A1 is an agent for PQR twice. But there is no way of eliminating this redundancy without losing information. Suppose that the table is decomposed into its two relations, **R1** and **R2**.

R1 :

Agent	**Company**
A1	PQR
A1	XYZ
A2	PQR

R2 :

Agent	**Product**
A1	Nut
A1	Screw
A1	Bolt
A2	Bolt

The redundancy has been eliminated by decomposing **ACP** relation, but the information about which companies make which products and which agents supply which product has been lost. The natural join of these relations over the 'agent' columns is:

R_{12} :		
Agent	**Company**	**Product**
A1	PQR	Nut
A1	PQR	Screw
A1	PQR	Bolt
A1	XYZ	Nut
A1	XYZ	Screw
A1	XYZ	Bolt
A2	PQR	Bolt

Hence, the decomposition of **ACP is a lossy join decomposition** as the natural join table is spurious, since it contains extra tuples(shaded) that gives incorrect information. But now, suppose the original relation **ACP** is decomposed into 3 relations :

- **R1(Agent, Company)**
- **R2(Agent, Product)**
- **R3(Company, Product)**

The result of the natural join of **R1** and **R2** over 'Agent' (already Calculated **R12**) and then, natural join of **R12** and **R3** over 'Company' & 'Product' is -

R_{123} :		
Agent	**Company**	**Product**
A1	PQR	Nut
A1	PQR	Screw
A1	PQR	Bolt
A1	XYZ	Bolt
A2	PQR	Bolt

Again, we get an extra tuple shown as by shaded portion. Hence, it has to be accepted that it is not possible to eliminate all redundancies using normalization techniques because it cannot be assumed that all decompositions will be non-loss. Hence again, the decomposition of **ACP is a lossy join decomposition**

Positive Example :

Consider the above schema, but with a different case as "if a company makes a product and an agent is an agent for that company, then he always sells that product for the company". Under these circumstances, the **ACP** table is shown as :

ACP :		
Agent	**Company**	**Product**
A1	PQR	Nut
A1	PQR	Bolt
A1	XYZ	Nut
A1	XYZ	Bolt
A2	PQR	Nut

The relation ACP is again decompose into 3 relations. Now, the natural Join of all the three relations will be shown as :

R_1 :	
Agent	**Company**
A1	PQR
A1	XYZ
A2	PQR

R_3 :	
Company	**Product**
PQR	Nut
PQR	Bolt
XYZ	Nut
XYZ	Bolt

R_2 :	
Agent	**Product**
A1	Nut
A1	Bolt
A2	Nut

Result of Natural Join of R_1 and R3 over 'Company' and then Natural Join of R_{13} and R_2 over 'Agent'and 'Product' ⇓

R_{123} :		
Agent	**Company**	**Product**
A1	PQR	Nut
A1	PQR	Bolt
A1	XYZ	Nut
A1	XYZ	Bolt
A2	PQR	Nut

Hence, in this example, all the redundancies are eliminated, and the decomposition of **ACP is a lossless join decomposition**. Hence the relation is in 5NF as it does not violate the property of lossless join

Normal Forms Shortcuts for Gate Students

I am trying to give you the normal forms shortcuts for Gate students. To identify that a relation is in which normal form, the first step is to Find all the Candidate Keys of the relation.

First Normal Form (1NF) :

It doesn't have multivalued attributes.

RollNo	Name	Subject	Marks
00001	AAAAA	Maths,C++	????
Error in 3rd Column (multivalue), as we can't assign marks for both subjects in one column.			

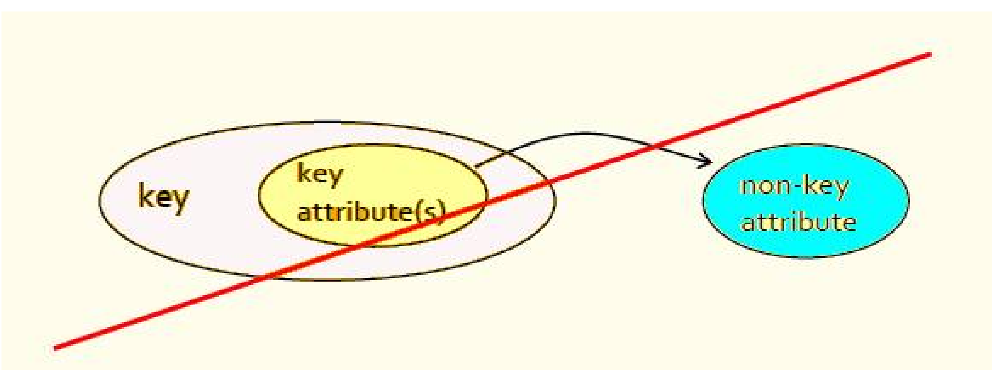
Solution :

