UNIT-4

SYLLABUS:

Database Design Theory: Functional Dependencies, Normal forms based on Primary Keys, Second and Third Normal Forms, Boyce-Codd Normal Form, Multi valued Dependencies and Fourth Normal Form, Join Dependencies and Fifth Normal Form.

Database Design Theory:

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables. This module discuss the basic and higher normal forms.

Introduction to DB design

Each relation schema consists of a number of attributes, and the relational database schema consists of a number of relation schemas. So far, we have assumed that attributes are grouped to form a relation schema by using the common sense of the database designer or by mapping a database schema design from a conceptual data model such as the ER or Enhanced-ER (EER) data model. These models make the designer identify entity types and relationship types and their respective attributes, which leads to a natural and logical grouping of the attributes into relations.

Database Design

There are two levels at which we can discuss the goodness of relation schemas:

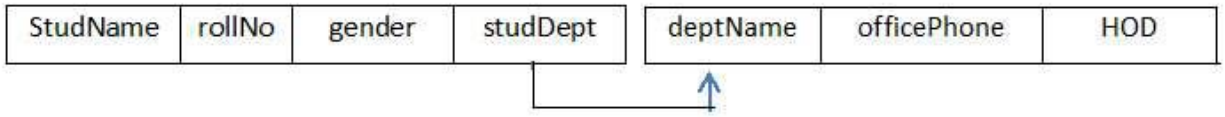
1. The logical (or conceptual) level how users interpret the relation schemas and the meaning of their attributes.
2. The implementation (or physical storage) level how the tuples in a base relation are stored and updated. This level applies only to schemas of base relations.

An Example

- STUDENT relation with attributes: studName, rollNo, gender, studDept
- DEPARTMENT relation with attributes: deptName, officePhone, hod
- Several students belong to a department
- studDept gives the name of the student's department

Correct schema:

Student **Department**



Incorrect schema:

Studdept

StudName	rollNo	gender	deptName	officePhone	HOD

Problems with bad schema

- Redundant storage of data:
  - Office Phone & HOD info -stored redundantly once with each student that belongs to the department
  - wastage of disk space
- A program that updates Office Phone of a department
  - must change it at several places
  - more running time
  - error -prone

2.2 Functional Dependencies

- Formal tool for analysis of relational schemas that enables us to detect and describe some of the problems in precise terms.

Definition of Functional Dependency

Definition : A functional dependency A->B in a relation holds if two tuples having same value of attribute A also have the same value for attribute B. It is denoted by A->B where A is called determinant and B is called dependent.

For example - Consider Student table as follows -

Roll	Name	City
1	AAA	Mumbai
2	BBB	Pune
3	CCC	Gandhinagar

Here

Roll ->Name **hold**

But

Name->City **does not hold**

- In above table, student roll number is unique hence each student's name and city can be uniquely identified using his roll number.
- But using name we cannot uniquely identify his/her city because there can be same names of the students. Similarly using city name we can not identify the student uniquely. As in the same city may belong to multiple students.
- A functional dependency is a constraint between two sets of attributes from the database.

- Given a relation R, a set of attributes X in R is said to functionally determine another attribute Y, also in R, (written X→Y) if and only if each X value is associated with at most one Y value.
- X is the determinant set and Y is the dependent attribute. Thus, given a tuple and the values of the attributes in X, one can determine the corresponding value of the Y attribute.
- The abbreviation for functional dependency is FD or f.d. The set of attributes X is called the left-hand side of the FD, and Y is called the right-hand side.
- A functional dependency is a property of the semantics or meaning of the attributes. The database designers will use their understanding of the semantics of the attributes of R to specify the functional dependencies that should hold on all relation states (extensions) r of R.

Consider the relation schema EMP_PROJ;

EMP_PROJ

SSN	PNUMBER	HOURS	ENAME	PNAME	PLOCATION

From the semantics of the attributes and the relation, we know that the following functional dependencies should hold:

a) Ssn → Ename
b) Pnumber →{Pname, Plocation}
c) {Ssn, Pnumber}→ Hours

- These functional dependencies specify that
  a) The value of an employee's Social Security number (Ssn) uniquely determines the employee name (Ename)
  b) The value of a projects number (Pnumber) uniquely determines the project name (Pname) and location (Plocation) and
  c) A combination of Ssn and Pnumber values uniquely determines the number of hours the employee currently works on the project per week (Hours).
- Alternatively,we say that Ename is functionally determined by (or functionally dependent on) Ssn, or given a value of Ssn, we know the value of Ename, and so on.
- Relation extensions r(R) that satisfy the functional dependency constraints are called legal relation states (or legal extensions) of R.
- A functional dependency is a property of the relation schema R, not of a particular legal relation state r of R.
- Therefore, a Functional Dependencies cannot be inferred automatically from a given relation extension r but must be defined explicitly by someone who knows the semantics of the attributes of R.

EXAMPLE:

Consider a relation in which the roll of the student and his/her name is stored as follows :

R	N
1	AAA
2	BBB
3	CCC
4	DDD
5	EEE

Fig. 5.2.1 : Table which holds functional dependency i.e. R->N

Here, is true. That means the functional dependency holds true here. Because for every assigned RollNuumber of student there will be unique name. For instance : The name of the Student whose RollNo is 1 is AAA. But if we get two different names for the same roll number then that means the table does not hold the functional dependency.

Following is such table -

R	N
1	AAA
2	BBB
3	CCC
1	XXX
2	YYY

Fig. 5.2.2 : Table which does not hold functional dependency

In above table for RollNumber 1 we are getting two different names - "AAA" and "XXX". Hence here it does not hold the functional dependency.

Trivial FD : The functional dependency A->B is trivial if B is a subset of A.

For example (A,B}->A

Non Trivial FD : The functional dependency A->B is non trivial if B is not a subset of A.

For example {A,B}->C

Diagrammatic notation for displaying Functional Dependencies

- Each Functional Dependencies is displayed as a horizontal line

- The left-hand-side attributes of the Functional Dependencies are connected by vertical lines to the line representing the Functional Dependencies
- The right-hand-side attributes are connected by the lines with arrows pointing toward the attributes.
- Each Functional Dependencies is displayed as a horizontal line
- The left-hand-side attributes of the FD are connected by vertical lines to the line representing the FD
- The right-hand-side attributes are connected by the lines with arrows pointing toward the attributes.

