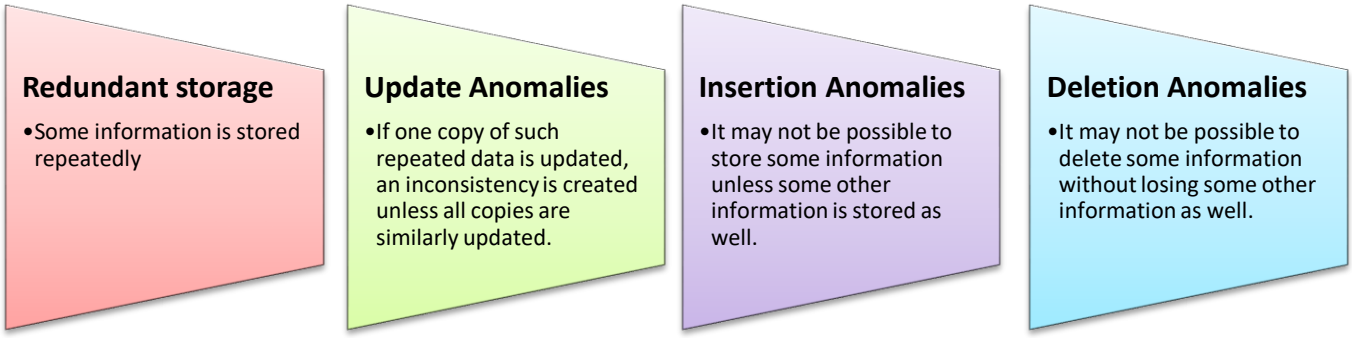
Schema Refinement (Normalization) – Purpose of Normalization or schema refinement, concept of functional dependency, normal forms based on functional dependency (1NF, 2NF and 3NF), concept of surrogate key, Boyce - codd normal form (BCNF), Lossless join and dependency preserving decomposition, Fourth normal form (4NF).

SCHEMA REFINEMENT

- The Schema Refinement refers to refine the schema by using some technique. The best technique of schema refinement is decomposition.
- Normalization or Schema Refinement is a technique of organizing the data in the database. It is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies. Redundancy refers to repetition of same data or duplicate copies of same data stored in different locations.
- Anomalies: Anomalies refers to the problems occurred after poorly planned and normalised databases where all the data is stored in one table which is sometimes called a flat file database.
- Problems caused by redundancy
  - ✓ **Redundant storage**
  - ✓ **Update Anomalies**
  - ✓ **Insertion Anomalies**
  - ✓ **Deletion Anomalies**



Consider the relation

Hourly_emps (ssn, name, lot, rating, hourly_wages, hours_worked)

The key for Hourly Emps is ssn. In addition, suppose that the hourly wages attribute is determined by the rating attribute. That is, for a given rating value, there is only one permissible hourly wages value. This IC is an example of a functional dependency. It leads to possible redundancy in the relation Hourly Emps

ssn	name	lot	Rating	Hourly_wages	Hours_worked
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

If the same value appears in the rating column of two tuples, the IC tells us that the same value must appear in the hourly wages column as well. This redundancy has several negative consequences:

- Some information is stored multiple times. For example, the rating value 8 corresponds to the hourly wage 10, and this association is repeated three times. In addition to wasting space by storing the same information many times,redundancy leads to potential inconsistency.
- For example, the *hourly wages* in the first tuple could be updated without making a similar change in the second tuple, which is an example of an ***update anomaly*.** Also, we cannot insert a tuple for an employee unless we know the hourly wage for the employee's rating value, which is an example of an *insertion anomaly*.
- If we delete all tuples with a given rating value (e.g., we delete the tuples for Smethurst and Guldu) we lose the association between that rating value and its hourly wage value (a **deletion anomaly**).

Null Values:

- ✓ null values cannot help eliminate redundant storage or update anomalies.
- ✓ null values can address insertion and deletion anomalies
- ✓ insertion anomaly example
  - we can insert an employee tuple with null values in the hourly wage field. null values cannot address all insertion anomalies. For example, we cannot record the hourly wage for a rating unless there is an employee with that rating
- ✓ deletion anomaly example
  - we might consider storing a tuple with null values in all fields except rating and hourly wages if the last tuple with a given rating would otherwise be deleted. However, this solution will not work because it requires the ssn value to be null, and primary key fields cannot be null.
- ✓ • Null values do not provide a general solution to the problems of redundancy,

Decomposition

- Technique used to eliminate the problems caused by redundancy.
- Def: A decomposition of a relation schema R consists of replacing the relation schema by two (or more) relation schemas that each contain a subset of the attributes of R and together include all attributes of R.
- in simple words, Splitting the relation into two or more sub tables

(or)

The essential idea is that many problems arising from redundancy can be addressed by replacing a relation with a collection of `smaller' relations. Each of the smaller relations contains a (strict) subset of the attributes of the original relation. We refer to this process as *decomposition* of the larger relation into the smaller relations.
For example:

We can deal with the redundancy in Hourly Emps by decomposing it into two relations:
Hourly Emps2(ssn, name, lot, rating, hours worked)
Wages(rating, hourly wages)

ssn	name	lot	Rating	Hours_worked
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Rating	Hourly_wages
8	10
5	7

Note that we can easily record the hourly wage for any rating simply by adding a tuple to Wages, even if no employee with that rating appears in the current instance of Hourly Emps. Changing the wage associated with a rating involves updating a single Wages tuple. This is more efficient than updating several tuples (as in the original design), and it also eliminates the potential for inconsistency. Notice that the insertion and deletion anomalies have also been eliminated.

Problems Related to Decomposition
Unless we are careful, decomposing a relation schema can create more problems than it solves. Two important questions must be asked repeatedly:

1. Do we need to decompose a relation?
2. What problems (if any) does a given decomposition cause?

To help with the first question, several *normal forms* have been proposed for relations. If a relation schema is in one of these normal forms, we know that certain kinds of problems cannot arise. Considering the normal form of a given relation schema can help us to decide whether or not to decompose it further. If we decide that a relation schema must be decomposed further, we must choose a particular decomposition (i.e., a particular collection of smaller relations to replace the given relation).

With respect to the second question, two properties of decompositions are of particular interest.
The ***lossless-join* property** enables us to recover any instance of the decomposed relation from corresponding instances of the smaller relations.
The ***dependency-preservation* property** enables us to enforce any constraint on the original relation by simply enforcing some constraints on each of the smaller relations. That is, we need not perform joins of the smaller relations to check whether a constraint on the original relation is violated.

FUNCTIONAL DEPENDENCIES

A functional dependency (FD) is a kind of Integrity constraint that generalizes the concept of a key.
Let R be a relation schema and let X and Y be nonempty sets of attributes in R. We say that an instance r of R satisfies the Functional Dependency **X → Y**
if the following holds for every pair of tuples t1 and t2 in r:

If t1:X = t2:X, then t1:Y = t2:Y .

We use the notation t1:X to refer to the projection of tuple t1 onto the attributes in X, in a natural extension of our TRC notation t:a for referring to attribute a of tuple t.
An FD X -> Y essentially says that if two tuples agree on the values in attributes X, they must also agree on the values in attributes Y.

Example:

Table Student

studID	lastname	status	credits
1234567	Smith	Undergraduate	254
1234568	Don	Undergraduate	149
1234569	Dallas	Graduate	543

Functional dependencies include:

{stuId}→{lastName}
{stuId} →{lastName, credits, status, stuId}
{credits}→{status} (but not {status}→{credits})

Example :

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

Instance r of R

AB → C	Holds on r
B → C	Doesn't Hold on r
A → B	Doesn't Hold on r
ABC → D	Doesn't Hold on r
CD → A	Holds on r

- A primary key constraint is a special case of an FD. The attributes in the key play the role of X, and the set of all attributes in the relation plays the role of Y.

