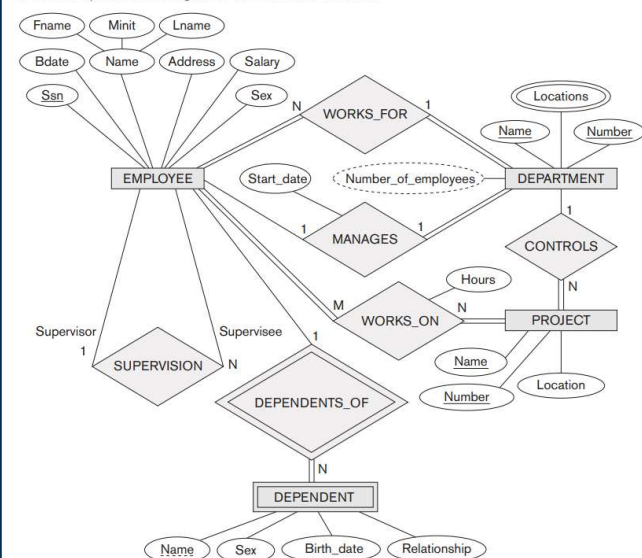


Unit-05: Relational Database Design

- Relational Database Design Using ER-to-Relational Mapping;
- Informal Design Guidelines for Relational Schemas;
- Functional Dependencies;
- Normal Forms Based on Primary Keys;
- General Definitions of Second and Third Normal Forms; Boyce-Codd Normal Form;
- Multivalued Dependency and Fourth Normal Form;
- Properties of Relational Decomposition

Unit-05: Relational Database Design

The ER conceptual schema diagram for the COMPANY database.



The COMPANY ER schema and the corresponding COMPANY relational database schema are shown in Figure.

The mapping will create tables with simple single valued attributes. The relational model constraints which include primary keys, unique keys (if any), and referential integrity constraints on the relations, will also be specified in the mapping results.

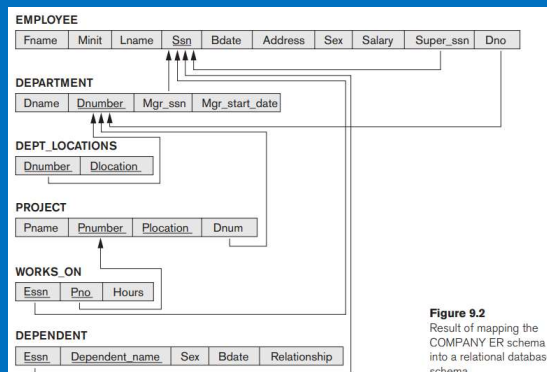


Figure 9.2
Result of mapping the
COMPANY ER schema
into a relational database
schema.

Unit - 5

The steps of an algorithm for **ER-to-Relational mapping** are as:

Step 1: Mapping of Regular Entity (Strong Entity) Types.

Step 2: Mapping of Weak Entity Types.

Step 3: Mapping of Binary 1:1 Relationship Types.

There are three possible approaches for ER-to-relational mapping. (1) the **foreign key approach**, (2) the **merged relationship approach**, and (3) the **crossreference or relationship relation approach (lookup table)**.

Step 4: Mapping of Binary 1:N Relationship Types.

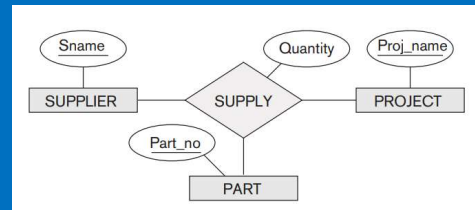
(1) the foreign key approach, and (2) the crossreference or relationship relation approach.

Step 5: Mapping of Binary M:N Relationship Types.

(1) the crossreference or relationship relation approach.

Step 6: Mapping of Multivalued Attributes.

Step 7: Mapping of N-ary Relationship Types.



Unit - 5

Figure 9.3

Illustration of some mapping steps.

(a) *Entity* relations after step 1.

(b) Additional *weak entity* relation after step 2.

(c) *Relationship* relations after step 5.

(d) Relation representing multivalued attribute after step 6.

