

Normalization

Module – 4

Database Design Theory

Introduction

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

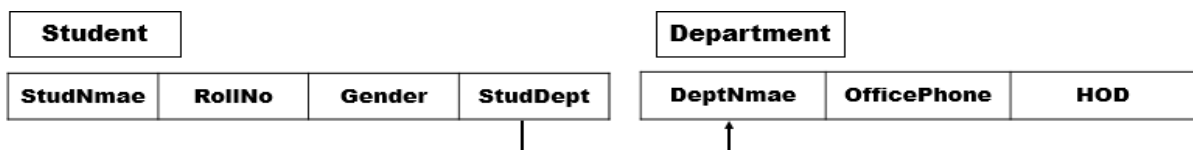
Database Design deals with coming up with a good schema. There are two levels at which we can discuss the goodness of relation schemas:

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

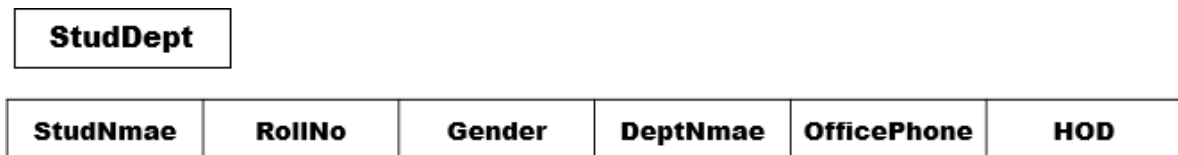
An Example

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

Correct schema:



Incorrect schema:



Problems with bad schema

- **Redundant storage of data:**
 - Office Phone & HOD info -stored redundantly once with each student that belongs to the department.
 - Wastage of disk space.
- **A program that updates Office Phone of a department**
 - Must change it at several places.
 - More running time.
 - Error –prone

(***VVIP)Informal Design Guidelines for Relation Schemas

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.

These measures are not always independent of one another.

1. Imparting Clear Semantics to Attributes in Relations

- Semantics of a relation refers to its meaning resulting from the interpretation of attribute values in a tuple.
- 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.
- The easier it is to explain the semantics of the relation, the better the relation schema design will be,

Guideline 1

- 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
 - If a relation schema corresponds to one entity type or one relationship type, it is straightforward to interpret and to explain its meaning.
 - If the relation corresponds to a mixture of multiple entities and relationships, semantic ambiguities will result and the relation cannot be easily explained.

Examples of Violating Guideline 1

EMP_DEPT						
Ename	SSN	Bdate	Address	Dnumber	Dname	Dmgr_SSN

EMP_PROJ					
SSN	Pnumber	Hours	Pname	Ename	Plocation

- Both the relation schemas have clear semantics
- A tuple in the EMP_DEPT relation schema represents a single employee but includes additional information—the name (Dname) of the department for which the employee works and the Social Security number (Dmgr_ssn) of the department manager.
- A tuple in the EMP_PROJ relates an employee to a project but also includes the employee name (Ename), project name (Pname), and project location (Plocation)
- Logically correct but they violate Guideline 1 by mixing attributes from distinct real-world entities:
 - EMP_DEPT mixes attributes of employees and departments
 - EMP_PROJ mixes attributes of employees and projects and the WORKS_ON relationship.
- They may be used as views, but they cause problems when used as base relations.

2. Redundant Information in Tuples and Update Anomalies

- One goal of schema design is to minimize the storage space used by the base relations
- Grouping attributes into relation schemas has a significant effect on storage space
- For example, compare the space used by the two base relations EMPLOYEE and DEPARTMENT with that for an EMP_DEPT base relation
- In EMP_DEPT, the attribute values pertaining to a particular department (Dnumber, Dname, Dmgr_ssn) are repeated for every employee who works for that department
- In contrast, each department's information appears only once in the DEPARTMENT relation. Only the department number Dnumber is repeated in the EMPLOYEE relation for each employee who works in that department as a foreign key.

EMP_DEPT				Data Redundancy		
Ename	<u>SSN</u>	Bdate	Address	Dnumber	Dname	Dmgr_SSN
Ravi	123456789	1986-05-10	21 Mysore	5	Research	987456321
Ramu	987456321	1987-06-11	22 Mandya	5	Research	987456321
Rohan	456789123	1988-07-12	23 Madhur	4	Administration	789456123
Sunil	789456123	1989-08-13	24 Bangalore	4	Administration	789456123
Smith	654321987	1990-09-14	25 Mangalore	5	Research	987456321

Storing natural joins of base relations leads to an additional problem referred to as update anomalies. These can be classified into:

- insertion anomalies
- deletion anomalies,
- modification anomalies

Insertion Anomalies

Insertion anomalies can be differentiated into two types, illustrated by the following examples based on the EMP_DEPT relation:

1. To insert a new employee tuple into EMP_DEPT, we must include either the attribute values for the department that the employee works for, or NULLs.
 - For example, to insert a new tuple for an employee who works in department number 5, we must enter all the attribute values of department 5 correctly so that they are consistent with the corresponding values for department 5 in other tuples in EMP_DEPT.
 - In the design of Employee in fig 1, we do not have to worry about this consistency problem because we enter only the department number in the employee tuple; all other attribute values of department 5 are recorded only once in the database, as a single tuple in the DEPARTMENT relation.
2. It is difficult to insert a new department that has no employees as yet in the EMP_DEPT relation. The only way to do this is to place NULL values in the attributes for employee.
 - This violates the entity integrity for EMP_DEPT because SSN is its primary key.
 - This problem does not occur in the design of Figure 1 because a department is entered in the DEPARTMENT relation whether or not any employees work for it, and whenever an employee is assigned to that department, a corresponding tuple is inserted in EMPLOYEE.

Deletion Anomalies

The problem of deletion anomalies is related to the second insertion anomaly situation just discussed.

- If we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database
- This problem does not occur in the database of Figure 2 because DEPARTMENT tuples are stored separately.

Modification Anomalies

- In 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.

Guideline 2

- Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations
- If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly
- The second guideline is consistent with and, in a way, a restatement of the first guideline
- These guidelines may sometimes have to be violated in order to improve the performance of certain queries.

3. NULL Values in Tuples

- If many of the attributes do not apply to all tuples in the relation, we end up with many NULLs in those tuples.
 - This can waste space at the storage level.
 - May lead to problems with understanding the meaning of the attributes.
 - May also lead to problems with specifying JOIN operations.
 - How to account for them when aggregate operations such as COUNT or SUM are applied.
- SELECT and JOIN operations involve comparisons; if NULL values are present, the results may become unpredictable.
- Moreover, NULLs can have multiple interpretations, such as the following:
 - The attribute does not apply to this tuple. For example, Ph.D Degree may not apply to Certain Applicants.
 - The attribute value for this tuple is unknown. For example, the Date_of_birth may be unknown for an employee.
 - The value is known but absent; that is, it has not been recorded yet. For example, the Home_Phone_Number for an employee may exist, but may not be available and recorded yet.

Guideline 3

- As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL
- If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation
- Using space efficiently and avoiding joins with NULL values are the two overriding criteria that determine whether to include the columns that may have NULLs in a relation or to have a separate relation for those columns with the appropriate key columns
- For example, if only 15 percent of employees have individual offices, there is little justification for including an attribute Office_number in the EMPLOYEE relation; rather, a relation EMP_OFFICES (Essn, Office_number) can be created to include tuples for only the employees with individual offices.

4. Generation of Spurious Tuples

Consider the two relation schemas EMP_LOCS and EMP_PROJ1 which can be used instead of the single EMP_PROJ.

- A tuple in EMP_LOCS means that the employee whose name is Ename works on some project whose location is Plocation.
- A tuple in EMP_PROJ1 refers to the fact that the employee whose Social Security number is Ssn works Hours per week on the project whose name, number, and location are Pname, Pnumber, and Plocation.

EMP_LOCAS

Ename	Plocation
Smith	Bangalore
Smith	Mysore
Ramu	Mandya
Ravi	Bangalore

EMP_PROJ1

SSN	Pnumber	Hours	Pname	Plocation
123456789	1	30	P1	Bangalore
123456789	2	10	P2	Mysore
654789321	3	20	P3	Mandya

- Suppose that we used EMP_PROJ1 and EMP_LOCS as the base relations instead of EMP_PROJ. This produces a particularly bad schema design because we cannot recover the information that was originally in EMP_PROJ from EMP_PROJ1 and EMP_LOCS.
- If we attempt a NATURAL JOIN operation on EMP_PROJ1 and EMP_LOCS, the result produces many more tuples than the original set of tuples in EMP_PROJ.
- Additional tuples that were not in EMP_PROJ are called spurious tuples because they represent spurious information that is not valid.
- The spurious tuples are marked by asterisks (*).

