
Chapter Four

Logical Database Design

The whole purpose of the data base design is to create an accurate representation of the data, the relationship between the data and the business constraints pertinent to that organization. Therefore, one can use one or more technique to design a data base. One such a technique was the E-R model. In this chapter we use another technique known as “Normalization” with a different emphasis to the database design---- defines the structure of a database with a specific data model.

Logical design is the process of constructing a model of the information used in an enterprise based on a specific data model (e.g. relational, hierarchical or network or object), but independent of a particular DBMS and other physical considerations.

The focus in logical database design is the *Normalization Process*

- ❖ Normalization process
 - Collection of Rules (Tests) to be applied on relations to obtain the minimal, non redundant set or attributes.
 - Discover new entities in the process
 - Revise attributes based on the rules and the discovered Entities
 - Works by examining the relationship between attributes known as functional dependency.

The purpose of normalization is to find the suitable set of relations that supports the data requirements of an enterprise.

A suitable set of relations has the following characteristics;

- *Minimal* number of attributes to support the data requirements of the enterprise
 - *Attributes* with close logical relationship (functional dependency) should be placed in the same relation.
 - *Minimal redundancy* with each attribute represented only once with the exception of the attributes which form the whole or part of the foreign key, which are used for joining of related tables.
-

The first step before applying the rules in relational data model is converting the conceptual design to a form suitable for relational logical model, which is in a form of tables.

Converting ER Diagram to Relational Tables

Three basic rules to convert ER into tables or relations:

Rule 1: Entity Names will automatically be table names

Rule 2: Mapping of attributes: attributes will be columns of the respective tables.

- ✓ Atomic or single-valued or derived or stored attributes will be columns
- ✓ Composite attributes: the parent attribute will be ignored and the decomposed attributes (child attributes) will be columns of the table.
- ✓ Multi-valued attributes: will be mapped to a new table where the primary key of the main table will be posted for cross referencing.

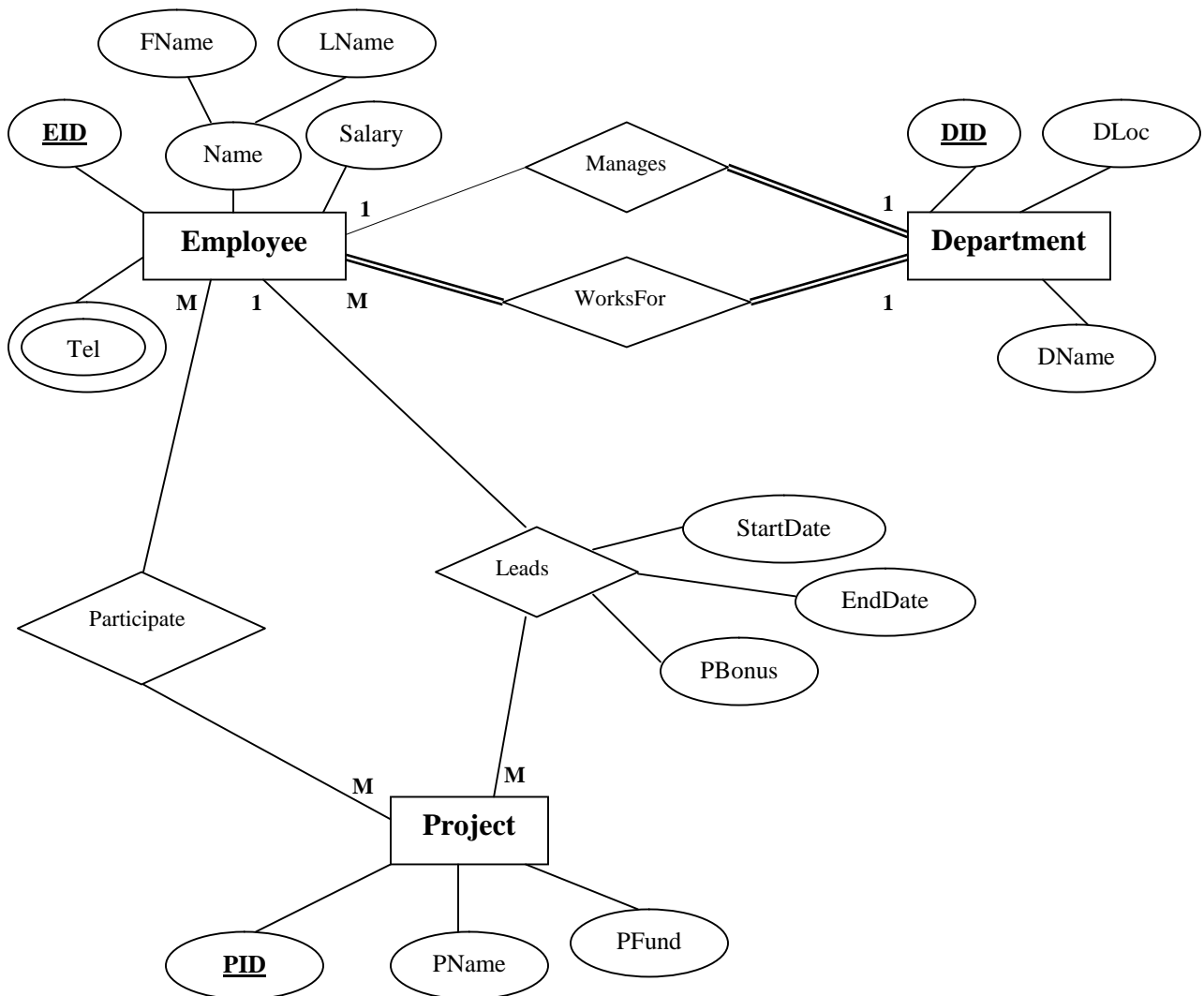
Rule 3: Relationships: relationship will be mapped by using a foreign key attribute. Foreign key is a primary or candidate key of one relation used to create association between tables.