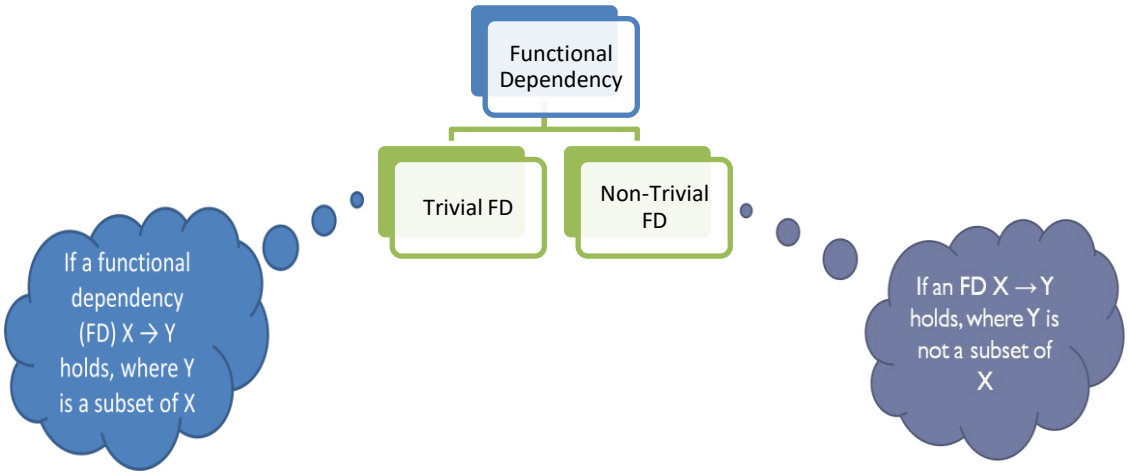
Closure of a set of FDs:

- The set of all FDs implied by a set F of FDs is called closure of F
- denoted as F+
- Three rules, called Armstrong Axioms can be applied repeatedly to infer all FDs implied by a set F of FDs.
- Armstrong Axioms
  - **Reflexivity : if X ⊇ Y, then X → Y**
  - **Augmentation : if X→ Y, then XZ → YZ for any Z**
  - **Transitivity : if X → Y, and Y → Z, then X →Z**
- Some additional rules while reasoning about FDs
  - **Union : if X → Y and X → Z, then X → YZ**
  - **Decomposition : if X → YZ, then X → Y and X → Z**
  - **Pseudo Transitivity Rule : if X → Y and YW → Z, then XW → Z**

Solve these:

- Let F = { A→ B, C → X, BX→Z }
  Show that AC → Z can be inferred from F using inference rules.
  From A→ B and BX→Z, we can apply **Pseudo Transitivity Rule**
  AX → Z ---(1)
  From (1) apply C → X
  AC→Z

- Let F = { A→ B, C → D } and C ⊆ B
  Show that A → D can be inferred from F using inference rules.



Trivial Functional dependency:

The **Trivial dependency** is a set of attributes which are called a trivial if the set of attributes are included in that attribute. So, X -> Y is a trivial functional dependency if Y is a subset of X.

For example:

Emp_id	Emp_name
AS555	Karthiika
AS811	Vedhansh
AS999	Vamsika

Consider this table with two columns Emp_id and Emp_name. {Emp_id, Emp_name} -> Emp_id is a trivial functional dependency as Emp_id is a subset of {Emp_id,Emp_name}.

Non trivial functional dependency:

Functional dependency which also known as a nontrivial dependency occurs when A->B holds true where B is not a subset of A. In a relationship, if attribute B is not a subset of attribute A, then it is considered as a non-trivial dependency.

For example:

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	38

(Company} -> {CEO} (if we know the Company, we knows the CEO name)
But CEO is not a subset of Company, and hence it's non-trivial functional dependency.

Advantages of Functional Dependency:

- Functional Dependency avoids data redundancy. Therefore same data do not repeat at multiple locations in that database.
- It helps you to maintain the quality of data in the database
- It helps you to defined meanings and constraints of databases
- It helps you to identify bad designs
- It helps you to find the facts regarding the database design

Attribute Closure:

- Attribute Closure of X (X^+) w.r.t to given set of FDs F, *is the set of all attributes functionally determined by X* under the set F.

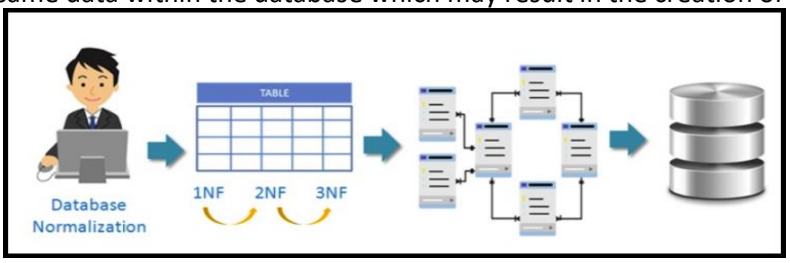
- Algorithm to find out the Attribute closure of attribute set X

closure = *X*;
repeat until there is no change: {
    if there is an FD $U \rightarrow V$ in *F* such that $U \subseteq$ *closure*,
        then set *closure* = *closure* ∪ *V*
}

- Applications of attribute closure.
  - To identify the additional FDs
  - To identify the keys
  - To identify the equivalence of the FDs
  - To identify the minimal set of FDs / irreducible set of FDs / canonical forms of FDs.

What is Normalization?

Normalization is a method of organizing the data in the database which helps you to avoid data redundancy, insertion, update & deletion anomaly. It is a process of analyzing the relation schemas based on their different functional dependencies and primary key. Normalization is inherent to relational database theory. It may have the effect of duplicating the same data within the database which may result in the creation of additional tables.



Normalization of Database :

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multistep process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

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

Problems Without Normalization:

If a table is not properly normalized and have data redundancy then it will not only eat up extra memory space but will also make it difficult to handle and update the database, without facing data loss. Insertion, Updation and Deletion Anomalies are very frequent if database is not normalized. To understand these anomalies let us take an example of a Student table.

ROLL NOR	NAME	BRANCH	HOD	PHONE NO
401	ANU	CSE	Mr. X	53337
402	AJAY	CSE	Mr. X	53337
403	RAHUL	CSE	Mr. X	53337
404	KIRAN	CSE	Mr. X	53337

In the table above, we have data of 4 Computer Sci. students. As we can see, data for the fields branch, hod(Head of Department) and office_tel is repeated for the students who are in the same branch in the college, this is **Data Redundancy.**

Insertion Anomaly-

- Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information as NULL.
- Also, if we have to insert data of 100 students of same branch, then the branch information will be repeated for all those 100 students.
- These scenarios are nothing but Insertion anomalies.

Updation Anomaly - What if Mr. X leaves the college? or is no longer the HOD of computer science department? In that case all the student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency. This is Updation anomaly.

Deletion Anomaly- In our Student table, two different informations are kept together, Student information and Branch information. Hence, at the end of the academic year, if student records are deleted, we will also lose the branch information. This is Deletion anomaly.