(a) **EMPLOYEE**

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary
-------	-------	-------	------------	-------	---------	-----	--------

DEPARTMENT

Dname	<u>Dnumber</u>
-------	----------------

PROJECT

Pname	<u>Pnumber</u>	Plocation
-------	----------------	-----------

(b) **DEPENDENT**

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------

(c) **WORKS_ON**

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

(d) **DEPT_LOCATIONS**

<u>Dnumber</u>	<u>Dlocation</u>
----------------	------------------

Unit - 5

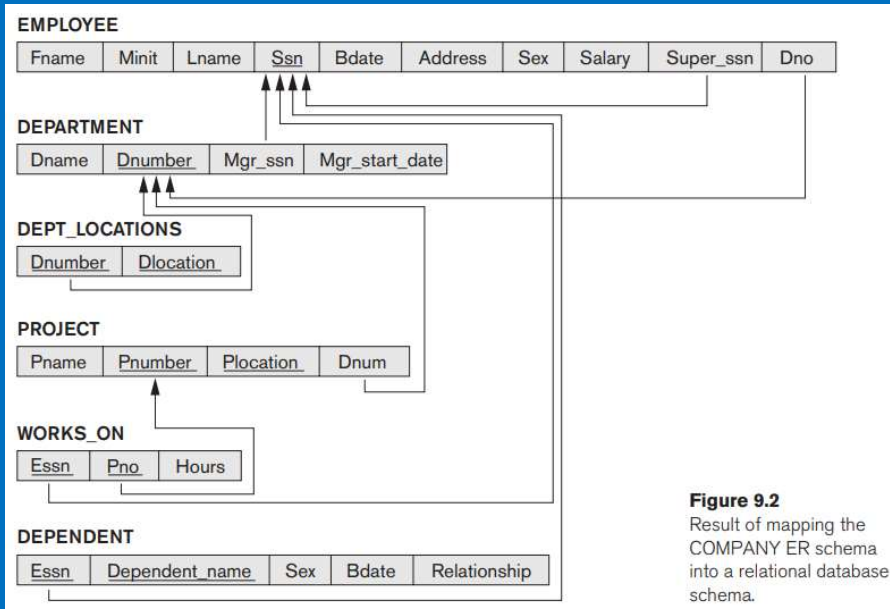
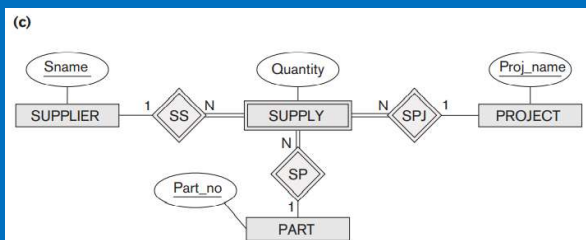
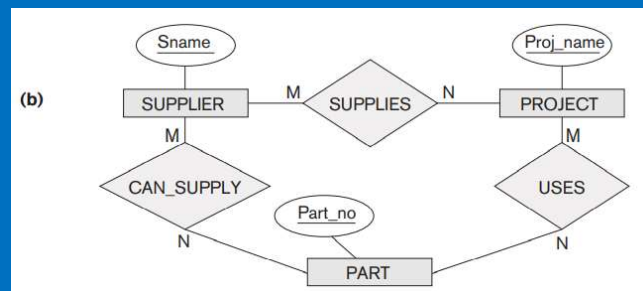
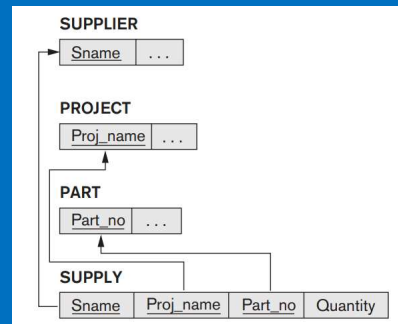
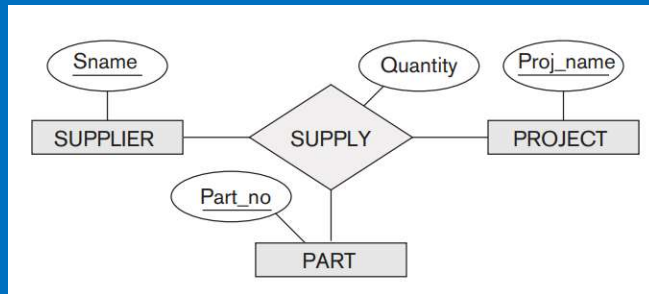


Figure 9.2

Result of mapping the COMPANY ER schema into a relational database schema.

