MODULE 4
NORMALIZATION

Relational database design
,
- The grouping of attributes to form good relation schemas
- We have two levels of relation schemas :
    - Logical "user view" level (conceptual schema)
    - Storage "base relation" level (physical schema)
- So the relational database design is concerned mainly with the base relations

Criteria for good base relations

- In database design**, two types of methodology** or we can design database in two ways :
    - Bottom up design (Design by synthesis)
    - Top down design (Design by analysis)
- Design by synthesis :
    - This approach start with individual data items and progressively builds up to larger structures like relations
    - Typically small data elements (attributes) are grouped into functional units (relations) based on their dependencies
    - This process often involves normalizations
- Design by analysis :
    - This approach begins with high level conceptual model (ER diagrams) and gradually breaks it down into detailed database tables

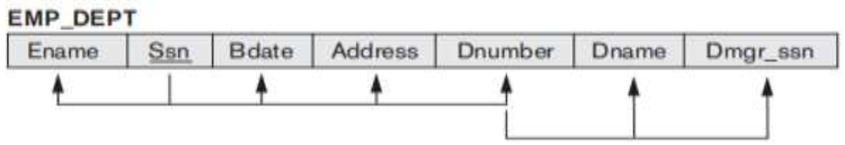
- In simple, relational database design ultimately produces a set of relations and which will preserve,
    - Information Preservation
    - Minimum Redundancy

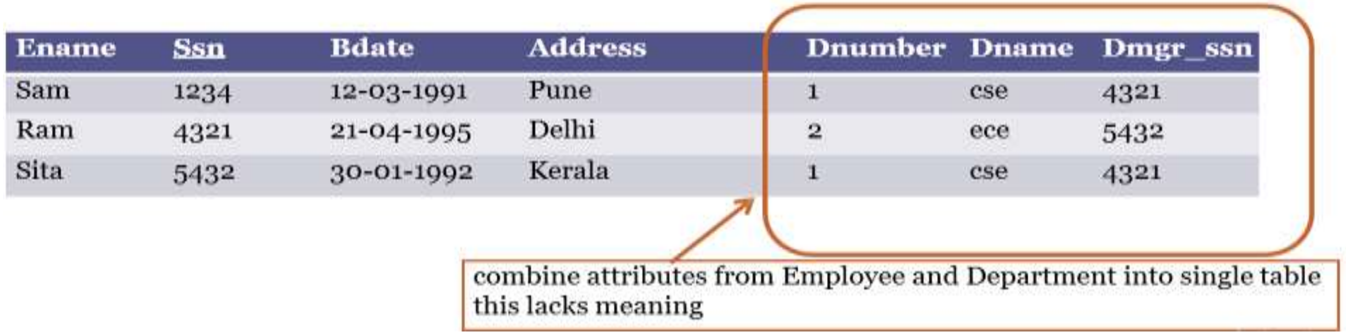
Informal Design Guidelines for Relation Schema

- Four informal guidelines that may be used as measures to determine the quality of relation schema design :
    - Making sure that the semantics of the attributes is clear in the schema
    - Reducing the redundant information in tuples
    - Reducing the NULL values in tuples
    - Disallowing the possibility of generating spurious tuples

1. Clear Semantics of the relation attributes

- Whenever we group attributes to form a relation schema , we assume that attributes belonging to one relation have certain real world meaning and a proper interpretation associated with them
- Design a relation schema so that it is easy to explain its meaning.
- Do not combine attributes from multiple entity types and relationship types into a single relation.
- Otherwise, if the relation corresponds to a mixture of multiple entities and relationships, semantic ambiguities will result and the relation cannot be easily explained.
- **In simple , attributes of different entities should not be mixed in the same relation**
- **Only foreign keys should be used to refer to other entities**
- **Entity and relationship attributes should be kept apart as much as possible**
- Eg :





Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Sam	1234	12-03-1991	Pune	1	cse	4321
Ram	4321	21-04-1995	Delhi	2	ece	5432
Sita	5432	30-01-1992	Kerala	1	cse	4321

combine attributes from Employee and Department into single table this lacks meaning

2. Redundant Information in Tuples and Update Anomalies

• Data redundancy is a condition created within a database or in which the same piece of data is held in different places.

• **Redundancy leads to Wastes storage Causes problems with update anomalies** such as:

- Insertion anomalies
- Deletion anomalies
- Modification anomalies

A) Insertion Anomalies

• Consider the relation:

EMP_PROJ (Empno, Projno, Ename, Pname, No_hours)

Insert Anomaly: Cannot insert a project unless an employee is assigned to it.

ie, Inserting a new employee who has not been assigned to a project becomes impossible

B) Deletion Anomalies

• Consider the relation:

EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)

• Delete Anomaly: When a project is deleted, it will result in deleting all the employees who work on that project. Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.

• ie, delete a record unintentionally removes useful information due to poor database design

C) Modification Anomalies

• This will arise when multiple types of data are stored in a single table

• For eg :

EMP_DEPT, if we change the value of one of the attributes of a particular department say, the manager of department 5 we must update the tuples of all employees who work in that department; otherwise, the database will become inconsistent.

• If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would be wrong

3. NULL Values in Tuples

- Reasons for nulls:
  - Attribute not applicable or invalid
  - Attribute value unknown (may exist)
  - Value known to exist, but unavailable

- NULL can waste space at the storage level and may also lead to problems with understanding the meaning of the attributes and with specifying JOIN operations at the logical level

- Another problem with NULLs is how to account for them when aggregate operations such as COUNT or SUM are applied.
  - if NULL values are present, the results may become unpredictable

4. Generation of Spurious Tuples – avoid at any cost

- Consider the tables

  EMP_LOCS (EName,PLocation)

  EMP_PROJ1 (SSN,PNumber,Hours, PName, PLocation)

- versus the table(result of natural join)

  EMP_PROJ(SSN,PNumber,Hours, EName, PName,PLocation)

- If we use the former as our base tables then we cannot recover all the information of the latter because trying to natural join the two tables will produce many rows not in EMP_PROJ.
- These extra rows are called **spurious tuples**.
- Another design guideline is that relation schemas should be designed so that they can be joined with equality conditions on attributes that are either primary keys or foreign keys in a way such that no spurious tuples are generated.

Summary of Design Guidelines

• Anomalies that cause redundant work to be done during insertion into and modification of a relation, and that may cause accidental loss of information during a deletion from a relation

• Waste of storage space due to NULLs and the difficulty of performing selections, aggregation operations, and joins due to NULL values

• Generation of invalid and spurious data during joins on base relations with matched attributes that may not represent a proper (foreign key, primary key) relationship

Functional dependencies

- Functional Dependencies :
    - Are used to specify formal measures of the "goodness" of relational designs
    - And keys are used to define normal forms for relations
    - Are constraints that are derived from the meaning and interrelationships of the data attributes
- A functional dependency is a constraint between two sets of attributes from the database.
- Suppose that our relational database schema has n attributes (A1, A2, ..., An)

A set of attributes X functionally determines a set of attributes Y ,

if the value of X determines a unique value for Y

Definition:

A functional dependency , denoted by X → Y, between two sets of attributes X and Y

that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R.