Candidate Key: Candidate Key is minimal set of attributes of a relation which can be used to identify a tuple uniquely.
Consider student table: ***student (sno, sname, sphone, age)*** we can take **sno** as candidate key.
we can have more than 1 candidate key in a table.
Types of candidate keys:

1. simple (having only one attribute)
2. composite (having multiple attributes as candidate key)

Example: *Finding candidate keys for a table*

R(A, B, C, D)
F = { AB → C, B → D, D → B}
How many candidate keys are there for R?
What are prime attributes in R

$(A)^+ = \{ A \}$
$(AB)^+ = \{ ABCD \}$ ✓
$(AC)^+ = \{ AC \}$
$(AD)^+ = \{ ADBC \}$ ✓

Super keys.....
ABC
ABD
ACD
ABCD

Prime Attributes
A, B, D

Super Key: Super Key is set of attributes of a relation which can be used to identify a tuple uniquely. Adding zero or more attributes to candidate key generates super key.

- A candidate key is a super key but vice versa is not true.
- Consider student table: student(sno, sname,sphone,age) we can take **sno**, **(sno, sname)** as super key.

Prime and non-prime attributes:

✓ A prime attribute is **an attribute that is part of any candidate key**. It can also be used to uniquely identify a tuple in the schema.
✓ A prime attribute in DBMS is also known as a key attribute.

✓ A non-prime attribute is one that is not part of one of the candidate keys.

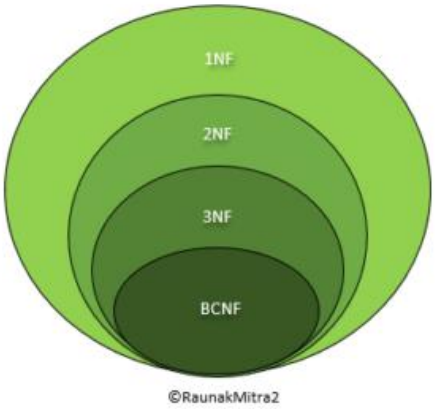
Non Prime Attribute

Roll_No	Name	Age	GPA
1	Arya	21	4
2	Bran	19	3
3	John	24	4.3
4	Max	24	1

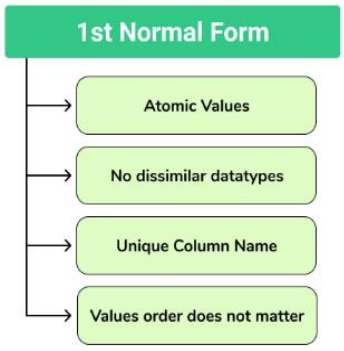
Non Prime Attributes

Normal Forms

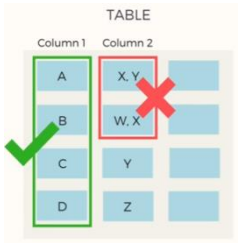
- Provides a guidance to decide whether a database design is good or needs decomposition.
- Can be used to identify the presence of redundancy in relations.
- Normal forms based on FDs are
  - First normal form (1 NF)
  - Second normal form (2 NF)
  - Third normal form (3 NF)
  - Boyce-Codd normal form (BCNF)
- A relational table is said to be in a particular normal form if it satisfied a certain set of constraints.
- If a relation is in one the of the higher normal forms, then it will be automatically in all lower normal forms
- The higher the normal form the lower the redundancy. Therefore higher normal forms are preferable.



First Normal Form (1 NF):

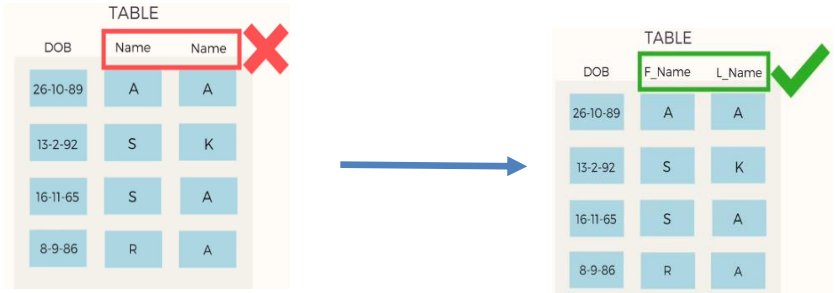


It is a level of normalization in DBMS. A relation is said to be in 1 normal form in DBMS (or 1NF) when it consists of an atomic value. In simpler words, 1NF states that a table's attribute would not be able to hold various values- it will only be able to hold an attribute of a single value.

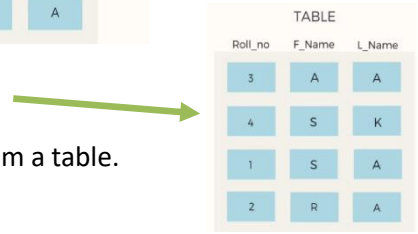


➢ A schema is in First normal form
  - There are 4 basic rules that a table should follow to be in 1st Normal Form
  - **Rule 1 :**
    - if the domain of every attribute is atomic
      - That is, no composite values (lists or sets)
      - Entries like X,Y and W,X violates the rule

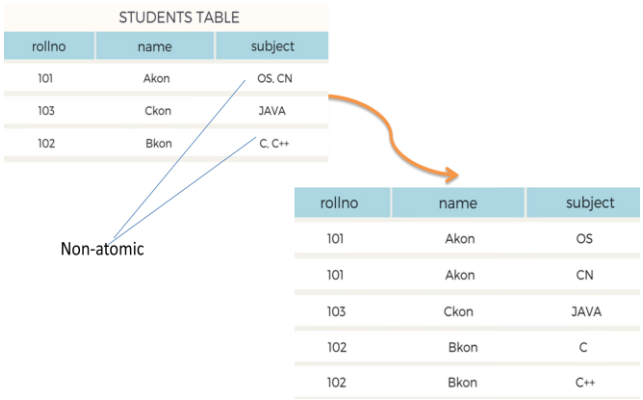
  - **Rule 2:**
    - All entries in any column must be of the same kind.
    - Do not inter-mix different types of values in any column
  - **Rule 3:**
    - Each column must have a unique name
    - Same name leads to confusion at the time of data retrieval



- Rule 4:
  - Order in which data stored doesn't matter.
  - No two rows are identical
  - Using SQL query, we can easily fetch data in any order from a table.



Example:



This way, although a few values are getting repeated, we can still see that there is just one value in every column.

Problem Table:

- Consider a table where a student can have multiple phone numbers

Roll no	Name	Phone
66	Trishaank	P1
73	Prashant	P2, P3
79	Sanjay	P4
82	Srinivas	P5

Atomic values

Divide the table into two parts such that all the ***multivalued attributes in one table at single-valued attributes in another table*** and add primary key attribute of the original table to each newly formed table

Roll no	Phone
66	P1
73	P2
73	P3
79	P4
82	P5

Roll no	Name
66	Trishaank
73	Prashant
79	Sanjay
82	Srinivas

Partial Dependency:

A functional dependency X->Y is a partial dependency if Y is functionally dependent on X and Y can be determined by any proper subset of X.

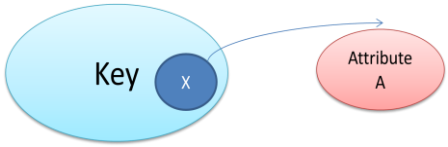
For example, we have a relationship AC->B, A->D, and D->B.

Now if we compute the closure of $\{A^+\}$ = ADB

Here A is alone capable of determining B, which means B is partially dependent on AC.

Example:

A is not part of key X-> A is a partial dependency



Let us take another example –

name	roll_no	course
Ravi	2	DBMS
Tim	3	OS
John	5	Java

Here, we can see that both the attributes name and roll_no alone are able to uniquely identify a course. Hence we can say that the relationship is partially dependent.