RollNo	Name	Subject	Marks
00001	AAAAA	Maths	89
00001	AAAAA	C++	98

Second Normal Form (2NF) :

Check right hand side of FD A → B ,

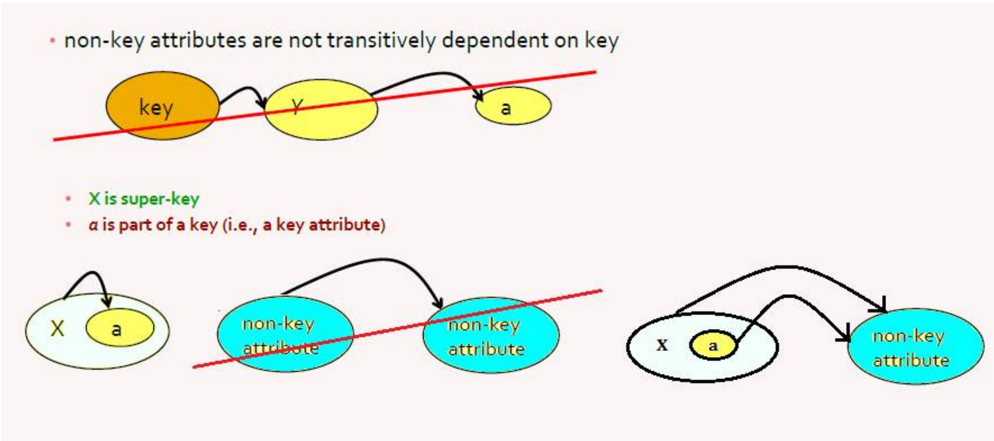
1. If B is a fullynonkey, then A must be either fullynonkey OR fully key. (i.e. it should not be partial)
2. If right hand side B is a key or part of key, then its okay - no need to check.
3. If its Nonkey → Nonkey, its fine - no need to check. Just check (1) is sufficient.



Third Normal Form (3NF) :

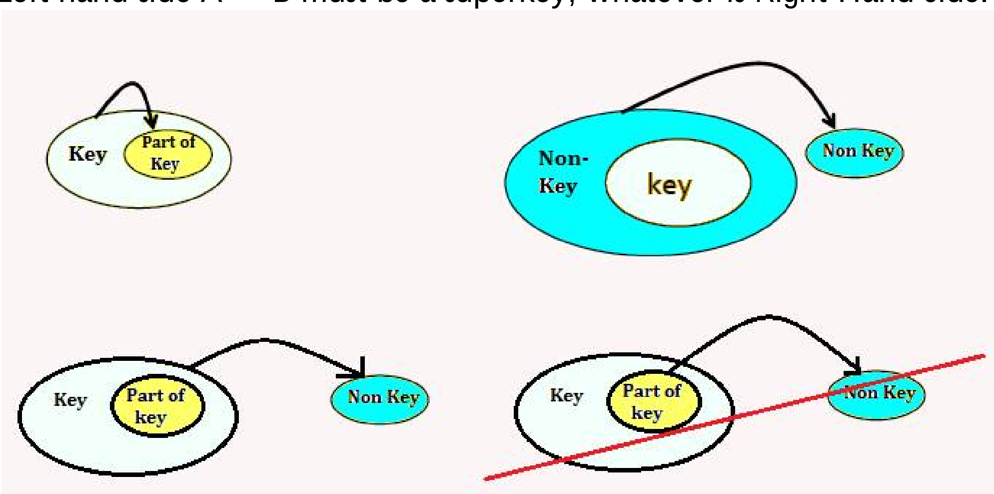
Check right hand side of FD A → B,

1. If B is fullynonkey, then left hand side must be either superkey or part of superkey.
2. Nonkey → Nonkey is not allowed in 3NF
3. Key or part of Key → Nonkey is allowed



Boyce-Codd Normal Form (BCNF) :

Left hand side A → B must be a superkey, whatever is Right Hand Side.