- ✓ ***For a relationship with One-to-One Cardinality:*** post the primary or candidate key of one of the table into the other as a foreign key. In cases where one entity is having partial participation on the relationship, it is recommended to post the candidate key of the partial participants to the total participant so as to save some memory location due to null values on the foreign key attribute. E.g.: for a relationship between Employee and Department where employee manages a department, the cardinality is one-to-one as one employee will manage only one department and one department will have one manager. here the PK of the Employee can be posted to the Department or the PK of the Department can be posted to the Employee. But the Employee is having partial participation on the relationship "Manages" as not all employees are managers of departments. thus, even though both way is possible, it is recommended to post the primary key of the employee to the Department table as a foreign key.
 - ✓ ***For a relationship with One-to-Many Cardinality:*** Post the primary key or candidate key from the "one" side as a foreign key attribute to the "many" side. E.g.: For a relationship called "Belongs To" between Employee (Many) and Department (One) the primary or candidate key of the one side which is Department should be posted to the many side which is Employee table.
-

-
- ✓ *For a relationship with Many-to-Many Cardinality:* for relationships having many to many cardinality, one has to create a new table (which is the associative entity) and post primary key or candidate key from the participant entities as foreign key attributes in the new table along with some additional attributes (if applicable). The same approach should be used for relationships with degree greater than binary.
 - ✓ *For a relationship having Associative Entity property:* in cases where the relationship has its own attributes (associative entity), one has to create a new table for the associative entity and post primary key or candidate key from the participating entities as foreign key attributes in the new table.
-

Example to illustrate the major rules in mapping ER to relational schema:

The following ER has been designed to represent the requirement of an organization to capture Employee Department and Project information. And Employee works for department where an employee might be assigned to manage a department. Employees might participate on different projects within the organization. An employee might as well be assigned to lead a project where the starting and ending date of his/her project leadership and bonus will be registered.



After we have drawn the ER diagram, the next thing is to map the ER into relational schema so as the rules of the relational data model can be tested for each relational schema. The mapping can be done for the entities followed by relationships based on the rule of mapping. the mapping has been done as follows.

✓ Mapping EMPLOYEE Entity:

There will be *Employee* table with *EID*, *Salary*, *FName* and *LName* being the columns. The composite attribute Name will be ignored as its decomposed attributes (*FName* and *LName*) are columns in the Employee Table. The *Tel* attribute will be a new table as it is multi-valued.

Employee

<u>EID</u>	FName	LName	Salary
------------	-------	-------	--------

Telephone

<u>EID</u>	<u>Tel</u>
------------	------------

✓ Mapping DEPARTMENT Entity:

There will be *Department* table with *DID*, *DName*, and *DLoc* being the columns.