Unit - 5



Theory of Database Design

Informal Design Guidelines for Relational Schemas

Two Approach of Database Design:

1. Top-down
2. Bottom-up

Top-down design methodology would start with a number of groupings of attributes into relations that have already been obtained from conceptual design and mapping activities i.e. using Data Models like E-R Diagram. **Design by analysis** is then applied to the relations individually and collectively to further decomposition until all desirable properties are met.

Theory of Database Design

Two Approach of Database Design:

- 1. Top-down
- 2. Bottom-up

In contrast, a **bottom-up design** methodology would consider the basic relations among individual attributes as the starting point, and it would use those to build up relations.

This approach is not very popular in practice and suffers from the problem of collecting a large number of binary attribute relationships as the starting point. This approach is also called **design by synthesis**.

Theory of Database Design

In **Bottom Up Approach**, four informal measures of quality for relation schema design are:

1. Semantics of the attributes.
2. Reducing the redundant values in tuples
3. Reducing the null values in tuples.
4. Disallowing the possibility of generating **spurious** tuples.

These measures are not always independent of one another.

Example: EMP_DEPT

Ename	SSN	Bdate	Address	Dnumber	Dname	DMGRSSN

Problems in Insertion, Deletion and Modification operation.

EMP_DEPT

Redundancy						
Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

Three types of anomalies are insertion anomalies, deletion anomalies, and modification anomalies.

Insertion anomalies:

Insertion anomalies can be differentiated into two types.

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 (if the employee does not work for a department as yet).
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.

Deletion Anomalies:

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 inadvertently from the database.

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.

Redundancy						
EMP_DEPT						
Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	281 Berry, Bellare, TX	4	Administration	987654321
Narayan, Ramoth K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

Theory of Database Design

Functional dependency:

An attribute or set of attributes X is said to functionally determine another attribute Y if and only if each X value is associated with at most one Y value. We call X determinant set and Y a dependent set.

So if we are given the value of X we can determine the value of Y.

Theory of Database Design

Functional dependency:

So if we are given the value of X we can determine the value of Y.

$S\# \rightarrow CITY$

$\{S\#, P\#\} \rightarrow QTY$

$S\# \rightarrow QTY$

S#	CITY	P#	QTY
S1	London	P1	100
S1	London	P2	100
S2	Paris	P1	200
S2	Paris	P2	200
S3	Paris	P2	300
S4	London	P2	400
S4	London	P4	400

Theory of Database Design

Types of Functional Dependencies:

1. Trivial Dependency
2. Non Trivial Dependency
3. Multivalued Dependency
4. Join Dependency
5. Transitive dependency

Theory of Database Design

Trivial Dependency:

An FD is trivial if and only if the right-hand side is a subset (not necessarily a proper subset) of the left hand side.

$\{S\#, P\# \} \rightarrow S\#$

A functional dependency FD: is trivial iff Y is a subset of X.

As the name implies, trivial dependencies are not very interesting in practice. We are usually more interested in non-trivial dependencies because these are the ones that correspond to “genuine” integrity constraints.

Theory of Database Design

Multivalued Dependency:

Smith have two dependent (john & Anna) and working in both project X and Y. This information can be represented in table form as

EmployeeNAME	ProjectNAME	DependentNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

Theory of Database Design

Multivalued Dependency:

If we have two or more multivalued independent attributes in the same relation schema, we get into a problem of having to repeat every value of one of the attributes with every value of the other attribute to keep the relation state consistent and to maintain the independence among the attribute involved. This constraint is specified by a multivalued dependency.

Whenever two independent 1:N relations A:B and A:C are mixed in the same relation, a MVD may arise.

Theory of Database Design

Join Dependency:

Let R be a relation and let A,B,...,Z be arbitrary subsets of the set of attributes R. Then we say that R satisfies the join Dependency, $JD^*(A,B,...,Z)$ if and only if R is equal to the join of its projections A,B,...,Z.