	SSN	Pnumber	Hours	Pname	Plocation	Ename
	123456789	1	30	P1	Bangalore	Smith
*	123456789	1	30	P1	Bangalore	Ravi
	123456789	2	10	P2	Mysore	Smith
	654789321	3	20	P3	Mandya	Ramu

- Decomposing EMP_PROJ into EMP_LOCS and EMP_PROJ1 is undesirable because when we JOIN them back using NATURAL JOIN, we do not get the correct original information.
- This is because in this case Plocation is the attribute that relates EMP_LOCS and EMP_PROJ1, and Plocation is neither a primary key nor a foreign key in either EMP_LOCS or EMP_PROJ1.

Guideline 4

- Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated.
 - Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

Functional Dependencies

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

Definition of Functional Dependency

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

When we can call it functional dependent i.e., $X \rightarrow Y$ is FD if it satisfy the constraint

If $T1.X = T2.X$
Then, $T1.Y = T2.Y$

Example 1:

		X	Y
T1	→	10	5
		20	6
		30	7
T2	→	10	5

$$T1.X = T2.X = T1.10 = T2.10$$

$$T1.Y = T2.Y = T1.5 = T2.5$$

Hence above **equation is Functional Dependent.**

Example 2:

		X	Y
T1	→	10	5
		20	6
		30	7
T2	→	10	8

$$T1.X = T2.X = T1.10 = T2.10$$

$$T1.Y = T2.Y = T1.58 = T2.58$$

Hence above equation is **not Functional Dependent**.

There two types of functional dependency such as,

Full Functional dependency

Consider X, Y is asset of attributes and if $X \rightarrow Y$ is Full FD i.e., if we remove any attribute of x violates the FD rules.

$$(A, B) \rightarrow C$$

$A \rightarrow C$ is not a FD

$B \rightarrow C$ is not a FD

Partial Functional dependency

Consider X, Y is asset of attributes and if $X \rightarrow Y$ is Partial FD i.e., if we remove any attribute of x does not violates the FD rules.

$$(A, B) \rightarrow C$$

$A \rightarrow C$ is a FD

$B \rightarrow C$ is a FD

Example: Consider the following table and find out FD.

Sid	Sname	Address	Course
101	Ramu	Mysore	Python
101	Ramu	Mysore	Java
102	Ravi	Mandya	Python
103	Ravi	Maddur	C
104	Rohan	Bangalore	Java
105	Sunil	Mysore	Python

Find out FD for single attributes for the following,

$$1. \text{ Sid} \rightarrow \text{Sname}$$

$$T1.X = T2.X = 101 = 101$$

$$T2.Y = T2.Y = \text{Ramu} = \text{Ramu}$$

Hence above **equation is FD**.

$$2. \text{ Sid} \rightarrow \text{Address}$$

$$T1.X = T2.X = 101 = 101$$

$$T2.Y = T2.Y = \text{Mysore} = \text{Mysore}$$

Hence above **equation is FD**.

3. $\text{Sid} \rightarrow \text{Course}$

$$T1.X = T2.X = 101 = 101$$

$$T2.Y = T2.Y = \text{Python} = \text{Java}$$

Hence above **equation is not FD.**

4. $\text{Course} \rightarrow \text{Sname}$

$$T1.X = T2.X = \text{Python} = \text{Java}$$

$$T2.Y = T2.Y = \text{Ramu} = \text{Ramu}$$

Hence above **equation is not FD.**

5. $\text{Sname} \rightarrow \text{Sid}$

$$T1.X = T2.X = \text{Ravi} = \text{Ravi}$$

$$T2.Y = T2.Y = 102 = 103$$

Hence above **equation is not FD.**

Find out FD for set of attributes for the following,

1. $(\text{Sid}, \text{Sname}) \rightarrow \text{Course}$

$$101, \text{Ramu} = \text{Python}$$

$$101, \text{Ramu} = \text{Java}$$

Hence above **equation is not FD.**

2. $(\text{Sid}, \text{Course}) \rightarrow \text{Sname}$

$$101, \text{Java} = \text{Ramu}$$

$$102, \text{Python} = \text{Ravi}$$

$$1032, \text{C} = \text{Ravi}$$

$$104, \text{Java} = \text{Rohan}$$

$$105, \text{Python} = \text{Sunil}$$

All are unique. Hence above **equation is FD.**

Find out Partial FD for set of attributes for the following,

1. $(\text{Sname}, \text{Course}) \rightarrow \text{Sname}$

If we remove the course and it is still a FD then we call it as partial FD.

i.e., after removing course then, $\text{Sname} \rightarrow \text{Sname}$ is a FD, so it is partial FD.

Find out Full FD for set of attributes for the following,

1. $(\text{Sname}, \text{Course}) \rightarrow \text{Sid}$

$$101, \text{Java} = \text{Ramu}$$

$$102, \text{Python} = \text{Ravi}$$

$$1032, \text{C} = \text{Ravi}$$

$$104, \text{Java} = \text{Rohan}$$

$$105, \text{Python} = \text{Sunil}$$

All are unique. Hence above **equation is FD.**

If we remove the course and it is still a FD then we call it as partial FD.

i.e., after removing course then, $\text{Sname} \rightarrow \text{Sid}$ is a FD, so it is partial FD.

$$T1.X = T2.X = \text{Ramu} = \text{Ramu}$$

$$T2.Y = T2.Y = 101 = 101$$

$$T1.X = T2.X = \text{Ravi} = \text{Ravi}$$

$$T2.Y = T2.Y = 102 = 103$$

Hence above **equation is not FD.**

If we remove the Sname and it is still a FD then we call it as partial FD.
i.e., after removing Sname then, Course \rightarrow Sid is a FD, so it is partial FD.

$$T1.X = T2.X = \text{Python} = \text{Python}$$

$$T2.Y = T2.Y = 101 = 102$$

Hence above **equation is not FD.**

Therefore, (Sname, Course) \rightarrow Sid is Full FD.

(VVIP)Types of Functional Dependency**

Consider the following table for all the types of functional dependency.

Sid	Sname	Age	College	Place
101	Ramu	25	NIT	Mandya
102	Ravi	24	MUSE	Mysore
103	Rohan	25	VVCE	Maddur
104	Ravi	20	VVIET	Mysore

1. TRIVIAL FD

If $A \rightarrow B$ is a FD, if B is subset of A then we call it as trivial FD, that means the dependent should be a subset of determinant. Here A is the determinant and B is the dependent. So A can have multiple attributes as well as B can have multiple attributes. So, whatever the attributes in B it should be a subset of A.

Example: Sid, Sname \rightarrow Sid (where dependent Sid is subset of determinant Sname and Sid).

2. Non-Trivial FD

If $A \rightarrow B$ is FD and if B is not a subset of A then we call it as non-trivial FD.

Example: Sid, Sname \rightarrow age (where dependent age is not a subset of determinant Sname and Sid).

3. Multivalued FD

If $A \rightarrow BC$ is a FD, if B and C should not be dependent then we call it as multivalued FD.
i.e., $B \rightarrow C$ and $C \rightarrow B$ both should not be FD.

Example: $Sid \rightarrow Sname, age$

i.e., $Sname \rightarrow age$

Ravi \rightarrow 24

Ravi \rightarrow 23

Hence above **equation is not FD.**

Age \rightarrow Sname

Ramu \rightarrow 25

Rohan \rightarrow 25

Hence above **equation is not FD.**

4. Transitive FD

If $A \rightarrow B$ is FD and if $B \rightarrow C$ is a FD, then $A \rightarrow C$ is also FD then we call it is as transitive FD.

Example: $Sid \rightarrow College$ ----- is a FD

$College \rightarrow Place$ -- is a FD then,

$Sid \rightarrow Place$ -----is also a FD.

Closure Attribute

It is represented by '+'. For each such set of attributes X, we determine the set X^+ of attributes that are functionally determined by X based on F; X^+ is called the closure of X under F. We apply this closure on single attribute and also on multiple attribute.