Department

<u>DID</u>	DName	DLoc
------------	-------	------

✓ Mapping PROJECT Entity:

There will be *Project* table with *PID*, *PName*, and *PFund* being the columns.

Project

<u>PID</u>	PName	PFund
------------	-------	-------

✓ Mapping the MANAGES Relationship:

As the relationship is having one-to-one cardinality, the PK or CK of one of the table can be posted into the other. But based on the recommendation, the Pk or CK of the partial participant (Employee) should be posted to the total participants (Department). This will require adding the PK of Employee (*EID*) in the Department Table as a foreign key. We can give the foreign key another name which is *MEID* to mean "managers employee id". this will affect the degree of the Department table.

Department

<u>DID</u>	DName	DLoc	MEID
------------	-------	------	------

✓ Mapping the WORKSFOR Relationship:

As the relationship is having one-to-many cardinality, the PK or CK of the "One" side (PK or CK of Department table) should be posted to the many side (Employee table). This will require adding the PK of Department (*DID*) in the Employee Table as a foreign key. We can give the foreign key another name which is *EDID* to mean "Employee's Department id". this will affect the degree of the Employee table.

Employee

<u>EID</u>	FName	LName	Salary	EDID
------------	-------	-------	--------	------

✓ **Mapping the PARTICIPATES Relationship:**

As the relationship is having many-to-many cardinality, we need to create a new table and post the PK or CK of the Employee and Project table into the new table. We can give a descriptive new name for the new table like Emp_Partc_Project to mean "Employee participate in a project".

Emp_Partc_Project

<u>EID</u>	<u>PID</u>
------------	------------

✓ **Mapping the LEADS Relationship:**

As the relationship is associative entity, we are supposed to create a table for the associative entity where the PK of Employee and Project tables will be posted in the new table as a foreign key. The new table will have the attributes of the associative entity as columns. We can give a descriptive new name for the new table like Emp_Lead_Project to mean "Employee leads a project".

Emp_Lead_Project

<u>EID</u>	<u>PID</u>	PBonus	StartDate	EndDate
------------	------------	--------	-----------	---------

At the end of the mapping we will have the following relational schema (tables) for the logical database design phase.

Department

<u>DID</u>	DName	DLoc	MEID
------------	-------	------	------

Project

<u>PID</u>	PName	PFund
------------	-------	-------

Telephone

<u>EID</u>	<u>Tel</u>
------------	------------

Employee

<u>EID</u>	FName	LName	Salary	EDID
------------	-------	-------	--------	------

Emp_Partc_Project

<u>EID</u>	<u>PID</u>
------------	------------

Emp_Lead_Project

<u>EID</u>	<u>PID</u>	PBonus	StartDate	EndDate
------------	------------	--------	-----------	---------

After converting the ER diagram in to table forms, the next phase is implementing the process of normalization, which is a collection of rules each table should satisfy.

Normalization

A relational database is merely a collection of data, organized in a particular manner. As the father of the relational database approach, **Codd** created a series of rules (tests) called *normal forms* that help define that organization

One of the best ways to determine what information should be stored in a database is to clarify what questions will be asked of it and what data would be included in the answers.

Database normalization is a series of steps followed to obtain a database design that allows for consistent storage and efficient access of data in a relational database. These steps reduce data redundancy and the risk of data becoming inconsistent.

NORMALIZATION is the process of identifying the logical associations between data items and designing a database that will represent such associations but without suffering the update anomalies which are;

1. **Insertion Anomalies**
2. **Deletion Anomalies**
3. **Modification Anomalies**

Normalization may reduce system performance since data will be cross referenced from many tables. Thus denormalization is sometimes used to improve performance, at the cost of reduced consistency guarantees.

Normalization normally is considered “good” if it is lossless decomposition.

All the normalization rules will eventually remove the update anomalies that may exist during data manipulation after the implementation. The update anomalies are;

The type of problems that could occur in insufficiently normalized table is called update anomalies which includes;

(1) **Insertion anomalies**

An "insertion anomaly" is a failure to place information about a new database entry into all the places in the database where information about that new entry needs to be stored. *Additionally, we may have difficulty to insert some data.* In a properly normalized database, information about a new entry needs to be inserted into only one place in the database; in an inadequately normalized database, information about a new entry may need to be inserted into more

than one place and, human fallibility being what it is, some of the needed additional insertions may be missed.