SPJ		
Supplier(S#)	Project(P#)	Part(J#)
S1	P1	J2
S1	P2	J1
S2	P1	J1
S1	P1	J1

SP	
S#	P#
S1	P1
S1	P2
S2	P1

PJ	
P#	J#
P1	J2
P2	J1
P1	J1

JS	
J#	S#
J2	S1
J1	S1
J1	S2

Theory of Database Design

Join Dependency:

SPJ

Supplier(S#)	Project(P#)	Part(J#)
S1	P1	J2
S1	P2	J1
S2	P1	J1
S1	P1	J1

JS

J#	S#
J2	S1
J1	S1
J1	S2

SP

S#	P#
S1	P1
S1	P2
S2	P1

PJ

P#	J#
P1	J2
P2	J1
P1	J1

SP * PJ = SPJ

Supplier(S#)	Project(P#)	Part(J#)
S1	P1	J2
S1	P2	J1
S2	P1	J1
S2	P1	J2
S1	P1	J1

Spurious Tuple

Theory of Database Design

Join Dependency:

JS

J#	S#
J2	S1
J1	S1
J1	S2

SP * PJ = SPJ

Supplier(S#)	Project(P#)	Part(J#)
S1	P1	J2
S1	P2	J1
S2	P1	J1
S2	P1	J2
S1	P1	J1

SPJ * JS = Gives initial Table

Supplier(S#)	Project(P#)	Part(J#)
S1	P1	J2
S1	P2	J1
S2	P1	J1
S1	P1	J1

Original Table

Theory of Database Design

Non-Loss Decomposition:

Normalization is a process of breaking down or decomposing a given relation into other relations (more than one relation) and the decomposition is required to be reversible so that no information is lost in the process. So we are only interested in decomposition that are non-loss. The decomposition of relation is non-loss, intimately bound up with the concept of functional dependency.

Theory of Database Design

Non-Loss Decomposition:

S#	STATUS	CITY
S3	30	Paris
S5	30	Athens

a)

SST	
S#	STATUS
S3	30
S5	30

SC	
S#	CITY
S3	Paris
S5	Athens

Non-Loss and reversible

b)

SST	
S#	STATUS
S3	30
S5	30

STC	
STATUS	CITY
30	Paris
30	Athens

Not Non-Loss and Non reversible

Theory of Database Design

Transitive dependency:

A transitive is a type of functional dependency which happens when it is indirectly formed by two functional dependencies.

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Alibaba	Jack Ma	54

{Company} → {CEO} (if we know the company, we know its CEO's name)

{CEO} → {Age} If we know the CEO, we know the Age

Therefore according to the rule of transitive dependency:

{Company} → {Age} should hold, that makes sense because if we know the company name, we can know his age.

Theory of Database Design

- Database Normalization
- Normal Forms
 1. First Normal Form (1NF)
 2. Second Normal Form (2NF)
 3. Third Normal Form (3NF)
 4. Boyce Code Normal Form (BCNF)
 5. Fourth Normal Form (4NF)

Theory of Database Design

- **Database Normalization**

Database normalization is a technique of splitting relational database's structure into multiple tables so that it reduces the redundancies in RDBS.

In the relational model, formal methods exist for quantifying "how normalized" a database is. These classifications are called **Normal Forms** (or NF), and there are algorithms for converting a given database between them.

Theory of Database Design

Normal Forms

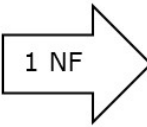
- Edgar F. Codd originally established three normal forms 1NF, 2NF and 3NF.
- There are now others that are generally accepted but 3NF is widely considered to be sufficient for many practical applications.
- Most tables when reaching 3NF are also in BCNF. 4NF and 5NF and 6NF only applied to temporal databases.

Theory of Database Design

First Normal Form (1NF):

A relation R is in first normal form (1NF) if and only if all underlying attributes contains atomic values only.