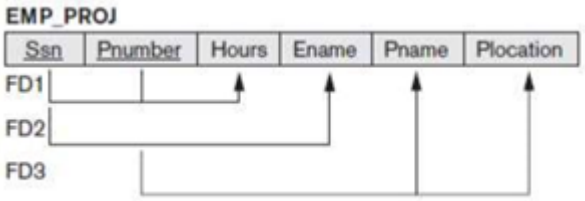


Fig: diagrammatic notation for displaying FDs

INFERENCE RULES:

The closure set is a set of all functional dependencies implied by a given set F. It is denoted by F^+

The closure set of functional dependency can be computed using basic three rules which are also called as **Armstrong's Axioms.**

These are as follows -

i) **Reflexivity :** If $X \supseteq Y$, then $X \rightarrow Y$

ii) **Augmentation :** If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z

iii) **Transitivity :** If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

In addition to above axioms some additional rules for computing closure set of functional dependency are as follows -

- **Union :** If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$
- **Decomposition :** If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

EXAMPLE: Given FD's for Relation R{A,C,D,E,F}, Find Closure of FD set by applying Amstrong's Axims.

A->B, A->C, CD->E, CD->F, B->E.

Solution:

Step 1 : A -> gives A attribute itself by reflexivity. It is called trivial production.

A -> B and A -> C, Hence by union rule A -> BC

A -> B and B -> E, Hence by transitivity rule A -> E

∴ $(A)^+ = \{A,B,C,E\}$

Step 2 : B -> gives B itself

And B -> E

∴ $(B)^+ = \{B,E\}$

Step 3 : CD -> gives CD itself. It is trivial.

CD -> E, CD -> F, Hence by union rule CD -> EF

∴ $(CD)^+ = \{C,D,E,F\}$

So by omitting trivial productions, we get

∴ $F^+ = \{A \rightarrow BC, A \rightarrow E, B \rightarrow E, CD \rightarrow EF\}$

EXAMPLE: Compute the closure of the following set F of functional dependencies for relational schema R = (A,B,C,D,E) A->BC, CD->E, B->D, E->A

Solution : The closure of F is denoted by F^+ and it can be computed in following steps

Step 1 : As A -> BC is given we get

A -> B and A -> C	By decomposition rule

Step 2 : As

A -> B	(Refer step 1)
B -> D	(given)
A -> D	(transitivity rule)

Step 3 :

A -> CD because A -> C and A -> D	(From step 1 and Step 2 applying union rule)

Step 4 :

CD -> E	(given)
A -> E	(transitive rule as A->CD, CD->E hence A->E)

Step 5 :

Since A → A,	we have (reflexive)
A → ABCDE	from the above steps (union)

Step 6 :

Since E → A, **E → ABCDE** (transitive)

Step 7 :

Since CD → E, **CD → ABCDE** (transitive)

Step 8 :

Since B → D and BC → CD, **BC → ABCDE** (augmentative, transitive)

Step 9 :

Also, C → C, D → D, BD → D

Thus any functional dependency with A, E, BC, or CD on the left hand side of the arrow is in F^+

EXAMPLE: Give Armstrong's axioms and using it find the closure of the following FD set.

A->B, AB->C, D->AC, D->E

Solution:

Step 1 : Consider the rules D -> AC and D -> E

∴ **D -> ACE** **(Union rule)**

Step 2 : Consider AB -> C then we get

∴ A -> C and B -> C **(decomposition rule)**

Thus F^+ ={A -> C, B -> C, D -> ACE}

EXAMPLE: R={A,B,C,D,E,F} and FDs are A->BC, E->CF, B->E, CD->EF compute closure of $\{A,B\}^+$

Solution:

Step 1 : A -> BC

∴ A -> B and A -> C (decomposition)

So we add A, B, C in closure set

Step 2 : E -> CF

∴ E -> C and E -> F (decomposition)

Step 3 : B -> E ∴ B -> C, F

So we add E and F in closure set

Hence

$\{A,B\}^+ = \{A,B,C,E,F\}$

EXAMPLE: Consider schema EMPLOYEE(E-ID, E-NAME, E-CITY, E-STATE) and

FD = {E-ID->E-NAME, E-ID->E-CITY, E-ID->E-STATE, E-CITY->E-STATE}

1) Find attribute of closure for $(E\text{-}ID)^+$

2) Find $(E\text{-}NAME)^+$

Solution:

1) Finding $(E\text{-}ID)^+$ means finding the closure. In this process we try to find out, all the attributes that can be derived from E-ID.
   As E-ID->E-NAME, E-ID->E-CITY, E-ID->E-STATE, We add E-NAME, E-ID, E-CITY, E-STATE to $(E\text{-}ID)^+$
   As E-ID->E-CITY, E-CITY->E-STATE, we add E-STATE to $(E\text{-}ID)^+$
   $\therefore (E\text{-}ID)^+ = \{E\text{-}ID, E\text{-}NAME, E\text{-}CITY, E\text{-}STATE\}$
2) **E-NAME** derives no rule. Hence $(E\text{-}NAME)+ = \{E\text{-}NAME\}$

KEYS AND FUNCTIONAL DEPENDENCIES:

PROPERTIES OF FUNCTIONAL DEPENDENCIES:

1) Reflexive: If $Y \subseteq X$ then $X \rightarrow Y$ is a Reflexive Functional Dependency.

Ex: $AB \rightarrow A$, $A \subseteq AB$ holds. Therefore $AB \rightarrow A$ is a Reflexive Functional Dependency.

2) Augmentation: If $X \rightarrow Y$ is a functional dependency then by augmentation, $XZ \rightarrow YZ$ is also a functional dependency.

3) Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$ are two functional dependencies then by transitivity, $X \rightarrow Z$ is also a functional dependency.

4) Union: If $X \rightarrow Y$ and $X \rightarrow Z$ are two functional dependencies then, $X \rightarrow YZ$ is also a functional dependency.

5) Decomposition: If $X \rightarrow YZ$ is a functional dependency then $X \rightarrow Y$ and $X \rightarrow Z$ are also functional dependencies.