The constraint is that, for any two tuples t1 and t2 in r that have

t1[X] = t2[X], they must also have t1[Y]= t2[Y].

- $X \rightarrow Y$ holds if whenever two tuples have the same value for X, they must have the same value for Y

  X is determinant

  Y is dependent

- For any two tuples t1 and t2 in any relation instance r (R):

  If t1[X] = t2[X], then t1 [Y] = t2 [Y]

- $X \rightarrow Y$ in R specifies a constraint on all relation instances r(R)
- FDs are derived from the real-world constraints on the attributes

- **Examples** of functional dependencies

  - Social security number determines employee name

    SSN→ENAME

  - Project number determines project name and location

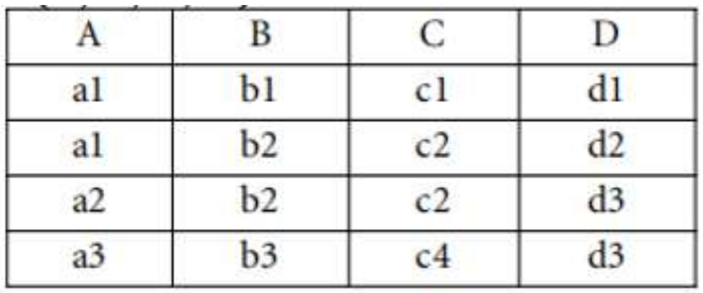
    PNUMBER →{PNAME, PLOCATION}

  - Employee ssn and project number determines the hours per week that the employee works on the project

    {SSN, PNUMBER}→HOURS

Eg :1

A relation schema R (A,B,C,D) with its extension

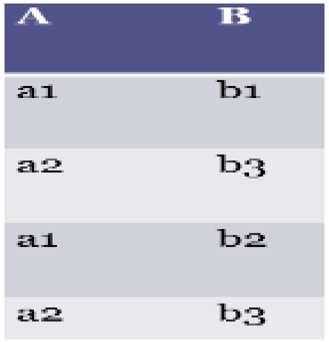


A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

- following FDs may hold because the four tuples in the current extension have **no violation** of these constraints
  - $B \rightarrow C$;
  - $C \rightarrow B$;
  - $\{A, B\} \rightarrow C$;
  - $\{A, B\} \rightarrow D$; and
  - $\{C, D\} \rightarrow B$.

- However, the following **do not hold** because we already have violations of them in the given extension:
  - $A \rightarrow B$ (tuples 1 and 2 violate this constraint);
  - $B \rightarrow A$ (tuples 2 and 3 violate this constraint);
  - $D \rightarrow C$ (tuples 3 and 4 violate it).

Eg 2:



A	B
a1	b1
a2	b3
a1	b2
a2	b3

- A → B So this is not a valid FD no unique matching
    - a1 (b1,b2) // not dependent
    - a2 b3 // functionally dependent
- B →A
    - b1 a1 // functionally dependent
    - b3 a2 // functionally dependent
    - b2 a1 // functionally dependent

So this is a valid FD

B $\rightarrow$C implies

- B functionally determines C
- C functionally depends on B
- C is functionally determined by B

• A functional dependency is a property of the semantics or meaning of the attributes.

• Functional Dependancy must be valid for every relation state.

• Whenever the semantics of two sets of attributes in R indicate that a functional dependency should hold, we specify the dependency as a constraint.

• Relation extensions r (R) that satisfy the functional dependency constraints are called legal relation states (or legal extensions) of R.

• Hence, the main use of functional dependencies is to describe further a relation schema R by specifying constraints on its attributes that must hold at all times

• A functional dependency is a property of the relation schema R, not of a particular legal relation state r of R.

• Therefore, an FD 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.

Types of Functional Dependancy

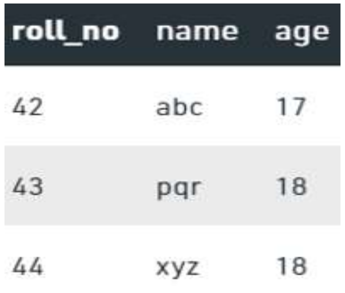
1. Trivial FD

- In Trivial Functional Dependency, **a dependent is always a subset of the determinant**.
- It is FD of the form X, Y→X
- Not a useful FD since we are not getting any important information here
- **In simple, dependency X→Y is trivial if Y is a subset of X**
- Eg: Eid, Ename→Ename // trivial

2. Non Trivial FD

- In Non-trivial functional dependency, **the dependent is strictly not a subset of the determinant**.
- i.e. **If X → Y and Y is not a subset of X**, then it is called Non-trivial functional dependency.

Eg:



roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

• roll_no → name is a non-trivial functional dependency, sincethe dependent name is not a subset of determinant roll_no

• Similarly,{roll_no, name} → age is also a non-trivial functional dependency, since age is not a subset of {roll_no,name}

3. Multivalued functional dependency

- A multivalued dependency X→ Y exists when,

  for a single X , multiple values of Y exist independently of other attributes
- **In simple, Attribute in dependent set are not dependent on each other**
- For eg : X → { Y, Z}

  There is a dependency between X and Y & X and Z

  But there is no dependency between Y and Z

4. Transitive functional dependency

- It occurs when **X→ Y and Y → Z, then X →Z**
- **ie, X is indirectly dependent on Z**
- Transitive dependency cause redundancy and should removed in 3NF

Properties of Functional Dependencies

- There are several useful rules that let you replace one set of functional dependencies with an equivalent set.
- Some of those rules are as follows:

1. **Reflexivity : If $Y \subseteq X$, then $X \rightarrow Y$**
2. **Augmentation : If $X \rightarrow Y$, then $XZ \rightarrow Y Z$**
3. **Transitivity : If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$**
4. **Union : If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow Y Z$**
5. **Decomposition : If $X \rightarrow Y Z$, then $X \rightarrow Y$ and $X \rightarrow Z$**

Implicit and Explicit FDs

- Given a set of explicit functional dependencies , we can determine the implicit ones
    - Explicit FDs : ID $\rightarrow$ level, level $\rightarrow$ salary
    - Implicit FD : ID $\rightarrow$ salary

- Implicit FDs are also called **inferred FDs**
- The notation $F \models X \rightarrow Y$ denotes that FD $X \rightarrow Y$ can be inferred from the set of functional dependencies F
    - X is called the left hand side (LHS)
    - Y is called the right hand side (RHS)
    - Y can be derived from X under F
    - X implies Y under F

Closure set of attribute

- Attribute closure of an attribute set can be **defined as set of attributes which can be functionally determined from it**.
- Closure is the set of all functional dependencies that can be impleed by F
- Closure is **denoted as** F^+
- F^+ includes both trivial and non trivial functional dependencies
- To **find attribute closure** of an attribute set:
  - Add elements of attribute set to the result set.
  - Recursively add elements to the result set which can be functionally determined from the elements of the result set
- Eg :

  R(A,B,C,D) with FD ={A→B, B→C,C→D,D→A}
- Closure of A, A+ = attribute which can be determined from A

  A+ = { A B C D } (ie using closure if we can cover all attribute then it is called Candidate Key CK)

  B+ = { B C D A }

  C+ = {C D A B }

  D+ = { D A B C }
- Candidate Keys of R are A,B,C,D

Eg 2 :

R (V W X Y Z) with FD's are { V Y → W , W X → Z , Z Y → V }

XY+ = { X Y }□

VXY+ = { V X Y W Z } // closure of VXY get same as the relation R , so it is a super key

To find CK , calculate the closure of each attributes in the closure set

ie, V + = { V }

X + = { X }

Y + = { Y }

VX + = { V X }

VY + = { V Y W }

XY + = { X Y }

Here, closure of each attribute set is not same as the original relation , so we can say that they are candidate keys

Therefore, VXY is a super key and candidate key

Similirly , WXY+ = { W X Y Z V } .