Full Functional Dependency:

In X → Y,

if Y cannot be determined by any of subsets of X, then Y is said to be fully functionally dependent on X

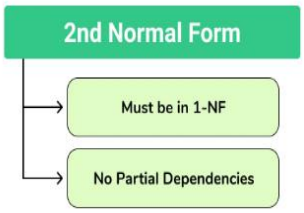
Example – In the relation ABC->D, attribute D is fully functionally dependent on ABC and not on any proper subset of ABC. That means that subsets of ABC like AB, BC, A, B, etc cannot determine D.

supplier_id	item_id	price
1	1	540
2	1	545
1	2	200
2	2	201

From the table, we can clearly see that neither supplier_id nor item_id can uniquely determine the price but both supplier_id and item_id together can do so. So we can say that price is fully functionally dependent on { supplier_id, item_id }. This summarizes and gives our fully functional dependency – { supplier_id , item_id } -> price.

Full Functional Dependency	Partial Functional Dependency
A functional dependency X->Y is a fully functional dependency if Y is functionally dependent on X and Y is not functionally dependent on any proper subset of X.	A functional dependency X->Y is a partial dependency if Y is functionally dependent on X and Y can be determined by any proper subset of X.
In full functional dependency, the non-prime attribute is functionally dependent on the candidate key.	In partial functional dependency, the non-prime attribute is functionally dependent on part of a candidate key.
In fully functional dependency, if we remove any attribute of X, then the dependency will not exist anymore.	In partial functional dependency, if we remove any attribute of X, then the dependency will still exist.
Full Functional Dependency equates to the normalization standard of Second Normal Form.	Partial Functional Dependency does not equate to the normalization standard of Second Normal Form. Rather, 2NF eliminates the Partial Dependency.
An attribute A is fully functional dependent on another attribute B if it is functionally dependent on that attribute, and not on any part (subset) of it.	An attribute A is partially functional dependent on other attribute B if it is functionally dependent on any part (subset) of that attribute.
Functional dependency enhances the quality of the data in our database.	Partial dependency does not enhance the data quality. It must be eliminated in order to normalize in the second normal form.

Second Normal Form:



It is a normalization level in DBMS. A relation is said to be in the 2nd Normal Form in DBMS (or 2NF) when it is in the First Normal Form but has no non-prime attribute functionally dependent on any candidate key's proper subset in a relation. A relation's non-prime attribute refers to that attribute that isn't a part of a relation's candidate key.

Uses of Second Normal Form in DBMS:

The concept of 2nd Normal Form in DBMS depends on full functional dependency. We apply 2NF on the relations that have composite keys or the relations that have a primary key consisting of two attributes or more. Thus, the relations having a primary key of a single attribute automatically get to their 2NF. Any relation that doesn't exist in the 2NF may eventually suffer from further update anomalies.

Not allowed FDs –

```
prime -> non prime
```

If say your functional dependency is of the form A->X where 'A' is a prime attribute but not a key and 'X' is a non prime attribute, then such an FD is not allowed in 2NF.

Allowed FDs –

```
Prime -> Prime
Non prime -> Prime/Non prime
Key -> Prime/Non prime
Prime + Non prime combination -> Prime/Non Prime
```

1. ***Is this relation in 2 NF***

   Consider the relational schema R (A, B, C, D) and the functional dependencies.

   F = { AB → C, BC → D, A → D}

   - *It is in 1NF*
   - *2NF:*

   $(A)^+ = \{A,D\}$
   $(B)^+ = \{B\}$
   $(C)^+ = \{C\}$
   $(D)^+ = \{D\}$

   $(AB)^+ = \{A,B,C,D\}$ ✓
   $(AC)^+ = \{A,C,D\}$
   $(AD)^+ = \{A,D\}$
   $(BC)^+ = \{B,C,D\}$
   $(BD)^+ = \{B,D\}$
   $(CD)^+ = \{C,D\}$

   Prime Attributes — A, B

   Non-Prime Attributes — C, D

   Clearly, prime attributes for Relation R are: {A,B} while non-prime attributes are: {C,D}.

   Check the partial Dependencies:

AB → C	✓	*AB is a key*
BC → D	✓	*A composite key determines non-prime attribute*
A → D	✗	*Partial dependency*

   As the partial dependency exists, therefore it is not in 2NF.

2. Consider schema R = (A, B, C, G, H, I) and the set F of functional dependencies {A→ B, A → C, CG → H, CG → I, B → H}. Compute the candidate keys of the schema. Check whether the schema is in 2NF or not

i. It is in 1NF

ii. 2NF:

$(A)^+ = \{A,B,C,H\}$
$(B)^+ = \{B,H\}$
$(C)^+ = \{C\}$
$(G)^+ = \{G\}$
$(H)^+ = \{H\}$
$(I)^+ = \{I\}$
$(AB)^+ = \{A,B,H\}$
$(AC)^+ = \{A,C,B,H\}$
$(AG)^+ = \{A,G,B,C,H,I\}$ ✓
$(AH)^+ = \{A,H,B,C\}$
$(HI)^+ = \{H,I\}$
$(AI)^+ = \{A,I,B,C,H\}$
$(BC)^+ = \{B,C,H\}$
$(BG)^+ = \{B,G,H\}$
$(BH)^+ = \{B,H\}$
$(BI)^+ = \{B,I,H\}$
$(CG)^+ = \{C,G,H,I\}$
$(CH)^+ = \{C,H\}$
$(CI)^+ = \{C,I\}$
$(GH)^+ = \{G,H\}$
$(GI)^+ = \{G,I\}$

Prime Attributes — A, G

Non-Prime Attributes — B, C, H, I

Clearly, prime attributes for Relation R are: {A,G} while non-prime attributes are: {B,C,H,I}.

Check the partial Dependencies:

$A \rightarrow B$	✗	*Partial dependency*
$A \rightarrow C$	✗	*Partial dependency*
$CG \rightarrow H$	✓	*A composite key determines non-prime attribute*
$CG \rightarrow I$	✓	*A composite key determines non-prime attribute*
$B \rightarrow H$	✓	*non-prime attribute determines prime attribute*

As the partial dependency exists, therefore it is not in 2NF.

3. Decompose the following table into 2NF:

Problem: Score Table

Student_id	Subject_id	Marks	teacher
191	13A	70	Java Teacher
191	15D	78	DBMS teacher
181	13A	80	Java Teacher

- So even if the table is in First Normal Form it suffers from problems due to insertion, updation and deletion hence we need to go for second normal form.
- Dependency are
  - Student_id, subject_id → marks
  - Subject_id → teacher
- Identify the partial and full dependencies and apply decomposition rule.

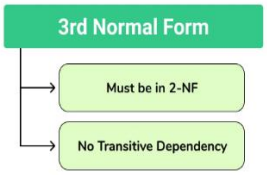
In the above table column teacher is dependent on subject_id

- The simple solution is to create a separate table for subject_id and teacher and removing it from Score table

Subject_id	teacher
13A	Java Teacher
15D	DBMS teacher
13A	Java Teacher

Student_id	Subject_id	Marks
191	13A	70
191	15D	78
181	13A	80

Third Normal Form:



A relation (table) is said to in 3NF:

- If it is in 2NF.
- There must be no transitive functional dependency for non prime attributes.

Transitive FD : "If non prime attribute is dependent on another non prime attribute then it is called transitive functional dependency."

- Let R be a relation schema,
- F be the set of FDs given to hold over R,
- X be a subset of the attributes of R,
- And A be an attribute of R.
- R is in third normal form if, for every FD X → A in F, one of the following statements is true:
- that is, it is a trivial FD, or
  - X is a super key , or
  - A is part of some key for R (prime attribute)

In simple words
- R should be in 2 NF
- No NPA should be transitively dependent on Candidate key

Q.1 Check whether the following relations are in 3NF or not R (A, B, C)

F = { A → B, B → C}

Solution: Firstly find the candidate key in the relation:

$(A)^+ = ABC$

A is the candidate key, because closure of A has all the attributes of R.