Algorithm for closure i.e., determining X^+ , the closure of X under F.

Input: A set of FDs on a relation schema R and set of attributes X, is subset of R.

$X^+ := X$;

repeat

 old $X^+ := X^+$;

 for each functional dependency $Y \rightarrow Z$ in f do

 if $X^+ \supseteq Y$ then $X^+ := X^+ \cup Z$;

until ($X^+ = \text{old}X^+$);

Example 1: In a given relation $R = (A, B, C, D)$ if the FD are $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$, $D \rightarrow A$. Determine the Candidate Key, Prime Attribute & Non-Prime Attribute

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

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

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

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

Candidate Key: $\{A, B, C, D\}$ Prime Attribute: $\{A, B, C, D\}$ Non-Prime Attribute: $\{\phi\}$

Example 2: In a given relation $R = (A, B, C, D, E)$ if the FD are $A \rightarrow B$, $BC \rightarrow D$, $E \rightarrow C$, $D \rightarrow A$. Determine the Candidate Key, Prime Attribute & Non-Prime Attribute

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

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

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

$CE^+ = \{C, E\}$

Candidate Key: $\{AE, BE, DE\}$ Prime Attribute: $\{A, B, E, D\}$ Non-Prime Attribute: $\{C\}$

Example3: In a given relation $R = (A, B, C, D, E)$ if the FD are $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$, $D \rightarrow E$. Determine the Candidate Key, Prime Attribute & Non-Prime Attribute

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

$B^+ = \{B, C, D, E\}$

$C^+ = \{C, D, E\}$

$D^+ = \{D, E\}$

Candidate Key: $\{A\}$ Prime Attribute: $\{A\}$ Non-Prime Attribute: $\{B, E, D, C\}$

Normal Forms Based on Primary Keys

We assume that a,

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

1. Normalization of Relations

- The normalization process, as first proposed by Codd (1972a), takes a relation schema through a series of tests to certify whether it satisfies a certain normal form.
- Initially, Codd proposed three normal forms, which he called first, second, and third normal form.
- All these normal forms are based on a single analytical tool: the functional dependencies among the attributes of a relation.
- A fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of multivalued dependencies and join dependencies, respectively.
- **Normalization of data can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of,**
 - Minimizing redundancy and
 - Minimizing the insertion, deletion, and update anomalies.
- **It can be considered as a —filtering‖ or —purification‖ process to make the design have successively better quality.**
- Unsatisfactory relation schemas that do not meet certain conditions—the normal form tests—are decomposed into smaller relation schemas that meet the tests and hence possess the desirable properties.
- Thus, the normalization procedure provides database designers with the following:
 - A formal framework for analyzing relation schemas based on their keys and on

the functional dependencies among their attributes.

- A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be normalized to any desired degree.

Definition: The normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.

2. Practical Use of Normal Forms

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

Definition: Denormalization is the process of storing the join of higher normal form relations as a base relation, which is in a lower normal form.

3. Definitions of Keys and Attributes Participating in Keys

Superkey: specifies a uniqueness constraint that no two distinct tuples in any state r of R can have the same value.

key K is a superkey with the additional property that removal of any attribute from K will cause K not to be a superkey anymore.

Example:

- The attribute set $\{Ssn\}$ is a key because no two employees tuples can have the same value for Ssn .
- Any set of attributes that includes Ssn —for example, $\{Ssn, Name, Address\}$ —is a superkey.

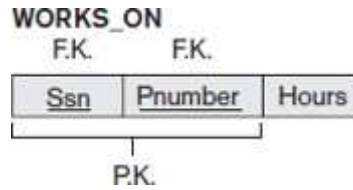
If a relation schema has more than one key, each is called a **candidate key**.

One of the candidate keys is arbitrarily designated to be the **primary key**, and the others are called **secondary keys**.

In a practical relational database, each relation schema must have a primary key. If no candidate key is known for a relation, the entire relation can be treated as a default superkey.

For example $\{Ssn\}$ is the only candidate key for EMPLOYEE, so it is also the primary key.

Definition. An attribute of relation schema R is called a prime attribute of R if it is a member of some candidate key of R . An attribute is called **nonprime** if it is not a **prime attribute**—that is, if it is not a member of any candidate key.



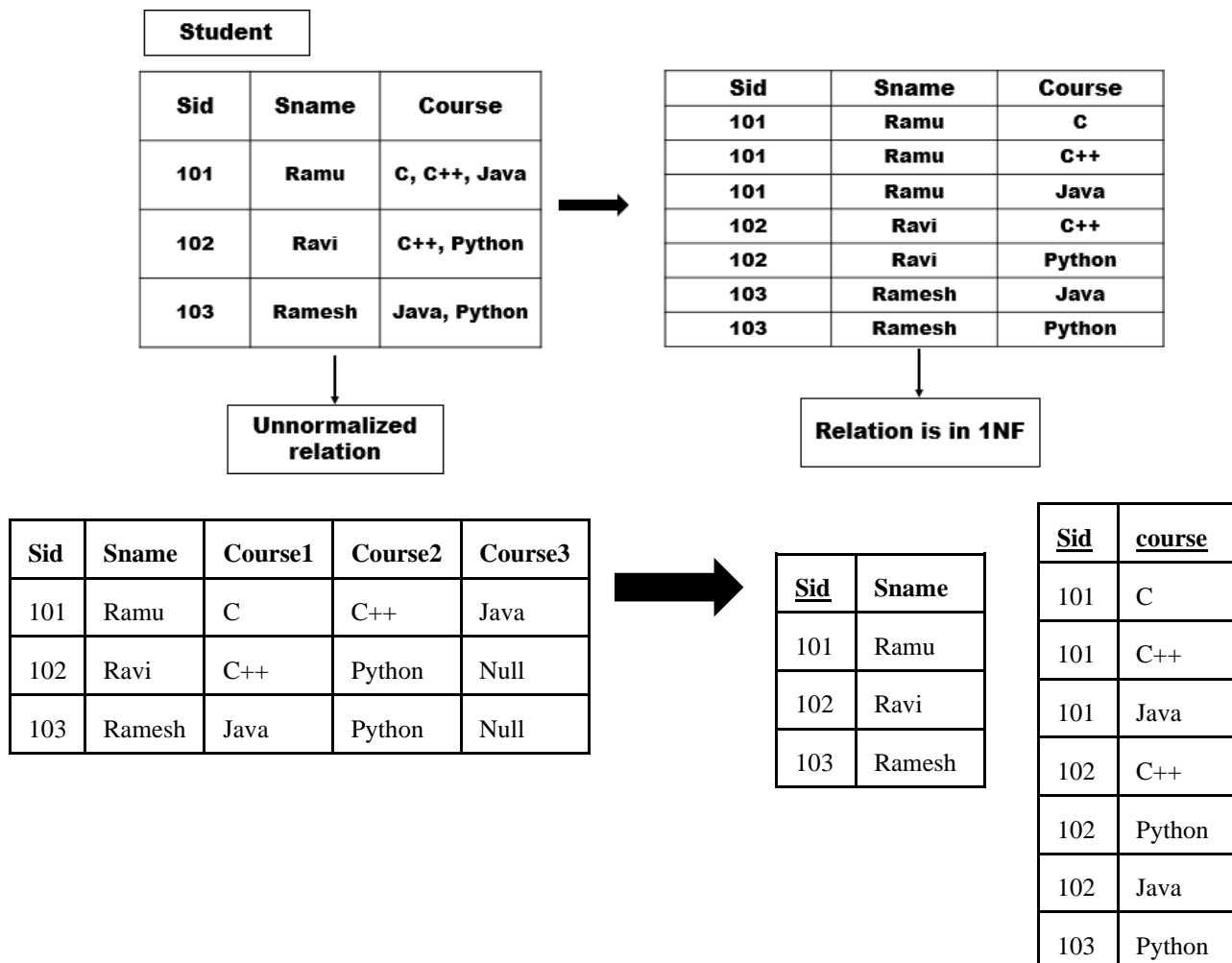
In the above figure, In WORKS_ON relation Both SSN and Pnumber are prime attributes whereas other attributes are nonprime.

(VVIP)First Normal Form (1NF)**

- Does not **multivalued attributes, composite attributes, and their combinations**
- It states that the domain of an attribute must include only **atomic** (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute
- 1NF disallows relations within relations or relations as attribute values within tuples
- The only attribute values permitted by 1NF **are single atomic (or indivisible) values**.