S#	PQ	
	P#	QTY
S1	P1	300
	P2	200
	P3	400
S2	P1	300
	P2	400
S3	P3	200



S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S2	P1	300
S2	P2	400
S3	P3	200

Theory of Database Design

Second Normal Form (2NF):

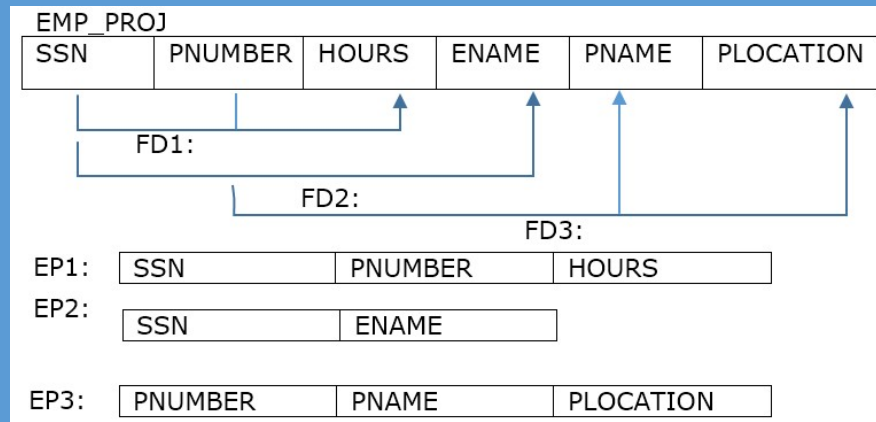
A relation R is in second normal form (2NF) if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key.

2NF must satisfy the additional constraint that all non-key attributes must be functionally dependent on the whole of each relation key. Relations in 2NF can not have non-key attributes that depend on only part of a relation key.

Theory of Database Design

Second Normal Form (2NF):

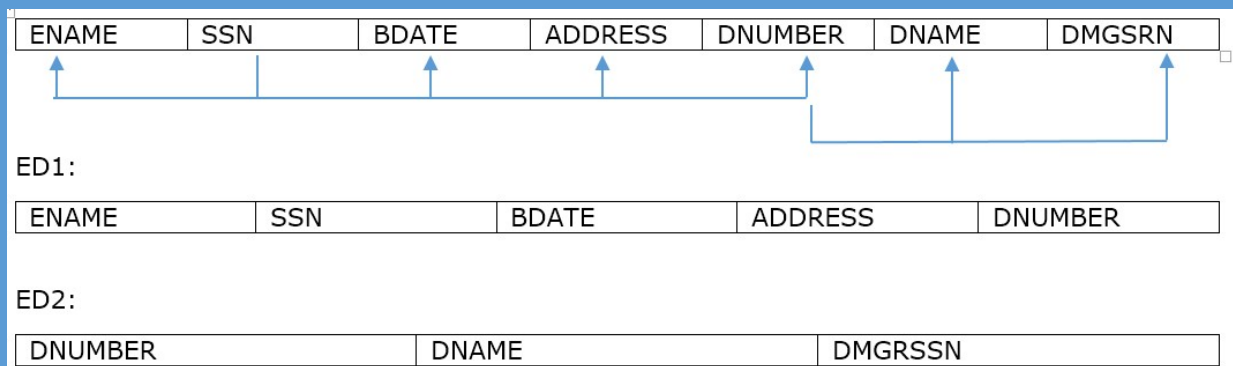
A relation R is in second normal form (2NF) if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key.



Theory of Database Design

Third Normal Form (3NF):

A relation R is in third normal form (3NF) if and only if it is in 2NF and there must be no dependencies between non-key attributes.



Theory of Database Design

Third Normal Form (3NF):

- Third normal form (3NF) is based on the concept of **transitive dependency**. A functional dependency $X \rightarrow Y$ 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 $X \rightarrow Z$ and $Z \rightarrow Y$ hold.

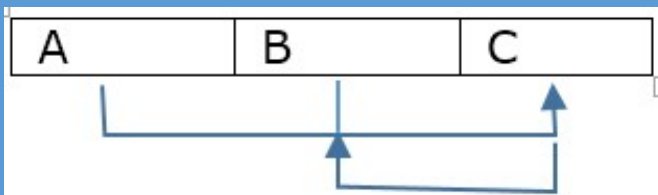
A relation R is in third normal form (3NF) if and only if it is in 2NF and there must be no dependencies between non-key attributes.

Theory of Database Design

Boyce Code Normal Form (BCNF):

BCNF is like 3NF but stricter than 3NF. Every relation in BCNF is also in 3NF, however a relation in 3NF is not necessary in BCNF.