(2) **Deletion anomalies**

A "deletion anomaly" is a failure to remove information about an existing database entry when it is time to remove that entry. *Additionally, deletion of one data may result in lose of other information.* In a properly normalized database, information about an old, to-be-gotten-rid-of entry needs to be deleted from only one place in the database; in an inadequately normalized database, information about that old entry may need to be deleted from more than one place, and, human fallibility being what it is, some of the needed additional deletions may be missed.

(3) **Modification anomalies**

A modification of a database involves changing some value of the attribute of a table. In a properly normalized database table, what ever information is modified by the user, the change will be effected and used accordingly.

In order to avoid the update anomalies we in a given table, the solution is to decompose it to smaller tables based on the rule of normalization. However, the decomposition has *two important properties*

- a. The **Lossless-join** property insures that any instance of the original relation can be identified from the instances of the smaller relations.
- b. The **Dependency preservation** property implies that constraint on the original dependency can be maintained by enforcing some constraints on the smaller relations. i.e. we don't have to perform Join operation to check whether a constraint on the original relation is violated or not.

The purpose of normalization is to reduce the chances for anomalies to occur in a database.

Example of problems related with Anomalies

<i>EmpID</i>	<i>FName</i>	<i>LName</i>	<i>SkillID</i>	<i>Skill</i>	<i>SkillType</i>	<i>School</i>	<i>SchoolAdd</i>	<i>Skill Level</i>
12	Abebe	Mekuria	2	SQL	Database	AAU	Sidist_Kilo	5
16	Lemma	Alemu	5	C++	Programming	Unity	Gerji	6
28	Chane	Kebede	2	SQL	Database	AAU	Sidist_Kilo	10
25	Abera	Taye	6	VB6	Programming	Helico	Piazza	8
65	Almaz	Belay	2	SQL	Database	Helico	Piazza	9
24	Dereje	Tamiru	8	Oracle	Database	Unity	Gerji	5
51	Selam	Belay	4	Prolog	Programming	Jimma	Jimma City	8
94	Alem	Kebede	3	Cisco	Networking	AAU	Sidist_Kilo	7
18	Girma	Dereje	1	IP	Programming	Jimma	Jimma City	4
13	Yared	Gizaw	7	Java	Programming	AAU	Sidist_Kilo	6

Deletion Anomalies:

If employee with *ID 16* is deleted then ever information about skill *C++* and the type of skill is deleted from the database. Then we will not have any information about *C++* and its skill type.

Insertion Anomalies:

What if we have a new employee with a skill called *Pascal*? We can not decide weather *Pascal* is allowed as a value for skill and we have no clue about the *type of skill* that *Pascal* should be categorized as.

Modification Anomalies:

What if the address for *Helico* is changed from *Piazza* to *Mexico*? We need to look for every occurrence of *Helico* and change the value of *School_Add* from *Piazza* to *Mexico*, which is prone to error.

Database-management system can work only with the information that we put explicitly into its tables for a given database and into its rules for working with those tables, where such rules are appropriate and possible.

Functional Dependency (FD)

Before moving to the definition and application of normalization, it is important to have an understanding of "functional dependency."

Data Dependency

The logical associations between data items that point the database designer in the direction of a good database design are referred to as determinant or dependent relationships.

Two data items A and B are said to be in a determinant or dependent relationship if certain values of data item B always appears with certain values of data item A. If the data item A is the determinant data item and B the dependent data item then the direction of the association is from A to B and not vice versa.

The essence of this idea is that if the existence of something, call it A, implies that B must exist and have a certain value, then we say that "**B is functionally dependent on A.**" We also often express this idea by saying that "A functionally determines B," or that "B is a function of A," or that "A functionally governs B." Often, the notions of functionality and functional dependency are expressed briefly by the statement, "If A, then B." It is important to note that the value of B must be *unique* for a given value of A, i.e., any given value of A must imply just one and only one value of B, in order for the relationship to qualify for the name "function." (However, this does not necessarily prevent different values of A from implying the same value of B.)

However, for the purpose of normalization, we are interested in finding 1..1 (one to one) dependencies, lasting for all times (*intension* rather than *extension* of the database), and the determinant having the minimal number of attributes.