Input for 1NF is unnormalized data and the output will be Relation in 1NF.

Example: Consider Student relation below which is in the form of unnormalized relation. Following are three ways to decompose the relation and the last is the best way



(**VVIP) Second Normal Form (2NF)

- Second normal form (2NF) is based on the concept of full functional dependency.
- A functional dependency $X \rightarrow Y$ 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 $A \in X$, $(X - \{A\})$ does not functionally determine Y .
- A functional dependency $X \rightarrow Y$ is a partial dependency if some attribute $A \in X$ can be removed from X and the dependency still holds; that is, for some $A \in X$, $(X - \{A\}) \rightarrow Y$.
- **Definition.** A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R

Example 1: FD are $\text{Student-ID} \rightarrow \text{Course-ID}$, $\text{Student-ID} \rightarrow \text{Course-Fee}$, $\text{Course-ID} \rightarrow \text{Course-Fee}$, $\text{Student-ID, Course-ID} \rightarrow \text{Course-Fee}$, $\text{Student-ID, Course-Fee} \rightarrow \text{Course-ID}$, $\text{Course-ID, Course-Fee} \rightarrow \text{Student-ID}$.

Student		
Student-ID	Course-ID	Course-Fee
101	Java	10000
102	Python	15000
101	C++	20000
103	C	10000
103	Java	10000
102	JavaScript	20000

check partial dependency for the equation which hold FD

$\text{Student-ID, Course-ID} \rightarrow \text{Course-Fee}$

$\text{Student-ID} \rightarrow \text{Course-Fee}$ (this equation is not FD).

$\text{Course-ID} \rightarrow \text{Course-Fee}$ (Hence this equation is FD).

$\text{Course-ID} \rightarrow \text{Course-Fee}$

Course fee is determined by **Course ID alone** (Partial Dependency)

So it is partial functional dependency but not in 2NF. Therefore the above student relation decomposed into two relation Student1 and Student2 as shown below which is in 2NF.

Student1		Student2	
Student-ID	Course-ID	Course-ID	Course-Fee
101	Java	Java	10000
102	Python	Python	15000
101	C++	C++	20000
103	C	C	10000
103	Java	Java	10000
102	JavaScript	JavaScript	20000

Example2: In a given relation $R = (A, B, C, D, E, F)$ if the FD are $C \rightarrow F$, $E \rightarrow A$, $EC \rightarrow D$, $A \rightarrow B$ Determine whether the relation is in **2NF**

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

Candidate Key: $\{EC\}$ Prime Attribute: $\{E, C\}$ Non-Prime Attribute: $\{A, B, F, D\}$

From the above Functional dependencies: $C \rightarrow F$, $E \rightarrow A$, $EC \rightarrow D$, $A \rightarrow B$

Part of Candidate Key ie, C alone is determining the non-Prime Attribute F

Hence the given relation is not in Second Normal Form (2NF)

(VVIP)Third Normal Form (3NF)**

1. A database table is in second normal form and is in first normal form.
2. There is no transitive dependency. If $A \rightarrow B$, then A can be a super key or candidate key.
3. Non-Prime Attribute should be determined by Prime Attribute

Example 1:

Score-id	Stu-id	Sub-id	Marks	Exam-name	Total-marks
101	201	1	6	Mid	20
102	201	2	69	Final	80
103	203	1	19	Mid	20

Candidate key: $\{Score-id, Stu-id, Sub-id\}$

Prime Attribute: $\{Score-id, Stu-id, Sub-id\}$

Non-Prime Attribute: $\{Exam-name, Total-marks\}$

Total marks is transitively dependent on Exam-name

Non-Prime Attribute is determined by a Non-Prime Attribute

The following table should be decomposed as below.

Score-id	Stu-id	Sub-id	Marks	Exam-id

Exam-id	Exam-name	Total-marks

Example2: In a given relation $R = (A, B, C, D)$ if the FD are $AB \rightarrow CD$, $C \rightarrow D$ Determine whether the relation is in **3NF**

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

Candidate Key: $\{AB\}$ Prime Attribute: $\{A, B\}$ Non-Prime Attribute: $\{C, D\}$

From the above Functional dependencies: $C \rightarrow D$

non-Prime Attribute D is determined by non-Prime Attribute C

Hence the given relation is not in Third Normal Form (3NF)

(VVIP)Boyce-Codd Normal Form (BCNF)**

Input for BCNF is relation should be in 3NF and the output will be Relation in BCNF.

- Boyce-Codd Normal Form (BCNF) is a stricter version of Third Normal Form (3NF) that ensures a more simplified and efficient database design .
- **A table is in BCNF if every functional dependency $X \rightarrow Y$, X is the super key of the table**
- **For BCNF, the table should be in 3NF, and for every FD. LHS is super key.**

Example: FD are Student \rightarrow Course, Student \rightarrow Tutor, Course \rightarrow Tutor, Student, Course \rightarrow Tutor, Student, Tutor \rightarrow Course, Course, Tutor \rightarrow Student.

Student		
Student	Course	Tutor
101	Java	Ravi
101	Python	Ramu
102	Java	Ravi
102	Python	Rohan

1. Student \rightarrow Course
101 \rightarrow Java
101 \rightarrow Python

Hence above **equation is not FD.**

2. Student \rightarrow Tutor
101 \rightarrow Ravi
101 \rightarrow Ramu

Hence above **equation is not FD.**

3. Course \rightarrow Tutor
Java \rightarrow Ravi
Java \rightarrow Ravi

But

Python \rightarrow Ramu
Python \rightarrow Rohan

Hence above **equation is not FD.**

4. Tutor \rightarrow Course
Ravi \rightarrow Java
Ravi \rightarrow Java

But

Ramu \rightarrow Python
Rohan \rightarrow Python

For Different X values it can be same y value. Hence above **equation is FD.**

5. Course \rightarrow Student

Java \rightarrow 101

Java \rightarrow 102

Hence above **equation is not FD.**

6. Tutor \rightarrow Student

Ravi \rightarrow 101

Ramu \rightarrow 101

Hence above **equation is not FD.**

7. Student, Course \rightarrow Tutor

101, Java \rightarrow Ravi

102, Java \rightarrow Ravi

Therefore no duplicates in x values for all the tuples. Hence above **equation is FD.**

8. Student, Tutor \rightarrow Course

101, Ravi \rightarrow Java

102, Ravi \rightarrow Java

Therefore no duplicates in x values for all the tuples. Hence above **equation is FD.**

9. Course, Tutor \rightarrow Student

Java, Ravi \rightarrow 101

Java, Ravi \rightarrow 102

Therefore there is a duplicates in x values for some tuples. Hence above **equation is not FD.**

Hence we have find out the FD such as Student, Course \rightarrow Tutor, Student, Tutor \rightarrow Course, Tutor \rightarrow Course.

Find the candidate keys

Student, ~~Course~~, Tutor = {Student, Course, Tutor}

Student, Tutor = {Student, Course, Tutor}

Now find out closure of Student and Tutor

Student⁺ = {Student}

Tutor⁺ = {Tutor, Course}

So Prime attributes are Student and Tutor replace tutor with Course.

So {Student, Tutor} are the candidate key.

Student, Tutor

Student, Course = {Student, Course, Tutor}

Now find out closure of Student and Course

Student⁺ = {Student}

Course⁺ = {Course}

So Prime attributes are Student and Course replace Course with Tutor. Already Student, Tutor are candidate key so stop at this point.

So {Student, Course} are the candidate key.

Hence in relation we have three attributes and all are present in candidate key. Therefore Non-Prime attributes are NIL in the relation.

Hence relation is in 3NF. Now check the relation is in BCNF are not.

If relation should be in BCNF then, for all the FD X should be a super key or candidate key.

For **FD** Tutor \rightarrow Course. Were tutor is not a candidate key, so relation is not in BCNF.

Now the relation is decompose the relation into two tables as Student Tutor and Course Tutor so that it will satisfy the BCNF.