A relation R is in BCNF if for every functional dependency ($X \rightarrow Y$) between any relation attributes, the attributes on the left hand side (that is X) are a relation (primary) key.



Example:

STUDENT	COURSE	INSTRUCTOR
Narayan	Database	AAA
Ram	Database	BBB
Ram	Operating	CCC
Ram	Theory	DDD
Shyam	Database	AAA
Shyam	Operating	EEE

FD1: {STUDENT, COURSE} -> INSTRUCTOR

FD2: INSTRUCTOR -> COURSE

Theory of Database Design

Fourth normal form (4NF):

Fourth normal form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key. It builds on the first three normal forms (1NF, 2NF and 3NF) and the Boyce-Codd Normal Form (BCNF). It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.

Theory of Database Design

Fourth normal form (4NF):

A relation R is in 4NF if and only if the following conditions are satisfied:

- It should be in the Boyce-Codd Normal Form (BCNF).
- the table should not have any Multi-valued Dependency.

A table with a multivalued dependency violates the normalization standard of Fourth Normal Form (4NF) because it creates unnecessary redundancies and can contribute to inconsistent data. To bring this up to 4NF, it is necessary to break this information into two tables.

Theory of Database Design

Example:

Smith have two dependent (john & Anna) and working in both project X and Y. This information can be represented in table form as

EmployeeNAME	ProjectNAME	DependentNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

Theory of Database Design

Multivalued Dependency:

If we have two or more multivalued independent attributes in the same relation schema, we get into a problem of having to repeat every value of one of the attributes with every value of the other attribute to keep the relation state consistent and to maintain the independence among the attribute involved. This constraint is specified by a multivalued dependency.

Whenever two independent 1:N relations A:B and A:C are mixed in the same relation, a MVD may arise. Create 2 tables as
 $A \twoheadrightarrow B$ and $A \twoheadrightarrow C$

Inference Rules for Functional Dependencies

We know the schema design based on the functional dependencies. Numerous other functional dependencies hold in all legal relation instances among sets of attributes that can be derived from and satisfy the dependencies in F . Those other dependencies can be inferred or deduced from the FDs in F . We call them as **inferred or implied functional dependencies**.

In real life, it is impossible to specify all possible functional dependencies for a given situation. For example, if each department has one manager, so that Dept_no uniquely determines Mgr_ssn ($\text{Dept_no} \rightarrow \text{Mgr_ssn}$), and a manager has a unique phone number called Mgr_phone ($\text{Mgr_ssn} \rightarrow \text{Mgr_phone}$), then these two dependencies together imply that $\text{Dept_no} \rightarrow \text{Mgr_phone}$. This is an inferred or implied FD and need not be explicitly stated in addition to the two given FDs.

Therefore, it is useful to define a concept called closure formally that includes all possible dependencies that can be inferred from the given set F .

Definition. Formally, the set of all dependencies that include F as well as all dependencies that can be inferred from F is called the closure of F ; it is denoted by F^+ .

Inference Rules for Functional Dependencies

The closure F^+ of F is the set of all functional dependencies that can be inferred from F . To determine a systematic way to infer dependencies, we must discover a set of inference rules that can be used to infer new dependencies from a given set of dependencies.

We discuss some of these inference rules next. We use the notation $F \models X \rightarrow Y$ to denote that the functional dependency $X \rightarrow Y$ is inferred from the set of functional dependencies F .

Let F be a set of FDs. The closure of F is the set of all FDs logically implied by F . We denote the **closure of F by F^+** . Given F , we can compute F^+ directly from the formal definition of FD.

The technique used to find F^+ is based on some axioms or of **inference rules** for FDs.

Inference Rules for Functional Dependencies

Inference Rules:

Armstrong's axioms are a set of inference rules used to infer all the functional dependencies on a relational database. They were developed by William W. Armstrong. The following describes what will be used, in terms of notation, to explain these axioms.

1. Reflexivity rule
2. Augmentation rule
3. Transitivity rule
4. Union rule
5. Decomposition rule
6. Pseudo transitivity rule

Inference Rules for Functional Dependencies

Inference Rules:

Let $R(U)$ be a relation scheme over the set of attributes U . We will use the letters X, Y, Z to represent any subset of and, for short, the union of two sets of attributes, instead of the usual $X \cup Y$.