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

The notation is: **$A \rightarrow B$** which is read as; B is functionally dependent on A

In general, a **functional dependency** is a relationship among attributes. In relational databases, we can have a determinant that governs one or several other attributes.

FDs are derived from the real-world constraints on the attributes and they are properties on the database intension **not** extension.

Example

<i>Dinner Course</i>	<i>Type of Wine</i>
Meat	Red
Fish	White
Cheese	Rose

Since the type of *Wine* served depends on the type of *Dinner*, we say *Wine* is functionally dependent on *Dinner*.

Dinner \rightarrow *Wine*

<i>Dinner Course</i>	<i>Type of Wine</i>	<i>Type of Fork</i>
Meat	Red	Meat fork
Fish	White	Fish fork
Cheese	Rose	Cheese fork

Since both *Wine* type and *Fork* type are determined by the *Dinner* type, we say *Wine* is functionally dependent on *Dinner* and *Fork* is functionally dependent on *Dinner*.

Dinner \rightarrow *Wine*

Dinner \rightarrow *Fork*

Partial Dependency

If an attribute which is not a member of the primary key is dependent on some part of the primary key (if we have composite primary key) then that attribute is partially functionally dependent on the primary key.

Let {A,B} is the Primary Key and C is no key attribute.

Then if **$\{A,B\} \rightarrow C$ and $B \rightarrow C$**

Then C is partially functionally dependent on {A,B}

Full Functional Dependency

If an attribute which is not a member of the primary key is not dependent on some part of the primary key but the whole key (if we have composite primary key) then that attribute is fully functionally dependent on the primary key.

Let $\{A,B\}$ be the Primary Key and C is a non- key attribute

Then if $\{A,B\} \rightarrow C$ and $B \rightarrow C$ and $A \rightarrow C$ does not hold

Then C Fully functionally dependent on $\{A,B\}$

Transitive Dependency

In mathematics and logic, a transitive relationship is a relationship of the following form: "If A implies B, and if also B implies C, then A implies C."

Example:

If Mr X is a Human, and if every Human is an Animal, then Mr X must be an Animal.

Generalized way of describing transitive dependency is that:

If A functionally governs B, AND

If B functionally governs C

THEN A functionally governs C

Provided that neither C nor B determines A i.e. $(B \not\rightarrow A \text{ and } C \not\rightarrow A)$

In the normal notation:

$\{(A \rightarrow B) \text{ AND } (B \rightarrow C)\} \Rightarrow A \rightarrow C$ provided that $B \not\rightarrow A$ and $C \not\rightarrow A$

Steps of Normalization:

We have various levels or steps in normalization called *Normal Forms*. The level of complexity, strength of the rule and decomposition increases as we move from one lower level Normal Form to the higher.

A table in a relational database is said to be in a certain normal form if it satisfies certain constraints.

A normal form below represents a stronger condition than the previous one

Normalization towards a logical design consists of the following steps:

UnNormalized Form(UNF):

Identify all data elements

First Normal Form(1NF):

Find **the key** with which you can find **all** data i.e. remove any repeating group

Second Normal Form(2NF):

Remove part-key dependencies (partial dependency). Make all data dependent on **the whole key**.

Third Normal Form(3NF)

Remove non-key dependencies (transitive dependencies). Make all data dependent on **nothing but the key**.

For most practical purposes, databases are considered normalized if they adhere to the *third normal form* (there is no transitive dependency).

First Normal Form (1NF)

Requires that all column values in a table are *atomic* (e.g., a number is an atomic value, while a list or a set is not).

We have two ways of achieving this:

1. Putting each repeating group into a separate table and connecting them with a *primary key-foreign key* relationship
2. Moving these repeating groups to a new row by repeating the **non-repeating attributes** known as “flattening” the table. If so then Find the key with which you can find all data

Definition: a table (relation) is in 1NF

If

- There are no duplicated rows in the table. Unique identifier
 - Each cell is single-valued (i.e., there are no repeating groups).
 - Entries in a column (attribute, field) are of the same kind.
-

Example for First Normal form (1NF)