Student1		Student2	
Student	Tutor	Course	Tutor
101	Ravi	Java	Ravi
101	Ramu	Python	Ramu
102	Ravi	Java	Ravi
102	Rohan	Python	Rohan

Example2:

Ck: {Roll no, Voter-id}

Fd: { Roll no \rightarrow name, Roll no \rightarrow voter-id, voter-id \rightarrow age, voter-id \rightarrow Roll-no }

Roll no	Name	Voter-id	Age
1	Ravi	K0123	21
2	Varun	M0234	21
3	Ravi	K0234	21

LHS of each FD is candidate key or Super key

Hence the above table is in BCNF

(VVIP)Multivalued Dependency and Fourth Normal Form (4NF)**

2NF, 3NF and BCNF are completely based on keys and functional dependency, whereas 4NF is based on keys and multivalued dependency.

Input for 4NF is relation should be in BCNF and the output will be Relation in 4NF.

Definition. A multivalued dependency $X \twoheadrightarrow Y$ specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R: If two tuples t1 and t2 exist in r such that $t1[X] = t2[X]$, then two tuples t3 and t4 should also exist in r with the following properties where we use Z to denote $(R - (X \cup Y))$

- $t3[X] = t4[X] = t1[X] = t2[X]$.
- $t3[Y] = t1[Y]$ and $t4[Y] = t2[Y]$.
- $t3[Z] = t2[Z]$ and $t4[Z] = t1[Z]$.

For a relation R to be in 4NF, it must meet two conditions –

- It should be in Boyce-Codd Normal Form (BCNF).
- It should not have any non-trivial multivalued dependencies.
- **Multivalued dependencies** are a consequence of 1NF which disallows an attribute in a tuple to have a set of values, and the accompanying process of converting an unnormalized relation into 1NF.

Example: For example, consider the relation EMP shown in Figure below:

- A tuple in this EMP relation represents the fact that an employee whose name is Ename works on the project whose name is Pname and has a dependent whose name is Dname.
- An employee may work on several projects and may have several dependents.
- The employee's projects and dependents are independent of one another.
- To keep the relation state consistent, and to avoid any spurious relationship between the two independent attributes, we must have a separate tuple to represent every combination of an employee's dependent and an employee's project.
- In the relation state shown in the EMP, the employee Smith works on two projects
- X and Y and has two dependents John and Anna and therefore there are 4 tuples to represent these facts together.
- The relation EMP is an all-key relation (with key made up of all attributes) and therefore no FDs and as such qualifies to be a BCNF relation.
- There is a redundancy in the relation EMP-the dependent information is repeated for every project and project information is repeated for every dependent.
- To address this situation, the concept of multivalued dependency.(MVD) was proposed and based on this dependency, the fourth normal form was defined.
- Informally, whenever two independent 1:N relationships are mixed in the same relation, $R(A, B, C)$, an MVD may arise.

Employee			
	X	Y	Z
	<u>Ename</u>	<u>Pname</u>	<u>Dname</u>
t1	Smith	X	John
t2	Smith	Y	Anna
t3	Smith	X	John
t4	Smith	Y	John

Let $X = \text{Ename}$, $Y = \text{Pname}$
 $t1[\text{Ename}] = t2[\text{ename}] = \text{Smith}$
 $Z = (\text{EMP} - (\text{Ename} \cup \text{Pname}))$
 $= \text{Dname}$

$t3(\text{Ename}) = t4(\text{Ename}) = t1(\text{Ename}) = t2(\text{Ename}) = \text{Smith}$
 $t3(\text{Pname}) = t1(\text{Pname}) = X$ and $t4(\text{Pname}) = t2(\text{Pname}) = Y$
 $t3(\text{Dname}) = t2(\text{Dname}) = \text{Anna}$ and $t4(\text{Dname}) = t1(\text{Dname}) = \text{John}$

- Whenever $X \twoheadrightarrow Y$ holds, we say that X multidetermines Y . Because of the symmetry in the definition, whenever $X \twoheadrightarrow Y$ holds in R , so does $X \twoheadrightarrow Z$. Hence, $X \twoheadrightarrow Y$ implies $X \twoheadrightarrow Z$, and therefore it is sometimes written as $X \twoheadrightarrow Y|Z$.
- An MVD $X \twoheadrightarrow Y$ in R is called a trivial MVD if,
 - Y is a subset of X , or
 - $X \cup Y = R$

EMP_PROJECTS

<u>Ename</u>	<u>Pname</u>
Smith	X
Smith	Y

- For the above example, the relation EMP_PROJECTS has the trivial MVD
 $\text{Ename} \twoheadrightarrow \text{Pname}$
- An MVD that satisfies neither (a) nor (b) is called a **nontrivial MVD**
- If we have a nontrivial MVD in a relation, we may have to repeat values redundantly in the tuples
- In the EMP relation the values 'X' and 'Y' of Pname are repeated with each value of Dname (or, by symmetry, the values 'John' and 'Anna' of Dname are repeated with each value of Pname)
- This redundancy is clearly undesirable.
- We now present the definition of **fourth normal form (4NF)**, which is violated when a relation has undesirable multivalued dependencies, and hence can be used to identify and decompose such relations.

Definition: A relation schema R is in 4NF with respect to a set of dependencies F (that includes functional dependencies and multivalued dependencies) if, for every nontrivial multivalued

dependency $X \twoheadrightarrow Y$ in F^+ X is a superkey for R .

The process of normalizing a relation involving the nontrivial MVDs that is not in 4NF consists of decomposing it so that each MVD is represented by a separate relation where it becomes a trivial MVD.

- We decompose EMP into EMP_PROJECTS and EMP_DEPENDENTS.
- Both EMP_PROJECTS and EMP_DEPENDENTS are in 4NF, because the MVDs $Ename \twoheadrightarrow Pname$ in EMP_PROJECTS and $Ename \twoheadrightarrow Dname$ in EMP_DEPENDENTS are trivial MVDs.
- No other nontrivial MVDs hold in either EMP_PROJECTS or EMP_DEPENDENTS. No FDs hold in these relation schemas either.

EMP_PROJECTS		EMP_DEPENDENTS	
<u>Ename</u>	<u>Pname</u>	<u>Ename</u>	<u>Dname</u>
Smith	X	Smith	John
Smith	Y	Smith	Anna

(VVIP)Join Dependencies and Fifth Normal Form (5NF)**

A **join dependency (JD)**, denoted by $JD(R_1, R_2, \dots, R_n)$, specified on relation schema R , specifies a constraint on the states r of R . The constraint states that every legal state r of R should have a nonadditive join decomposition into R_1, R_2, \dots, R_n . Hence, for every such r we have,

$$* \pi_{R_1}(r), \pi_{R_2}(r), \dots, \pi_{R_n}(r) = r$$

A join dependency $JD(R_1, R_2, \dots, R_n)$, specified on relation schema R , is a trivial JD if one of the relation schemas R_i in $JD(R_1, R_2, \dots, R_n)$ is equal to R .

Fifth normal form (project-join normal form)

- A relation schema R is in fifth normal form (5NF) (or project-join normal form (PJNF)) with respect to a set F of functional, multivalued, and join dependencies if, for every nontrivial join dependency $JD(R_1, R_2, \dots, R_n)$ in F^+ every R_i is a superkey of R .
- **A database is said to be in 5NF, if and only if,**
 - **It's in 4NF**
 - **If we can decompose table further to eliminate redundancy and anomaly**
 - **when we re-join the decomposed tables by means of candidate keys, we should not be losing the original data or any new record set should not arise.**
 - **In simple words, joining two or more decomposed table should not lose records nor create new records.**

Example: The relation supply with no Multivalued dependency in 4NF but not in 5NF if it has the $JD(R_1, R_2, R_3)$. Decompose the relation supply into 5NF relation (R_1, R_2, R_3) .

Supply

<u>Sname</u>	<u>Part_name</u>	<u>Project_name</u>
Smith	Bolt	PX
Smith	Nut	PY
Adamsky	Bolt	PY
Walton	Nut	PZ
Adamsky	Nail	PX
Adamsky	Bolt	PX
Smith	Bolt	PY

R1

<u>Sname</u>	<u>Part_name</u>
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R2

<u>Sname</u>	<u>Project_name</u>
Smith	PX
Smith	PY
Adamsky	PY
Walton	PZ
Adamsky	PX

R3

<u>Part_name</u>	<u>Project_name</u>
Bolt	PX
Nut	PY
Bolt	PY
Nut	PZ
Nail	PX

Normalization Algorithms

(**VVIP)Inference Rules for Functional Dependencies

Three rules IR1 through IR3 are well-known inference rules for functional dependencies. They are proposed by Armstrong and hence known as Armstrong's axioms.