Prime Attributes *Non-Prime Attributes*

Clearly, prime attributes for Relation R are: {A} while non-prime attributes are: {B, C}.

A relation is said to be **2NF**, if it is 1NF and for dependency X -> a, there should not be any partial dependency.

Check the partial Dependencies:

A → B ✓ *fully functional dependency*

B → C ✓ *Non-prime determines non-prime attribute*

As the no partial dependency exists , therefore it is in 2NF.

A relation is said to be **3NF**, if it holds at least one of the following for every non trivial functional dependency X -> a:

X is super key.

a is prime attribute.

	Trivial Dependency		Super Key	Prime Attributes
A -> B	No	→	A ✓	B ✗
B -> C	No	→	B ✗	C ✗

A -> B – A is super key and B is not a prime attribute.
B -> C -- Neither B is super key, nor C is prime attribute.
So, the relation is not in 3NF as it is not following the rules of 3NF.

Therefore, R(ABC) needs to be divided into following:
R1(A B)
R2 (B C)
Now R1 and R2 are in 3NF.

Q.2 Suppose a relational schema R (A B C D E) and set of functional dependencies
F: { A -> B B-> E C-> D }
Check out that relation is in 3NF or not? If not decompose it in 3NF.

Solution: Firstly find the candidate key in the relation:
$(AC)^+$ = ABCDE
AC is the candidate key, because closure of AC has all the attributes of R.

Prime Attributes *Non-Prime Attributes*

Clearly, prime attributes for Relation R are: {A,C} while non-prime attributes are: {B,D,E}.

A relation is said to be 3NF, if it holds at least one of the following for every non trivial functional dependency X -> a:
X is super key.
a is prime attribute.
A -> B – Neither A is super key nor B is prime attribute.
B -> E -- Neither B is super key, nor E is prime attribute.
C-> D -- Neither C is super key, nor D is prime attribute.

	Trivial Dependency		Super Key	Prime Attributes
A -> B	No	→	A ✗	B ✗
B -> E	No	→	B ✗	E ✗
C -> D	No	→	C ✗	D ✗

So, the relation is not in 3NF as it is not following the rules of 3NF.
Therefore, R(ABCDE) needs to be divided into following:
R1(A B E) → R11 (A B)
R12 (B E)
R2 (C D)
R3(A C)
Now R11, R12, R2, R3 are in 3NF.

Surrogate Key

- A artificial primary key which is generated automatically by the system
- The value of surrogate key is numeric
- It is automatically incremented for each new row
- Surrogate key is called the factless key as it is added just for our ease of identification of unique values and contains no relevant fact (or information) that is useful for the table.
- Example: Suppose we have two tables of two different colleges having the same column registration_no , name and percentage , each table having its own natural primary key, that is registration_no.

Table of College VIT

registration_no	name	percentage
210101	Harry	90
210102	Maxwell	65
210103	Lee	87
210104	Chris	76

Table of College SVECW

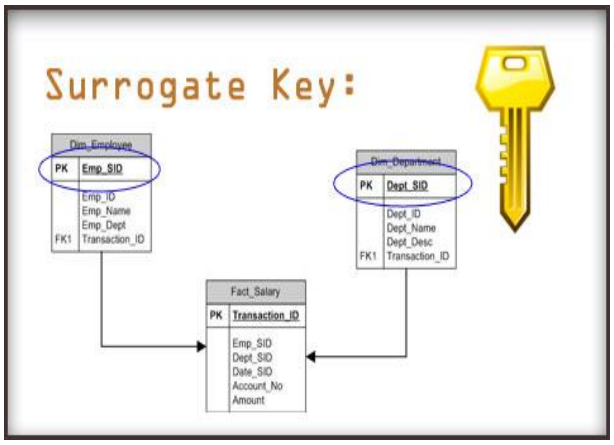
registration_no	name	percentage
CS107	Taylor	49
CS108	Simon	86
CS109	Sam	96
CS110	Andy	58

- Now, suppose we want to merge the details of both
- Resulting table will be –

surr_no	registration_no	name	percentage
1	210101	Harry	90
2	210102	Maxwell	65
3	210103	Lee	87
4	210104	Chris	76
5	CS107	Taylor	49
6	CS108	Simon	86
7	CS109	Sam	96
8	CS110	Andy	58

Registration_no cannot be the primary key of the table as it does not match with all the records of the table though it is holding all unique values of the table .

Now , in this case, we have to artificially primary key for this table. We can do this by adding a column surr_no in the table that contains anonymous integers and has no direct relation with other columns .



Some examples of Surrogate key are :

- System date & time stamp
- Random alphanumeric string

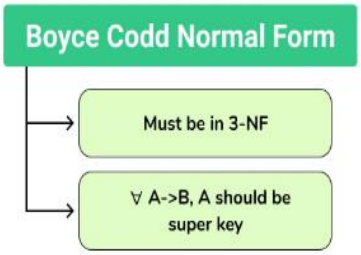
Advantages of the surrogate key :

- Enables us to run fast queries
- Performance is enhanced as the value of the key is relatively smaller.
- The key value is guaranteed to contain unique information .

Disadvantages of the surrogate key :

- Can never be used as a search key.
- As the key value has no relation to the data of the table, so third normal form is violated.
- The extra column for surrogate key will require extra disk space.
- We will need extra IO when we have to insert or update data of the table.

Boyce-Codd Normal Form:



- Let R be the relation schema, F be the set of FDs that hold on R we say that R is in BCNF w.r.t F, iff, for every FD in F, one of the following 2 rules conditions is true.
  - A ϵ X if X →A is a trivial FD or
  - X is a super key.
- BCNF is based on functional dependencies, and all the candidate keys of the relation are taken into consideration. BCNF is stricter than 3NF and has some additional constraints along with the general definition of 3NF.

Example: Consider a relation R with attributes (student, subject, teacher).

Student	Teacher	Subject
Jhansi	P.Naresh	Database
jhansi	K.Das	C
subbu	P.Naresh	Database
subbu	R.Prasad	C

F: { **(student, Teacher) -> subject**
(student, subject) -> Teacher
Teacher -> subject}

- Candidate keys are (student, teacher) and (student, subject).
- The above relation is in 3NF [since there is no transitive dependency]. A relation R is in BCNF if for every non-trivial FD X->Y, X must be a key.
- The above relation is not in BCNF, because in the FD (teacher->subject), teacher is not a key. This relation suffers with anomalies –
- For example, if we try to delete the student Subbu, we will lose the information that R. Prasad teaches C. These difficulties are caused by the fact the teacher is determinant but not a candidate key.

Decomposition for BCNF

Teacher-> subject violates BCNF [since teacher is not a candidate key].

If X->Y violates BCNF then divide R into R1(X, Y) and R2(R-Y).

So R is divided into two relations R1(Teacher, subject) and R2(student, Teacher).

R1

Teacher	Subject
P.Naresh	database
K.DAS	C
R.Prasad	C

R2

Student	Teacher
Jhansi	P.Naresh
Jhansi	K.Das
Subbu	P.Naresh
Subbu	R.Prasad

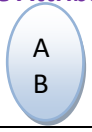
All the anomalies which were present in R, now removed in the above two relations.

Example: find the highest normalization form, and for that, we are given a relation **R(A, B, C)** with functional dependencies as follows: {AB ->C, C ->B, AB ->B}

Sol: $(AB)^+ = \{A,B,C\}$

Candidate Key (given): **{AB}**

Prime Attributes



Non-Prime Attributes

Clearly, prime attributes for Relation R are: {A,B} while non-prime attributes are: {C}.