WXY is a super key and candidate key

F + can be computed via Armstrong Axioms inference rules, but it is difficult and time consuming to do

Armstrong’s Axioms in Functional Dependency

- The term **Armstrong axioms refer** to the sound and **complete set of inference rules** or axioms
- Armstrong, that is used to test the logical implication of functional dependencies.
- If F is a set of functional dependencies then the closure of F, denoted as F+, is the set of all functional dependencies logically implied by F.
- **Armstrong’s Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.**

- **Basic Armstrong Axioms**

1. **Axiom of reflexivity**
    - If A is a set of attributes and B is subset of A, then A holds B.
    - **If $B \subseteq A$ then $A \rightarrow B$**
    - This property is trivial property.

2. **Axiom of augmentation**
    - **If $A \rightarrow B$ holds and Y is attribute set, then $AY \rightarrow BY$ also holds.**
    - That is adding attributes in dependencies, does not change the basic dependencies.
    - If $A \rightarrow B$, then $AC \rightarrow BC$ for any C.

3. Axiom of transitivity

- Same as the transitive rule in algebra
- **If A→B holds and B→C holds, then A→C also holds**.
- A→B is called as A functionally that determines B.
- If X→Y and Y→Z , then X→Z

- **Secondary Rules**

1. Union

- **If A→B holds and A→C holds, then A→BC holds**.
- If X→Y and X→Z then X→YZ

2. Composition

- **If A→B and X→Y holds, then AX→BY hol**ds.

3. Decomposition

- **If A→BC holds then A→B and A→C hol**ds**.**

4. Pseudo Transitivity

- **If A→B holds and BC→D holds, then AC→D holds**.
- If X→Y and YZ→W then XZ→W.

Equivalence of Sets of Functional Dependencies

- **Definition:** A set of functional dependencies **F is said to cover** another set of functional dependencies E if every FD in E is also in F+; that is, **if every dependency in E can be inferred from F**
- Alternatively, we can say that E is covered by F.

- **Definition :** Two sets of functional dependencies **E and F are equivalent if E+ = F+.**

  Therefore, equivalence means that every FD in E can be inferred from F,

  and every FD in F can be inferred from E

  ie, **E is equivalent to F if both the conditions holds**

    - **E covers F and**
    - **F covers E**

- We can determine whether F covers E by calculating X+ with respect to F for each FD
- $X \rightarrow Y$ in E, and then checking whether this X+ includes the attributes in Y.
- If this is the case for every FD in E, then F covers E.
- We determine whether E and F are equivalent by checking that E covers F and F covers E.

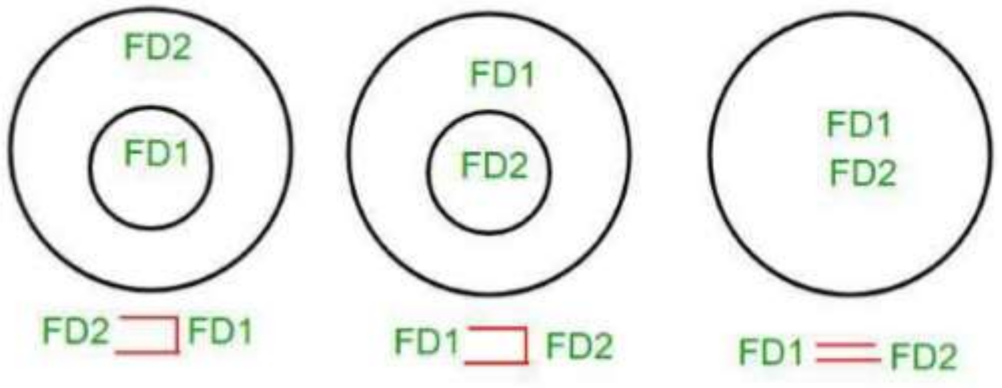
How to find relationship between two FD sets?

- Let FD1 and FD2 are two FD sets for a relation R.

  1. If all FDs of FD1 can be derived from FDs present in FD2,we can say that FD2 ⊃ FD1**.**

  2. If all FDs of FD2can be derived from FDs present in FD1, we can say that FD1 ⊃ FD2.

  3. If 1 and 2 both are true, FD1=FD2.



Q) A relation R (A,B,C,D) having two FD sets FD1 = {A -> B, B -> C, AB -> D} and FD2 = {A -> B, B- > C, A -> C, A -> D}

Solun :

- Step 1 : **Checking whether all FDs of FD2 are present inFD1**
  - A->B in set FD2 is present in set FD1.
  - B->C in set FD2 is also present in set FD1.
  - A->C is present in FD2 but not directly in FD1 but we will check whether we can derive it or not.
  - For set FD1,

    (A)+ ={A,B,C,D}.
  - It means that A can functionally determine A, B, C and D.
  - So A -> C will also hold in set FD1.
  - A->D is present in FD2 but not directly in FD1 but we will check whether we can derive it or not.
  - For set FD1,

    (A)+ = {A,B,C,D}.
  - It means that A can functionally determine A, B, C and D.
  - So A->D will also hold in set FD1.
  - As all FDs in set FD2 also hold in set FD1, FD1 ⊃ FD2 is true.

    FD1 = {A->B, B->C, AB->D} and

    FD2 = {A->B, B->C, A->C, A->D}

- Step 2 : **Checking whether all FDs of FD1 are present in FD2**
    - A->B in set FD1 is present in set FD2.
    - B->C in set FD1 is also present in set FD2.
    - AB->D in FD1 is present but not directly in FD2 but we will check whether we can derive it or not.
    - For set FD2,

      (AB)+ = {A,B,C,D}.
    - It means that AB can functionally determine A, B, C and D.
    - So AB->D will also hold in set FD2.
    - As all FDs in set FD1 also hold in set FD2, FD2 ⊃ FD1 is true.

Step 3 :

- As FD1 ⊃ FD2 and FD2 ⊃ FD1 both are true FD2 =FD1 is true.
- These two FD sets are semantically equivalent.

Minimal Sets of Functional Dependencies

- A minimal cover of a set of FDs F is a minimal set of functional dependencies Fmin that is equivalent to F.
- We can think of a **minimal set of dependencies as being a set of dependencies in a standard or canonical form and with no redundancies.**
- A canonical cover is a simplified and reduced version of the given set of functional dependencies.
- Since it is a reduced version, **it is also called as Irreducible set.**
- Canonical cover is free from all the extraneous functional dependencies
- To satisfy these properties, we can **formally define a set of functional dependencies F to be minimal if it satisfies the following conditions:**

1. Every dependency in F has a **single attribute for its right-hand side**.
2. We **cannot replace** any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$,

   where Y is a proper subset of X, and still have a set of dependencies that is equivalent to F.
3. We **cannot remove** any dependency from F and still have a set of dependencies that is equivalent to F.