Decomposition in All Normal Form Violation

If A → B is violating normal form in R(ABCD), then decompose into

- R1 = AB
- R2 = BCD

Some Notes about Normal Forms - Useful for Gate Students

	1NF	2NF	3NF	BCNF
Redundancy	Hgh	<1NF	<2NF	0% if it doesn't contain MVD, Otherwise, it may contain Redundancy
Lossless Join	Always	Always	Always	Always
Dependency Preserving	Always	Always	Always	Some relation may not possible to decompose in BCNF by preserving dependency

	4NF
0% redundancy	Yes
Lossless	Yes
Dependency Preserving	may or may not be
Problem may exist ??	Join Dependency

Questions on Normal Forms -Part 1

Identify the Normal Forms of the relation R :

Question 1 :
R(ABCD)
FD : {A → B, B → C}
Solution :
(A)+ = {ABC}
As no dependency defined for D, hence R is in 1NF.
Question 2:
R(ABCD)
FD1 : A → B, FD2 : B → C, FD3 : AD → BC

Solution :

Step 1 :	**Find Candidate Key :** $(AD)^{+}$ = {ADBC}		
Step 2 :	**Functional Dependency**	**Find Highest Normal Form**	**Reason**
	FD1 : A → B,	1NF	Partial Dependency as A is part of Key
	FD2 : B → C,	2NF	Nonkey Nonkey allowed
	FD3 : AD → BC	BCNF	LHS is the SuperKey

Question 3:

R(ABCDE)
FD1 : AB → C, FD2 : C → D, FD3 : D → E,FD4 : E → A

Solution :

<table>
<tr><td>Step 1 :</td><td colspan="3">Find Candidate Key : $(AB)^+$ = {ABCDE} Other Candidate Keys derived from AB {AB,EB,DB,CB}</td></tr>
<tr><td rowspan="5">Step 2 :</td><td>Functional Dependency</td><td>Find Highest Normal Form</td><td>Reason</td></tr>
<tr><td>FD1 : AB → C,</td><td>BCNF</td><td>LHS is the SuperKey</td></tr>
<tr><td>FD2 : C → D,</td><td>3NF</td><td>Part of Key → Part of Key is allowed.</td></tr>
<tr><td>FD3 : D → E</td><td>3NF</td><td>Part of Key → Part of Key is allowed.</td></tr>
<tr><td>FD3 : E → A</td><td>3NF</td><td>Part of Key → Part of Key is allowed.</td></tr>
<tr><td colspan="4">R is in 3NF.</td></tr>
</table>

Question 4:

R(ABCDEF)
FD1 : AB → C, FD2 : C → D, FD3 : B → E,FD4 : B → F

Solution :

<table>
<tr><td>Step 1 :</td><td colspan="3">Find Candidate Key : $(AB)^+$ = {ABCDEF}</td></tr>
<tr><td rowspan="5">Step 2 :</td><td>Functional Dependency</td><td>Find Highest Normal Form</td><td>Reason</td></tr>
<tr><td>FD1 : AB → C,</td><td>BCNF</td><td>LHS is the SuperKey</td></tr>
<tr><td>FD2 : C → D,</td><td>2NF</td><td>NonKey → NonKey is allowed.</td></tr>
<tr><td>FD3 : B → E</td><td>1NF</td><td>Partial Dependency as B is part of Key</td></tr>
<tr><td>FD3 : B → F</td><td>1NF</td><td>Partial Dependency as B is part of Key</td></tr>
<tr><td colspan="4">R is in 1NF.</td></tr>
</table>

Question 5 :

a) If relation R consists of only simple candidate keys then R should be in ....?

b) If relation R consists of only prime attributes, then R should be in ....?

c) If relation R is in 3NF and every CK is simple CK, then relation is in ....?

d) If relation R with no non trivial FD, then R is in ......?

Solution :

a) If relation R consists of only simple candidate keys then R should be in <u>2NF but may or may not be in BCNF,3NF</u>

b) If relation R consists of only prime attributes, then R should be in <u>3NF but may or may not be in BCNF</u>

$X \rightarrow Y$ {X: Candidate Key , Y : Prime Attribute }

Example of this type of relation is defined in Question 3, i.e.

R(ABCDE)

FD : $\{AB \rightarrow C, C \rightarrow D, D \rightarrow E, E \rightarrow A\}$

CK : {AB,EB,DB,CB}

c) If relation R is in 3NF and every CK is simple CK, then relation is in <u>BCNF</u>.

Example of this type of relation is :

R(ABCD)

FD : $\{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$

d) If relation R with no non trivial FD, then R is in <u>always BCNF</u>.

