12. LOGICAL DATABASE DESIGN AND THE RELATIONAL MODEL

Logical database design: is the process of transforming the conceptual data model (ER-Model) into a logical data model.

There are two reasons for emphasizing on the logical model:

1. It is the most used in contemporary database applications.
2. Some of the principles of logical data base design for the relational model apply to the other logical model as well.

12.1. The Relational Data Model:

E. F. Codd first introduced the relational data model in 1970. Today RDBMSs have been the dominate technology for database management.

Basic definitions:

The relational model based on mathematical theory and therefore has a solid theoretical foundation, it consists of the following components:

a) **Data Structure**: Data are organized in the form of tables with rows and columns.

b) **Data manipulation:** Powerful operations (using SQL Language)are used to manipulate data stored in the relation.

c) **Data integrity:** Facilities are included to specify business rules that maintain the integrity of data when they are manipulated.

12.2. Relational Data Structure:

A Relation is a named two-dimensional table of data , each relation consists of a set of named columns and an arbitrary number of unnamed rows .

An Attribute (consistent of its previous definition) is a named column of a relation.

Each row of the relation corresponds to a record that contains data (attributes) values for a single entity. See EMPLOYEE1:

EMPLOYEE1

Emp_ID	**Name**	**Dept_Name**	**Salary**
100	Mohammad Zaki	Marketing	530.000
140	Sufyan Salim	Accounting	480.000
110	Sinan Zohair	Info systems	225.000
190	Muthanna Mohammad	Finance	400.000
150	Mahmmod Khalid	Marketing	75.000

We can express the structure of a relation by a shorthand notation in which the name of the relation is followed (in parentheses) by the names of the attributes in that relation. For EMPLOYEE1 Example we would have:

EMPLOYEE1 (Emp_id, Name,Dept_Name,Salary).

12.3. Relational Keys:

We must be able to store and retrieve a row of data in a relation based on the data values stored in that row, to achieve this goal, every relation must have a primary key .

A **Primary Key** : is an attribute (or combination of attributes) that uniquely identifies each row in a relation.

We designate a Primary Key by underlining the attribute name. For example, the primary key for the relation EMPLOYEE is Emp_id and we represent it as follows:

EMPLOYEE1 (Emp_id, Name,Dept_Name,Salary).

The Concept of a primary key is related to the term "Identifier" in many cases the same attribute (or attributes) indicates as an entity's identifier in an E R-diagram will be the same attributes that compose the primary key for the relation representing that entity .

There are exceptions, for example associated entities is not have to had an identifier and the identifier of a weak entity forms only part of a weak entity's primary key.

In addition, there may be several attributes of an entity that may be serve as the associated relation's primary key.

A composite key: is a primary key that consists of more than one attribute.

For example, the primary key for a relation DEPENDENT would likely consists of the combination of Emp_id and Dependent_name.

Foreign key an attribute in a relation of a database that serves as the primary key in another relation in the same database.

Some authors emphasize the fact that an attribute is a foreign key by using dashed underline, such as :

EMPLOYEE1(Emp_id,Name,Dept_Name,Salary).
DEPARTMENT(Dept_Name,Location,Fax)

The attribute Dept_Name is a foreign key in EMPLOYEE . It allows a user to associate any employee with the department to which he or she is assigned.

12.4. Properties of the Relations:

Relation has several properties; we summarize them in the following:

1. Each relation (or table) in a database has a unique name.
2. Any entry at the intersection of each row and column is atomic (or single valued) there can be no multivalued attributes in a relation.
3. Each row is a unique, no two rows in a relation are identical.
4. Each attribute (or column) within a table has a unique name.
5. The sequence of columns (left to right) in insignificant. The columns of a relation can be interchanged without changing the meaning or use of the relation.
6. The sequence of rows (top to bottom). As with columns. The rows of a relation may be interchanged or stored in any sequence.

12.5. Removing Multivalued Attributes from Tables:

The second property of the relation above states that can be no multivalued attributes in a relation. Thus, a table that contains one or more multivalued attributes is not a relation. As an example, the next figure shows the employee data from the EMPLOYEE1 relation extended to include courses that may have been taken by those employees. Since a given employee may have taken more than one course, the attributes Course_Title and Date_Completed are multivalued attributes. For example, the employee with Emp_IdD 100 has taken two courses . If an employee has not taken the any courses, the Course_Title and Date_Completed attributes (see employee with Emp_ID 190 for an example).

EMPLOYEE2

Emp_ID	Name	Dept_Name	Salary	Course_Title
100	Mohammad Zaki	Marketing	530.000	SPSS Cisco
140	Sufyan Salim	Accounting	480.000	CNC
110	Sinan Zohair	Info systems	225.000	Tax Acc SPSS
190	Muthanna Mohammad	Finance	400.000	
150	Mahmmod Khalid	Marketing	75.000	SPSS Java

We show how to eliminate the multivalued attributes in the next figure, by filling the relevant data values into the previously vacant cells of previous figure. As a result, the table in figure below has only single-valued attributes and now satisfies the atomic property of relation, but still have some undesirable properties.

EMPLOYEE2

Emp_ID	Name	Dept_Name	Salary	Course_Title
100	Mohammad Zaki	Marketing	530.000	SPSS
100	Mohammad Zaki	Marketing	530.000	Cisco
140	Sufyan Salim	Accounting	480.000	CNC
110	Sinan Zohair	Info systems	225.000	Tax Acc
110	Sinan Zohair	Info systems	225.000	SPSS
190	Muthanna Mohammad	Finance	400.000	
150	Mahmmod Khalid	Marketing	75.000	SPSS
150	Mahmmod Khalid	Marketing	75.000	Java

12.6. Example Database:

A relational data base consists of any number of relations. The structure of the database is described using a conceptual schema. There are two common methods for expressing a (conceptual) schema:

A- Short text statement, in which each relation is named, and the names of its attributes follow in parentheses such as: EMPLOYEE(Emp_id,Name,Dept_Name,Salary).

B- Graphical representation, in which each relation is represented by a rectangle containing the attributes for the relation such as .

CUSTOMER

Customer_Id	Customer_Name	Adress	City

ORDER

Order_Id	Order_Date	Customer_Id

ORDER LINE

Order_Id	Product_Id	Quantity

PRODUCT

Product_Id	Product_description	Product_Finish	Unit_Price	On_Hand

It is a good idea to create an instance of the relational schema with sample data for three reasons:

1. The sample data provide a conventional way to check the accuracy of the design.
2. The sample data help improve communications with users in discussing the design.
3. Sample data can be used to develop prototype applications and test queries.

12.7. Integrity Constrains:

The relational data model includes several types of constrains whose purpose is to facilitate maintaining the accuracy and integrity of data in the database.

The main types of integrity constrains are: domain constrains, entity integrity, referential integrity, and operational constrains.

1. Domain Constrains:

All the values that appear in a column of a relation must be taken from the same domain.

A domain is the set of values that may be assigned to an attribute, it consists of the following components:

1. Domain name
2. Data type
3. Size (or length)
4. Allowable values (or allowable range)

The table below shows domain definition for domain associated with attributes for EMPLOYEE1:

Attribute	**Domain name**	**Description**	**Domain**
Emp_id	Emp_id	Set of all possible employee IDs	Character Size 6
Name	Name	Set of all possible employee names	Character size 30
Dept_Name	Dept_Name	Set of all possible departments	Character size 20
Salary	Salary	Set of all possible e Salaries	Numeric size 8

2. Entity Integrity:

The entity integrity rule is designed to assure that every relation has a primary key, and that the data values for the primary key are all valid. It guarantees that every primary key attribute is non-null.