CLOSURE SET OF A FUNCTIONAL DEPENDENCY (F+)

It is a set of all functional dependencies that can be determined using the given set of dependencies. It is denoted by F+.

Attribute Closure (X+): It is a set of all the attributes that can be determined using X. It is denoted by X+, where X is any set of attributes.

Example:

R(A,B,C) F:{A→B , B→C}

A+={A,B,C} B+={B,C} C+={C}

AB+={A,B,C} AC+={A,C,B} BC+={B,C} ABC+={A,B,C}

Identifying keys in the given relation based on Functional Dependencies associated with it

X+ is a set of attributes that can be determined using the given set X of attributes.

- If X+ contains all the attributes of a relation, then X is called "Super key" of that relation.
- If X+ is minimal set, then X is called "Candidate Key" of that relation.

If no closure contains all the elements then in such a case we can find independent attributes of that relation i.e., the attributes that which are not in the R.H.S. of any dependency.

If the closure of the Independent attributes contains all the elements then it can be treated as a candidate key. If the closure of independent attributes also doesn't contain all the elements then we try to find the key by adding dependent attributes one by one. If we couldn't find key then we can add groups of dependent attributes till we find a key to that relation.

PROPERTIES OF RELATIONAL DECOMPOSITION:

- Decomposition is the process of breaking down one table into multiple tables.
- **Formal definition of decomposition is -**
- A decomposition of relation schema R consists of replacing the relation schema by two relation schema that each contain a subset of attributes of R and together include all attributes of R by storing projections of the instance.
- For example **-** Consider the following table

Employee_Department table as follows **-**

Eid	Ename	Age	City	Salary	Deptid	DeptName
E001	ABC	29	Pune	20000	D001	Finance
E002	PQR	30	Pune	30000	D002	Production
E003	LMN	25	Mumbai	5000	D003	Sales
E004	XYZ	24	Mumbai	4000	D004	Marketing
E005	STU	32	Hyderabad	25000	D005	Human Resource

We can decompose the above relation Schema into two relation schemas as **Employee (Eid, Ename, Age, City, Salary)** and **Department (Deptid, Eid, DeptName)** as follows **-**

Employee Table

Eid	Ename	Age	City	Salary
E001	ABC	29	Pune	20000
E002	PQR	30	Pune	30000
E003	LMN	25	Mumbai	5000
E004	XYZ	24	Mumbai	4000
E005	STU	32	Hyderabad	25000

Department Table

Deptid	Eid	DeptName
D001	E001	Finance
D002	E002	Production
D003	E003	Sales
D004	E004	Marketing
D005	E005	Human Resource

- The decomposition is used for eliminating redundancy.
- **For example :** Consider following relation **Schema R** in which we assume that the grade determines the salary, the redundancy is caused

Schema R

Name	eid	deptname	Grade	Salary
AAA	121	Accounts	2	8000
AAA	132	Sales	3	7000
BBB	101	Marketing	4	7000
CCC	106	Purchase	2	8000

- Hence, the above table can be decomposed into two Schema S and T as follow :

Schema S

Name	eid	deptname	Grade
AAA	121	Accounts	2
AAA	132	Sales	3
BBB	101	Marketing	4
CCC	106	Purchase	2

Schema T

Grade	Salary
2	8000
3	7000
4	7000
2	8000

Problems related to decomposition :

Following are the potential problems to consider :

1) Some queries become more **expensive.**
2) Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
3) Checking some dependencies may require joining the instances of the decomposed relations.
4) There may be loss of information during decomposition.

Properties associated with decomposition

There are two properties associated with decomposition and those are -

1) **Loss-less join or non loss decomposition :** When all information found in the original database is preserved after decomposition, we call it as loss less or non loss decomposition.
2) **Dependency preservation :** This is a property in which the constraints on the original table can be maintained by simply enforcing some constraints on each of the smaller relations.

LOSS-LESS JOIN:

The lossless join can be defined using following three conditions :

i) **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.

$Att(R1) \cup Att(R2) = Att(R)$

ii) **Intersection** of attributes of R1 and R2 must not be NULL.

$Att(R1) \cap Att(R2) \neq \Phi$

iii) **Common attribute** must be a key for at least one relation (R1 or R2)

$Att(R1) \cap Att(R2) \rightarrow Att(R1)$

or $Att(R1) \cap Att(R2) \rightarrow Att(R2)$

EXAMPLE: Consider the following relation R(A, B, C, D) and FDs A->BC, is the decomposition of R into R1(A, B, C), R2(A, D). Check if the decomposition is lossless join or not.

Solution:

Step 1 : Here Att(R1) ∪ Att(R2) = Att(R) i.e R1(A,B,C) ∪ R2(A,D) = (A,B,C,D) i.e R.
Thus first condition gets satisfied.

Step 2 : Here R1 ∩ R2 = {A}. Thus Att(R1) ∩ Att(R2) ≠ □. Here the second condition gets satisfied.

Step 3 : Att(R1) ∩ Att(R2) -> {A}. Now (A)+ = {A,B,C} □ attributes of R1. Thus the third condition gets satisfied.

This shows that the given decomposition is a **lossless join.**

Normal forms based on Primary Keys

NORMAL FORMS:

- Normalization is the process of reorganizing data in a database so that it meets **two basic requirements :**

1) There is **no redundancy** of data (all data is stored in only one place) and
2) **Data dependencies** are logical (all related data items are stored together)

Need for normalization

1) **It eliminates redundant data.**
2) **It reduces chances of data error.**
3) The normalization is important because it allows database to take up **less disk space.**
4) It also help in increasing the **performance.**
5) It improves the data integrity and consistency.

We assume that a

- Set of functional dependencies is given for each relation Each relation has a designated primary key.
- This information combined with the tests (conditions) for normal forms drives the normalization process for relational schema design.
- First three normal forms for relation takes into account all candidate keys of a relation rather than the primary key.