Definition :

A minimal cover of a set of functional dependencies E is a minimal set of dependencies (in the standard canonical form and without redundancy) that is equivalent to E. We can always find at least one minimal cover F for any set of dependencies E

Example:

Find the minimal cover of set of FDs be E : $\{B \rightarrow A, D \rightarrow A, AB \rightarrow D\}$

Solution :

- Step 1 : All above dependencies are in canonical form that is, they have only one attribute on the right-hand side
- Step 2 we need to determine if $AB \rightarrow D$ has any redundant attribute on the left-hand side

that is, can it be replaced by $B \rightarrow D$ or $A \rightarrow D$

$\{B \rightarrow A, D \rightarrow A, AB \rightarrow D\}$

Since $B \rightarrow A$, by augmenting with B on both sides (IR2), we have

$BB \rightarrow AB$, or $B \rightarrow AB$

(i). However, $AB \rightarrow D$ as given

(ii). Hence by the transitive rule (IR3), we get from (i) and (ii), $B \rightarrow D$. Thus $AB \rightarrow D$ may be replaced by $B \rightarrow D$.

We now have a set equivalent to original E, say

E’: $\{B \rightarrow A, D \rightarrow A, B \rightarrow D\}$.

No further reduction is possible in step 2 since all FDs have a single attribute on the left-hand side.

- Step 3 : we look for a redundant FD in E’.

By using the transitive rule on $B \rightarrow D$ and $D \rightarrow A$, we derive $B \rightarrow A$.

Hence $B \rightarrow A$ is redundant in E’ and can be eliminated.

Therefore, the minimal cover of E is $\{B \rightarrow D, D \rightarrow A\}$.

“Key” Concepts

<u>Definitions of Keys and Attributes Participating in Keys</u>

1. Super Key

• A super key of a relation schema R = {A1, A2, ... , An} is a set of attributes S ⊆ R with the property that no two tuples t1 and t 2 in any legal relation state r of R will have t 1[S] = t 2[S].

• ie , **A Superkey is a set of attributes such that no two tuples have the same values for these attributes**

• A key K is a super key with the additional property that removal of any attribute from K will cause K not to be a super key any more

• **The difference between a key and a super key is that a key has to be minimal**; that is, if we have a key

K = {A1, A2, ..., Ak} of R, then K – {Ai } is not a key of R for any Ai , 1 ≤ i ≤ k.

• For eg,

{Ssn} is a key for EMPLOYEE, whereas {Ssn}, {Ssn, Ename}, {Ssn, Ename, Bdate}, and any set of attributes that includes Ssn are all superkeys

2. Candidate key

• 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

• ie , Candidate key is a minimal superkey where none of the attributes can be removed to create another superkey

There can be many candidate keys for one relation and any superkey must contain atleast one candidate key

3. Primary Key

• They are One of the selected candidate keys

• There can be only one primary key for a relation

4. **Prime and Non Prime Attributes**

• An attribute of relation schema R is called a **prime attribute** of R if it is a member of some candidate key of R.

In simple , **Prime attribute is an attribute in any candidate key**

• 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 simple , **Non prime attribute is an attribute that is not a member of any candidate key**

Full and Partial Functional Dependencies

- A functional dependency X → Y is a **full functional dependency** if removal of any attribute A from X means that the dependency doesnot hold anymore
  - ie , A € X,

    (X – {A}) doesnot functionally determine Y

- A functional dependency X → Y is a **partial dependency** if some attribute A can be removed from X and the dependency still holds
  - ie, A € X,

    (X – {A}) → Y // functionally determine Y

Example1 :

Consider a relation Student (Courseid, Studid, Grade)

Let’s assume :

- A student can enroll in multiple courses
- A grade is awarded based on both Courseid, and Studid

So,

- { Courseid,Stdid } $\rightarrow$ Grade is a full functional dependency because :
    - Grade depends on the combination of both Courseid, and Stdid
    - Grade cannot be determined by just Courseid or just Studid
- If we remove any part of the key, the dependency breaks. Hence, it is a full dependency

Example 2 :

Consider a relation Student (Courseid, Studid, Studname, Grade)

where (Courseid,Studid) is the composite primary key

Studid $\rightarrow$ Studname is a partial dependency , because

- Studname depends only on Studid, which is the part of primary key

NORMALIZATION OF RELATIONS

• **Normalization**: The process of **decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations**

• It is the process of minimizing redundancies from a relation or set of relations.

• **Normal form:** Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form

• **Normal forms are used to eliminate or reduce redundancy in database tables**

• **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

• **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**

- The process of normalization through decomposition should satisfy the following two properties :
    - The **non additive join or lossless join property**, which guarantees that the spurious tuple generation problem does not occur with respect to the relation schemas created after decomposition.
    - The **dependency preservation property**, which ensures that each functional dependency is represented in some individual relation resulting after decomposition

- **Denormalization** is the process of storing the join of higher normal form relations as a base relation

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
- The practical utility of these normal forms becomes questionable when the constraints on which they are based are hard to understand or to detect
- The database designers need not normalize to the highest possible normal form
    - ➢ usually up to 3NF and BCNF. 4NF rarely used in practice.

NORMALIZATION

- Normalization can be defined as:

A process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly and deletion anomaly

- Data modification anomalies can be categorized into three types:
    - **Insertion Anomaly**: Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
    - **Deletion Anomaly**: The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
    - **Updation Anomaly**: The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Levels of Normalization

- Levels of normalization based on the amount of redundancy in the database.
- Various levels of normalization are:

First Normal Form (1NF)

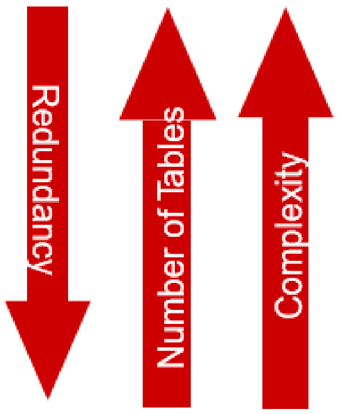
Second Normal Form (2NF)

Third Normal Form (3NF)

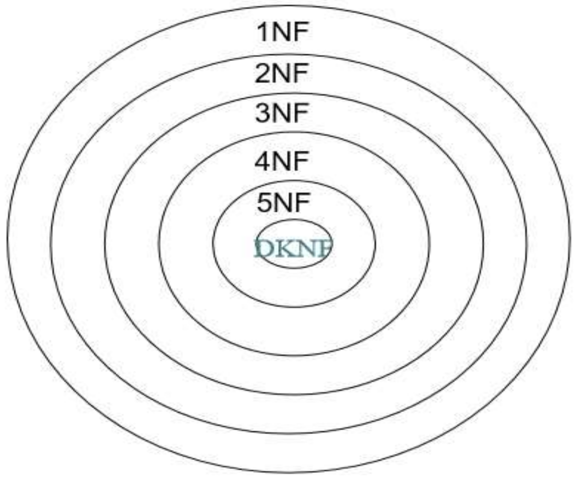
Boyce Codd Normal Form (BCNF)

Fourth Normal Form (4NF) etc

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form (BCNF)
- Fourth Normal Form (4NF)
- Fifth Normal Form (5NF)
- Domain Key Normal Form (DKNF)



Most databases should be 3NF or BCNF in order to avoid the database anomalies.