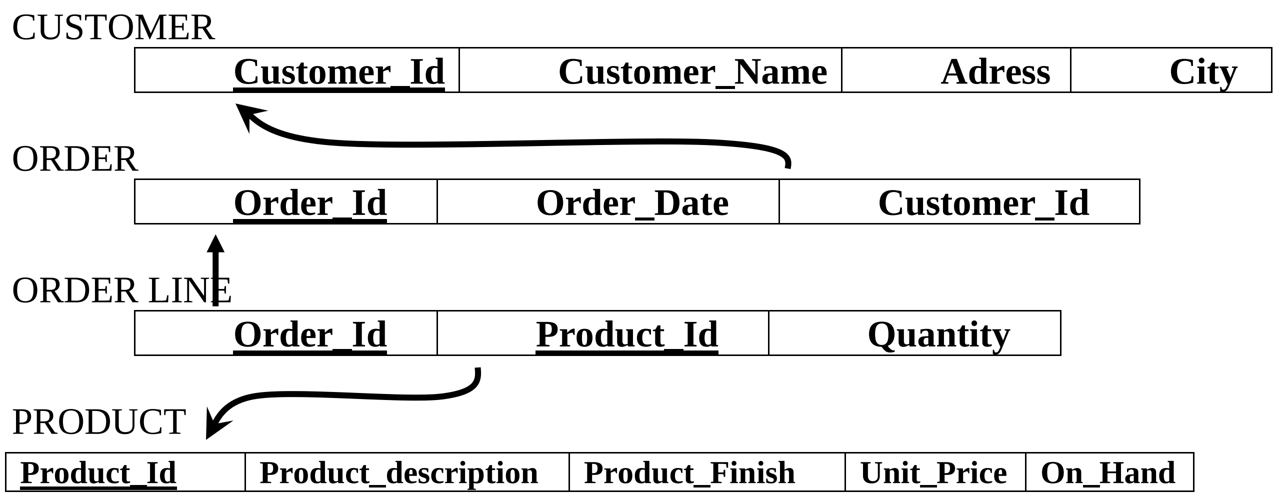
NULL: a value that may be assigned to an attribute when no other value applies , or when the application value is unknown.

3. Referential Integrity:

In the relational data model, the association between tables are defined through the use of foreign keys.

A **referential** integrity constraint is rule that maintains consistency among the rows of two relations. The rule states that if there is a foreign key in one relation, either each foreign key value must match a primary key value in the other relation or else the foreign key value must be NULL.

The graphical version of the relational schema provides a simple technique for identifying associations where referential integrity must be enforced, the figure below shows the schema for the relations introduced early. An arrow has been drawn from each foreign key to the associated primary key. A referential integrity constraint must be defended for each of these arrows in the schema.


CUSTOMER
Customer_Id | Customer_Name | Adress | City
ORDER
Order_Id | Order_Date | Customer_Id
ORDER LINE
Order_Id | Product_Id | Quantity
PRODUCT
Product_Id | Product_description | Product_Finish | Unit_Price | On_Hand

4. **Operational Constraints:** these constraints belong to business rules during operations .as an example (A person may purchase a ticket for all –star game only if that person is a season –ticket holder)

12.8. Creating Relational Tables:

At this point, we create table definition for the four tables above. These definitions are created using CREATE TABLE statements from T-SQL data definition language ,these table definitions are created during the implementation phase later in the database development process.

General syntax for create table is :

```
CREATE TABLE table_name (
   column1 datatype constraint,
   column2 datatype constraint,
   column3 datatype constraint,
   ....
);
```

The SQL table definition is shown in the relational schema , each attribute for a table is then defined. Notice that the data type and length for each attribute is taken from the domain definition.

```
Create database SALES;
Use SALES;
CREATE TABLE  CUSTOMER
(CUSTOMER_ID        INT Identity PRIMARY KEY,
CUSTOMER_NAME       VARCHAR(50) Not Null ,
CUSTOMER_ADDRESS    VARCHAR(60) Not Null ,
);

CREATE TABLE  ORDERS
(ORDER_ID       INT Identity PRIMARY KEY,
ORDER_DATE      DATE Not Null      ,
CUSTOMER_ID     Integer Not Null   ,
FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER (CUSTOMER_ID));

CREATE TABLE PRODUCT
(PRODUCT_ID     INT Identity PRIMARY KEY,
PRODUCT_NAME     VARCHAR (30) Unique Not Null,
PRODUCT_DESCRIPTION    VARCHAR (25) Not Null,
PRODUCT_FINISH      VARCHAR (12)Not Null,
UNIT_PRICE      DECIMAL (8, 2)Not Null ,
ON_HAND         INT  Not Null);


CREATE TABLE ORDERS_LINE
(ORDER_ID       INT  Not Null,
PRODUCT_ID      INT  Not Null
QUANTITY  INT  Not Null
PRIMARY KEY (ORDER_ID, PRODUCT_ID),
FOREIGN KEY (ORDER_ID) REFERENCES ORDERS (ORDER_ID),
FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT (PRODUCT_ID));
```

12.9. Well Structured Relations:

A relation that contains minimal redundancies and allows users to insert, modify , and delete the rows in a table without errors or inconsistencies . Redundancies in a table may results in errors or inconsistencies called anomalies.

Anomalies : errors or inconsistencies that may result when a user attempts to update a table, there are three types of anomalies:

1. ***Insertion anomaly***: Suppose that we need to add a new employee to EMPLOYEE2 . The primary key for this relation id Emp_Id and Course_Title, therefore to insert anew row, the user must supply values for both Emp_Id and Course_Title(since primary key values cannot be null or nonexistent)This is an anomaly ,since the user should be able to enter employee data without supplying course data.

2. ***Deletion anomaly***: Suppose that data for employee number 140 are deleted from the table, this will result in losing the information that this employee completed a course (CNC) on 12/8/2021. It results in losing the information that this course had an offering that completed on that date.

3. ***Modification anomaly****:* suppose that employee 100 *gates* a salary increase. We must record this increase in each of the rows for that employee (two occurrences); otherwise, the data will be inconsistent.

These anomalies indicate that EMPLOYEE2 is not well-structured relation. The problem with this relation is that it contains data about two entities: EMPLOYEE and COURSE, It should be normalized using normalization theory to divide EMPLOYEE2 into two relations, one of the resulting relation is EMPLOYEE1 , the other will be EMP_COURSE, which appears with sample data in next figure :

EMP_COURSE

Emp_ID	**Course_Title**	**Date_Completed**
100	SPSS	6/19/2021
100	Cisco	10/7/2022
140	CNC	12/8/2021
110	Tax Acc	1/12/2020
110	SPSS	4/22/2021
150	SPSS	6/19/2021
150	Java	8/12/2022

The primary key of this relation is the combination of Emp_Id and Course_Title, and we underline these attribute names to highlight this fact, so it is a well-structured relation.

12.10. Transforming Er- Diagrams into Relations:

In the logical design the E-R (and EER) Diagram transforms to a relational database schema, the input to this process is the ER-Diagram and the output is the relational schema. It is a straightforward process with a well-defined set of rules.

Three types of entities will be discussed:

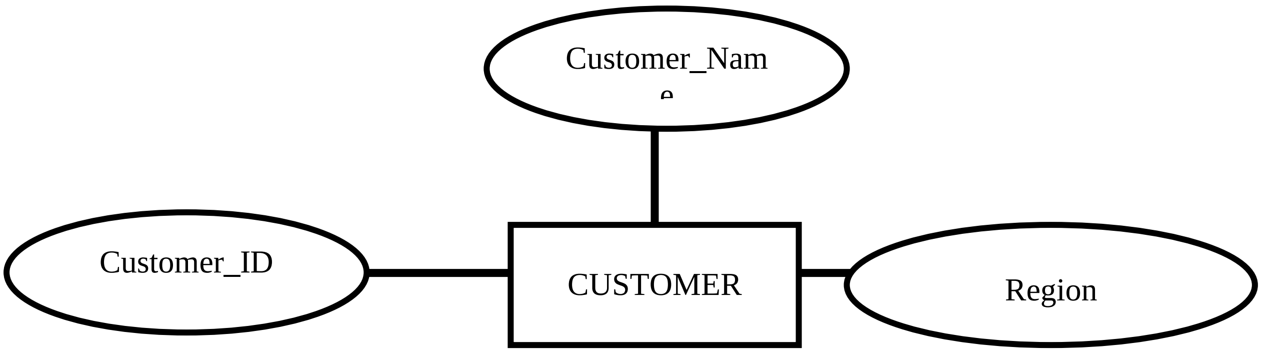
1. ***Regular entities***: are entities that have an independent existence and generally represent real-word objects such as persons and products. Regular entity types are represented by rectangles with single line.