Normalization of Relations

- The normalization process, as first proposed by Codd (1972a), takes a relation schema through a series of tests to certify whether it satisfies a certain normal form.
- Initially, Codd proposed three normal forms, which he called first, second, and third normal form

- All these normal forms are based on a single analytical tool: the functional dependencies among the attributes of a relation
- A fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of multivalued dependencies and join dependencies, respectively Normalization of data can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of
    1) minimizing redundancy and
    2) minimizing the insertion, deletion, and update anomalies
- It can be considered as a "filtering" or "purification" process to make the design have successively better quality.
- Unsatisfactory relation schemas that do not meet certain conditions the normal form tests are decomposed into smaller relation schemas that meet the tests and hence possess the desirable properties.
- Thus, the normalization procedure provides database designers with the following:
    - A formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes.
    - A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be normalized to any desired degree.
- **Definition:** The **normal form** of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.

Practical Use of Normal Forms

- Normalization is carried out in practice so that the resulting designs are of high quality and meet the desirable properties
- Database design as practiced in industry today pays particular attention to normalization only up to 3NF, BCNF, or at most 4NF.
- The database designers need not normalize to the highest possible normal form
- Relations may be left in a lower normalization status, such as 2NF, for performance reasons
- **Definition: Denormalization** is the process of storing the join of higher normal form relations as a base relation, which is in a lower normal form.

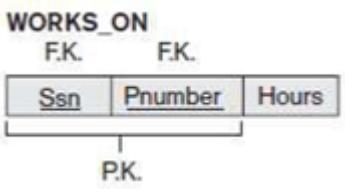
Definitions of Keys and Attributes Participating in Keys

- Superkey: specifies a uniqueness constraint that no two distinct tuples in any state r of R can have the same value
- key K is a superkey with the additional property that removal of any attribute from K will
- cause K not to be a superkey any more

Example:

- The attribute set {Ssn} is a key because no two employees tuples can have the same value for Ssn
- Any set of attributes that includes Ssn for example, {Ssn, Name, Address} is a superkey
- If a relation schema has more than one key, each is called a candidate key
- One of the candidate keys is arbitrarily designated to be the primary key, and the others are called secondary keys
- In a practical relational database, each relation schema must have a primary key
- If no candidate key is known for a relation, the entire relation can be treated as a default superkey

- For example {Ssn} is the only candidate key for EMPLOYEE, so it is also the primary key
- **Definition.** An attribute of relation schema R is called a **prime attribute** of R if it is a member of some candidate key of R. An attribute is called **nonprime** if it is not a prime attribute that is, if it is not a member of any candidate key.

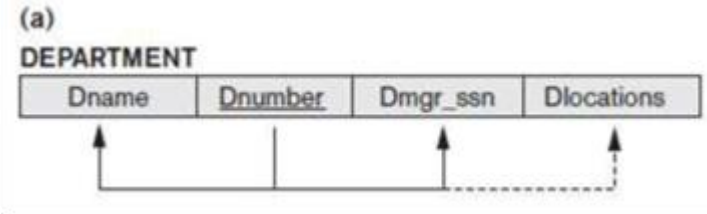


- In WORKS_ON relation Both Ssn and Pnumber are prime attributes whereas other attributes are nonprime.

First Normal Form

- Defined to disallow multivalued attributes, composite attributes, and their combinations.
- It states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute 1NF disallows relations within relations or relations as attribute values within tuples.
- The only attribute values permitted by 1NF are single atomic (or indivisible) values.

Consider the DEPARTMENT relation schema shown in Figure below



✓ Primary key is Dnumber

We assume that each department can have a number of locations

- The DEPARTMENT schema and a sample relation state are shown in Figure below

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

- As we can see, this is not in 1NF because Dlocations is not an atomic attribute, as illustrated by the first tuple in Figure
- There are two ways we can look at the Dlocations attribute:

- The domain of Dlocations contains atomic values, but some tuples can have a set of these values. In this case, Dlocations is not functionally dependent on the primary key Dnumber.
- The domain of Dlocations contains sets of values and hence is nonatomic. In this

- In either case, the DEPARTMENT relation is not in 1NF

There are three main techniques to achieve first normal form for such a relation:

1) Remove the attribute Dlocations that violates 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key Dnumber of DEPARTMENT. The primary key of this relation is the combination {Dnumber, Dlocation}. A distinct tuple in DEPT_LOCATIONS exists for each location of a department. This decomposes the non-1NF relation into two 1NF relations.
2) Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT. In this case, the primary key becomes the combination {Dnumber, Dlocation}. This solution has the disadvantage of introducing redundancy in the relation

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

3) If a maximum number of values is known for the attribute for example, if it is known that at most three locations can exist for a department replace the Dlocations attribute by three atomic attributes: Dlocation1, Dlocation2, and Dlocation3. This solution has the disadvantage of introducing NULL values if most departments have fewer than three locations. Querying on this attribute becomes more difficult;
Example, consider how you would write the query: List the departments that have Ballare as one of their locations in this design.

- Of the three solutions, the first is generally considered best because it does not suffer from redundancy and it is completely general, having no limit placed on a maximum number of values.
- First normal form also disallows multivalued attributes that are themselves composite.
- These are called **nested relations** because each tuple can have a relation within it.

(a)

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

- Figure above shows how the EMP_PROJ relation could appear if nesting is allowed Each tuple represents an employee entity, and a relation PROJS(Pnumber, Hours)

within each tuple represents the employee's projects and hours per week that employee work on each project.

- The schema of this EMP_PROJ relation can be represented as follows:
  EMP_PROJ(Ssn, Ename, {PROJS(Pnumber, Hours)}
- Ssn is the primary key of the EMP_PROJ relation and Pnumber is the partial key of the nested relation; that is, within each tuple, the nested relation must have unique values of Pnumber.
- To normalize this into 1NF, we remove the nested relation attributes into a new relation and propagate the primary key into it; the primary key of the new relation will combine the partial key with the primary key of the original relation.
- Decomposition and primary key propagation yield the schemas EMP_PROJ1 and EMP_PROJ2.

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, Alicia J.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

EXAMPLE:

The table is said to be in 1NF if it follows following rules -

i) It should only have single (atomic) valued attributes/columns.

ii) Values stored in a column should be of the same domain.

iii) All the columns in a table should have unique names.

iv) And the order in which data is stored, does not matter.