- **It is in First Normal Form**
- **Second normal form**

	Partial Dependency
AB->C	No ✗ (fully functionally dependencies)
AB->B	No ✗ (fully functionally dependencies)
C->B	No ✗ (non-prime attribute determines prime attribute)

As there is no partial dependency , the given relation schema is in **2NF.**

- **Third normal form**

	Trivial Dependency	Super Key	Prime attribute
AB->C	No ✗	AB ✓	C ✗
AB->B	Yes ✓	AB ✓	B ✓
C->B	No ✗	C ✗	B ✓

As there is no transitive dependency , the given relation schema is in **3NF.**

- **Boyce-Codd normal form**

	Trivial Dependency	Super Key
AB->C	No ✗	AB ✓
AB->B	Yes ✓	AB ✓
C->B	No ✗	C ✗

- Clearly, {AB->C} and {AB->B} are in BCNF because
  - AB is the candidate key present on the LHS of both dependencies.
  - The second dependency, {C->B}, however, is not in BCNF because C is neither a super key nor a candidate key.
- C->B is, however, present in 3NF because B is a prime attribute that satisfies the conditions of 3NF. Hence, relation R has 3NF as the highest normalization form.

Decomposition:

R consists of replacing the relation schema by two (or more) relation schemas that each contain a subset of the attributes of R and together include all attributes in R. Intuitively, we want to store the information in any given instance of R by storing projections of the instance. This section examines the use of decompositions.

- A tool that allows us to eliminate redundancy.
- Properties of decomposition
  - Lossless-Join Decomposition
  - Dependency-Preserving Decomposition

- **Lossless-Join Decomposition:** Let R be a relation schema, F be the set of FDs that hold on R.

- The decomposition of R into 2 sub-schemas with attribute sets X and Y is said to be loss-less join decomposition w.r.t. 'F', iff, for every instance 'r' of 'R' that satisfies the dependencies in F, the following condition should be true.

$$\pi_X(r) \bowtie \pi_Y(r) = r$$

- i.e., we should be able to recover the original relation from decomposed relations
- Loss-less decomposition also called as non-additive join decomposition.
- No extraneous tuples appear after joining of the sub-relations.

- Decomposition of $r = (A, B, C)$ into:
  $r_1 = (A, B)$ and $r_2 = (B, C)$

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_{r_1}(r) \bowtie \Pi_{r_2}(r)$

A	B	C
α	1	A
β	2	B

This decomposition is loss-less join decomposition

As it retains the original relation after joining the decomposed relations.

Note: All decompositions used to eliminate redundancy must be lossless join decomposition.

Let R be a relation and F be a set of FDs that hold over R. decomposition of R into relations with attribute sets R_1 and R_2 is lossless if and only if F^+ contains either the FD $R_1 \cap R_2 \rightarrow R_1$ or the FD $R_1 \cap R_2 \rightarrow R_2$

In other words
R_1 and R_2 must have one attribute in common and
The common attribute must be a super key for either R_1 or R_2

Example 1:

R(A, B, C, D)
F = { A →B, B → C, C→A}
R is decomposed into R1(A,B,C) and R2(C,D).
Find whether the decomposition is loss-less or lossy join decomposition?

Solution:

- **Attributes(R1) ∪ Attributes(R2) = Attributes(R)**
  - {A, B, C} ∪ {C,D} = {A,B,C,D} = Attributes(R) ✓
- **Attributes(R1) ∩ Attributes(R2) ≠ ϕ**
  - {A, B, C} ∩ {C,D} = {C} ≠ ϕ ✓
- **Attributes(R1) ∩ Attributes(R2) = Super key of either R1 or R2**

  {A, B, C} ∩ {C, D} = {C}
  R1(A,B,C) :
  (C)$^+$ = {C, A, B} can determine all the attributes of R1, C is a super key for R1 ✓
  R2(C,D) :
  (C)$^+$ = {C, A, B} cannot determine all the attributes of R2, C is not a super key for R2 ✗

All the properties were satisified. Therefore it is a loss-less join decomposition.

Example 2:

R(X, Y, Z)
F = { X →Y, Y → Z}
R is decomposed into R1(X,Y) and R2(Y,Z). Find whether the decomposition is loss-less or lossy?

Solution:

- **Attributes(R1) ∪ Attributes(R2) = Attributes(R)**
  - {X,Y} ∪ {Y,Z} = {X,Y,Z} = Attributes(R) ✓
- **Attributes(R1) ∩ Attributes(R2) ≠ ϕ**
  - {X,Y} ∩ {Y,Z} = {Y} ≠ ϕ ✓
- **Attributes(R1) ∩ Attributes(R2) = Super key of either R1 or R2**
  - {X,Y} ∩ {Y,Z} = {Y}
    R1(X,Y) :
    $(Y)^+$ = {Y} cannot determine X, Y soo it is not a super key for R1 ✗
    R2(Y,Z)
    $(Y)^+$ = {Y,Z} can determine all the attributes of R2, Y is a super key for R2 ✓

All the properties were satisified. Therefore it is a loss-less join decomposition.

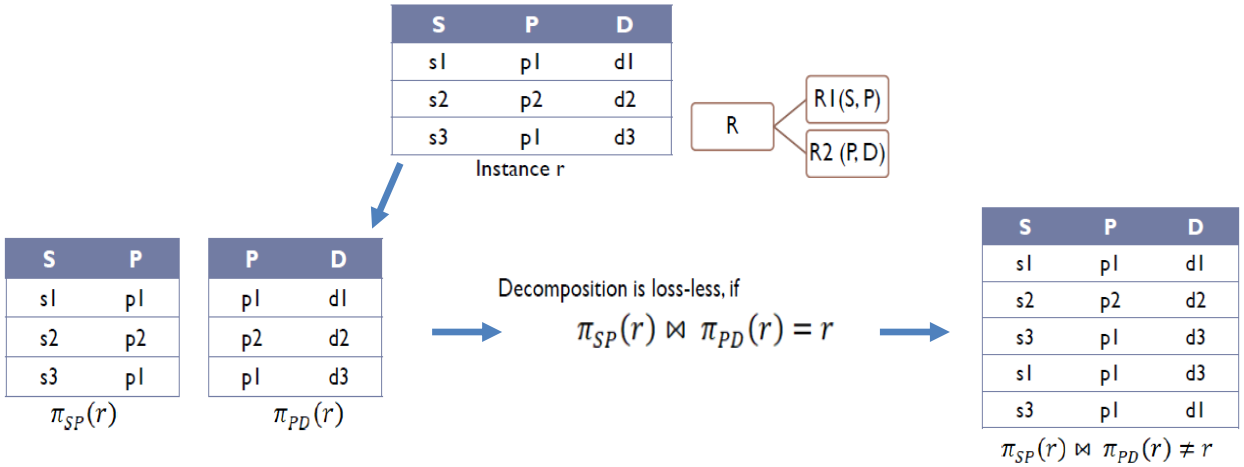
Lossy- Join decomposition: A relation X is decomposed into sub relations X_1, X_2, … X_n. This decomposition is called lossy join decomposition. When the join of the sub relations does not result in the same relation R that was decomposed.

- Contains extraneous tuples

$$X1 \bowtie X2 \bowtie X3 \ldots\ldots \bowtie Xn \supset X$$

Here, the operator ⋈ acts as a natural join operator.

- Lossy decomposition also called as "careless decomposition".
- Because of extraneous tuples, identification of the original tuples is difficult.
- Example:



This decomposition is not loss-less
It is a lossy decomposition.