2. ***Weak entities***: are entities that cannot exist except with an identifying relationship with an owner (regular)entity type. Weak entities identified by a rectangle with a double line.

3. ***Associative entities*** (gerunds): they formed from many-to-many relationships between other entity types .Associative entities represented by a rectangle with single line that encloses the diamond relationship symbol.

STEP1: Map regular Entities:
Each regular entity type in an ER-diagram is transformed into a relation .The name given to the relation is generally the same as the entity type. Each simple attribute of the entity type becomes an attribute of the relation. The identifier of the entity type becomes the primary key satisfies the desirable properties of identifiers.

The figure below shows a representation of CUSTOMER entity type, and the corresponding relation shown in graphical form, (we take only few attributes to simplify the figure).

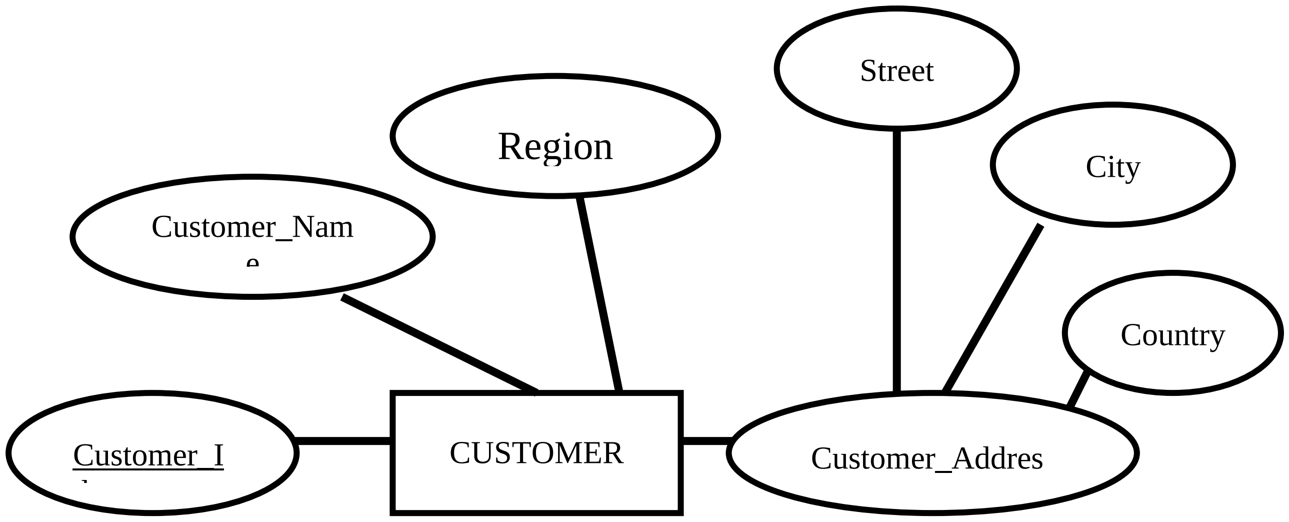


CUSTOMER

Customer_Id	Customer_Name	Region

Composite attribute:

When the regular entity type has composite attributes, only the simple components attributes of the composite attribute are included in the new relation. Figure below shows a variation of the previous example:



CUSTOMER

<u>**Customer_Id**</u>	**Customer_Name**	**Region**	**Street**	**City**	**Country**

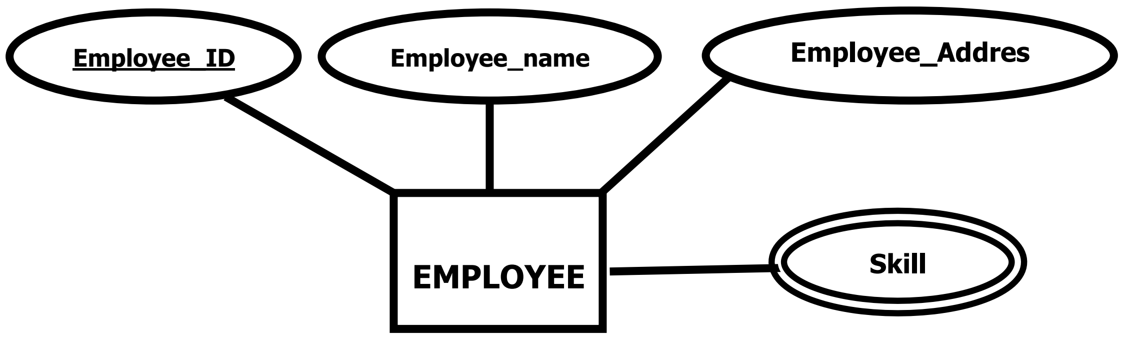
We see that Customer_Address, which is a composite attribute, with components Street, City, Country, mapped to the relation with its simple attributes only.

Multivalued attributes:

When regular entity contains multivalued attribute, two new relations (rather than one) are created. The first relation contains all of the attributes of the entity type except the multivalued attribute. The Second relation contains two attributes that form the primary key of the second relation. The first of these attributes is the primary key from the first relation, which becomes a foreign key in the second relation. The second is the multivalued attribute .the name of the second relation should capture the meaning of the multivalued attribute.

As an example of this procedure, in the figure below, this is the EMPLOYEE entity type for a company. As shown in part (a) EMPLOYEE has Skill as a multivalued attribute. Part (b) shows the two relations that are created. The first (called EMPLOYEE) has the primary key Employee_ID. The second relation (called EMPLOYEE_SKILL) has the two attributes Employee_ID and Skill, which form the primary key. The relationship between foreign and primary keys is indicated by arrow in the figure.

Part (a)



Part (b)

EMPLOYEE

<u>Employee_ID</u>	Employee_Name	Employee_Address

EMPLOYEE_SKILL

<u>Employee_ID</u>	<u>Skill</u>

STEP 2: Map Weak Entities:

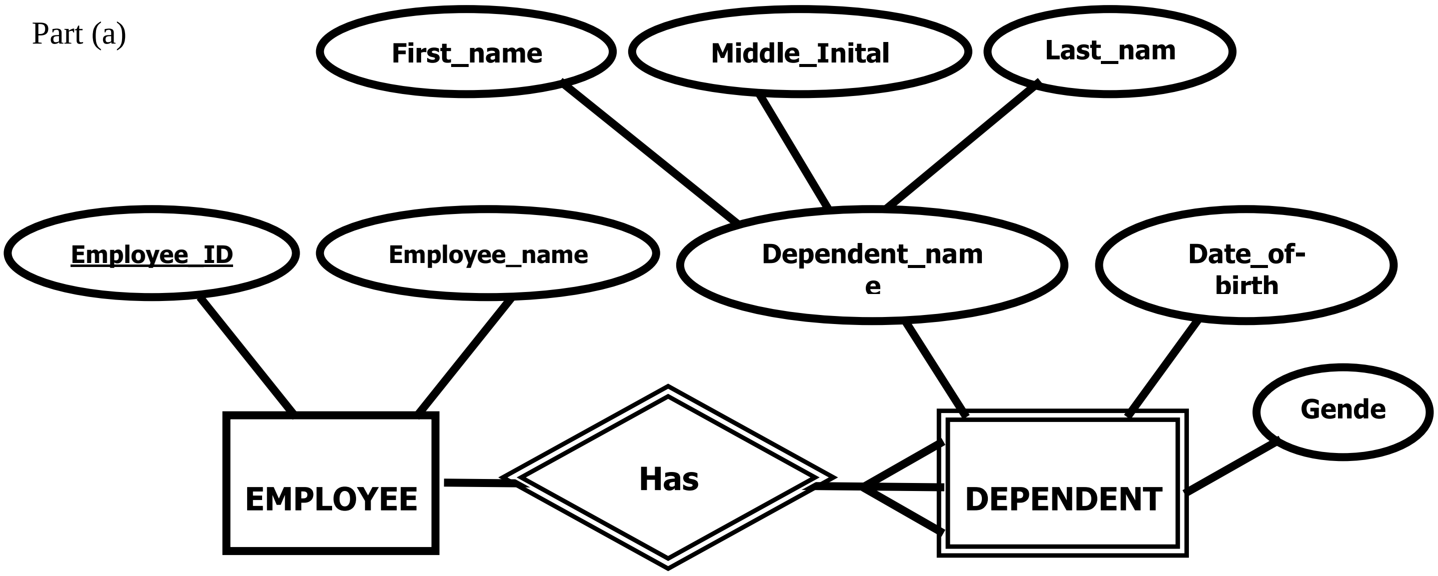
a) For each weak entity type, create a new relation and include all of the simple attributes(or the simple components of composite attributes) as attributes of this relation.

b) Include the primary key of the *owner* relation as a foreign key attribute in this new relation. The primary key of the new relation is the combination of this primary key of the owner and the partial identifier of the weak entity type.