Consider following student table

Student

sid	sname	Phone
1	AAA	11111 22222
2	BBB	33333
3	CCC	44444 55555

As there are multiple values of phone number for sid 1 and 3, the above table is not in 1NF. We can make it in 1NF. The conversion is as follows -

sid	sname	Phone
1	AAA	11111
1	AAA	22222
2	BBB	33333
3	CCC	44444
3	CCC	55555

Second Normal Form

Before understanding the second normal form let us first discuss the concept of partial functional dependency and prime and non prime attributes.

Concept of partial functional dependency

Partial dependency means that a nonprime attribute is functionally dependent on part of a candidate key.

For example : Consider a relation R(A,B,C,D) with functional dependency {AB->CD, A->C}

Here (AB) is a candidate key because

$$(AB)^+ = \{ABCD\} = \{R\}$$

Hence {A,B} are prime attributes and {C,D} are non prime attribute. In A->C, the non prime attribute C is dependent upon A which is actually a part of candidate key AB. Hence due to A->C we get partial functional dependency.

Prime and non prime attributes

- **Prime attribute :** An attribute, which is a part of the candidate-key, is known as a prime attribute.
- **Non-prime attribute :** An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.
- **Example : Consider a Relation** R = {A,B,C,D} and candidate key as AB, the Prime attributes : A, B.

Non Prime attributes : C, D

The second normal form

For a table to be in the Second Normal Form, following conditions must be followed

i) It should be in the First Normal form.

ii) It should not have partial functional dependency.

For example : Consider following table in which every information about a the Student is maintained in a table such as student id(sid), student name(sname), course id(cid) and course name(cname).

Student_Course

sid	sname	cid	cname
1	AAA	101	C
2	BBB	102	C++
3	CCC	101	C
4	DDD	103	Java

This table is not in 2NF. For converting above table to 2NF we must follow the following steps -

Step 1 : The above table is in 1NF.

Step 2 : Here **sname** and **sid** are associated similarly **cid** and **cname** are associated with each other. Now if we delete a record with **sid = 2,** then automatically the course C++ will also get deleted. Thus,

sid->sname or **cid->cname** is a partial functional dependency, because **{sid,cid}** should be essentially a candidate key for above table. Hence to bring the above table to 2NF we must decompose it as follows :

Student

sid	sname	cid
1	AAA	101
2	BBB	102
3	CCC	101
4	DDD	103

Here candidate key is (sid, cid) and (sid, cid)->sname

Course

cid	cname
101	C
102	C++
101	C
103	Java

Here candidate key is cid here cid- > cname

Thus now table is in 2NF as there is no partial functional dependency.

EXAMPLE:

Study the relation given below and state what level of normalization can be achieved and normalize it up to that level.

Order no.	*Order date*	*Item lines*		
		Item code	*Quantity*	*Price/Unit*
1456	*26-12-1999*	*3687*	*52*	*50.4*
		4627	*38*	*60*
		3214	*20*	*20.00*
1886	*04-03-1999*	*4629*	*45*	*20.25*
		4627	*30*	*60.20*
1788	*04-04-1999*	*4627*	*40*	*60.20*

Solution:

Reason for the given relation being unnormalized

1. Observe order for many items.
2. Item lines has many attributes-called composite attributes.
3. Each tuple has variable length.
4. Difficult to store due to non-uniformity.
5. Given item code difficult to find qty-ordered and hence called Unnormalized relation.

For conversion to first normal form -

- Identify the composite attributes, convert the composite attributes to individual attributes.
- Duplicate the common attributes as many times as lines in composite attribute.
- Every attribute now describes single property and not multiple properties, some data will be duplicated.
- Now this is called First normal form (1NF) also called flat file.

Order no.	Order date	Item lines		
		Item code	Quantity	Price/Unit
1456	26-12-1999	3687	52	50.4
1456	26-12-1999	4627	38	60
1456	26-12-1999	3214	20	20.00
1886	04-03-1999	4629	45	20.25
1886	04-03-1999	4627	30	60.20
1788	04-04-1999	4627	40	60.20

Table in first normal form

- The above table has insertion, deletion and update anomalies. For instance - if we delete order no. 1886, then the item code 4629 gets lost. Similarly if we update 4627, then all instances of 4627 need to be changed.
- We need to convert 2NF if it is in 1NF. The non-key attributes are functionally dependent on key attribute and if there is a composite key then no non-key attribute is functionally depend on one part of the key.

Orders

OrderNo	OrderDate
1456	26-12-1999
1886	04-03-1999
1788	04-04-1999

Order details

OrderNo	ItemCode	Qty
1456	3687	52
1886	4629	45
1788	4627	40

Prices

ItemCode	Price/Unit
3687	50.4
4627	60
3214	20
4629	20.25

Third Normal Forms

Before understanding the third normal form let us first discuss the concept of transitive dependency, super key and candidate key.

Concept of transitive dependency

A functional dependency is said to be transitive if it is indirectly formed by two functional dependencies. For example -

X -> Z is a transitive dependency if the following functional dependencies hold true :

X->Y

Y->Z

Concept of super key and candidate key

Superkey : A super key is a set or one of more columns (attributes) to uniquely identify rows in a table.

Candidate key : The minimal set of attribute which can uniquely identify a tuple is known as candidate key. For example consider following table

RegID	RollNo	Sname
101	1	AAA
102	2	BBB
103	3	CCC
104	4	DDD

Superkeys

- {RegID}
- {RegID, RollNo}
- {RegID, Sname}
- {RollNo, Sname}
- {RegID, RollNo, Sname}

Candidate keys

- {RegID}
- {RollNo}

Third normal form

A table is said to be in the third normal form when,

i) It is in the second normal form.(i.e. it does not have partial functional dependency).

ii) It doesn't have transitive dependency.

In other words 3NF can be defined as : A table is in 3NF if it is in 2NF and for each functional dependency

X-> Y

at least one of the following conditions hold :

i) X is a super key of table.

ii) Y is a prime attribute of table.

For example : Consider following table **Student_details** as follows -

sid	sname	zipcode	cityname	state
1	AAA	11111	Pune	Maharashtra
2	BBB	22222	Surat	Gujarat
3	CCC	33333	Chennai	Tamilnadu
4	DDD	44444	Jaipur	Rajastan
5	EEE	55555	Mumbai	Maharashtra

Here

Super keys : {sid},{sid,sname},{sid,sname,zipcode}, {sid,zipcode,cityname}… and so on.