Example of this type of relation is :

R(ABC)

FD : $\{A \rightarrow A, B \rightarrow B, C \rightarrow C, AB \rightarrow AB, AC \rightarrow AC, BC \rightarrow BC, ABC \rightarrow ABC\}$

Questions on Lossless Join

To Identify whether a decomposition is lossy or lossless, it must satisfy the following conditions :

1. $R_1 \cup R_2 = R$
2. $R_1 \cap R_2 \neq \Phi$ and
3. $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$

Question 1 :
R(ABC)
F = {A → B, A → C} decomposed into
D = R_1(AB), R_2(BC)
Find whether D is Lossless or Lossy ?
Solution :
D = {AB, BC}
Step 1: AB ∪ BC = ABC
Step 2: AB ∩ BC = B //Intersection
Step 3: B^+ = {B} //Not a super key of R_1 or R_2
⇒ Decomposition is lossy.
Question 2 :
R(ABCDEF)
F = {A → B, B → C, C → D, E → F} decomposed into
D = R_1(AB), R_2(BCD), R_3(DEF).
Find whether D is Lossless or Lossy ?
Solution :
Step 1: AB ∪ BCD ∪ DEF = ABCDEF = R // Condition 1 satisfies
step 2: AB ∩ BCD = B
B^+ = {BCD} //superkey of R_2
⇒ R_{12}(ABCD)

ABCD ∩ DEF = D
D^+ = {D} // Not a superkey of R_{12} or R_3
⇒ Decomposition is Lossy.
Question 3 :

R(ABCDEF)
F = {A → B, C → DE, AC → F} decomposed into
D = R_1(BE), R_2(ACDEF).
Find whether D is Lossless or Lossy ?
Solution :
Step 1: BE ∪ ACDEF = ABCDEF = R // Condition 1 satisfies
step 2: BE ∩ ACDEF = E
E^+ = {E} //Not a superkey of R_1 or R_2
⇒ Decomposition is Lossy.

Question 4 :

R(ABCDEG)
F = {AB → C, AC → B, AD → E, B → D, BC → A, E → G} decomposed into
(i) D1 = R_1(AB), R_2(BC), R_3(ABDE), R_4(EG).
(ii) D2 = R_1(ABC), R_2(ACDE), R_3(ADG).
Find whether D1 and D2 is Lossless or Lossy ?
Solution (i) :
Step 1: AB ∪ BC ∪ ABDE ∪ EG = ABCDEG = R // Condition 1 satisfies
step 2: AB ∩ BC = B
B^+ = {BD} //Not a superkey of R1 or R_2
⇒ Decomposition is Lossy. No need to check further.

Solution (ii) :
Step 1: ABC ∪ ACDE ∪ ADG = ABCDEG = R // Condition 1 satisfies
step 2: ABC ∩ ACDE = AC
AC^+ = {ACBDEG} //superkey
⇒ R_{12}(ABCDE)

ABCDE ∩ ADG = AD
AD^+ = {ADEG} //Superkey of R_3
⇒ R_{123}(ABCDEG)
⇒ Decomposition is LossLess.

Question 5 :
R(ABCDEFGHIJ)
F = {AB → C, B → F, D → IJ, A → DE, F → GH} decomposed into
(i) D1 = R_1(ABC), R_2(ADE), R_3(BF), R_4(FGH),R_5(DIJ).
(ii) D2 = R_1(ABCDE), R_2(BFGH), R_3(DIJ).
(iii) D3 = R_1(ABCD), R_2(DE), R_3(BF), R_4(FGH),R_5(DIJ).
Find whether D1, D2 and D3 is Lossless or Lossy ?
Solution (i) :
Step 1: ABC ∪ ADE ∪ BF ∪ FGH ∪ DIJ = ABCDEFGHIJ = R // Condition 1 satisfies
step 2: ABC ∩ ADE = A
A^+ = {ADEIJ} //Superkey of R_2
⇒ R_{12}(ABCDE)

ABCDE ∩ BF = B
B^+ = {BFGH} //superkey of R_3
⇒ R_{123}(ABCDEF)

ABCDEF ∩ FGH = F
F^+ = {FGH} //Superkey of R_4
⇒ R_{1234}(ABCDEGH)

ABCDEFGH ∩ DIJ = D

D^+ = {DIJ} //Superkey of R_5

⇒ R_{12345}(ABCDEGHIJ)

⇒ Decomposition is LossLess.

Solution (ii) :

Step 1: ABCDE ∪ BFGH ∪ DIJ = R // Condition 1 satisfies

step 2: ABCDE ∩ BFGH = B

B^+ = {BFGH} //Superkey of R_2

⇒ R_{12}(ABCDEFGH)

ABCDEFGH ∩ DIJ = D

D^+ = {DIJ} //superkey of R_3

⇒ R_{123}(ABCDEFGHIJ)

⇒ Decomposition is LossLess.