Reflexivity rule:

This axiom says, if Y is a subset of X , then X determines Y

reflexivity if $Y \subseteq X$ then $X \rightarrow Y$

Inference Rules for Functional Dependencies

Inference Rules:

Augmentation rule:

The axiom of augmentation, also known as a partial dependency, says if X determines Y, then XZ determines YZ for any Z.

$$\text{if } X \rightarrow Y \text{ then } XZ \rightarrow YZ$$

The axiom of augmentation says that every non-key attribute must be fully dependent on the PK.

Inference Rules for Functional Dependencies

Inference Rules:

Transitivity rule:

The axiom of transitivity says if X determines Y, and Y determines Z, then X must also determine Z.

$$\text{If } X \rightarrow Y \text{ and } Y \rightarrow Z, \text{ then } X \rightarrow Z$$

Inference Rules for Functional Dependencies

Inference Rules:

Union rule:

This rule suggests that if two tables are separate, and the PK is the same, you may want to consider putting them together. It states that if X determines Y and X determines Z then X must also determine Y and Z .

$$\text{If } X \rightarrow Y \text{ and } X \rightarrow Z \text{ then } X \rightarrow YZ$$

Inference Rules for Functional Dependencies

Inference Rules:

Decomposition

Decomposition is the reverse of the Union rule. If you have a table that appears to contain two entities that are determined by the same PK, consider breaking them up into two tables. This rule states that if X determines Y and Z , then X determines Y and X determines Z separately.

$$\text{If } X \rightarrow YZ \text{ then } X \rightarrow Y \text{ and } X \rightarrow Z$$

Inference Rules for Functional Dependencies

Inference Rules:

Pseudo transitivity rule:

This rule states that if X determines Y (i.e. $X \rightarrow Y$) and $ZY \rightarrow P$ holds, then XZ determines P (i.e. $XZ \rightarrow P$)

Inference Rules for Functional Dependencies

Example: Let R be a relation: $R = (A, B, C, D, E, F)$ having the functional dependencies:

- a) $A \rightarrow BC$
- b) $B \rightarrow E$
- c) $C \rightarrow EF$
- d) $E \rightarrow CF$

Find the closure of $\{A, B\}$ over the given set of FDs.

Sol: We start with initial closure for $\{A, B\}$ and then proceed as follows:

Inference Rules for Functional Dependencies

Example:

Sol: We start with initial closure for $\{A,B\}$ and then proceed as follows:

$[AB]^+$	$= \{A,B\}$	
	$\{A,B,C\}$	(AS $A \rightarrow BC$)
	$\{A,B,C,E\}$	(AS $B \rightarrow E$)
	$\{A,B,C,E,F\}$	(AS $C \rightarrow EF$)
	$\{A,B,C,E,F\}$	(AS $E \rightarrow CF$)

So $[AB]^+ = \{A,B,C,E,F\}$

Equivalence of Sets of Functional Dependencies

Two sets of functional dependencies are equivalent or not is determined as follows:

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.

Equivalence of Sets of Functional Dependencies

<https://www.youtube.com/watch?v=eIXC6NfKno4>

Example-1: Let us take an example to show the relationship between two FD sets. A relation $R(A,B,C,D)$ having two FD sets

$FD1 = \{A \rightarrow B, B \rightarrow C, AB \rightarrow D\}$ and $FD2 = \{A \rightarrow B, B \rightarrow C, A \rightarrow C, A \rightarrow D\}$

Solution:

Step 1. Checking whether all FDs of $FD1$ is present in $FD2$

- $A \rightarrow B$ in set $FD1$ is present in set $FD2$.
- $B \rightarrow C$ in set $FD1$ is also present in set $FD2$.
- $AB \rightarrow D$ is present in set $FD1$ 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 $AB \rightarrow D$ will also hold in set $FD2$.
- As all FDs in set $FD1$ also hold in set $FD2$, $FD2$ covers $FD1$ is true.