Candidate keys : {sid}

Non-prime attributes : {sname,zipcode,cityname,state}

The dependencies can be denoted as

sid->sname
sid->zipcode
zipcode->cityname
cityname->state

The above denotes the transitive dependency. Hence above table is not in 3NF. We can convert it into 3NF as follows :

Student

sid	sname	zipcode
1	AAA	11111
2	BBB	22222
3	CCC	33333
4	DDD	44444
5	EEE	55555

Zip

zipcode	cityname	state
11111	Pune	Maharashtra
22222	Surat	Gujarat
33333	Chennai	Tamilnadu
44444	Jaipur	Rajasthan
55555	Mumbai	Maharashtra

EXAMPLE:

A software contract and consultancy firm maintains details of all the various projects in which its employees are currently involved. These details comprise:

- **Employee number**
- **Employee name**
- **Date of birth**
- **Department code**
- **Department name**
- **Project code**
- **Project description**
- **Project supervisor**

Assume the following:

- **Each employee number is unique.**
- **Each department has a single department code.**
- **Each project has a single code and supervisor.**
- **Each employee may work on one or more projects.**
- **Employee names need not necessary be unique.**
- **Project code, project supervisor and project description are repeting fields.**

Normalize this data to third normal form.

Solution:

Un-Normalized Form

Employee Number, Employee Name_Date of Birth_Department Code_Department Name_Project Code_Project Description_Project Supervisor

1NF

Employee Number, Employee Name_Date of Birth
Department Code, Department Name
Employee Number, Project Code, Project Description_Project Supervisor

2NF

Employee Number, Employee Name_Date of Birth_Department Code_Department Name
Employee Number,Project Code,
Project Code, Project Description,Project Supervisor

3NF

Employee Number, Employee Name_Date of Birth_*Department Code
Department Code, Department Name
Employee Number, Project Code
Project Code, Project Description, Project Supervisor

EXAMPLE:

What is normalization? Normalize below given relation upto 3NF STUDENT.

StudID	*StudName*	*City*	*Pincode*	*ProjectID*	*ProjectName*	*Course*	*Content*
S101	*Ajay*	*Surat*	*326201*	*P101*	*Health*	*Programming*	*C++, Java, C*
S102	*Vijay*	*Pune*	*325456*	*P102*	*Social*	*WEB*	*HTML, PHP, ASP*

Solution : For converting the given schema to first normal form, we will arrange it in such a way that have each tuple contains single record. For that purpose we need to split the schema into two tables namely **Student** and **Projects.**

1NF

Student

StudID	StudName	Pincode	City
S101	Ajay	326201	Surat
S102	Vijay	325456	Pune

Projects

StudID	ProjectID	ProjName	Course	Content
S101	P101	Health	Programming	C++
S101	P101	Health	Programming	Java
S101	P101	Health	Programming	C
S102	P102	Social	WEB	HTML
S102	P102	Social	WEB	PHP
S102	P102	Social	WEB	ASP

2NF

For a table to be in 2NF, there should not be any partial dependency.

Student

StudID	StudName	Pincode	City
S101	Ajay	326201	Surat
S102	Vijay	325456	Pune

Project

StudID	ProjectID	ProjName	CourseID
S101	P101	Health	C101
S101	P101	Health	C102
S101	P101	Health	C103
S102	P102	Social	C104
S102	P102	Social	C105
S102	P102	Social	C106

CourseDetails

CourseID	Course	Content
C101	Programming	C++
C102	Programming	Java
C103	Programming	C
C104	WEB	HTML
C105	WEB	PHP
C106	WEB	ASP

3NF

There was a transitive dependency in 2NF tables because city is associated with student ID and city depends upon zip code. Hence the transitive dependency is removed to covert table into 3NF. The required 3NF schema is as below -

Student

StudID	StudName	Pincode
S101	Ajay	326201
S102	Vijay	325456

Student_Address

Pincode	City
326201	Surat
325456	Pune

Project

StudID	ProjectID	ProjName	CourseID
S101	P101	Health	C101
S101	P101	Health	C102
S101	P101	Health	C103
S102	P102	Social	C104
S102	P102	Social	C105
S102	P102	Social	C106

CourseDetails

CourseID	Course	Content
C101	Programming	C++
C102	Programming	Java
C103	Programming	C
C104	WEB	HTML
C105	WEB	PHP
C106	WEB	ASP

EXAMPLE:

What is the need for normalization? Consider the relation: Emp-proj = {ssn, Pnumber, Hours, Ename, Pname, Plocation}

Assume {ssn,Pnumber} as primary key.
The dependencies are:
{ssn,Pnumber}->Hours
Ssn-> Ename
Pnumber ->{Pname,Plocation}
Normalize the above relation to 3NF.

Solution:

Need for normalization

1) **It eliminates redundant data.**
2) **It reduces chances of data error.**
3) The normalization is important because it allows database to take up **less disk space.**
4) It also help in increasing the **performance.**
5) It improves the data integrity and consistency.

Consider the given dependencies

(1) {ssn,Pnumber}->Hours

(2) ssn-> Ename

(3) Pnumber ->{Pname,Plocation}

The dependencies 2 and 3 represents the partial dependency. Hence we convert the relation into second normal form by splitting the given Emp-proj into three relations

Emp = {ssn, Ename}

Proj = {Pnumber, Pname, Plocation}

Works = {ssn, Pnumber, Hours}

EXAMPLE:

Normalize the below relation upto 3NF.

Module	*Dept*	*Lecturer*	*Text*
M1	*D1*	*L1*	*T1*
M1	*D1*	*L1*	*T2*
M2	*D1*	*L1*	*T1*
M2	*D1*	*L1*	*T3*
M3	*D1*	*L2*	*T4*
M4	*D2*	*L3*	*T1*
M4	*D2*	3	*T5*
M5	*D2*	*L4*	*T6*

Solution : The given relation is already in 1st normal form. But it has Insert, delete and update anomalies. Because -

1) Insert anomalies : We can not add a module(M) with no texts(T).