1. Reflexivity Rule (IR1)

If A is a set of attributes and B is a subset of A, Then $A \rightarrow B$ is a FD. i.e., **Dependent is a subset of determinant.**

2. Augmentation Rule (IR2)

If $A \rightarrow B$ is FD and if C is attribute or set of attributes added to both determinant and dependent (i.e., LHS = RHS) then, $AC \rightarrow CD$ is also FD.

3. Transitive Rule (IR3)

If $A \rightarrow B$ is FD and $B \rightarrow C$ is also FD then, $A \rightarrow C$ is also FD.

- The reflexive rule (IR1) states that a set of attributes always determines itself or any of its subsets, which is obvious.
- Because IR1 generates dependencies that are always true, such dependencies are called trivial.
- Formally, a functional dependency $X \rightarrow Y$ is trivial if $X \supseteq Y$; otherwise, it is nontrivial.
- The augmentation rule (IR2) says that adding the same set of attributes to both the left- and right-hand sides of a dependency results in another valid dependency.
- According to IR3, functional dependencies are transitive.

There are three other inference rules that follow from IR1, IR2 and IR3. They are also called as Secondary Rules and it is derived from Armstrong Axioms.

1. Decomposition or projective rule (IR4)

If $A \rightarrow BC$ is a FD then, $A \rightarrow B$, $A \rightarrow C$ is FD is also FD.

2. Union or additive Rule (IR5)

If $A \rightarrow B$ is an FD and $A \rightarrow C$ is a FD then, $A \rightarrow BC$ is also an FD.

3. Pseudotransitive Rule (IR6)

If $A \rightarrow B$ is a FD and $BC \rightarrow D$ is a FD then $AC \rightarrow D$ is also FD.

- The decomposition rule (IR4) says that we can remove attributes from the right-hand side of a dependency; applying this rule repeatedly can decompose the FD $X \rightarrow \{A_1, A_2, \dots, A_n\}$ into the set of dependencies $\{X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n\}$.

- The union rule (IR5) allows us to do the opposite; we can combine a set of dependencies $\{X \rightarrow A1, X \rightarrow A2, \dots, X \rightarrow An\}$ into the single FD $X \rightarrow \{A1, A2, \dots, An\}$.
- The pseudotransitive rule (IR6) allows us to replace a set of attributes Y on the left hand side of a dependency with another set X that functionally determines Y, and can be derived from IR2 and IR3 if we augment the first functional dependency $X \rightarrow Y$ with W (the augmentation rule) and then apply the transitive rule.
- In other words, the set of dependencies F^+ , which we called the closure of F, can be determined from F by using only inference rules IR1 through IR3.
- A systematic way to determine these additional functional dependencies is first to determine each set of attributes X that appears as a left-hand side of some functional dependency in F and then to determine the set of all attributes that are dependent on X.

Definition. For each such set of attributes X, we determine the set X^+ of attributes that are functionally determined by X based on F; X^+ is called the closure of X under F.

Algorithm for closure i.e., determining X^+ , the closure of X under F.

Input: A set of FDs on a relation schema R and set of attributes X, is subset of R.

```

 $X^+ := X;$ 
repeat
  old  $X^+ := X^+;$ 
  for each functional dependency  $Y \rightarrow Z$  in f do
    if  $X^+ \supseteq Y$  then  $X^+ := X^+ \cup Z;$ 
until ( $X^+ = \text{old}X^+$ );

```

- Algorithm starts by setting X^+ to all the attributes in X.
- By IR1, we know that all these attributes are functionally dependent on X.
- Using inference rules IR3 and IR4, we add attributes to X^+ , using each functional dependency in F.
- We keep going through all the dependencies in F (the repeat loop) until no more attributes are added to X^+ during a complete cycle (of the for loop) through the dependencies in F.

For example, consider the relation schema EMP_PROJ. From the semantics of the attributes, we specify the following set F of functional dependencies that should hold on EMP_PROJ:

$FD = \{ Ssn \rightarrow Ename, Pnumber \rightarrow \{Pname, Plocation\}, \{Ssn, Pnumber\} \rightarrow Hours \}$

Using Algorithm, we calculate the following closure sets with respect to F:

```

 $\{Ssn\}^+ = \{Ssn, Ename\}$ 
 $\{Pnumber\}^+ = \{Pnumber, Pname, Plocation\}$ 
 $\{Ssn, Pnumber\}^+ = \{Ssn, Pnumber, Ename, Pname, Plocation, Hours\}$ 

```

Finding candidate key using closure method

1. In given relation R (A, B, C, D) the FD are $\{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$ then find the closure of A, B and C.

$$A^+ = \{A, B, C, D\} \quad B^+ = \{B, C, D\} \quad C^+ = \{C, D\}.$$

Here in the above relation A is candidate key. Whereas candidate key it that one which determine rest of an attributes in relation schema R.

(**VVIP)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; that is, E is equivalent to F if both the conditions—E covers F and F covers E—hold.

Example 1: The FD of F and G are given as $F = \{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}$ and $G = \{A \rightarrow B, B \rightarrow C, A \rightarrow B\}$ then check whether F and G are equivalent.

Step 1: To check whether FDs in F can be inferred using FDs in G. So, take the attributes from the LHS of FDs in F and compute closure for each using FDs in G.

A^+ using $G = \{A, B, C, D\}$ i.e., $A \rightarrow A, \underline{A \rightarrow B}, A \rightarrow C, A \rightarrow D$

B^+ using $G = \{B, C\}$ i.e., $\underline{B \rightarrow C}, B \rightarrow B$.

AC^+ using $G = \{A, B, C, D\}$ i.e., $AC \rightarrow A, AC \rightarrow B, AC \rightarrow C, \underline{AC \rightarrow D}$

Step 2: To check whether FDs in G can be inferred using FDs in F. So, take the attributes from the LHS of FDs in G and compute closure for each using FDs in F.

A^+ using $F = \{A, B, C, D\}$ i.e., $A \rightarrow A, \underline{A \rightarrow B}, A \rightarrow C, \underline{A \rightarrow D}$

B^+ using $F = \{B, C\}$ i.e., $\underline{B \rightarrow C}, B \rightarrow B$.

Since all FDs can be obtained from G and vice versa, we conclude the F and G are equivalent.

Example 2: The FD of X and Y are given as $X = \{A \rightarrow B, B \rightarrow C\}$ and $Y = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$ then check whether X and Y are equivalent.

Step 1: To check whether FDs in X can be inferred using FDs in Y.

A^+ using $X = \{A, B, C\}$ i.e., $A \rightarrow A, \underline{A \rightarrow B}, \underline{A \rightarrow C}$

B^+ using $X = \{B, C\}$ i.e., $\underline{B \rightarrow C}, B \rightarrow B$.

Step 2: To check whether FDs in Y can be inferred using FDs in X.

A^+ using $Y = \{A, B, C\}$ i.e., $A \rightarrow A$, $\underline{A \rightarrow B}$, $A \rightarrow C$
 B^+ using $Y = \{B, C\}$ i.e., $\underline{B \rightarrow C}$, $B \rightarrow B$.

Since all FDs can be obtained from X and vice versa, we conclude the X and Y are equivalent.

Example 3: The FD of X and Y are given as $X = \{A \rightarrow C, AC \rightarrow D, E \rightarrow H\}$ and $Y = \{A \rightarrow CD, E \rightarrow AH\}$ then check whether X and Y are equivalent.

Step 1: To check whether FDs in X can be inferred using FDs in Y.

A^+ using $Y = \{A, C, D\}$ i.e., $A \rightarrow A$, $\underline{A \rightarrow CD}$,
 AC^+ using $Y = \{A, C, D\}$ i.e., $\underline{A \rightarrow CD}$
 E^+ using $Y = \{A, H\}$ i.e., $\underline{E \rightarrow AH}$, .

Step 2: To check whether FDs in Y can be inferred using FDs in X.

A^+ using $X = \{A, C, D\}$ i.e., $A \rightarrow C$, $\underline{AC \rightarrow D}$
 E^+ using $X = \{A, H\}$ i.e., $\underline{E \rightarrow AH}$.

Since all FDs can be obtained from X and vice versa, we conclude the X and Y are equivalent.

(*VVIP)Minimal Cover of Functional Dependencies(Canonical Cover)**

A set of functional dependencies F to be minimal if it satisfies the following conditions:

- Step1: Every dependency in F has a single attribute for its right-hand side.
- Step2: There should not be any redundant attributes on left hand side(LHS).
- Step3: There should not be any redundant functional dependencies.