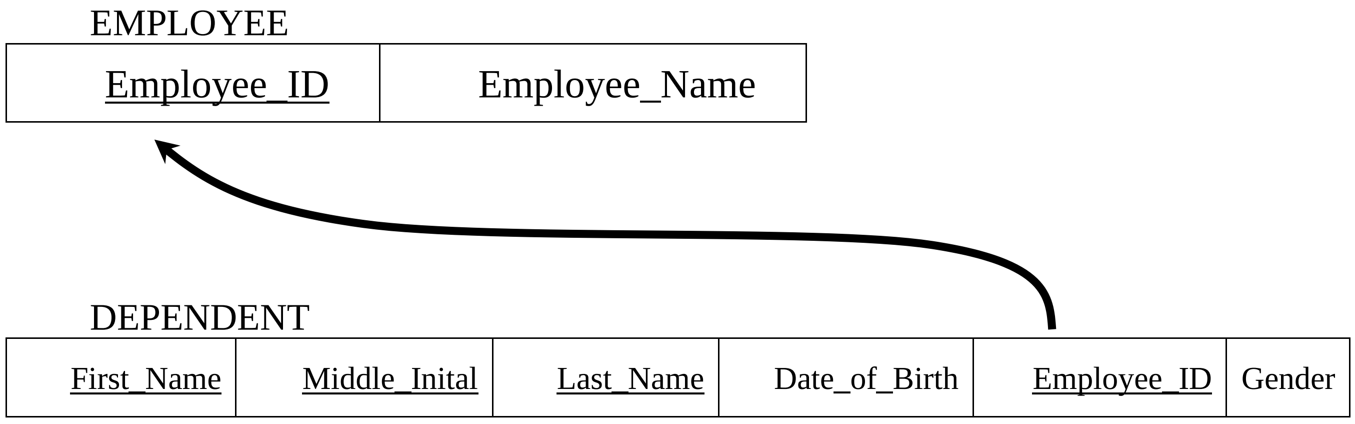
An example of this process is shown in figure bellow; in part(a) we see the weak entity type DEPENDENT and its owner entity type EMPLOYEE, linked by the identifying relationship "Has", notice that the attribute Dependent_name ,which is the partial identifier for this relation , is a composite attribute with components First_Name, Middle_Initial, and Last_Name .

Thus, we assume that *for a given employee* these items will uniquely identify a dependent. In Part (b) we see two relations that result from mapping this E-R segment.
The primary key of DEPENDENT consists of four attributes: Employee_ID, First_Name , Middle_Initial, and Last_Name, Date_of_Birth and GENDER are the nonkey attributes. The foreign key relationship with its primary key is indicated by the arrow in the figure.

Part (a)



Part (b)



EMPLOYEE

Employee_ID	Employee_Name

DEPENDENT

First_Name	Middle_Inital	Last_Name	Date_of_Birth	Employee_ID	Gender

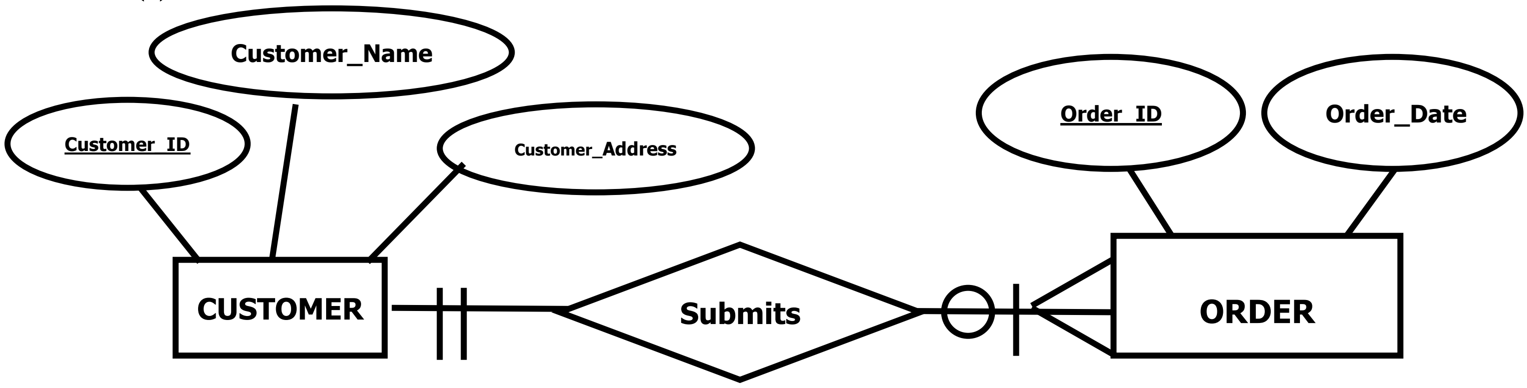
STEP 3 : Map Binary Relationships : The procedure for representing relationships depends on both the degree of the relationships (unary, binary and ternary)and the cardinalities of the relationships.

A : Map Binary One-to-Many Relationships :

a. For each binary 1:M Relationship create a relation for each of the two entity types participating in the relationship using the procedure in Step 1.

b. Include the primary key attribute (or attributes) of the entity on the one-side of the relationship as a foreign key in the relation that is one to many – side of the relationship (to remember: The primary key migrates to the many side)
We illustrate the above rules using the relationships between CUSTOMER and ORDER, which is 1:M relationship, in the figure bellow. The primary key Customer_ID of CUSTOMER (the one side) is included as a foreign key in ORDER(the many -side) .The foreign key relationship is indicated with an arrow.

Part (a)



Part (b)

CUSTOMER

<u>Customer_Id</u>	Customer_Name	Customer_Address

ORDER

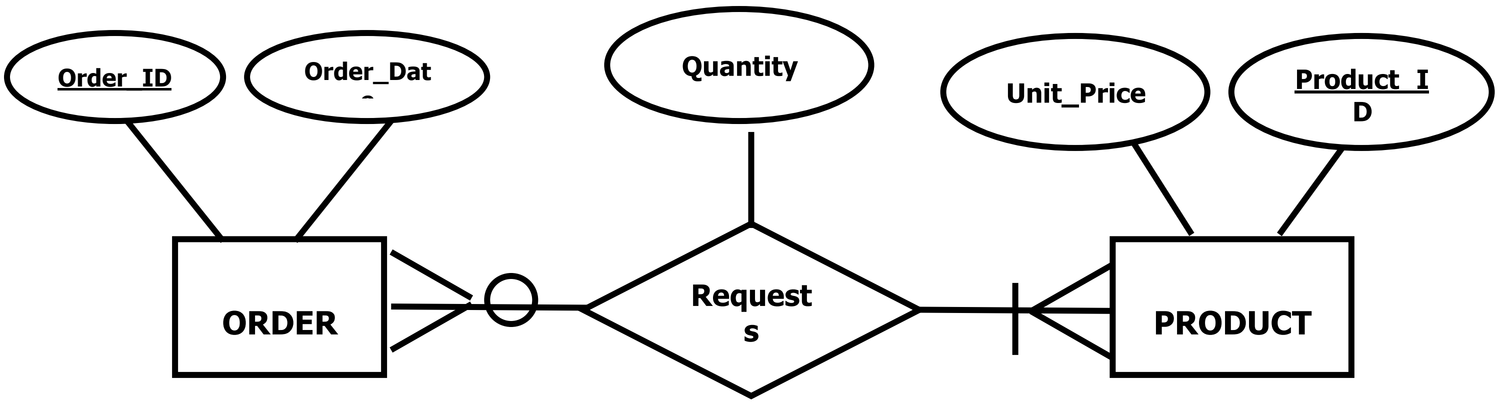
<u>Order_ID</u>	Order_Date	<u>Customer_Id</u>