Solution (iii) :

Step 1: ABCD ∪ DE ∪ BF ∪ FGH ∪ DIJ = ABCDEFGHIJ = R // Condition 1 satisfies

step 2: ABCD ∩ DE = D

D^+ = {DIJ} //Not a super key of R_1 or R_2

⇒ Decomposition is Lossy. No need to check further.

Question on Decomposition of Normal Forms -5

Question 4 :

R(ABCDEFGH)

FD : {AB → C, AC → B, AD → E, B → D, BC → A, E → G}

Decompose the Relation R till BCNF.

Solution :

Step 1 : Find all the candidate keys of R.

Candidate Key : {ABFH}, {BCFH} and {ACFH}

Step 2 : Checking For 2NF :

(a) FD which violates 2NF :

AD → E

B → D

(b)

Applying Decomposition Algorithm to FD: AD → E $\boxed{ABCDEFGH}$		
Compute Closure of LHS	**Relation$_i$** = All Attributes in Closure	**Relation$_j$** = All attributes on LHS of FD ∪ All attributes of R not in Closure
$(AD)^+$ = {ADEG}	ADEG AD : CK	ADBCFH {ABFH,ACFH.BCFH} : CK
	AD → E √ E → G √	AB → **C** √ **AC** → B √ **B** → **D** **x** **BC** → A √

Since all are in 2NF except B → **D,**

Applying Decomposition Algorithm to FD: B → D $\boxed{ABCDFH}$		
Compute Closure of LHS	**Relation$_i$** = All Attributes in Closure	**Relation$_j$** = All attributes on LHS of FD ∪ All attributes of R not in Closure
$(B)^+$ = {BD}	BD B : CK	BACFH {ABFH, ACFH, BCFH} : CK
	B → D √	AB → C √ BC → A √ AC → B √

Check the CK of R is preserved in the decomposed relations- Yes,

Hence the decomposition in 2NF :

CK : AD	CK : B	CK : {ABFH, BCFH, ACFH}
ADEG	BD	ABCFH
AD → E √ E → G √	B → D √	AB → C √ AC → B √ BC → A √

Step 3 : <u>Checking For 3NF</u> :

FD which violates 3NF : E → G

Applying Decomposition Algorithm to FD: E → G $\boxed{\text{ADEG}}$		
Compute Closure of LHS	**Relation$_i$ = All Attributes in Closure**	**Relation$_j$ = All attributes on LHS of FD ∪ All attributes of R not in Closure**
$(E)^+$ = {EG}	EG E : CK	EAD AD : CK
	E → G √	AD → E √

Check the CK of R is preserved in the decomposed relations- Yes

Hence the decomposition in 3NF :

CK : E	CK : AD	CK : B	CK : ABFH, BCFH, ACFH
EG	ADE	BD	ABCFH
E → G √	AD → E √	B → D √	AB → C √ AC → B √ BC → A √

Step 4 : <u>Checking For BCNF</u> :

FD which violates BCNF :

AB → C

AC → B

BC → A

Applying Decomposition Algorithm to FD: AB → C $\boxed{\text{ABCFH}}$		
Compute Closure of LHS	**Relation$_i$ = All Attributes in Closure**	**Relation$_j$ = All attributes on LHS of FD ∪ All attributes of R not in Closure**
$(AB)^+$ = {ABC}	ABC AB : CK	ABFH {ABFH} : CK
	AB → C √ BC → A x AC → B x	No FD exist for relation "ABFH".But {ABFH} is a candidate key. So, remains in Decomposition.

Applying Decomposition Algorithm to FD: BC → A $\boxed{ABC}$		
Compute Closure of LHS	**Relation$_i$ = All Attributes in Closure**	**Relation$_j$ = All attributes on LHS of FD ∪ All attributes of R not in Closure**
$(BC)^+ = \{ABC\}$	$\boxed{ABC}$ AB : CK BC : CK	$\boxed{BC}$
	AB → C √ BC → A √ AC → B x	No FD exist for relation "BC". So Discarded.

Applying Decomposition Algorithm to FD: AC → B $\boxed{ABC}$		
Compute Closure of LHS	**Relation$_i$ = All Attributes in Closure**	**Relation$_j$ = All attributes on LHS of FD ∪ All attributes of R not in Closure**
$(AC)^+ = \{ABC\}$	$\boxed{ABC}$ AB : CK BC : CK AC : CK	$\boxed{AC}$
	AB → C √ BC → A √ AC → B √	No FD exist for relation "AC". So Discarded.