Algorithm: Finding minimal cover F for a set of functional dependencies E.

Input: A set of Functional dependencies E.

1. Set $F := E$.
 2. Replace each functional dependency $X \rightarrow \{A_1, A_2, \dots, A_n\}$ in F by the n functional dependencies $X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n$.
 3. For each functional dependencies $X \rightarrow A$ in F
 for each attribute B that is an element X
 if $\{ \{F - \{X \rightarrow A\} \} \cup \{X - \{B\} \rightarrow A\} \}$ is equivalent to F
 then replace $X \rightarrow A$ with $(X - \{B\}) \rightarrow A$ in F.
 4. For each meaning functional dependency $X \rightarrow A$ in F
 if $\{F - \{X \rightarrow A\}\}$ is equivalent to F,
 then remove $X \rightarrow A$ from F.
- Step 2 places FDs in a canonical form for subsequent testing.
 - Step 3 constitutes removal of an extraneous attribute B contained in the left-hand side X of a functional dependency $X \rightarrow A$ from F when possible.

- Step 4 constitutes removal of a redundant functional dependency $x \rightarrow A$ from F when possible.

Example: Let the given set of FDs be $E: \{B \rightarrow A, D \rightarrow A, AB \rightarrow D\}$. We have to find the minimal cover of E .

Step 1: All above dependencies are in canonical form (that is, they have only one attribute on the right-hand side), so we have completed step 1 of Algorithm and can proceed to step 2.

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

Step 2: In step 2 we need to determine if $AB \rightarrow D$ has any redundant attribute on the left-hand side; Hence take the closure of A and B individually to check if we can determine the other attribute.

$A^+ = A$ (only A can be determined)

$B^+ = BA$ (**A can be determined using B**)

Hence we discard attribute A as we cannot determine B from it.

we get $B \rightarrow D$. 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 $\{B \rightarrow D, D \rightarrow A\}$.

Now check if the new FD is equivalent to old FD

old FD: $\{B \rightarrow A, D \rightarrow A, AB \rightarrow D\}$

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

Check Old FD can be inferred by new FD

$B^+ = BDA$ ($B \rightarrow D, D \rightarrow A$)

$D^+ = DA$ ($D \rightarrow A$)

$AB^+ = ABD$ ($B \rightarrow D$)

Check new FD can be inferred by old FD

$B^+ = ABD$ ($B \rightarrow A, AB \rightarrow D$)

$D^+ = DA$ ($D \rightarrow A$)

Since all new FDs can be obtained from old FD and vice versa, we conclude the new FD is equivalent to old FD.

Properties of Relational Decompositions

1. Relational Decompositions & Insufficiency of Normal forms

- A Universal relation schema $R = \{A_1, A_2, \dots, A_n\}$ includes all the attributes of the database. Universal relation assumption: every attribute name is unique
- The set F of functional dependencies that should hold on the attributes of R is

specified by the database designers.

- Using the functional dependencies, the algorithms decompose the universal relation schema R into a set of relation schemas $D = \{R_1, R_2, \dots, R_m\}$ that will become the relational database schema. D is called a decomposition of R .
- Each attribute in R will appear in at least one relation schema R_i in the decomposition so that no attributes are lost (Attribute Preserving Property)
- each individual relation R_i in the decomposition D be in BCNF or 3NF.

Attribute Preservation condition of a Decomposition

Each attribute in R will appear in at least one relation schema R_i in the decomposition so that no attributes are lost; formally, we have

$$\bigcup_{i=1}^m R_i = R$$

- Another goal of decomposition is to have each individual relation R_i in the decomposition D be in BCNF or 3NF.
- Additional properties of decomposition are needed to prevent from generating spurious tuples.

Example: If $R(ABCDE)$ after decomposition to $R_1(AB)$ and $R_2(CD)$ then $(R_1 \cup R_2 \dots R_n) = R$.

While decomposition none of the attribute should be lost. But in above example attribute E is missing after decomposition.

- **Additional properties of decomposition are needed to prevent from generating spurious tuples.**

2. Dependency Preservation Property

- Each functional dependency $X \rightarrow Y$ specified in F either appeared directly in one of the relation schemas R_i in the decomposition D or could be inferred from the dependencies that appear in some R_i .
- We want to preserve the dependencies because each dependency in F represents a constraint on the database.
- If one of the dependencies is not represented in some individual relation R_i of the decomposition, we cannot enforce this constraint by dealing with an individual relation
- It is not necessary that the exact dependencies specified in F appear themselves in individual relations of the decomposition D .
- It is sufficient that the union of the dependencies that hold on the individual relations in D be equivalent to F .

Points to remember:

1. Write down all the non-trivial dependencies present in the child table.
2. Check the dependency hold true or not.
 - Dependency can be directly present.
 - Else take the closure of the determinant w.r.t FD given in the question and check the possibility.
3. Take union of all the possible FD.
4. Check all the original dependencies are preserved are not w.r.t possible FD.

- Dependency can be directly present.
- Else the closure of the determinant w.r.t derived possible FD.

Example 1: Let the relation be $R(XYZ)$ and given set of FD are $X \rightarrow Y$, $Y \rightarrow Z$, $Z \rightarrow X$ and check whether it is dependency preserving decomposition or not.

First decomposition of given relation to child table as $R_1(XY)$ and $R_2(YZ)$.

Next check lossless or not for each child table,

1. XY can drive FDs as,

$X \rightarrow Y$, $Y \rightarrow X$

Now check these FDs are present in the original question FDs. $X \rightarrow Y$ is preserved directly but $Y \rightarrow X$ not preserved directly.

So check whether it is preserved indirectly by **finding closure of Y**

$Y^+ = YZ$ but X is not in the table discard it.

2. YZ can drive FDs as,

$Y \rightarrow Z$, $Z \rightarrow Y$

Now check these FDs are present in the original question FDs. $Y \rightarrow Z$ is preserved directly but $Z \rightarrow Y$ not preserved directly.

So check whether it is preserved indirectly by finding closure of Z

$Z^+ = ZXY$ but X is not in the table discard it.

Then, $Z^+ = ZXY$ So, $Z \rightarrow X$ preserved indirectly.

All the FDs are preserved as in original FDs from the given question. Therefore decomposition is Dependency preserving.

Example 2: Let the relation be $R(A, B, C, D)$ and given set of FD are $A \rightarrow B$, $C \rightarrow D$ and check whether it is dependency preserving decomposition or not.

First decomposition of given relation to child table as $R_1(AC)$ and $R_2(BD)$.

Next check lossless or not for each child table,

1. AC can drive FDs as,

Find closure for A and C using the given FDs.

$A^+ = A$, where $A \rightarrow A$ is trivial function and B is not part of child table so discard it. Hence we can't find any FD from this table.

$C^+ = C$, where $C \rightarrow C$ is trivial function and D is not a part of child table so discard it. Hence we can't find any FD from this table.

$F_1 = \{ \}$ empty set of FDs.

1. BD can drive FDs as,

Find closure for B and D using the given FDs.

$B^+ = B$, where $B \rightarrow B$ is trivial function. Hence we can't find any FD from this table.

$D^+ = D$, where $D \rightarrow D$ is trivial function. Hence we can't find any FD from this table.

$F_2 = \{\}$ empty set of FDs.

$F_1 \cup F_2 = \{\}$ is also empty set of FDs.

Therefore decomposition is not Dependency preserving.

Dependency Preservation Property

Let $R(ABCD)$ with Functional Dependencies

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

R is decomposed into $R_1(AB), R_2(BC), R_3(BD)$

$R_1(AB)$ $A \rightarrow B \checkmark$ $B \rightarrow A \times$ $B^+ = BCD$ (A is not coming) Check if dependency is preserved Take Union of all	$R_2(BC)$ $B \rightarrow C \checkmark$ $C \rightarrow B \checkmark$ $C^+ = CDB$	$R_3(BD)$ $B \rightarrow D \checkmark$ $D \rightarrow B \checkmark$ $B^+ = BCD$ Hence this is also Possible
---	--	---

Now check if the original Dependencies are present in the new

Original: $\{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow B\}$ New: $\{A \rightarrow B, B \rightarrow C, C \rightarrow B, B \rightarrow D, D \rightarrow B\}$

$C \rightarrow D$ not there $C^+ = \text{none}$ Hence in C^+ we are getting D
 $C^+ = CDB$ Hence $C \rightarrow D$ is also preserved

Lossless Join Property(Nonadditive)

- The nonadditive join property ensures that no spurious tuples result after the application of PROJECT and JOIN operations.
- The term lossy design refer to a design that represents a loss of information.
- If a decomposition does not have the lossless join property, we may get additional spurious tuples after the PROJECT (π) and NATURAL JOIN ($*$) operations are applied; these additional tuples represent erroneous or invalid information.