UNNORMALIZED

<i>EmpID</i>	<i>FirstName</i>	<i>LastName</i>	<i>Skill</i>	<i>SkillType</i>	<i>School</i>	<i>SchoolAdd</i>	<i>SkillLevel</i>
12	Abebe	Mekuria	SQL, VB6	Database, Programming	AAU, Helico	Sidist_Kilo Piazza	5 8
16	Lemma	Alemu	C++ IP	Programming Programming	Unity Jimma	Gerji Jimma City	6 4
28	Chane	Kebede	SQL	Database	AAU	Sidist_Kilo	10
65	Almaz	Belay	SQL Prolog Java	Database Programming Programming	Helico Jimma AAU	Piazza Jimma City Sidist_Kilo	9 8 6
24	Dereje	Tamiru	Oracle	Database	Unity	Gerji	5
94	Alem	Kebede	Cisco	Networking	AAU	Sidist_Kilo	7

FIRST NORMAL FORM (1NF)

Remove all repeating groups. Distribute the multi-valued attributes into different rows and identify a unique identifier for the relation so that it can be said is a relation in relational database. Flatten the table.

<i>EmpID</i>	<i>FirstName</i>	<i>LastName</i>	<i>SkillID</i>	<i>Skill</i>	<i>SkillType</i>	<i>School</i>	<i>SchoolAdd</i>	<i>SkillLevel</i>
12	Abebe	Mekuria	1	SQL	Database	AAU	Sidist_Kilo	5
12	Abebe	Mekuria	3	VB6	Programming	Helico	Piazza	8
16	Lemma	Alemu	2	C++	Programming	Unity	Gerji	6
16	Lemma	Alemu	7	IP	Programming	Jimma	Jimma City	4
28	Chane	Kebede	1	SQL	Database	AAU	Sidist_Kilo	10
65	Almaz	Belay	1	SQL	Database	Helico	Piazza	9
65	Almaz	Belay	5	Prolog	Programming	Jimma	Jimma City	8
65	Almaz	Belay	8	Java	Programming	AAU	Sidist_Kilo	6
24	Dereje	Tamiru	4	Oracle	Database	Unity	Gerji	5
94	Alem	Kebede	6	Cisco	Networking	AAU	Sidist_Kilo	7

Second Normal form 2NF

No partial dependency of a non key attribute on part of the primary key. This will result in a set of relations with a level of Second Normal Form.

Any table that is in 1NF and has a single-attribute (i.e., a non-composite) key is automatically also in 2NF.

Definition: a table (relation) is in 2NF

If

- It is in 1NF and
- If all non-key attributes are dependent on the entire primary key.
i.e. no partial dependency.

Example for 2NF:

EMP_PROJ

<u>EmpID</u>	EmpName	<u>ProjNo</u>	ProjName	ProjLoc	ProjFund	ProjMangID	Incentive
--------------	---------	---------------	----------	---------	----------	------------	-----------

EMP_PROJ rearranged

<u>EmpID</u>	<u>ProjNo</u>	EmpName	ProjName	ProjLoc	ProjFund	ProjMangID	Incentive
--------------	---------------	---------	----------	---------	----------	------------	-----------

Business rule: Whenever an employee participates in a project, he/she will be entitled for an incentive.

This schema is in its 1NF since we don't have any repeating groups or attributes with multi-valued property. To convert it to a 2NF we need to remove all partial dependencies of non key attributes on part of the primary key.

{EmpID, ProjNo} → EmpName, ProjName, ProjLoc, ProjFund, ProjMangID, Incentive

But in addition to this we have the following dependencies

FD1: {EmpID} → EmpName

FD2: {ProjNo} → ProjName, ProjLoc, ProjFund, ProjMangID

FD3: {EmpID, ProjNo} → Incentive

As we can see, some non key attributes are partially dependent on some part of the primary key. This can be witnessed by analyzing the first two functional dependencies (FD1 and FD2). Thus, each Functional Dependencies, with their dependent attributes should be moved to a new relation where the Determinant will be the Primary Key for each.

EMPLOYEE

<u>EmpID</u>	EmpName
--------------	---------

PROJECT

<u>ProjNo</u>	ProjName	ProjLoc	ProjFund	ProjMangID
---------------	----------	---------	----------	------------

EMP_PROJ

<u>EmpID</u>	<u>ProjNo</u>	Incentive
--------------	---------------	-----------

Third Normal Form (3NF)

Eliminate Columns dependent on another non-Primary Key - If attributes do not contribute to a description of the key; remove them to a separate table.

This level avoids update and deletes anomalies.