2) Delete anomalies : If we remove M3, we remove L2 as well.

3) Update anomalies : To change lecturer for M1, we have to change two rows.

Hence we will convert it to second normal form.

Step 1 : We can define the functional dependency FD as

{Module, Text}->{Lecturer, Dept}

But

{Module}->{Lecturer, Dept}

That means Lecturer and Dept are partially dependent on the primary key. Hence for conversion of first normal form to second normal form we will decompose the give table into two tables as

Table 2a

Module	Dept	Lecturer
M1	D1	L1
M2	D1	L1
M3	D1	L2
M4	D2	L3
M5	D2	L4

Table 2b

Module	Text
M1	T1
M1	T2
M2	T1
M2	T3
M3	T4
M4	T1
M4	T5
M5	T6

The relation is now in second normal form.

Step 3 : The table 2a has Insert, Delete and Update anomalies. Because -

1) INSERT anomalies : We can't add lecturers who teach no modules.

2) UPDATE anomalies : To change the department for L1 we must alter two rows.

3) DELETE anomalies : If we delete M3 we delete L2 as well.

Hence, to eliminate these anomalies, we decompose table 2a into two tables and convert it to third normal form.

Step 4 : Hence we get

Table 3a

Lecturer	Dept.
L1	D1
L2	D1
L3	D2
L4	D2

Table 3b

Module	Lecturer
M1	L1
M2	L1
M3	L2
M4	L3
M5	L4

Step 5 : Thus now the complete relation is decomposed into three tables and it is in third normal form. It is summarized as below

Table 3a

Lecturer	Dept.
L1	D1
L2	D1
L3	D2
L4	D2

Table 3b

Module	Lecturer
M1	L1
M2	L1
M3	L2
M4	L3
M5	L4

Table 2b

Module	Text
M1	T1
M1	T2
M2	T1
M2	T3
M3	T4
M4	T1
M4	T5
M5	T6

Boyce/Codd Normal Form(BCNF)

Boyce and Codd Normal Form is a **higher version** of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF.

A 3NF table which **does not have multiple overlapping** candidate keys is said to be in BCNF.

Or in other words,

For a table to be in BCNF, following conditions must be satisfied :

i) R must be in 3rd Normal Form

ii) For each functional dependency ($X \rightarrow Y$), X should be a super key. In simple words if Y is a prime attribute then X can not be non prime attribute.

For example **-** Consider following table that represents that a student enrollment for the course **-**

Enrollment table

sid	Course	Teacher
1	C	Ankita
1	Java	Poonam
2	C	Ankita
3	C++	Supriya
4	C	Archana

From above table following observations can be made :

- One student can enroll for multiple courses. For example student with sid = 1 can enroll for C as well as Java.
- For each course, a teacher is assigned to the student.
- There can be multiple teachers teaching one course for example course C can be taught by both the teachers namely - Ankita and Archana.
- The candidate key for above table can be (sid, course), because using these two columns we can find,
- The above table holds following dependencies
  - (sid, course)->Teacher
  - Teacher->course
- The above table is not in BCNF because of the dependency teacher->course. Note that the teacher is not a superkey or in other words, teacher is a non prime attribute and course is a prime attribute and non-prime attribute derives the prime attribute.
- To convert the above table to BCNF we must decompose above table into student and course tables

Student

sid	Teacher
1	Ankita
1	Poonam
2	Ankita
3	Supriya
4	Archana

Course

Teacher	Course
Ankita	C
Poonam	Java
Ankita	C
Supriya	C++
Archana	C

Now the table is in BCNF

EXAMPLE:

Consider the relation R(ABC) with following FD A->B, B->C and C->A. What is the normal form of R?

Solution:

Step 1 : We will find the candidate key

$(A)^+ = \{ABC\} = R$

$(B)^+ = \{ABC\} = R$

$(C)^+ = \{ABC\} = R$

Hence A, B and C all are candidate keys

Prime attributes = {A,B,C}

Non prime attribute{}

Step 2 : For R being in BCNF for X->Y the X should be candidate key or super key.

From above FDs

- Consider A->B in which A is a candidate key or super key. Condition for BCNF is satisfied.
- Consider B->C in which B is a candidate key or super key. Condition for BCNF is satisfied.
- Consider C->A in which C is a candidate key or super key. Condition for BCNF is satisfied.

This shows that the given relation **R is in BCNF.**

EXAMPLE:

A college maintains details of its lecturer's subject areas skills. These details comprise:

- **Lecture number**
- **Lecture name**
- **Lecture grade**
- **Department code**
- **Department name**
- **Subject code**
- **Subject name**
- **Subject level**

Assume that each lecture may teach many subjects but may belong to more than one department.

Subject code, Subject name, Subject level are repeating fields.

Normalize this data to third normal form.

Solution:

Unnormalized form

Lecturer Number, Lecturer Name_Lecturer Grade_Department Code,Department Name_Subject Code, Subject Name_Subject Level

1NF

LecturerNumber	Lecturer Name	Lecturer Grade	Department Code	Department Name

Lecturer Number	Subject Code	Subject Name	Subject Level

2NF

Lecturer Number	_Lecturer Name	_Lecturer Grade	_Department Code	_Department Name

Lecturer Number	Subject Code

Subject Code	_Subject Name	_Subject Level

3NF

Lecturer Number	Lecturer Name	Lecturer Grade

*Department Code

Department Code	_Department Name

Lecturer Number	Subject Code

Subject Code	Subject Name	Subject Level

EXAMPLE: Prove that any relational schema with two attributes is in BCNF.

Solution : Here, we will consider R={A,B} i.e. a relational schema with two attributes. Now various possible FDs are A->B, B->A.

From the above FDs

- Consider A->B in which A is a candidate key or super key. Condition for BCNF is satisfied.
- Consider B->A in which B is a candidate key or super key. Condition for BCNF is satisfied.
- Consider both A->B and B->A with both A and B is candidate key or super key. Condition for BCNF is satisfied.
- No FD holds in relation R. In this {A,B} is candidate key or super key. Still condition for BCNF is satisfied.

This shows that any relation R **is in BCNF** with two attributes.