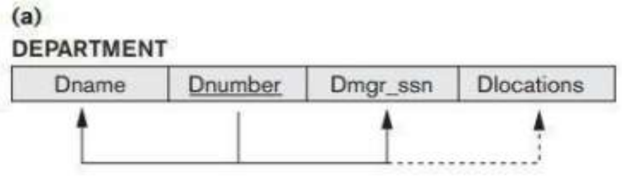
Each higher level is a subset of the lower level

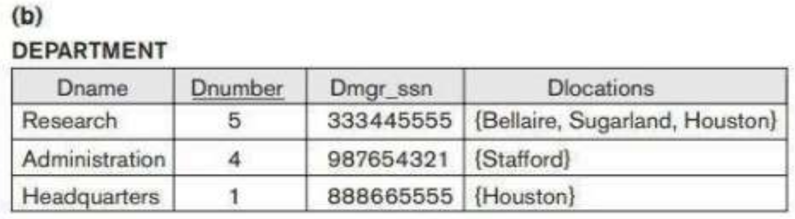
Normal Form	Description
1NF	A relation is in 1NF if it contains atomic values.
2NF	• A relation will be in 2NF if it is in 1NF • And all non-key attributes are fully functional dependent on the primary key
3NF	• A relation will be in 3NF if it is in 2NF • And no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form
4NF	• A relation will be in 4NF if it is in Boyce Codd's normal form • And has no multi-valued dependency.
5 NF (or called as Projection- Join Normal Form (PJNF))	• Must be in 4NF • It cannot be further divided into smaller tables without losing data

FIRST NORMAL FORM

- Disallow multi valued 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.
- Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple.
- In other words, 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

Example :




(b)

DEPARTMENT

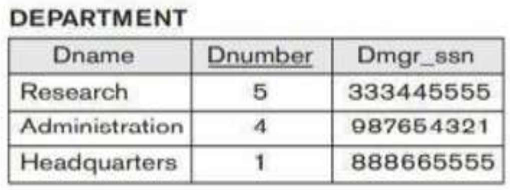
Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

this is not in 1NF because Dlocations is not an atomic attribute

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

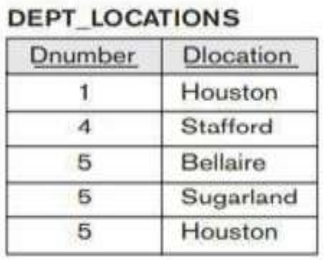
1. **Remove the attribute** Dlocations **that violates 1NF and place it in a separate relation** DEPT_LOCATIONS along with the primary key Dnumber of DEPARTMENT.

The primary key of this relation is the combination {Dnumber, Dlocation},



DEPARTMENT

Dname	Dnumber	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

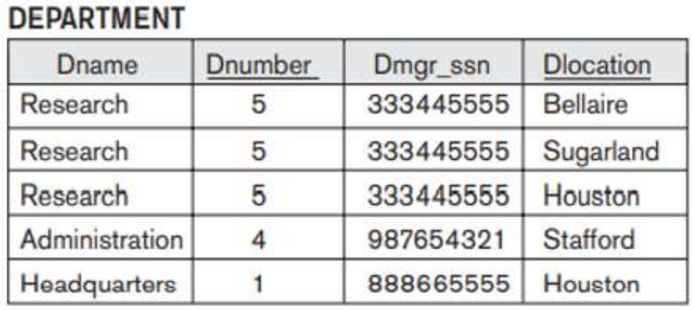


DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

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

DName	**<u>Dnumber</u>**	**Dmr_ssn**	**Dlocation1**	**Dlocation2**	**Dlocation3**

• Of the three solutions above, 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 multi valued attributes that are themselves composite. These are called nested relations because each tuple can have a relation within it.

SECOND NORMAL FORM

• Second normal form (2NF) is **based on the concept of full functional dependency**.

• A relation will be in 2NF if **it is in 1NF and all non-key attributes are fully functional dependent on the primary key**.

• A functional dependency X → Y is a **full functional dependency** ,

if removal of any attribute A from X means that the dependency does not hold any more

• **that is, for any attribute A ε X, (X – {A}) does not functionally determine Y.**

• A functional dependency X → Y is a **partial dependency,**

if some attribute A ε X can be removed from X and the dependency still holds

• **that is, for some A ε X, (X – {A}) → Y holds**

• A non prime attribute partially depends on key then it is called Partial Fuctional Dependency

• If there is Partial dependency then R not in 2NF

• 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.**

Definition.

• A relation schema R is in 2NF if every non prime attribute A in R is fully functionally dependent on the primary key of R

•A relation schema R is in second normal form (2NF) if every nonprime attribute A in R is not partially dependent on any key of R

• The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key.

• If the primary key contains a single attribute, the test need not be applied at all.

$$\text{SSn} \rightarrow \text{Ename}$$

• **If a relation schema is not in 2NF, it can be second normalized or 2NF normalized into a number of 2NF relations in which nonprime attributes are associated only with the part of the primary key on which they are fully functionally dependent**

Qn.

Consider Relation Std_Faculty(Sid,Cid, Mark, Faculty).With FD = {Sid,Cid → Marks, Cid → Faculty}. Is this in 2NF?

Ans.

1. Find CK,

    (Sid,Cid)+ = {Sid, Cid, Marks, Faculty }

    So (Sid,Cid) is a candidate key here

    Here Prime attribute= { Sid,Cid }

    Non Prime attribute= { Marks, Faculty }

    Now **check for Parial functional dependency**

    Sid,Cid → Marks // Full functional dependent

    Cid → Faculty □ // Partial dependency

    So not in 2NF

Things to Remember ;

1. **If a non prime attribute depends on a proper subset of any key of R then there is Partial Dependency**

   Proper subset :A proper subset of a set A is a subset of A that is not equal to A.

   In other words, if B is a proper subset of A, then all elements of B are in A but A contains atleast one element that is not in B.

   {B,C}'s proper subset has {B,C} BC not included.

To check whether a relation is in 2NF with some functional dependencies :

1. Subset of Candidate key $\rightarrow$ Non prime attribute , does not hold
2. A Candidate key $\rightarrow$ Non prime attribute, holds
3. Non prime attribute $\rightarrow$ Non prime attribute , holds

THIRD NORMAL FORM

- Third normal form (3NF) is **based on the concept of transitive dependency**.
- Conditions for 3NF
  - **Relation should be in 2NF**
  - **No transitive dependency is allowed**

Transitive Dependency

- A functional dependency X → Y in a relation schema R is a transitive dependency if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of R and both X → Z and Z → Y hold.