Dependency Preserving Decomposition:

- Let R be a relation schema and F be the set of FDs that hold over R.
- The decomposition of R into sub schemas with attribute sets X and Y is dependency preserving, if

$$(F_X \cup F_Y)^+ = F^+$$

  - F_X is the set of FDs in F^+ that includes only attributes of X
  - F_Y is the set of FDs in F^+ that includes only attributes of Y

Intuitively, a dependency-preserving decomposition allows us to enforce all FDs by examining a single relation instance on each insertion or modification of a tuple.

- In the dependency preservation, **at least one decomposed table must satisfy every dependency**. If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.

Example 1:

R (A, B, C)

F = { A → B, B → C, C→A} R is decomposed into R1(A,B) and R2(B,C). Find whether the decomposition is dependency preserving or not?

Solution: The decomposition of R into sub schemas with attribute sets X and Y is dependency preserving,if

$$(F_{R1} \cup F_{R2})^+ = F^+$$

R1(A,B)			R2(B,C)		
$(A)^+ = \{A,B,C\}$	$(A)^+ = \{A,B,C\}$ (skip the attributes which are not in the R1 relation and also same attribute of left side).	**A → B**	$(B)^+ = \{B,C,A\}$	$(B)^+ = \{B,C,A\}$ (skip the attributes which are not in the R2 relation and also same attribute of left side).	**B → C**
$(B)^+ = \{B,C,A\}$	$(B)^+ = \{B,C,A\}$ (skip the attributes which are not in the R1 relation and also same attribute of left side).	**B → A**	$(C)^+ = \{C,A,B\}$	$(C)^+ = \{C,A,B\}$ (skip the attributes which are not in the R2 relation and also same attribute of left side).	**C → B**
$(AB)^+ = \{A,B,C\}$	A is determining all the attributes of R1, so no need to check the combinations	-	$(BC)^+ = \{B,C,A\}$	B is determining all the attributes of R2, so no need to check the combinations	-

$(F_{R1} \cup F_{R2})$ = { **A → B, B → A, B → C, C → B** }

Now we have to check whether $(F_{R1} \cup F_{R2})$ covers F

Consider functional dependencies of R

F : FD's of R	$(F_{R1} \cup F_{R2})$ covers F { A → B, B → A, B → C, C → B } Covers F
A → B	$(A)^+ = \{A,B,C\}$ ✓
B → C	$(B)^+ = \{B,C,A\}$ ✓
C→ A	$(C)^+ = \{A,B,C\}$ ✓

- As all the original dependencies are available in decomposed dependencies. Therefore we can say that it is a dependency preserving decomposition.

Multi-value dependency

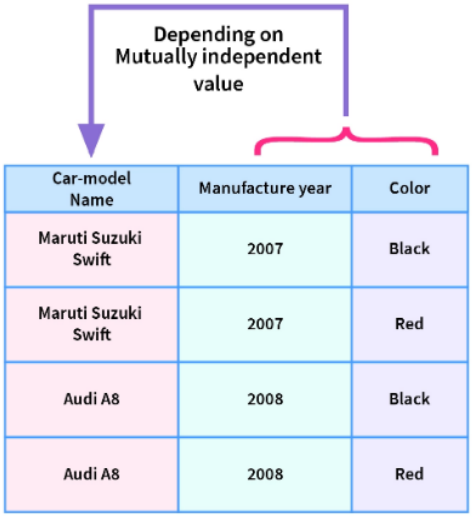
- Multivalued dependency would occur whenever two separate attributes in a given table happen to be independent of each other. And yet, both of these depend on another third attribute.
- The multivalued dependency contains at least two of the attributes dependent on the third attribute. This is the reason why it always consists of at least three of the attributes.

Conditions for Multivalued Dependency in DBMS

- An MVD would mean that for some single value of the attribute 'x', multiple values of attribute 'y' can exist. Thus, we will write it as follows:

  x –> –> y
- So, it is actually read as: x is multivalued dependent only.

Example: Suppose that there is a car manufacturing company that produces two of the colors in the market, i.e., red and black of each of their models.



Car-model Name	Manufacture year	Color
Maruti Suzuki Swift	2007	Black
Maruti Suzuki Swift	2007	Red
Audi A8	2008	Black
Audi A8	2008	Red

In this case, the columns COLOR and MANUFACTURE YEAR are dependent on CAR_MODEL NAME, and they are independent of each other. Thus, we can call both of these columns multivalued. These are, as a result, dependent on CAR_MODEL NAME. Here is a representation of the dependencies we discussed above:

CAR_MODEL NAME →→ MANUFACTURE YEAR

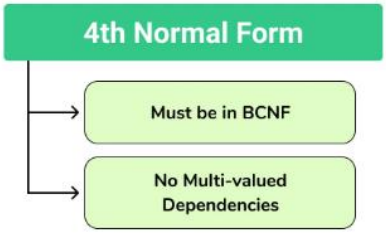
CAR_MODEL NAME →→ COLOR

We can read this as "CAR_MODEL NAME multidetermined MANUFACTURE YEAR" and "CAR_MODEL NAME multidetermined COLOUR".

We always use multi-valued conditions when we encounter these two different ways:

- When we want to test the relations or decide if these happen to be lawful under some arrangement of practical as well as multi-valued dependencies.
- When we want to determine what limitations are there on the arrangement of the lawful relations. Thus, we will concern ourselves with just the relations that fulfil a given arrangement of practical as well as multi-valued dependencies.

FOURTH NORMAL FORM:



- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Examples of 4NF:

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

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data. So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_HOBBY

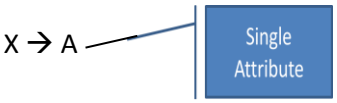
STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

STUDENT_COURSE

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

Decomposition into BCNF

- Let R be the relation schema and F be the set of FDs that hold on R.
- Assume that R is not in BCNF



  - Replace R with (R-A), XA
  - Apply this algorithm recursively.

Example: Let R(A,B,C,D,E) be a relational schema with set of FD as
F : { A -> BC , CD -> E, B->D, E -> A} decompose into bcnf
Sol:

A -> BC ---- A -> B and A -> C

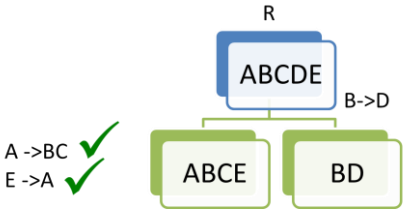
LHS should be a super key

$(A)^+ = \{A,B,C,D,E\}$ A -> BC ✓

$(CD)^+ = \{C,D,E,A,B\}$ CD -> E ✓

$(B)^+ = \{B,D\}$ B->D ✗

$(E)^+ = \{E,A,B,C,D\}$ E -> A ✓



Decomposition into 3NF:

1. Dependent reserving decomposition into 3NF.
2. 3NF Synthesis

Algorithm:

Step 1: Eliminate redundant FDs, resulting in a canonical cover F_c of F.
Step 2: Create a relation $R_i = XY$ for each FD $X \rightarrow Y$ in F_c
Step 3: If the key of R does not occur in any relation R_i, create one more relation $R_i = K$.

Example: Let R(A,B,C,D,E) be a relational schema with FDs as
F = { A → B , BC → D, A → C } convert into 3NF.