Check the CK of R is preserved in the decomposed relations- No
{BCFH and ACFH} are two candidate keys which are not preserving dependency. So, make new relations for each CK which is not preserved :

CK : E	CK : AD	CK : B	CK : AB,BC,AC
$\boxed{EG}$	$\boxed{EAD}$	$\boxed{BD}$	$\boxed{ABC}$
E → G √	AD → E √	B → D √	AB → C √ AC → B √ BC → A √

CK : ABFH	CK : BCFH	CK : ACFH
$\boxed{ABFH}$	$\boxed{BCFH}$	$\boxed{ACFH}$

Question on Decomposition of Normal Forms -

Question 4 :

R(ABCDEFGH)

FD : {AB → C, AC → B, AD → E, B → D, BC → A, E → G}

Decompose the Relation R till BCNF.

Solution :

Step 1 : Find all the candidate keys of R.

Candidate Key : {ABFH}, {BCFH} and {ACFH}

Step 2 : Checking For 2NF :

(a) FD which violates 2NF :

AD → E

B → D

(b)

Applying Decomposition Algorithm to FD: AD → E ABCDEFGH		
Compute Closure of LHS	**Relation$_i$ = All Attributes in Closure**	**Relation$_j$ = All attributes on LHS of FD ∪ All attributes of R not in Closure**
$(AD)^+$ = {ADEG}	ADEG AD : CK	ADBCFH {ABFH,ACFH.BCFH} : CK
	AD → E √ E → G √	AB → **C** √ **AC** → B √ **B** → **D** x **BC** → A √

Since all are in 2NF except B → **D,**

Applying Decomposition Algorithm to FD: B → D ABCDFH		
Compute Closure of LHS	**Relation$_i$ = All Attributes in Closure**	**Relation$_j$ = All attributes on LHS of FD ∪ All attributes of R not in Closure**
$(B)^+$ = {BD}	BD B : CK	BACFH {ABFH, ACFH, BCFH} : CK
	B → D √	AB → C √ BC → A √ AC → B √

Check the CK of R is preserved in the decomposed relations- Yes, Hence the decomposition in 2NF :

CK : AD	CK : B	CK : {ABFH, BCFH, ACFH}
ADEG	BD	ABCFH
AD → E √ E → G √	B → D √	AB → C √ AC → B √ BC → A √

Step 3 : <u>Checking For 3NF</u> :

FD which violates 3NF : E → G

Applying Decomposition Algorithm to FD: E → G ADEG		
Compute Closure of LHS	**Relation$_i$ = All Attributes in Closure**	**Relation$_j$ = All attributes on LHS of FD ∪ All attributes of R not in Closure**
$(E)^+$ = {EG}	EG E : CK	EAD AD : CK
	E → G √	AD → E √

Check the CK of R is preserved in the decomposed relations- Yes

Hence the decomposition in 3NF :

CK : E	CK : AD	CK : B	CK : ABFH, BCFH, ACFH
EG	ADE	BD	ABCFH
E → G √	AD → E √	B → D √	AB → C √ AC → B √ BC → A √

Step 4 : <u>Checking For BCNF</u> :

FD which violates BCNF :

AB → C

AC → B

BC → A

Applying Decomposition Algorithm to FD: AB → C $\boxed{ABCFH}$		
Compute Closure of LHS	**Relation$_i$** = All Attributes in Closure	**Relation$_j$** = All attributes on LHS of FD ∪ All attributes of R not in Closure
$(AB)^+$ = {ABC}	ABC AB : CK	ABFH {ABFH} : CK
	AB → C √ BC → A x AC → B x	No FD exist for relation "ABFH".But {ABFH} is a candidate key. So, remains in Decomposition.

Applying Decomposition Algorithm to FD: BC → A $\boxed{ABC}$		
Compute Closure of LHS	**Relation$_i$** = All Attributes in Closure	**Relation$_j$** = All attributes on LHS of FD ∪ All attributes of R not in Closure
$(BC)^+$ = {ABC}	ABC AB : CK BC : CK	BC
	AB → C √ BC → A √ AC → B x	No FD exist for relation "BC". So Discarded.

Applying Decomposition Algorithm to FD: AC → B $\boxed{ABC}$		
Compute Closure of LHS	**Relation$_i$** = All Attributes in Closure	**Relation$_j$** = All attributes on LHS of FD ∪ All attributes of R not in Closure
$(AC)^+$ = {ABC}	ABC AB : CK BC : CK AC : CK	AC
	AB → C √ BC → A √ AC → B √	No FD exist for relation "AC". So Discarded.