- If a non prime attribute is transitively depends on Key through another non prime attribute then there is transitive dependency, such a dependency is not allowed in 3NF

Definition. : According to Codd's original definition, a relation schema R is in 3NF if it satisfies 2NF and no non prime attribute of R is transitively dependent on the primary key

A relation R is in 3NF

if for every FD X → Y

Either X is a SK

or Y is a prime attribute of R

Qn :

Consider Relation Employee_Dept (Eid,Ename,Dno, Dmgrid), With FD={ Eid → Ename, Eid → Dno, Dno → Dmgrid }. Is this in 3NF?

Ans.

1. First check for 2NF

   Here, No partial dependency so is in 2NF

2. Find CK,

   Eid + = { Eid,Ename,Dno,Dmgrid }

   Eid is CK

   So, Prime Attribute = {Eid} and

   Non Prime Attributes = { Ename,Dno, Dmgrid }

3. Now check for transitive dependency,

   Eid → Dno, Dno → Dmgrid // This is transitive dependency.

   So not in 3NF

- **To make this 3NF** we have to decompose relation Employee_Dept into two relations

  Employee(Eid,Ename,Dno) and

  Dept(Dno, Dmgrid)

BOYCE-CODD NORMAL FORM (BCNF)

- Boyce-Codd normal form (BCNF) was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF.
- That is, every relation in BCNF is also in 3NF; however,a relation in 3NF is not necessarily in BCNF.
- It is an advance version of 3NF that's why it is also referred as 3.5NF.
- **BCNF is stricter than 3NF.**
- BCNF is relatively easy to understand and ensures that data redundancy is eliminated
- However, using BCNF all FDs may not be preserved

- **A table complies with BCNF**
    - **if it is in 3NF and**
    - **for every functional dependency X->Y, X should be the super key of the table.**

Example :1

Teach (stdid. cid, lecturer)

FD are { stdid , cid } → lecturer

lecturer → cid

Keys are stdid,cid

Lecturer is not a superkey

Lecturer → cid violates BCNF

Therefore, Teach is not in BCNF

Example :2

For the given relation schema R (A, B, C, D) with the set of FDs F below, determine whether or not R is in BCNF

F = { A, D} → { B,C}

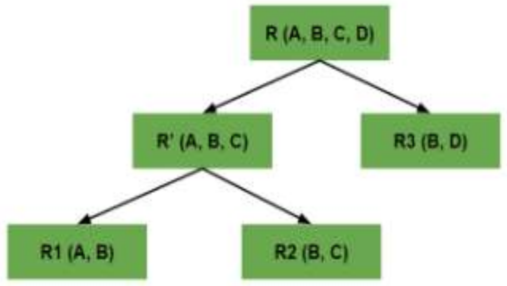
B → A

Here, B → A is a non trivial FD, B is not a Superkey

So it violates BCNF condition

Therefore, R is not in BCNF

DECOMPOSING A RELATION



Decomposition in DBMS

A Decomposition of R replaces R by two or more relations such that :

- Each new relation contains a subset of the attributes of R
- Every attribute of R appears in atleast one new relation

Properties of Decomposition

- **Lossless:** All the information should not be lost while performing the join on the sub-relation to get back the original relation. It helps to remove the redundant data from the database.
- **Dependency Preservation:** It ensures that the functional dependencies between the entities is maintained while performing decomposition. It helps to improve the database efficiency, maintain consistency and integrity.
- **Lack of Data Redundancy:** This property states that the decomposition performed should not suffer redundant data. It will help us to get rid of unwanted data and focus only on the useful data or information.

- Decomposition in DBMS removes redundancy, anomalies and inconsistencies from a database by dividing the table into multiple tables.
- There are mainly **two types of decompositions** in DBMS

1. Lossless Decomposition

2. Lossy Decomposition

LOSSLESS JOIN DECOMPOSITION

• Consider there is a relation R which is decomposed into sub relations R1,R2,..Rn

• This decomposition is called lossless join decomposition **when the join of the sub relations results in the same relation R that was decomposed**

• For lossless join decomposition , we always have

$$R1 \bowtie R2 \bowtie R3 \ldots\ldots \bowtie Rn = R$$

• Lossless join decomposition is also known as **non- additive join decomposition**

• This is because the resultant relation after joining the sub relations is same as the decomposed relation

• No extraneous tuples appear after joining the sub relations

• **In simple , by lossless decomposition , it becomes feasible to reconstruct the relation R from decomposed tables by using joins**

• Only 1NF , 2NF , 3NF and BCNF are valid for lossless join decomposition

• In Lossless Decomposition, we select the common attribute and the criteria for selecting a common attribute is that the common attribute must be a candidate key or super key in either relation R1, R2, or both.

• Decomposition of a relation R into R1 and R2 is a lossless-join decomposition if at least one of the following functional dependencies is in F+ (Closure of functional dependencies)

Advantages of Lossless Decomposition

- **Reduced Data Redundancy:** This helps in improving the efficiency of the database system by reducing storage requirements and improving query performance.
- **Maintenance and Updates:** Easier to maintain and update the database since it allows for more granular control over the data.
- **Improved Data Integrity:** Help to improve data integrity by ensuring that each relation contains only data that is relevant to that relation. This can help to reduce data inconsistencies and errors.
- **Improved Flexibility:** Improve the flexibility of the database system by allowing for easier modification of the schema.

Disadvantages of Lossless Decomposition

- **Increased Complexity:** Making it harder to understand and manage.
- **Increased Processing Overhead:** This can lead to slower query performance and reduced efficiency.
- **Join Operations:** This can also result in slower query performance.
- **Costly:** Decomposing relations can be costly, especially if the database is large and complex.

LOSSY DECOMPOSITION

- Consider there is a relation R which is decomposed into sub relations R1,R2, …..Rn
- 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**
- The natural join of the sub relations is always found to have some extraneous tuples (ie, addition of spurious information)
- These extra tuples genrates difficulty for the user to identify the original tuples.
- For a lossy join decomposition , we always have $R \subset R1 \bowtie R2 \bowtie \ldots\ldots \bowtie Rn$
- Lossy decomposition is also known as **careless decomposition**