B: Map Binary Many –to Many Relations:

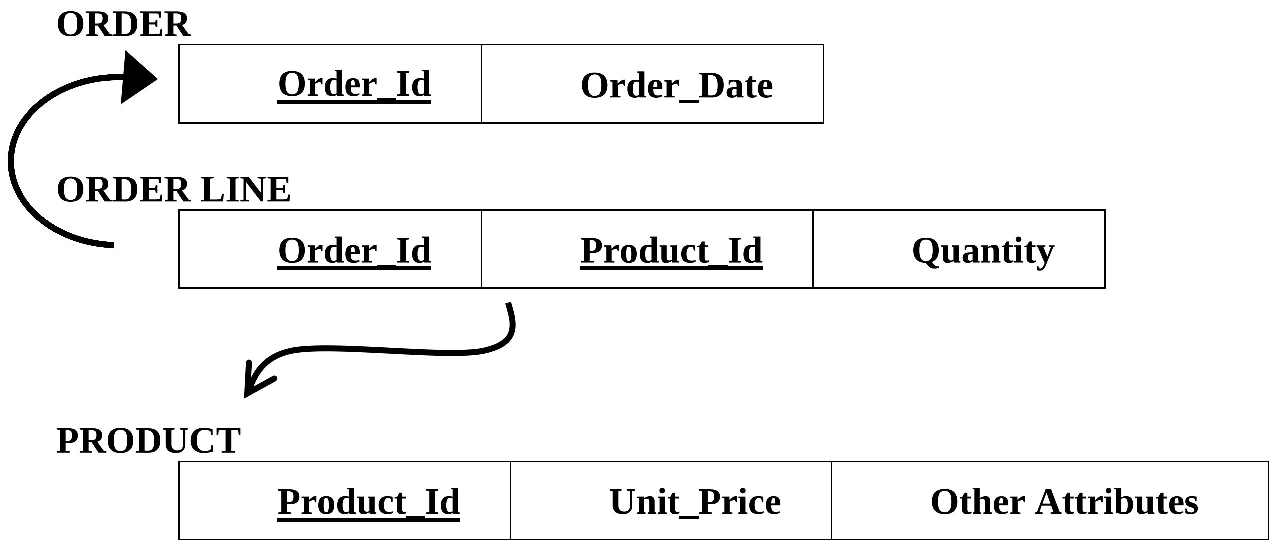
Supposing we have binary (M: N) relationship between two entity types A and B. For such relationship we create a new relation C. Includes as foreign key attributes in C the primary key for each of the two participating entity types. These attributes become the primary key of C. Any non-key attributes are associated with M: N relationships are included with the relation C.

An example of applying this rule is in the figure below, part (a) shows the requests relationship between entity types of ORDER and PRODUCT, part(b) shows three relations (ORDER, ORDER LINE and PRODUCT) that are formed from the entity types and requests relationships. First, a relation created for each of the two regular entity types of ORDER, PRODUCT. Then a relation (named ORDER_LINE) is created for the request's relationship. The primary key of ORDER_LINE is the combination of Order_ID and Product_ID, which are the respective primary keys of ORDER and PRODUCT. As indicated in the diagram, these attributes are foreign keys that "point to "the respective keys. The non-key attribute Quantity also appears in ORDER_LINE.

Part (a)



Part (b)



ORDER

Order_Id	Order_Date

ORDER LINE

Order_Id	Product_Id	Quantity

PRODUCT

Product_Id	Unit_Price	Other Attributes

C: Map Binary One-to one Relationships:

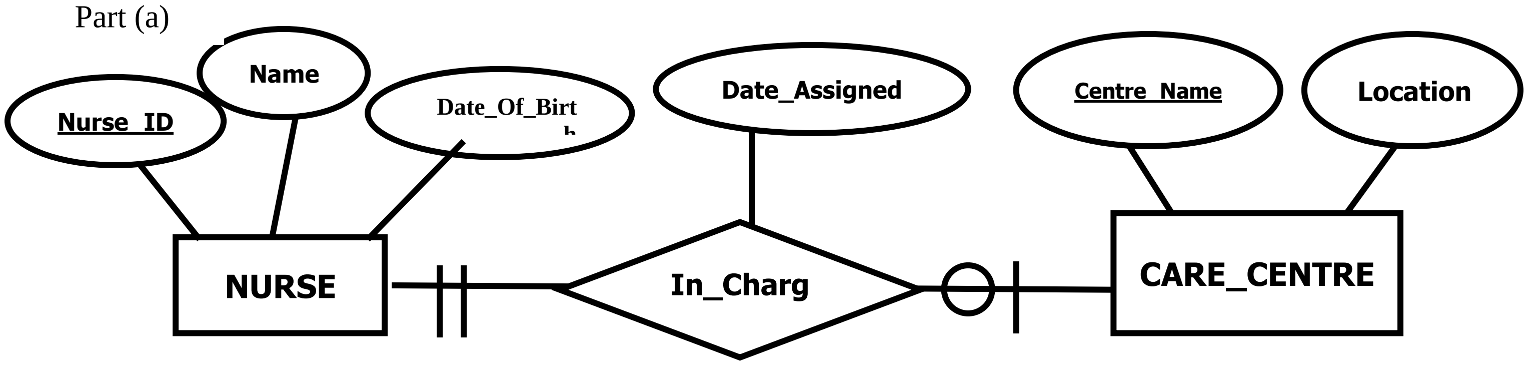
Binary one- to -one relationships can be viewed as a special case of one- to many relationships .The process of mapping such a relationship to relation requests two steps :

1. First two relations are created , one for each of the participating entity types .
2. Second , the primary key of one of the relations is included as a foreign key in the other relation.

In 1: 1 relationship, the association in one direction is nearly always optional one , while the association in the other direction is mandatory one . The foreign key of the entity type that has a mandatory participation in the 1:1 relationship should be included in the relation on the optional side of the , relationship. This approach will avoid the need to store null values in the foreign key attribute .Any attributes associated with the relationship itself are also included in the same relation as the foreign key.

As an example of applying the procedure is shown below, part (a) shows a binary 1:1 relationship between the entity type NURSE and CARE_CENTRE. Each care center must have a nurse who is in charge of that center. Thus, the association from CARE_CENTRE to NURSE is a mandatory one, while the association from NURSE to CARE_CENTRE is an optional one(since any nurse may or may not be in charge of a care center). The attribute Date_Assigned is attached to the In_Charge relationship.
The result of mapping this relationship to a set of relations is shown in part (b). The two relations NURSE and CARE_CENTRE are created from the two entity types. Since CARE_CENTRE is the optional participant, the foreign key is placed in this relation. In this case, the foreign key called Nurse_in_Charge. It has the same domain as Nurse_ID , and the relationship with the primary key is shown in the figure. The attribute Date_assigned is also located in CARE_CENTRE.