Check the CK of R is preserved in the decomposed relations- No
{BCFH and ACFH} are two candidate keys which are not preserving dependency. So, make new relations for each CK which is not preserved :

CK : E	CK : AD	CK : B	CK : AB,BC,AC
EG	EAD	BD	ABC
E → G √	AD → E √	B → D √	AB → C √ AC → B √ BC → A √

CK : ABFH	CK : BCFH	CK : ACFH
ABFH	BCFH	ACFH

Question on Decomposition of R till BCNF-4

Question 3 :
R(ABCDEFGH)
FD : {ABC → DE, E → BCG, F → AH}
Decompose the Relation R till BCNF.
Solution :
Step 1 : Find all the candidate keys of R.
Candidate Key : {EF} AND {BCF}

Step 2 : <u>Checking For 2NF</u> :
(a) FD which violates 2NF :
ABC → DE, E → BCG, F → AH
(b)

Applying Decomposition Algorithm to FD: F → AH ABCDEFGH		
Compute Closure of LHS	**Relation**$_i$ **= All Attributes in Closure**	**Relation**$_j$ **= All attributes on LHS of FD ∪ All attributes of R not in Closure**
$(F)^+$ = {FAH}	FAH	FBCDEG
	F → AH √	E → BCG ×

Since ABC → DE is functionally dependency is lost in this decomposition step. So, to preserve the dependency, make a relation for ABC → DE

ABCDE
ABC → DE √

Applying Decomposition Algorithm to FD: E → BCG $\boxed{\text{FBCDEG}}$		
Compute Closure of LHS	**Relation$_i$ = All Attributes in Closure**	**Relation$_j$ = All attributes on LHS of FD ∪ All attributes of R not in Closure**
$(E)^+$ = {EBCG}	EBCG	EFD
	E → BCG √	There is no FD exist for this relation. So, discard it.

Check the CK of R is preserved in the decomposed relations- NO,
So, make new relations for each CK which is not preserved.
Here, 2 Candidate Keys are there - {EF} and {BCF} and both are not preserved in any of the decomposition. So, making relations for each one as :

$\boxed{\text{EF}}$ $\boxed{\text{BCF}}$

Hence the decomposition in 2NF :

CK : F	CK : ABC	CK : E	CK : EF	CK : BCF
FAH	ABCDE	EBCG	EF	BCF

F → AH √ ABC → DE √ E → BCG √

Step 3 : <u>Checking For 3NF</u> :
FD which violates 3NF : None
Hence the decomposition is already in 3NF.

Step 4 : <u>Checking For BCNF</u> :
FD which violates BCNF : None
Hence the decomposition is already in BCNF also.

Question on Decomposition of R till BCNF

Question 3 :
R(ABCDEFGH)
FD : {ABC → DE, E → BCG, F → AH}
Decompose the Relation R till BCNF.
Solution :
Step 1 : Find all the candidate keys of R.
Candidate Key : {EF} AND {BCF}

Step 2 : <u>Checking For 2NF</u> :
(a) FD which violates 2NF :
ABC → DE
E → BCG
F → AH

(b)

Applying Decomposition Algorithm to FD: F → AH $\boxed{\text{ABCDEFGH}}$		
Compute Closure of LHS	**Relation$_i$ = All Attributes in Closure**	**Relation$_j$ = All attributes on LHS of FD ∪ All attributes of R not in Closure**
$(F)^+$ = {FAH}	FAH	FBCDEG
	F → AH √	E → BCG ×

Since ABC → DE is functionally dependency is lost in this decomposition step. So, to preserve the dependency, make a relation for ABC → DE

$\boxed{\text{ABCDE}}$

ABC → DE √

Applying Decomposition Algorithm to FD: E → BCG $\boxed{\text{FBCDEG}}$		
Compute Closure of LHS	**Relation$_i$ = All Attributes in Closure**	**Relation$_j$ = All attributes on LHS of FD ∪ All attributes of R not in Closure**
$(E)^+$ = {EBCG}	EBCG	EFD
	E → BCG √	There is no FD exist for this relation. So, discard it.

Check the CK of R is preserved in the decomposed relations- NO,
So, make new relations for each CK which is not preserved.
Here, 2 Candidate Keys are there - {EF} and {BCF} and both are not preserved in any of the decomposition. So, making relations for each one as :

$\boxed{\text{EF}}$ $\boxed{\text{BCF}}$

Hence the decomposition in 2NF :

CK : F	CK : ABC	CK : E	CK : EF	CK : BCF
FAH	ABCDE	EBCG	EF	BCF

F → AH √ ABC → DE √ E → BCG √

Step 3 : <u>Checking For 3NF</u> :

FD which violates 3NF : None

Hence the decomposition is already in 3NF.