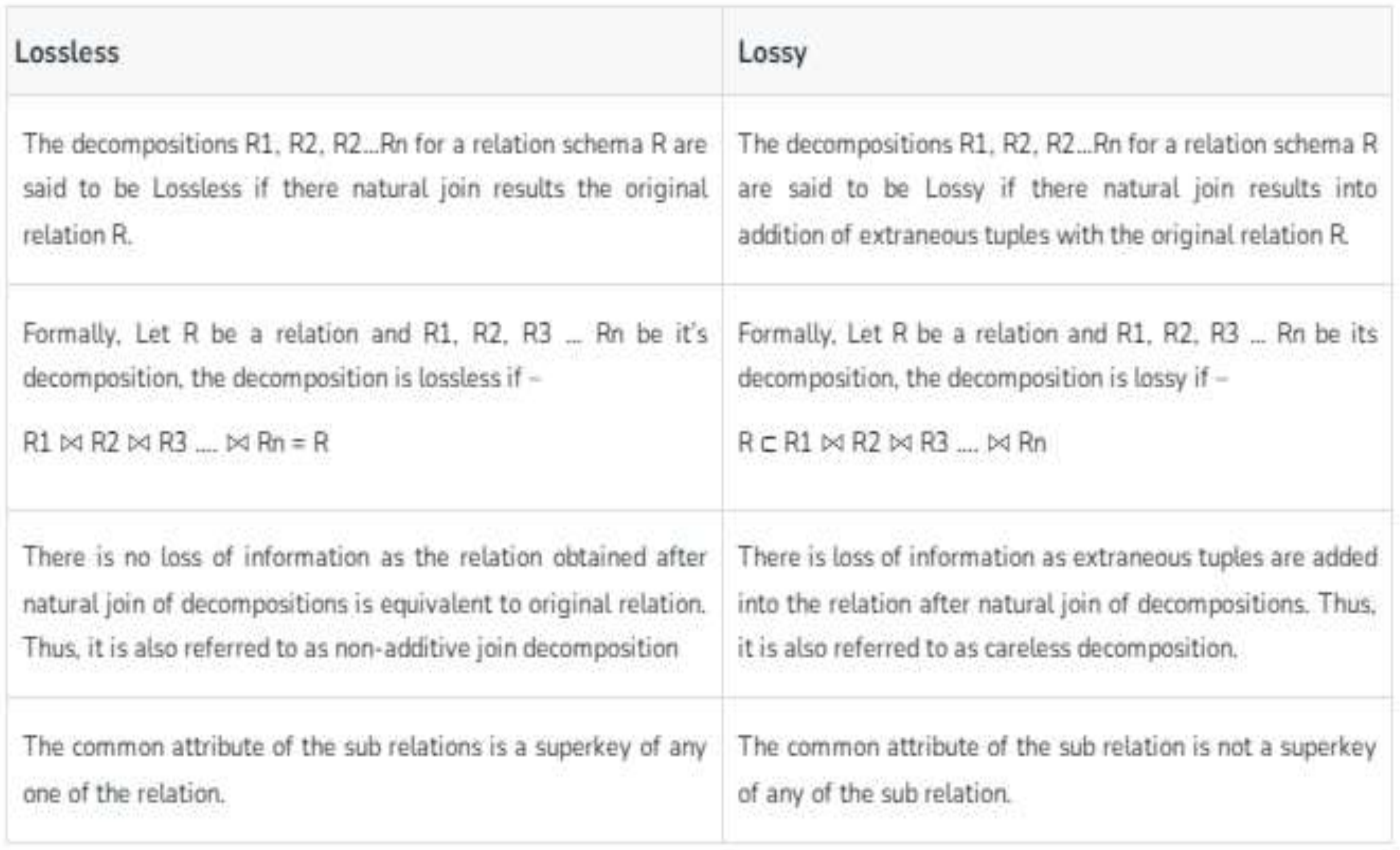
Advantages of Lossy Join Decomposition

- **Structure is Simple:** The result of decomposing the relations will give simple and smaller sub tables and this helps in reducing the complexity in some cases.
- **Redundancy is Less**

Disadvantage of Lossy Join Decomposition

- **Loss of Data:** When tables are joined back together then some of the information will be loosed permanently
- **Inconsistency :** Due to information loss the data integrity problem will arise.
- **Hard to manage:** It can be difficult to maintain data consistency

Difference Between Lossless and Lossy Join Decomposition



Lossless	Lossy
The decompositions R1, R2, R2...Rn for a relation schema R are said to be Lossless if there natural join results the original relation R.	The decompositions R1, R2, R2...Rn for a relation schema R are said to be Lossy if there natural join results into addition of extraneous tuples with the original relation R.
Formally, Let R be a relation and R1, R2, R3 ... Rn be it's decomposition, the decomposition is lossless if – $R1 \bowtie R2 \bowtie R3 .... \bowtie Rn = R$	Formally, Let R be a relation and R1, R2, R3 ... Rn be its decomposition, the decomposition is lossy if – $R \subset R1 \bowtie R2 \bowtie R3 .... \bowtie Rn$
There is no loss of information as the relation obtained after natural join of decompositions is equivalent to original relation. Thus, it is also referred to as non-additive join decomposition	There is loss of information as extraneous tuples are added into the relation after natural join of decompositions. Thus, it is also referred to as careless decomposition.
The common attribute of the sub relations is a superkey of any one of the relation.	The common attribute of the sub relation is not a superkey of any of the sub relation.

Algorithm For Testing for Lossless Join Property

Input: A universal relation R, a decomposition D = {R1, R2, ..., Rm} of R, and a set F of functional dependencies.

```
1.  Create an initial matrix S with one row i for each relation Ri in D, and one column j for each attribute Aj in R.
2.  Set S(i,j):  =  bij for all matrix entries.  (* each bij is a distinct symbol associated with indices (i,j) *).
3.  For each row i representing relation schema Ri
            {  for each column j representing attribute Aj
               { if (relation Ri includes attribute Aj) then set S(i,j):= aj; }; };
 (* each aj is a distinct symbol associated with index (j) *)
4.  Repeat the following loop until a complete loop execution results in no changes to S
            { for each functional dependency X→Y in F
              { for all rows in S which have the same symbols in the columns corresponding to attributes in X
                 { make the symbols in each column that correspond to an attribute in Y be the same in all these rows
as follows :
            If any of the rows has an "a" symbol for the column, set the other rows to that same "a"
            symbol in the column.
            If no "a" symbol exists for the attribute in any of the rows, choose one of the "b" symbols that
            appear in one of the rows for the attribute and set the other rows to that same "b" symbol in
            the column ;}; }; };
5. If a row is made up entirely of "a" symbols, then the decomposition has the lossless join property; otherwise it does
not.
```

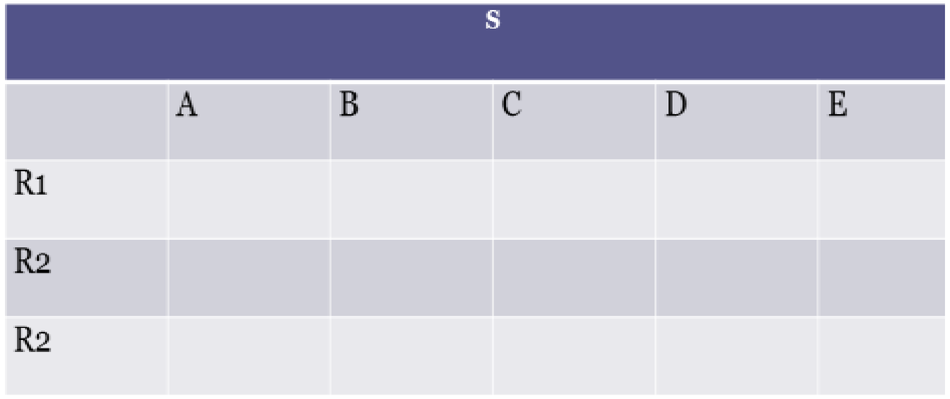
Q) Using algorithm find decomposition is Lossless or not

- Given R(A,B,C,D,E)
- Decomposed to R1(A,B,C), R2(B,C,D), R3(C,D,E)
- FD={AB→CD, A→E, C→D}

Solution :

Let's construct a table of the above relation R, R1 R2 and R3 and insert value in form of bij or aj using ALGO STEP1

(Create an initial matrix S with one row i for each relation in Ri in D, and one column j for each attribute aj in R).