EXAMPLE:

Prove the statement "Every relation which is in BCNF is in 3NF but the converse is not true."

Solution : For a relations to be in 3NF

A table is said to be in the Third Normal Form when,

i) It is in the Second Normal form. (i.e. it does not have partial functional dependency)

ii) It doesn't have transitive dependency.

Or in other words

In other words 3NF can be defined as : A table is in 3NF if it is in 2NF and for each functional dependency X-> Y

at least one of the following conditions hold :

iii) X is a super key of table

iv) Y is a prime attribute of table

For a relation to be in BCNF

1) It should be in 3NF
2) **A 3NF** table which **does not have multiple overlapping** candidate keys is said to be in BCNF.

For proving that the table can be in 3NF but not in BCNF consider Following relation

R(Student, Subject, Teacher) . Consider following are FDs

(Subject, Student)-> Teacher

Because subject and student combination gives unique teacher.

Teacher -> Subject

Because each teacher teaches only Subject.

(Teacher, Student)->Subject

- So, this relation is in 3NF as every non-key attribute is non-transitively fully functional dependent on the primary key.
- But it is not in BCNF. Because this is a case of **overlapping of candidate keys** because there are two composite candidate keys :
  - (Subject, Student)
  - (Teacher, Student)

And student is a common attribute in both the candidate keys.

So we need to normalize the above table to BCNF. For that purpose we must set Teacher to be a candidate key

The decomposition of above takes place as follows

R1(Student, Teacher)

R2(Teacher, Subject)

Now table is in 3NF, as well as in BCNF.

This show that the relation Every relation which is in BCNF is in 3NF but the converse is not true.

Multivalued Dependencies and Fourth Normal Form

Concept of multivalued dependencies

- 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 values** of B exists, then the table may have multi-values 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.

- In simple terms, if there are two columns A and B - and for column A if there are multiple values of column B then we say that MVD exists between A and B
- The multivalued dependency is denoted by ↠
- If there exists a multivalued dependency then the table is **not in 4^{th} normal form.**
- **For example :** Consider following table for information about student

Student

sid	Course	Skill
1	C C++	English German
2	Java	English French

Here sid = 1 leads to multiple values for courses and skill. Following table shows this

sid	Course	Skill
1	C	English
1	C++	German
1	C	German
1	C++	English
2	Java	English
2	Java	French

Here **sid** and **course** are dependent but the **Course** and **Skill** are independent. The multivalued dependency is denoted as :

sid ↠ Course

sid ↠ Skill

Fourth normal form

Definition : For a table to satisfy the fourth normal form, it should satisfy the following two conditions :

1) It should be in the Boyce-Codd Normal Form(BCNF).
2) And, the table should not have any multi-valued dependency.

For example : Consider following student relation which is not in 4NF as it contains multivalued dependency.

Student table

sid	Course	Skill
1	C	English
1	C++	German
1	C	German
1	C++	English
2	Java	English
2	Java	French

Now to convert the above table to 4NF we must decompose the table into following two tables.

Student_Course Table

Key : (sid,Course)

sid	Course
1	C
1	C++
2	Java

Student_Skill Table

Key : (sid,Skill)

sid	Skill
1	English
1	German
2	English
2	French

Thus the tables are now in 4NF.

Join Dependencies and Fifth Normal Form

- Join decomposition is a further generalization of multivalued dependencies.
- If the join of R1 and R2 over C is equal to relation R, then we can say that a Join Dependency (JD) exists.
- Where R1 and R2 are the decompositions R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D).
- Alternatively, R1 and R2 are a lossless decomposition of R.
- A JD ⋈ {R1, R2,..., Rn} is said to hold over a relation R if R1, R2,....., Rn is a lossless-join decomposition.
- The *(A, B, C, D), (C, D) will be a JD of R if the join of join's attribute is equal to the relation R.
- Here, *(R1, R2, R3) is used to indicate that relation R1, R2, R3 and so on are a JD of R.

Concept of fifth normal form

The database is said to be in 5NF if -

i) It is in 4^{th} normal form

ii) If we can decompose table further to eliminate redundancy and anomalies and when we rejoin the table we should not be losing the original data or get a new record **(join Dependency Principle).**

The fifth normal form is also called as **project join normal form**

For example - Consider following table

Seller	Company	Product
Rupali	Godrej	Cinthol
Sharda	Dabur	Honey
Sharda	Dabur	Hairoil
Sharda	Dabur	Rosewater
Sunil	Amul	Icecream
Sunil	Britania	Biscuits

Here we assume the keys as{Seller, Company, Product}

The above table has multivalued dependency as

Seller→→{Company, Product}. Hence table is not in 4^{th} normal form. To make the above table in 4^{th} normal form we decompose above table into two tables as

Seller_Company	
Seller	**Company**
Rupali	Godrej
Sharda	Dabur
Sunil	Amul
Sunil	Britania

Seller_Product	
Seller	**Product**
Rupali	Cinthol
Sharda	Honey
Sharda	Hairoil
Sharda	Rosewater
Sunil	Icecream
Sunil	Biscuits

The above table is in 4^{th} normal form as there is no multivalued dependency. But it is not in 5^{th} normal form because if we join the above two table we may get

Seller	Company	Product
Rupali	Godrej	Cinthol
Sharda	Dabur	Honey
Sharda	Dabur	Hairoil
Sharda	Dabur	Rosewater
Sunil	Amul	Icecream
Sunil	Amul	Biscuits
Sunil	Britania	Icecream
Sunil	Britania	Biscuits

Newly added records which are not present in original table

To avoid the above problem we can decompose the tables into three tables as Seller_Company, Seller_Product, and Company Product table

Seller_Company

Seller	Company
Rupali	Godrej
Sharda	Dabur
Sunil	Amul
Sunil	Britania

Seller_Product

Seller	Product
Rupali	Cinthol
Sharda	Honey
Sharda	Hairoil
Sharda	Rosewater
Sunil	Icecream
Sunil	Biscuit

Company_Product

Company	Product
Godrej	Cinthol
Dabur	Honey
Dabur	Hairoil
Dabur	Rosewater
Amul	Icecream
Britania	Biscuit

Thus the table in in 5^{th} normal form.