Equivalence of Sets of Functional Dependencies

<https://www.geeksforgeeks.org/equivalence-of-functional-dependencies-sets/>

$FD1 = \{A \rightarrow B, B \rightarrow C, AB \rightarrow D\}$ and $FD2 = \{A \rightarrow B, B \rightarrow C, A \rightarrow C, A \rightarrow D\}$

Step 2. Checking whether all FDs of $FD2$ are present in $FD1$

- $A \rightarrow B$ in set $FD2$ is present in set $FD1$.
- $B \rightarrow C$ in set $FD2$ is also present in set $FD1$.
- $A \rightarrow C$ 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 $A \rightarrow C$ will also hold in set $FD1$.
- $A \rightarrow D$ 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 $A \rightarrow D$ will also hold in set $FD1$.
- As all FDs in set $FD2$ also hold in set $FD1$, $FD1$ covers $FD2$ is true.

Equivalence of Sets of Functional Dependencies

<https://www.geeksforgeeks.org/equivalence-of-functional-dependencies-sets/>

FD1 = {A→B, B→C, AB→D} and **FD2** = {A→B, B→C, A→C, A→D}

Step 2. Checking whether all FDs of FD2 are present in FD1

- 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 covers FD2 is true.

Step 3. As FD2 covers FD1 and FD1 covers FD2 both are true FD2 = FD1 is true. These two FD sets are semantically equivalent.

Equivalence of Sets of Functional Dependencies

Example-2: Let us take another example to show the relationship between two FD sets. A relation R2(A,B,C,D) having two FD sets

FD1 = {A→B, B→C, A→C} and **FD2** = {A→B, B→C, A→D} .

Step 1. Checking whether all FDs of FD1 is present in FD2

- A→B in set FD1 is present in set FD2.
- B→C in set FD1 is also present in set FD2.
- A→C is present in FD1 but not directly in FD2 but we will check whether we can derive it or not. For set FD2, (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 FD2.

As all FDs in set FD1 also hold in set FD2, FD2 ⊇ FD1 is true.

Equivalence of Sets of Functional Dependencies

FD1 = {A→B, B→C, A→C} and FD2 = {A→B, B→C, A→D}.

Step 2. Checking whether all FDs of FD2 are present in FD1

- A→B in set FD2 is present in set FD1.,
- B→C in set FD2 is also present 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}. It means that A can't functionally determine D. SO A→D will not hold in FD1.

As all FDs in set FD2 do not hold in set FD1, FD2 not cover FD1.

Step 3. In this case, FD2 cover FD1 and FD2 not cover (⊄) FD1, these two FD sets are not semantically equivalent.

Minimal Sets of Functional Dependencies

We applied inference rules to expand on a set F of FDs to arrive at F⁺, its closure, it is possible to think in the opposite direction to see if we could shrink or reduce the set F to its minimal form so that the minimal set is still equivalent to the original set F.

A **minimal cover** of a set of functional dependencies E is a set of functional dependencies F that satisfies the property that every dependency in E is in the closure F⁺ of F.

We will use the concept of an **extraneous attribute** in a functional dependency for defining the minimum cover.

Minimal Sets of Functional Dependencies

Definition: An attribute in a functional dependency is considered an extraneous attribute if we can remove it without changing the closure of the set of dependencies.

Formally, given F , the set of functional dependencies, and a functional dependency $X \rightarrow A$ in F , attribute Y is extraneous in X if $Y \subset X$, and F logically implies $(F : (X \rightarrow A) \cup \{ (X - Y) \rightarrow A \})$.

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 .

Minimal Sets of Functional Dependencies

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.

Condition 1 just represents every dependency in a canonical form with a single attribute on the right-hand side, and it is a preparatory step before we can evaluate if conditions 2 and 3 are met.

Conditions 2 and 3 ensure that there are no redundancies in the dependencies either by having redundant attributes (referred to as extraneous attributes) on the left-hand side of a dependency (Condition 2) or by having a dependency that can be inferred from the remaining FDs in F (Condition 3).

Minimal Sets of Functional Dependencies

Example 1: 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 and can proceed to step 2.

Step 2: We need to determine if $AB \rightarrow D$ has any redundant (extraneous) attribute on the left-hand side; that is, can it be replaced by $B \rightarrow D$ or $A \rightarrow D$?

➤ Since $B \rightarrow A$, by augmenting with B on both sides, 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.

Minimal Sets of Functional Dependencies

➤ In 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 $F: \{B \rightarrow D, D \rightarrow A\}$. The reader can verify that the original set F can be inferred from E ; in other words, the two sets F and E are equivalent.