Example: Consider a given relation R.

R		
SNO	Sname	Branch
1	X	CS
2	X	EC

R1	
SNO	Sname
1	X
2	X

R2	
Sname	Branch
X	CS
X	EC

FD are given as, $\{SNO \rightarrow Branch\}$. Find the branch of student with SNO=1 and check for Lossless Join Property.


$D\{R1, R2\}$

```
SELECT R2.Branch
FROM R1 Natural Join R2
Where R1.SNO = 1;
```

So join $R1 \bowtie R2$

R1 \bowtie R2		
SNO	Sname	Branch
1	X	CS
1	X	EC
2	X	CS
2	X	EC

**Spurious
Tuples**

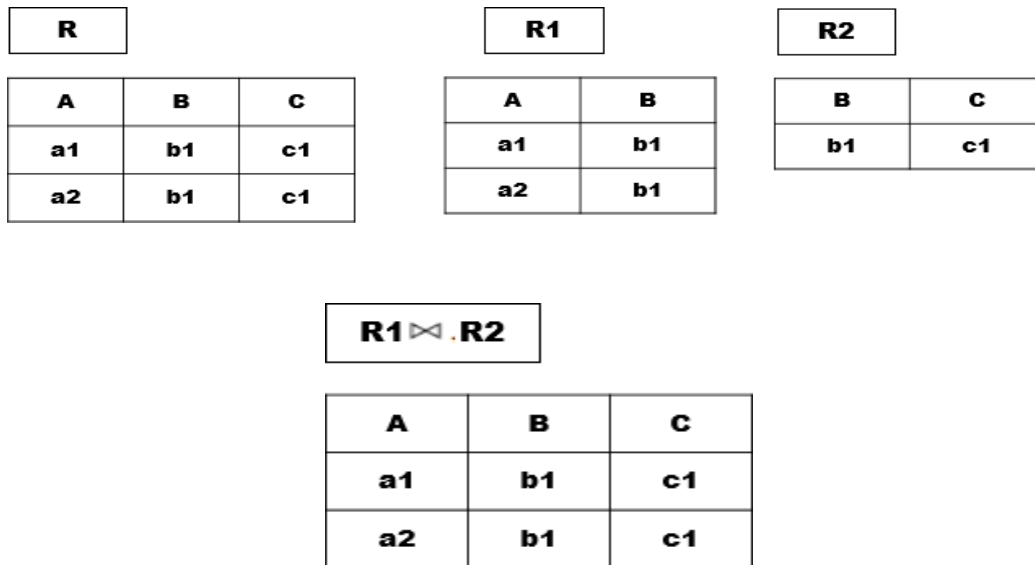


Output: This is lossy decomposition.

Branch
CS
EC

Example 2: Consider a given relation R and FD are given as, $\{B \rightarrow C\}$ and check for Lossless Join Property.

$D\{R1, R2\}$

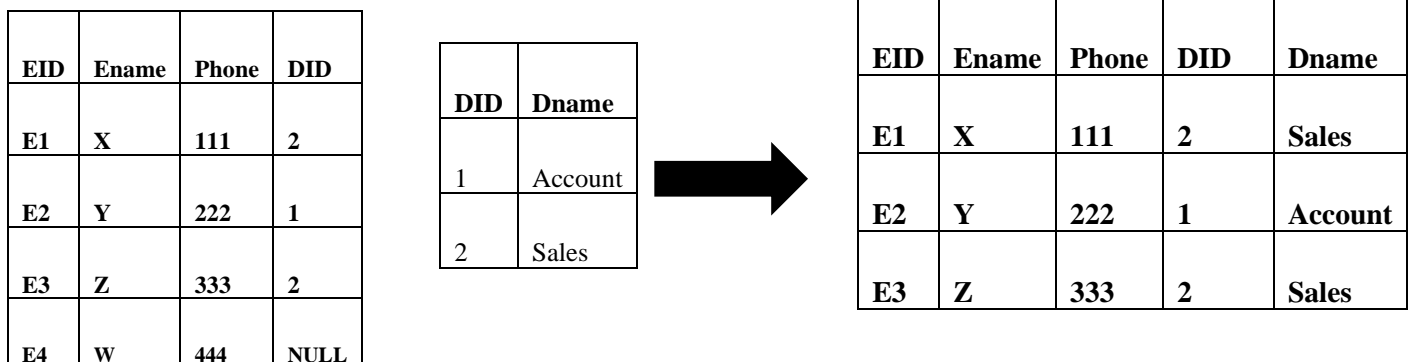


Therefore this decomposition is **lossless or Nonadditive join decomposition**.

If $B^+ = BC$ then, B is key or superkey and satisfy the conditions $(R1 \cap R2) \rightarrow R1$ (OR) $(R1 \cap R2) \rightarrow R2$.

Problems with NULL Values and Dangling Tuples

- when some tuples have NULL values for attributes that will be used to join individual relations in the decomposition that may lead to incomplete results.
- Newly hired employee who have not been assigned any department
- Suppose we want to join Employee and Department using natural join

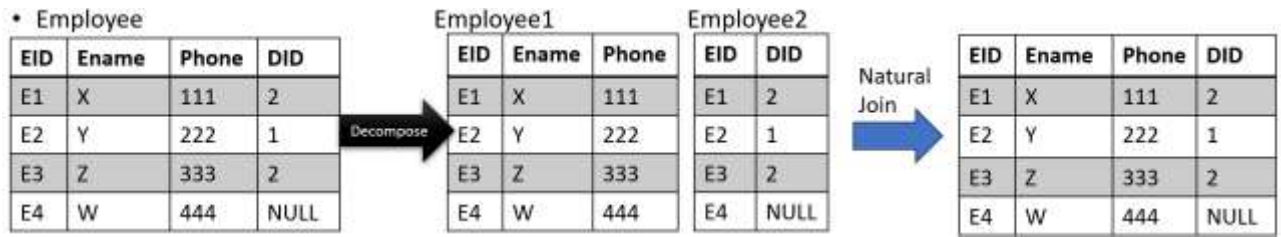


The information about the employee E4 is lost

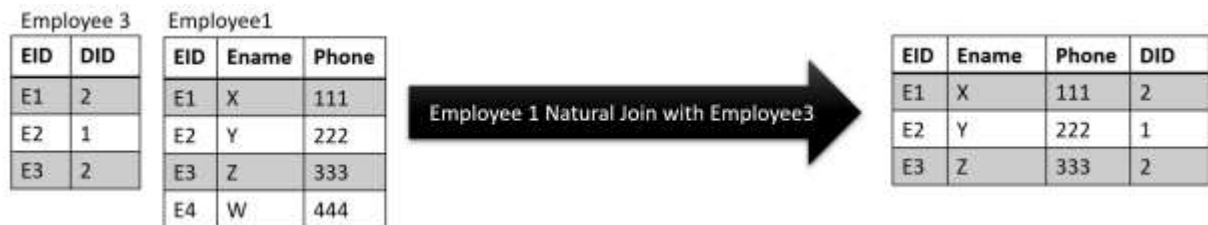
In general, whenever a relational database schema is designed in which two or more relations are interrelated via foreign keys, particular care must be devoted to watching for potential NULL values in foreign keys. This can cause unexpected loss of information in queries that involve joins on that foreign key.

Problems with Dangling Tuples

Are those tuples in the referencing relation that do not appear in the referenced relation



- But if we do not include tuples for which DID has null value in Employee 2



Domain-Key Normal Form

- takes into account all possible types of dependencies and constraints.
- A relation schema is said to be in DKNF if all constraints and dependencies that should hold on the valid relation states can be enforced simply by enforcing the domain constraints and key constraints on the relation
- For a relation in DKNF, it becomes very straightforward to enforce all database constraints by simply checking that **each attribute value in a tuple is of the appropriate domain and that every key constraint is enforced**