Part (a)



Part (b)

NURSE

<u>**Nurse_Id**</u>	**Name**	**Date_Of_Birth**

CARE_CENTRE

<u>**Centre_Name**</u>	**Location**	**Nurse_In_Charge**	**Date_Assigned**

Step 4: Map Associative Entities:

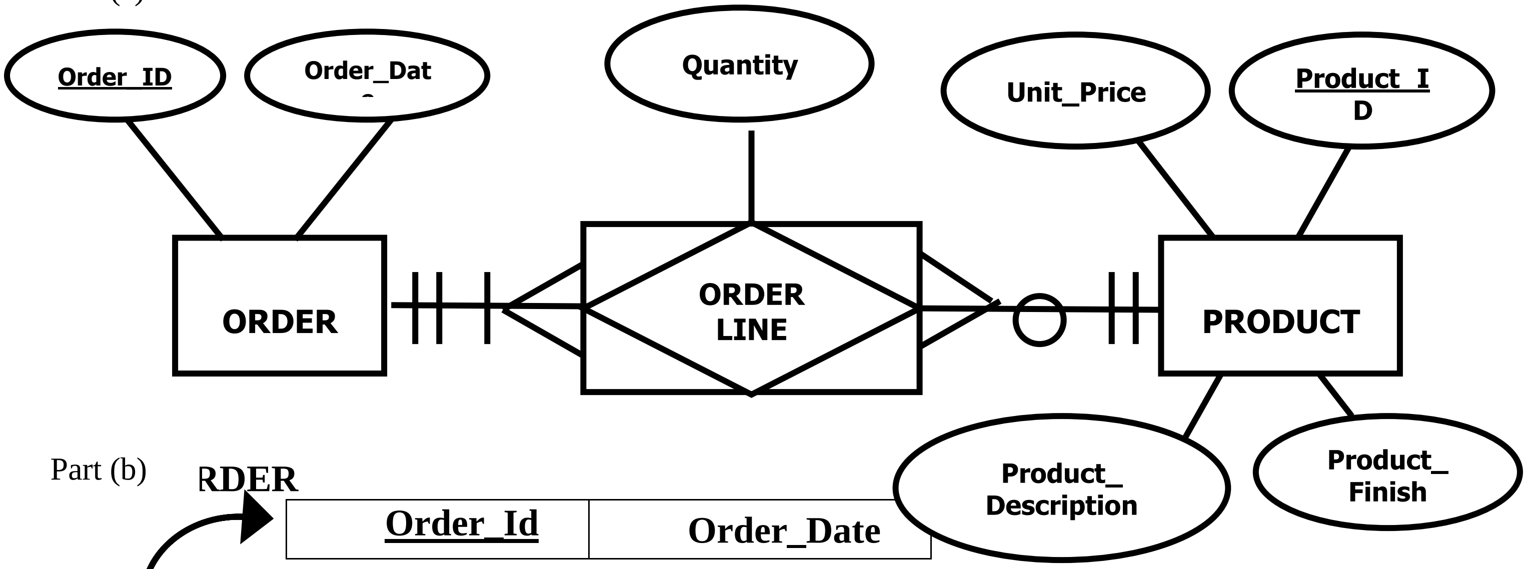
It is a good approach when the end user can best visualize the relationship as an entity type rather than as an M:N relationship.

1. create three relations: one for each of the participating entity types ,and the third for the associative entity. We refer to the relation formed from the associative entity as the *associative relation*.

2. Depending on weather the on the E-R diagram an identifier was assigned to the associative entity.

CASE A: Identifier Not Assigned:
If an identifier was not assigned, the default primary key for the associative relation consists of the two primary key attributes from the other two relations.
These attributes are then foreign keys that reference the other two relations.
An example of this case is shown in figure below ,part(a)shows the associative entity ORDER_LINE that links the ORDER and PRODUCT entity types, part(b) shows the three relations that result from this mapping.

Part (a)



Part (b) RDER

<u>Order_Id</u>	Order_Date

ORDER LINE

<u>Order_Id</u>	<u>Product_Id</u>	Quantity

PRODUCT

<u>Product_Id</u>	Unit_Price	Product_Description	Product_Finish

CASE B: Identifier Assigned:

Sometimes the data modeler will assign an identifier (called *surrogate* identifier or key) to the associative entity type on the E-R Diagram .There are two reasons that may motivate this approach :

1. The associative entity type has a natural identifier that is familiar to the end users.

2. The default identifier (consisting of the identifiers for each of the participating entity types) may not uniquely identify instances of the associative entity.

The process for mapping the associative entity in this case is now modified as follows. As before, a new (associative) relation is created to represent the associative entity. However, the primary key for this relation is the identifier assigned on E-R diagram (rather than the default key).The primary key for the two participating entity types are included as foreign keys in the associative relation.

An example of this process is in the next figure, part (a) shows the associative entity type SHIPMENT that links the CUSTOMER and VENDOR entity types. Shipment_No has been chosen as the identifier for SHIPMENT, for two reasons:

1. Shipment_No is a natural identifier for this entity type that is familiar to end users.

2. The default identifier consisting of the combination of Customer_ID and Vendor_Id does not uniquely identify the instances of SHIPMENT .

In fact, a given vendor will make many shipments to a given customer .

Even including the attribute date does not guarantee uniqueness, since there may be more than one shipment by vendor on a given date , but the surrogate key Shipment_No will uniquely identify each shipment.

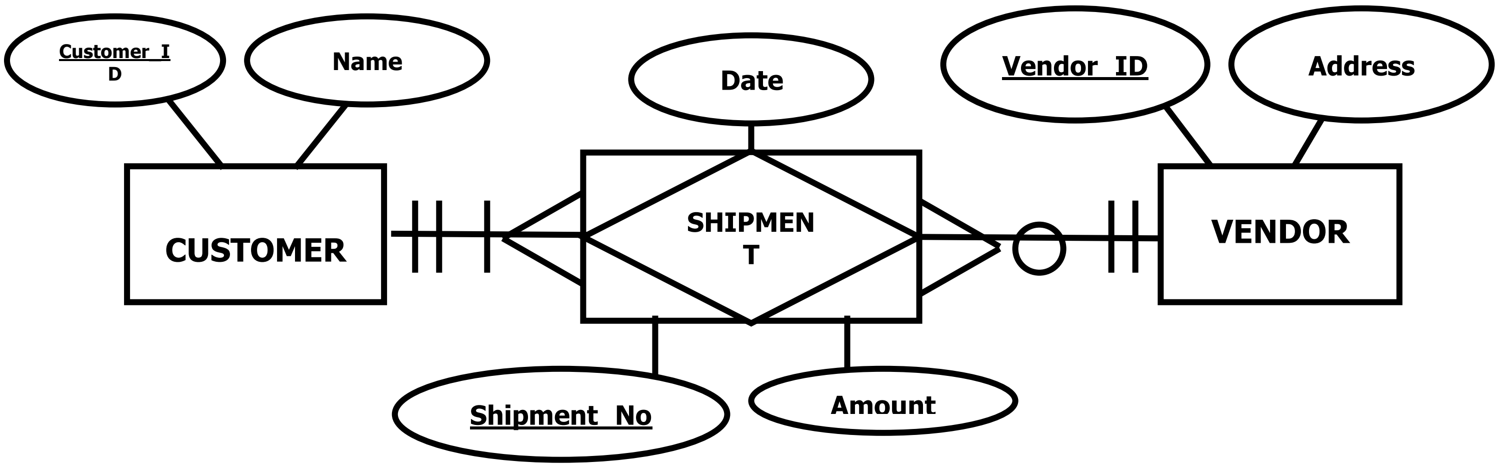
Tow non-key attributes associated with SHIPMENT are Date and Amount .

The result of mapping this entity to a set of relations is in part (b).

The new associative relation is named SHIPMENT.

The primary key is Shipment_No. Customer_ID and Vendor_ID are included as foreign keys in this relation, and Date and Amount are non-key attributes.