Definition: a Table (Relation) is in 3NF

If

- **It is in 2NF and**
- **There are no transitive dependencies between a primary key and non-primary key attributes.**

Example for (3NF)

Assumption: Students of same batch (same year) live in one building or dormitory

STUDENT

<u>StudID</u>	Stud_F_Name	Stud_L_Name	Dept	Year	Dormitory
125/97	Abebe	Mekuria	Info Sc	1	401
654/95	Lemma	Alemu	Geog	3	403
842/95	Chane	Kebede	CompSc	3	403
165/97	Alem	Kebede	InfoSc	1	401
985/95	Almaz	Belay	Geog	3	403

This schema is in its 2NF since the primary key is a single attribute and there are no repeating groups (multi valued attributes).

Let's take *StudID*, *Year* and *Dormitory* and see the dependencies.

StudID* → *Year* AND *Year* → *Dormitory

And Year can not determine StudID and Dormitory can not determine StudID Then transitively StudID → Dormitory

To convert it to a 3NF we need to remove all transitive dependencies of non key attributes on another non-key attribute.

The non-primary key attributes, dependent on each other will be moved to another table and linked with the main table using Candidate Key- Foreign Key relationship.

STUDENT

<u>StudID</u>	<i>Stud F_Name</i>	<i>Stud L_Name</i>	<i>Dept</i>	<i>Year</i>
125/97	Abebe	Mekuria	<i>Info Sc</i>	<i>1</i>
654/95	Lemma	Alemu	<i>Geog</i>	<i>3</i>
842/95	Chane	Kebede	<i>CompSc</i>	<i>3</i>
165/97	Alem	Kebede	<i>InfoSc</i>	<i>1</i>
985/95	Almaz	Belay	<i>Geog</i>	<i>3</i>

DORM

<u>Year</u>	<i>Dormitory</i>
<i>1</i>	<i>401</i>
<i>3</i>	<i>403</i>

Generally, eventhough there are other four additional levels of Normalization, a table is said to be normalized if it reaches 3NF. A database with all tables in the 3NF is said to be Normalized Database.

Mnemonic for remembering the rationale for normalization up to 3NF could be the following:

1. No Repeating or Redundancy: *no repeting fields in the table.*
2. The Fields Depend Upon the Key: *the table should solely depend on the key.*
3. The Whole Key: *no partial keybdependency.*
4. And Nothing But the Key: *no inter data dependency.*
5. So Help Me Codd: *since Codd came up with these rules.*

Other Levels of Normalization

Boyce-Codd Normal Form (BCNF):

BCNF is based on functional dependency that takes in to account all the candidate keys in a relation.

So, **table is in BCNF if it is in 3NF and if every determinant is a candidate key.**

Violation of the BCNF is very rare. The potential sources for violation of this rule are

1. The relation contains two (or more) composite candidate keys
2. The candidate keys overlap i.e. have common attribute.

The issue is related to:

Isolating Independent Multiple Relationships - No table may contain two or more 1:N or N:M relationships that are not directly related.

The correct solution, to cause the model to be in 4th normal form, is to ensure that all M:M relationships are resolved independently if they are indeed independent, as shown below.

Forth Normal form (4NF)

Isolate Semantically Related Multiple Relationships - There may be practical constraints on information that justify separating logically related many-to-many relationships.

MVD(Multi-Valued Dependency) : represents a dependency between attributes(for example A, B,C) in a relation such that for every value of A there is a set of values for B and there is a set of values for C but the sets B and C are independent to each other.

MVD between attributes A, B, and C in a relation is represented as follows

A----->>B
A----->>C

Def: A table is in 4NF if it is in BCNF and if it has no multi-valued dependencies.

Fifth Normal Form (5NF)

Sometimes called the Project -Join -Normal Form (PJNF)

5NF is based on the Join dependency.

Join Dependency: a property of decomposition that ensures that no spurious are generated when rejoining to obtain the original relation

Def: A table is in 5NF, also called "Projection-Join Normal Form" (PJNF), if it is in 4NF and if every join dependency in the table is a consequence of the candidate keys of the table.