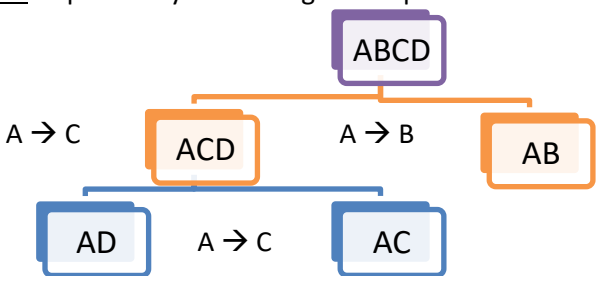
Solution: Candidate key – AE

Checking for 3NF:

A → B (violates 3NF rules)
Therefore R is not in 3NF.

3NF decomposition:

Method 1: Dependency Preserving decomposition into 3NF.



Therefore Loss-less join decomposition of R is R1(A,D,E) R2(A,C) R3(A,B)

$(F_{ADE} \cup F_{AC} \cup F_{AB})^+ = F^+$

A → B , BC → D, A → C

R1(A,D,E)		R2(A,C)		R3(A,B)	
$(A)^+ = \{A,B,C,D\}$	**A -> D**	$(A)^+ = \{A,B,C,D\}$	**A → C**	$(A)^+ = \{A,B,C,D\}$	**A → B**
$(D)^+ = \{D\}$	-	$(C)^+ = \{C\}$	-	$(B)^+ = \{B\}$	-
$(E)^+ = \{E\}$	-				
$(AD)^+ = \{A,D,B,C\}$	-				
$(AE)^+ = \{A,E,B,C,D\}$	**AE -> D**				

$(F_{ADE} \cup F_{AC} \cup F_{AB})^+ = \{A\text{->}D, AE \text{ -> } D, A \text{ -> } C, A\text{-> } B\}$

The functional dependencies of Relational schema are F =

A → B ✓

BC → D $(BC)^+ = \{B,C\}$ ✗

A → C ✓

$(F_{ADE} \cup F_{AC} \cup F_{AB})^+$ not equal to F^+

Here decomposition of R into R1,R2 and R3 is not dependency preserving and BC -> D not enforcing, so we add another table R4(B,C,D)

The decomposed relations are

R1(A,D,E)
R2(A,C)
R3(A,B)
R4(B,C,D)

} Loss-less and dependency preserving

Method 2: 3NF Synthesis

F = { A → B , BC → D, A → C }

The decomposed relations can be R1(A,B) R2(B,C,D) R3 (A,C)

R1(A,B)		R2(B,C,D)		R3(A,C)	
$(A)+ = \{A,B,C,D\}$	**A → B**	$(B)^+ = \{B\}$		$(A)^+ = \{A,B,C,D\}$	**A → C**
$(B)+ = \{B\}$	-	$(C)^+ = \{C\}$		$(C)^+ = \{C\}$	
		$(D)^+ = \{D\}$			
		$(BC)^+ = \{B,C,D\}$	**BC->D**		
		$(CD)^+ = \{C,D\}$			

$(F_{AB} \cup F_{BCD} \cup F_{AC})^+ = \{A\text{->}B, BC \text{ -> } D, A \text{ -> } C\}$

The functional dependencies of Relational schema are F =

A → B ✓

BC → D

A → C ✓

$(F_{ADE} \cup F_{AC} \cup F_{AB})^+ = F^+$

Here decomposition of R into R1,R2 and R3 is dependency preserving.

Now determine loss-less or not

	A	B	C	D	E
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}
R_2	b_{21}	a_2	a_3	a_4	b_{25}
R_3	a_1	b_{32}	a_3	b_{34}	b_{35}

A -> B

	A	B	C	D	E
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}
R_2	b_{21}	a_2	a_3	a_4	b_{25}
R_3	a_1	a_2	a_3	b_{34}	b_{35}

A -> C

	A	B	C	D	E
R_1	a_1	a_2	a_3	b_{14}	b_{15}
R_2	b_{21}	a_2	a_3	a_4	b_{25}
R_3	a_1	a_2	a_3	b_{34}	b_{35}

BC -> D

	A	B	C	D	E
R_1	a_1	a_2	a_3	a_4	b_{15}
R_2	b_{21}	a_2	a_3	a_4	b_{25}
R_3	a_1	a_2	a_3	a_4	b_{35}

No single row contains all 'a' values. So the decomposition is not loss-less. In order to make it loss-less, add any key of original table R_4(A,E).

The decomposed relations are

R1(A,B)
R2(B,C,D)
R3(A,C)
R4(A,E)

} Loss-less and dependency preserving

QUESTION BANK

1. What is Normalization ? Explain 1NF with Example.
2. Explain different anomalies handled using Normalization with suitable examples.
3. Consider the Table:
   EMP(EMP_ID,ENAME,SALARY,AADHAR,MOBILE,HOBBIES)
   Normalize the table to 2NF
4. What is functional Dependency? Explain its properties along with a example.
5. Why is normalization needed? Explain the process of normalization.
6. What is partial dependency? Explain how to use normalization to avoid it.
7. Consider schema R = (A, B, C, G, H, I) and the set F of functional dependencies {A→ B, A → C, CG → H, CG → I, B → H}. Compute the candidate keys of the schema. Check whether the schema is in 2NF or not
8. Consider the table : Table_Name : STUDENT

Stud_ID	Stud_Name	Aadhar	Section	Hobbies
1	Akshar	499977268949	A	Reading Books, Swimming
2	Nayan	315165254889	B	Music, Fiction Stories
3	Ritu	133126294520	B	Swimming

4	Kumar	488877295645	C	Yoga

Perform Normalization on the above table so that it can be in 1NF

9. Consider the table : Table_Name : PRODUCT

Product_Id	Product_Name	Price	Stock_Available
101	Apple	65000,70000	3
102	Mac	50100	3
104	Dell	45000,24000	3
103	HP	32000	3

Perform normalization on the above table and bring it into 2NF.

10. What is meant by the closure of functional dependencies? Illustrate with an example.
11. Explain briefly about 3NF, 4NF and BCNF with suitable examples?
12. Explain about Boyce Codd normal form with an example.
13. What is Functional Dependency? Explain types and properties of FD's.
14. Given a Relation R=(X,Y,Z) and Functional Dependencies are
    F={ {X,Y}→{Z}, {Z}→{X} }
    Determine all Candidate keys of R and the normal form of R with proper explanation.
15. How to compute closure of set of functional dependency? Explain with a suitable example schema.
16. What is multi valued dependency? State and explain fourth normal form based on this concept.
17. Explain the advantages of decomposition? Discuss the problems faced in decomposition.
18. Explain the role of functional dependencies in normalization with suitable examples.
19. State BCNF. How does it differ from 3NF?
20. Define functional dependency? How can you compute the minimal cover for a set of functional dependencies? Explain it with an example.
21. Elaborate the importance of computing closure of functional dependencies. Explain the procedure with an example.
22. What is lossless join decomposition? Explain the same with an example.
23. Define normalization. Explain the conditions that are required for a relation to be in 2NF, 3NF and BCNF with suitable examples.
24. What is lossless join decomposition? Explain the same with an example.
25. Explain the problems related to decomposition.
26. Explain BCNF and the properties of decompositions.
27. Explain FOURTH and THIRD normal forms with examples.
28. Elaborate the importance of computing closure of functional dependencies. Explain the procedure with an example.
29. Given Relation, R=(A,B,C,D,E,F,G) and Functional Dependencies
    F={ {A,B}→{C}, { A,C}→{B}, {A,D}→{E}, {B}→{D}, { B,C}→{A},
    {E}→{F}}.
    Check whether the following decomposition of R into
    R1=(A,B,C), R2=(A,C,D,E) and R3=(A,D,F) is satisfying the lossless Decomposition property.
30. What is dependency preservation property for decomposition? Explain why it is important.
31. Give relation schemas for the following normal forms
    i)2NF but not in 3NF ii) 3NF but not in BCNF