Part (a)



Part (b)

CUSTOMER

<u>Customer_Id</u>	Name	Other Attributes

SHIPMENT

<u>Shipment_No</u>	Customer_Id	Vendor_ID	Date	Amount

VENDOR

<u>Vendor_ID</u>	Address	Other Attributes

Step 5: Map Unary Relationships:

Unary relationships = recursive relationships

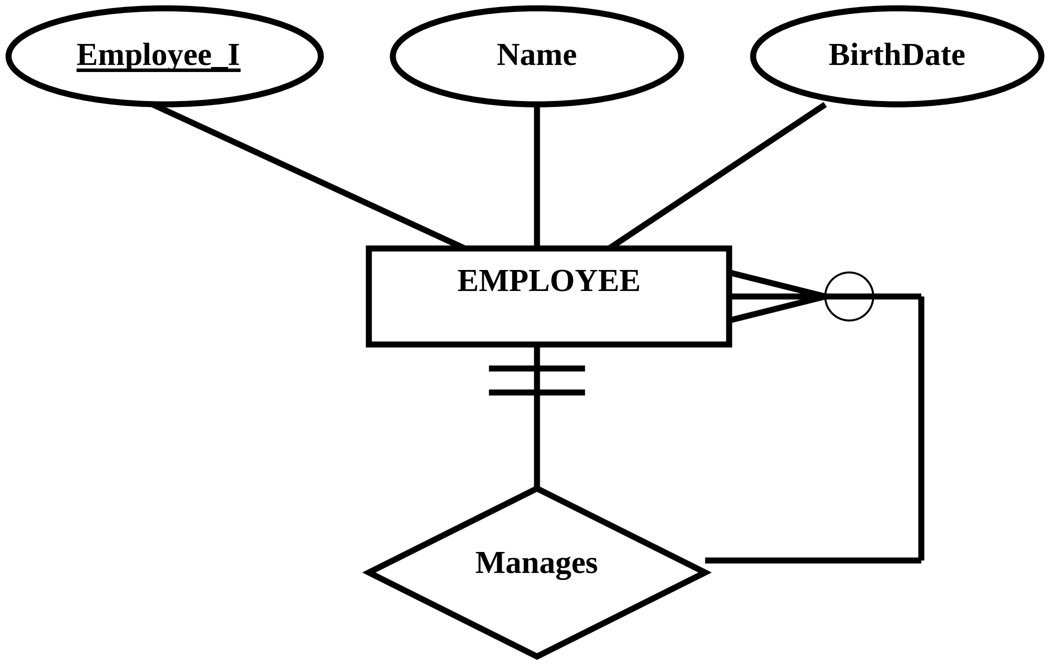
Two most important unary relationship cases are: one-to–many and many–to–many.

1. Unary One- To – Many Relationships:

The entity type in the unary relationship mapped to a relation using the procedure in step (1). Then a foreign key attribute added *within* the same relation that references the primary key values (this foreign key must have the same domain as the primary key). A recursive foreign key is a foreign key in a relation that references the primary key values of that same relation .

The figure below shows in part(a) a unary one –to-many relationships named Manages that associate each employee of an organization with another employee who is his or her manager. Each employee has exactly one manager; a given employee may manage zero to many employees .

Part (a)



The EMPLOYEE relation that results from mapping this entity and relationship is shown in part(b) . The (recursive) foreign key in the relation named Manager_ID. This attribute has the same domain as the primary key Employee_ID.
Each row of this relation stores the following data

Part (b)

EMPLOYEE

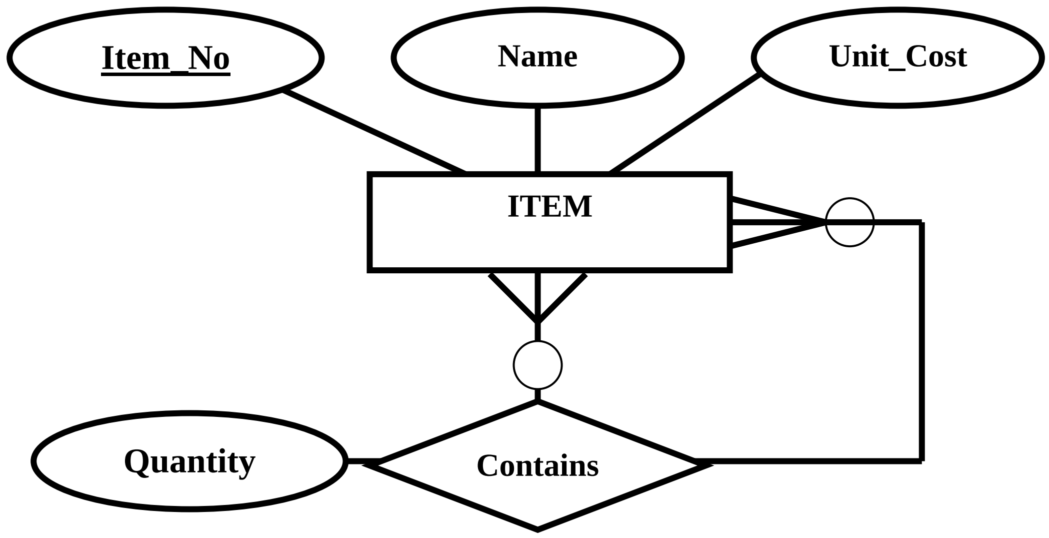
<u>**Employee_ID**</u>	**Name**	**BirthDate**	**Manager_ID**

2. Unary Many-To Many Relationships:

In these type two relations are created: one to represent the entity type in the relation and the other an associative relation to represent the M: N relationship itself. The primary key of the associative relation consists of two attributes .These attributes (Which need not to have the same name) both take their values from the primary key of the other relation. Any non-key attribute of the relationship is included in the associative relation.
An example of mapping a unary M:N relationship is shown in figure below, part(a) shows a bill - -materials among items that are assembled from other items

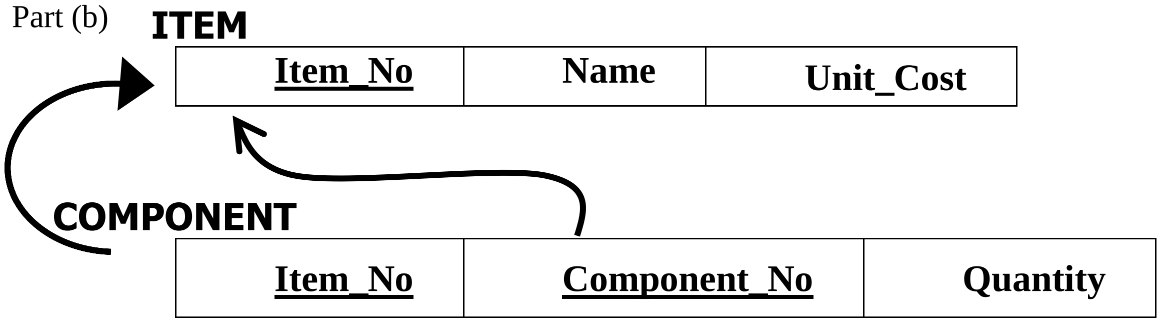
or components, the relationship (called Contains) is M:N since a given item can contain numerous component items, and conversely an item can be used as a component in numerous other items.

Part (a)



The relation that results from mapping this entity and its relationship are shown in part (b). The ITEM relation is mapped directly from the same entity type .COMPONENTS is an associative relation whose primary key consists of two attributes that are arbitrary named Item_No and Component_No . The attribute quantity is non-key attribute of this relation that for a given item, records the quantity of particular component item used in that item. Notice that both Item_No and Component_No reference the primary key (Item_No)of the ITEM relation.