Step 4 : <u>Checking For BCNF</u> :

FD which violates BCNF : None

Hence the decomposition is already in BCNF also.

Question on Dependency Preserving Decomposition

Question 1:
R(ABCD)
F = {A → B, B → C, C → D, D → A}
D = {AB, BC, CD}
Check whether the decomposition is preserving dependency or not ?
Solution :
The following dependencies can be projected into the following decomposition :

R1(AB)	R2(BC)	R3(CD)
A→B	B→C	C → D

Inferring reverse FDs which fits into the decomposition-
B+ w.r.t F = {BCDA} ⇒ B → A
C+ w.r.t F = {CDAB} ⇒ C → B
D+ w.r.t F = {DABC} ⇒ D → C

So, the table will be updated as -

R1(AB)	R2(BC)	R3(CD)
A→B B→A	B→C C→B	C → D D→C

Checking D → A preserves dependency or not -
Compute D+ w.r.t updated table FDs :
D+ = {DCBA}
as closure of D w.r.t updated table FDs contains A. So D→A preserves dependency.
Question 2:
R(ABCDEF)
F = {AB → CD, C → D, D → E, E → F}
D = {AB, CDE, EF}
Check whether the decomposition is preserving dependency or not ?
Solution :
The following dependencies can be projected into the following decomposition :

R1(AB)	R2(CDE)	R3(EF)
	C→D D→E	E → F

Inferring reverse FDs which fits into the decomposition-
D+ w.r.t F = {DEF}
E+ w.r.t F = {EF}
F+ w.r.t F = {F}
No reverse FDs can be derived.

<u>Checking AB → CD preserves dependency or not</u> -
Compute AB+ w.r.t table FDs :
AB+ = {AB}
as closure of AB w.r.t table FDs does not contains CD. So AB→CD preserves dependency.
Question 3:
R(ABCDEG)
F = {AB → C, AC → B, BC → A, AD → E, B → D, E → G}
D = {ABC, ACDE, ADG}

Check whether the decomposition is preserving dependency or not ?
Solution :
The following dependencies can be projected into the following decomposition :

R1(ABC)	R2(ACDE)	R3(ADG)
AB→C AC→B BC→A	AD→E	

Inferring reverse FDs which fits into the decomposition-
C+ w.r.t F = {C}
B+ w.r.t F = {BD}
A+ w.r.t F = {A}
E+ w.r.t F = {EG}
No reverse FDs can be derived.

<u>Checking B → D preserves dependency or not</u> -
Compute B+ w.r.t updated table FDs :
B+ = {B}
as closure of B w.r.t table FDs doesn't contains D. So B→D doesn't preserves dependency.

<u>Checking E → G preserves dependency or not</u> -
Compute E+ w.r.t updated table FDs :
E+ = {E}
as closure of E w.r.t table FDs doesn't contains G. So E→G doesn't preserves dependency.
Question 4:
Let R(ABCD) be a relational schema with the following functional dependencies :
F = {A → B, B → C, C → D, D → B}. The decomposition of R into
D = {AB, BC, BD}
Check whether the decomposition is preserving dependency or not ?
Solution :
The following dependencies can be projected into the following decomposition :

R1(AB)	R2(BC)	R3(BD)
A→B	B→C	D → B

Inferring reverse FDs which fits into the decomposition-
B+ w.r.t F = {BCD} ⇒ B → D
C+ w.r.t F = {CDB} ⇒ C → B

So, the table will be updated as -

R1(AB)	R2(BC)	R3(CD)
A→B	B→C C→B	D→B B→D

<u>Checking C → D preserves dependency or not</u> -
Compute C+ w.r.t updated table FDs :
C+ = {CBD}
as closure of C w.r.t updated table FDs contains D. So C→D preserves dependency.
Question 5:
R(ABCDE)
F = {A → BC, CD → E, B → D, E → A}
D = {ABCE, BD}

Check whether the decomposition is preserving dependency or not ?
Solution :
The following dependencies can be projected into the following decomposition :

R1(ABCE)	R2(BD)
A→B A→C E→A	B→D

Inferring reverse FDs which fits into the decomposition-
B+ w.r.t F = {BD}
C+ w.r.t F = {C}
A+ w.r.t F = {ABCDE} ⇒ A → E
D+ w.r.t F = {D}

So, the table will be updated as -

R1(AB)	R2(BC)
A→B A→C E→A A→E	B→D

<u>Checking CD → E preserves dependency or not</u> -
Compute CD+ w.r.t updated table FDs :
CD+ = {CD}
as closure of D w.r.t updated table FDs doesn't contains E. So CD→E doesn't preserves dependency.