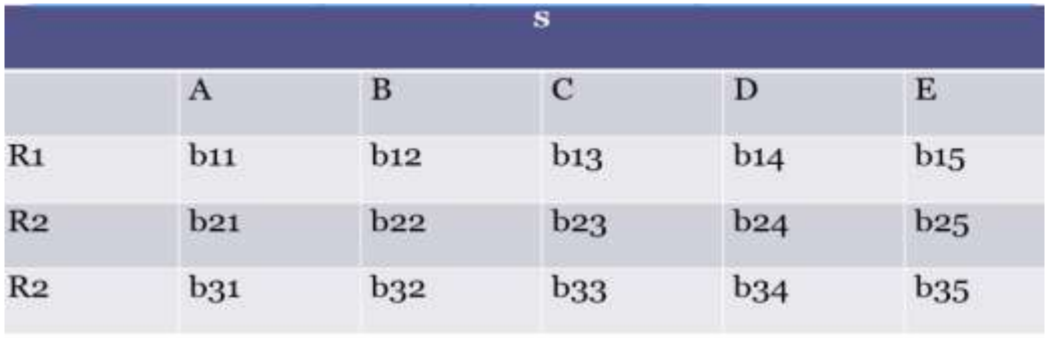


S					
	A	B	C	D	E
R1					
R2					
R2					

• Created a table using R = {AB C D E } where every attribute of R is represented in each column.

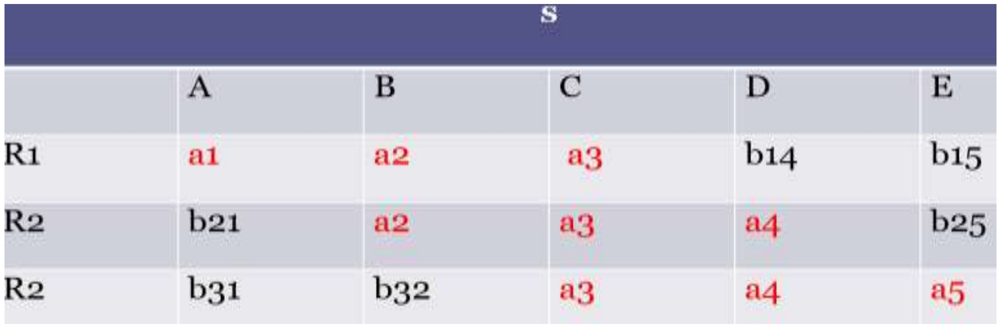
• And initial value of each decomposed table R1 R2 and R3 in the format of bij, where i is the row and j is the column using ALGO STEP2

• (Set S(i, j) := bij for all matrix entries. (* each bij is a distinct symbol associated with



S					
	A	B	C	D	E
R1	b11	b12	b13	b14	b15
R2	b21	b22	b23	b24	b25
R2	b31	b32	b33	b34	b35

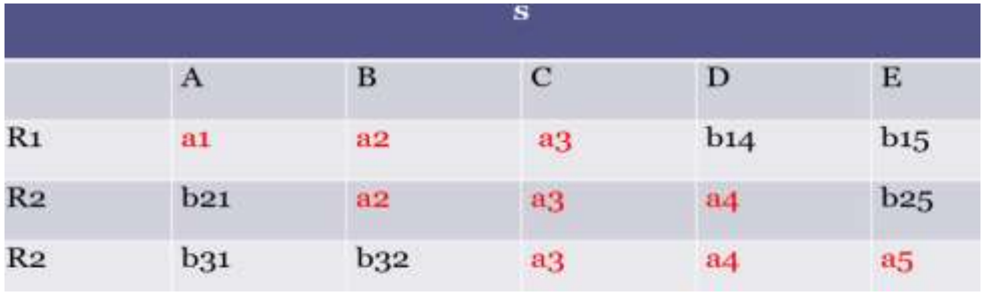
• Now insert value in row R1 R2 and R3 as "aj" using R1 = {A,B ,C } R2 = { B,C,D} and R3 = { C,D,E } using ALGO STEP3 For each row i representing relation schema Ri{for each column j representing attribute Aj {if (relation Ri includes attribute Aj) then set S(i, j):=aj;};}; (* each aj is a distinct symbol associated



S					
	A	B	C	D	E
R1	a1	a2	a3	b14	b15
R2	b21	a2	a3	a4	b25
R2	b31	b32	a3	a4	a5

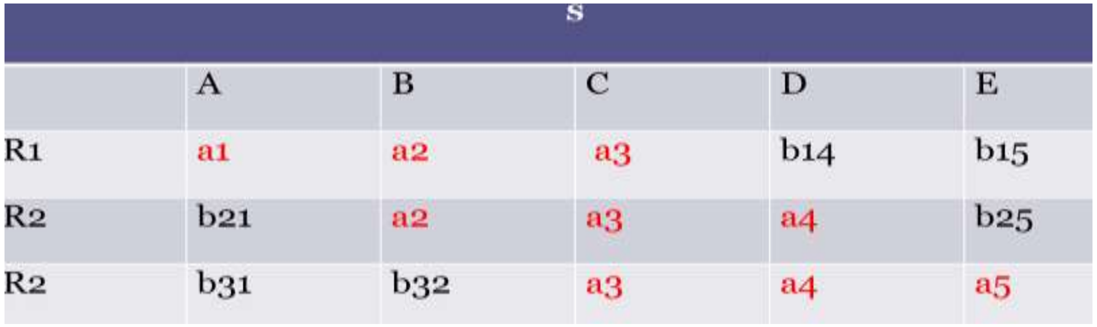
- Given Functional Dependencies are FD={AB→CD, A→E, C→D}
- Using step 4 of above algorithm, if there exist a functional dependency X →Y , and for two tuples t1, and t2 if
- t1 [X] = t2 [X] then we must have
- t1 [Y] = t2 [Y]

- Find in the above table that is there any FD X →Y whose X are equal then make Y also equal.
- Step A : By using the above FD AB→CD, rows of A and B column do not have any same value, No action taken



S					
	A	B	C	D	E
R1	a1	a2	a3	b14	b15
R2	b21	a2	a3	a4	b25
R2	b31	b32	a3	a4	a5

- Step B: By using FDA→ E on the above table we found that A has no same values so No action taken



S					
	A	B	C	D	E
R1	a1	a2	a3	b14	b15
R2	b21	a2	a3	a4	b25
R2	b31	b32	a3	a4	a5

- Step C: Since by using above FD: C → D IN C all tuple have same value a3 so we will make b values in D to a value

	A	B	C	D	E
R1	a1	a2	a3	b14 a4	b15
R2	b21	a2	a3	a4	b25
R2	b31	b32	a3	a4	a5

Now look for any row with all ***a*** values.

Here there is no such row so we can conclude this is a lossy join

IF any row with all ***a*** values , then we can say that this relation is a lossless dependency