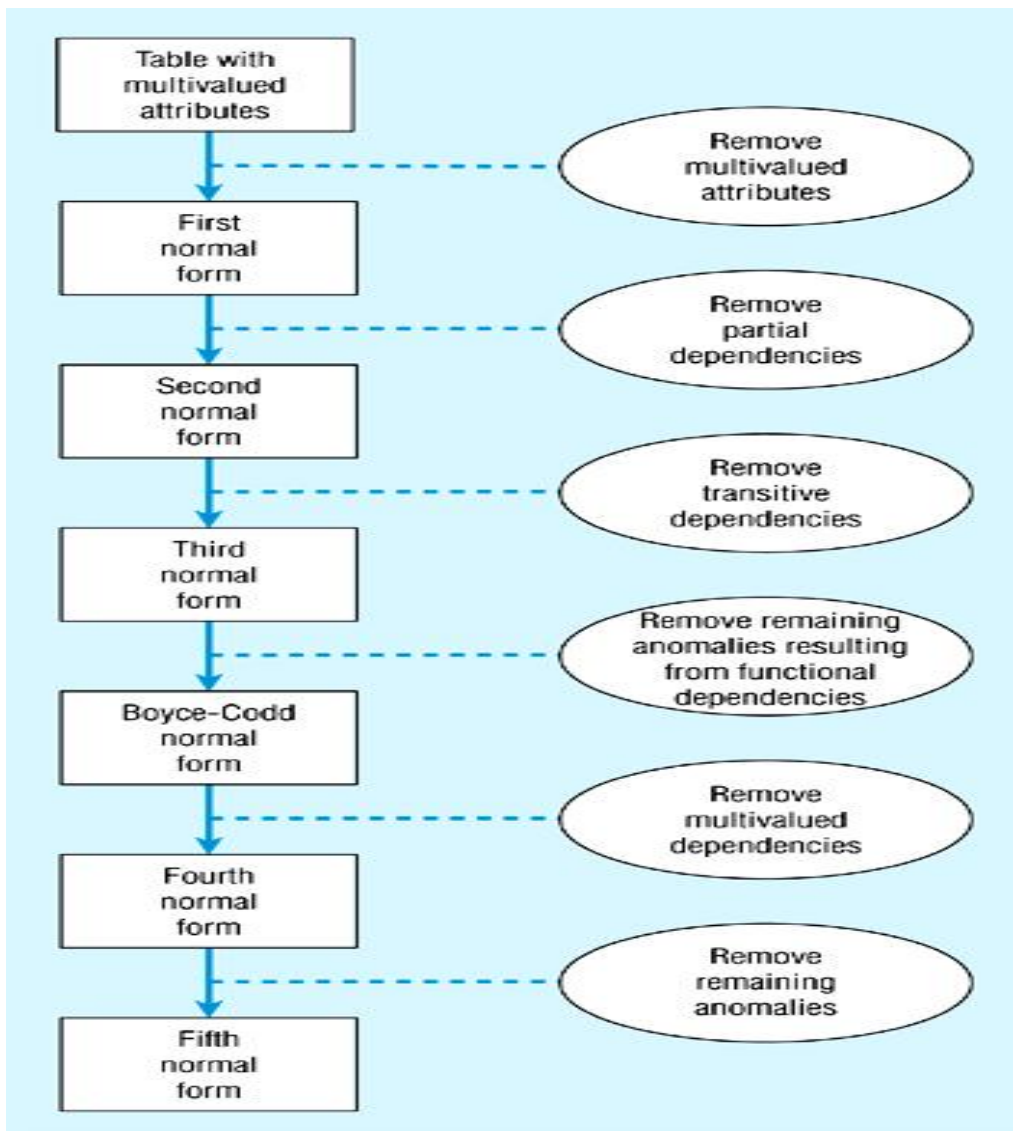
Domain-Key Normal Form (DKNF)

A model free from all modification anomalies.

Def: A table is in DKNF if every constraint on the table is a logical consequence of the definition of keys and domains.

The underlying ideas in normalization are simple enough. Through normalization we want to design for our relational database a set of tables that;

- (1) Contain all the data necessary for the purposes that the database is to serve,
- (2) Have as little redundancy as possible,
- (3) Accommodate multiple values for types of data that require them,
- (4) Permit efficient updates of the data in the database, and
- (5) Avoid the danger of losing data unknowingly.



Problems associated with normalization

- Requires data to see the problems
 - May reduce performance of the system
 - Is time consuming,
 - Difficult to design and apply and
 - Prone to human error
-