Part (b)



We can easily query the above relation to determine (for example) the components of a given item using SQL query to list the immediate components and their quantity for item number 100:

```
SELECT Component_No,Quantity FROM COMPONENT WHERE Item_No =100
```

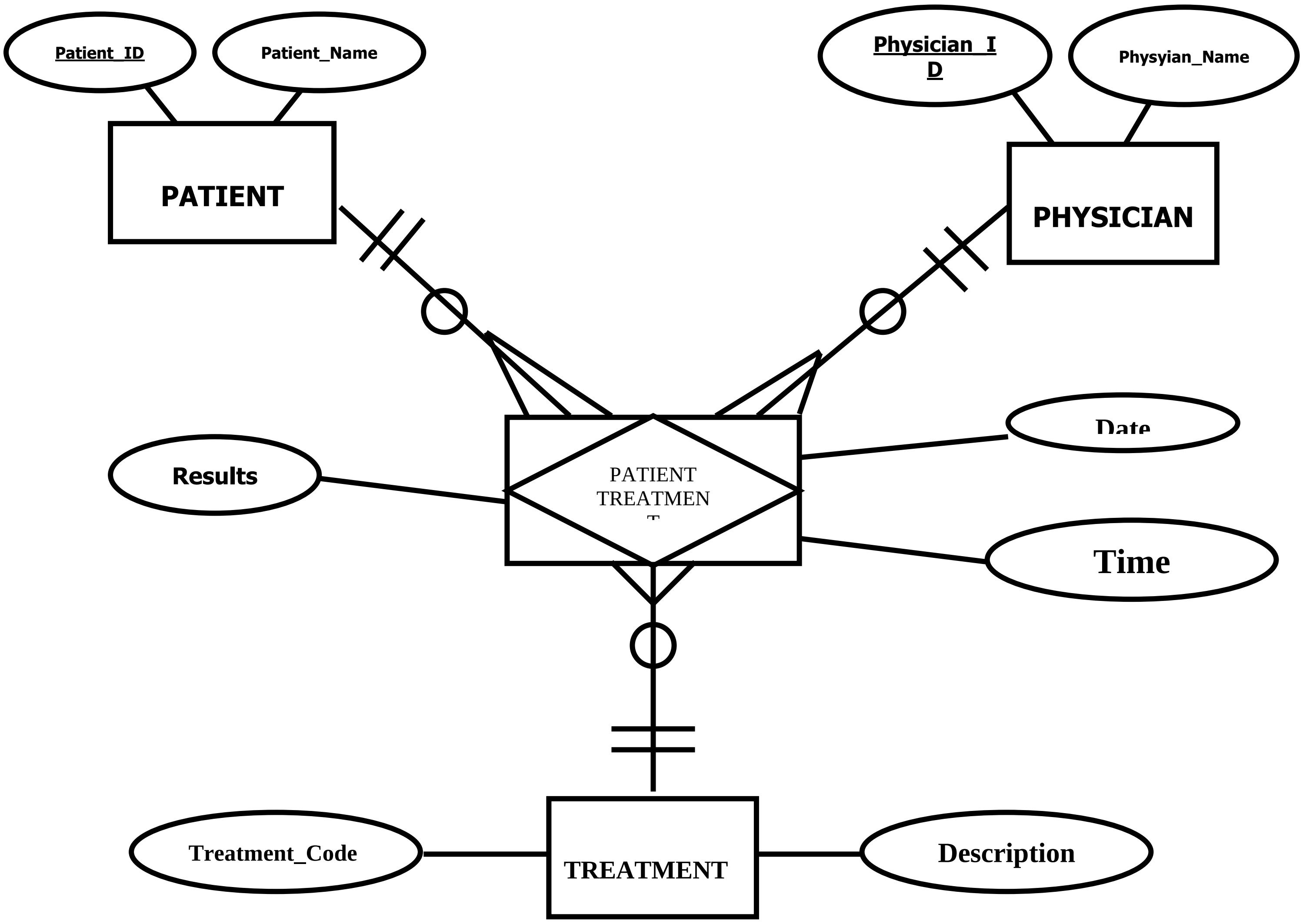
Step 6: Map Ternary (and *N*-ary) Relationships:

Recall that a ternary relationship is a relationship among three entity types.
In fact, it is recommended to convert a ternary relationship to an associative entity in order to represent participation constrains more accurately.

To map associative entity, type that links three regular entity types or more we create a new associative relation the default primary key of this relation consists of the three primary key attributes for the participating entity types (in some cases additional attributes are required to form a unique primary key) . These attributes then act in the role of foreign keys that reference the individual primary keys of the participating entity types any attributes of the associative entity type become attributes of the new relation.

An example of mapping ternary relationship (represented as an associative entity type) is shown in figure below, in part(a) is an E-R segment (or view) that represents *patient* receiving a *treatment* from a *physician* the associative entity type PATIENT TREATMENT has the attributes Date, Time, and results ; values are recorded for these attributes for each instance of PATIENT TREATMENT .

Part (a)



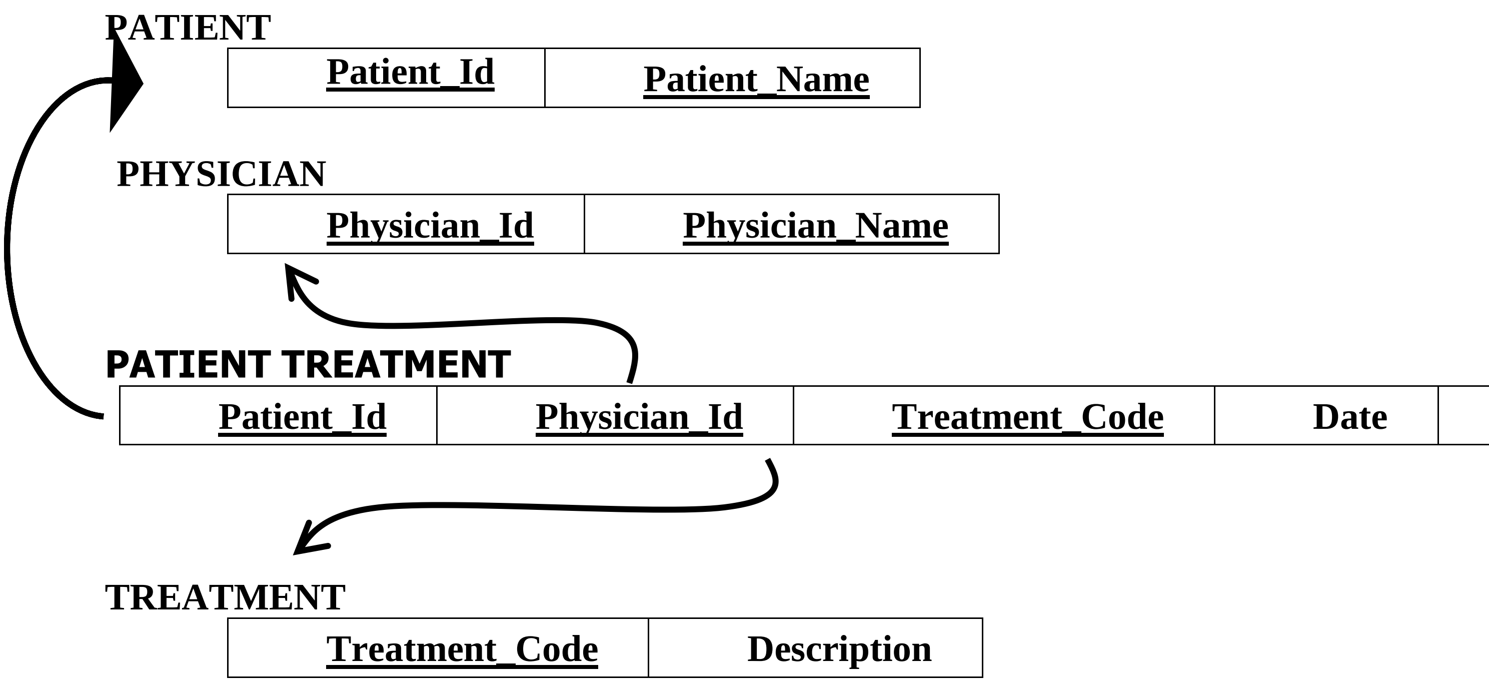
The result of mapping this view is shown in part (b) . The primary key attributes Patient_ID ,Physician_ID,and Treatment_Code become foreign keys in PATIENT TREATMENT. These attributes are components of the primary key of PATIENT TREATMENT.

However, they do not uniquely identify a given treatment since a patient may receive the same treatment from the same physician on more than one occasion .

Does including the attribute Date as part of the primary key (along with the other three attributes) result in primary key ?
This would be so if a given patient receives only one treatment from a particular physician on a given date.

However, this is not likely to be the case. For example, a patient may be receiving a treatment in the morning , then the same treatment in the afternoon .To resolve this issue we include time as a part of the primary key. Therefore ,a primary key of PATIENT TREATMENT consists of the five attributes shown in part(b) : Patient_UD,Physician_ID,Treatment_Code,Date,and Time. The only non-key attribute in the relation is Results.

Part (b)



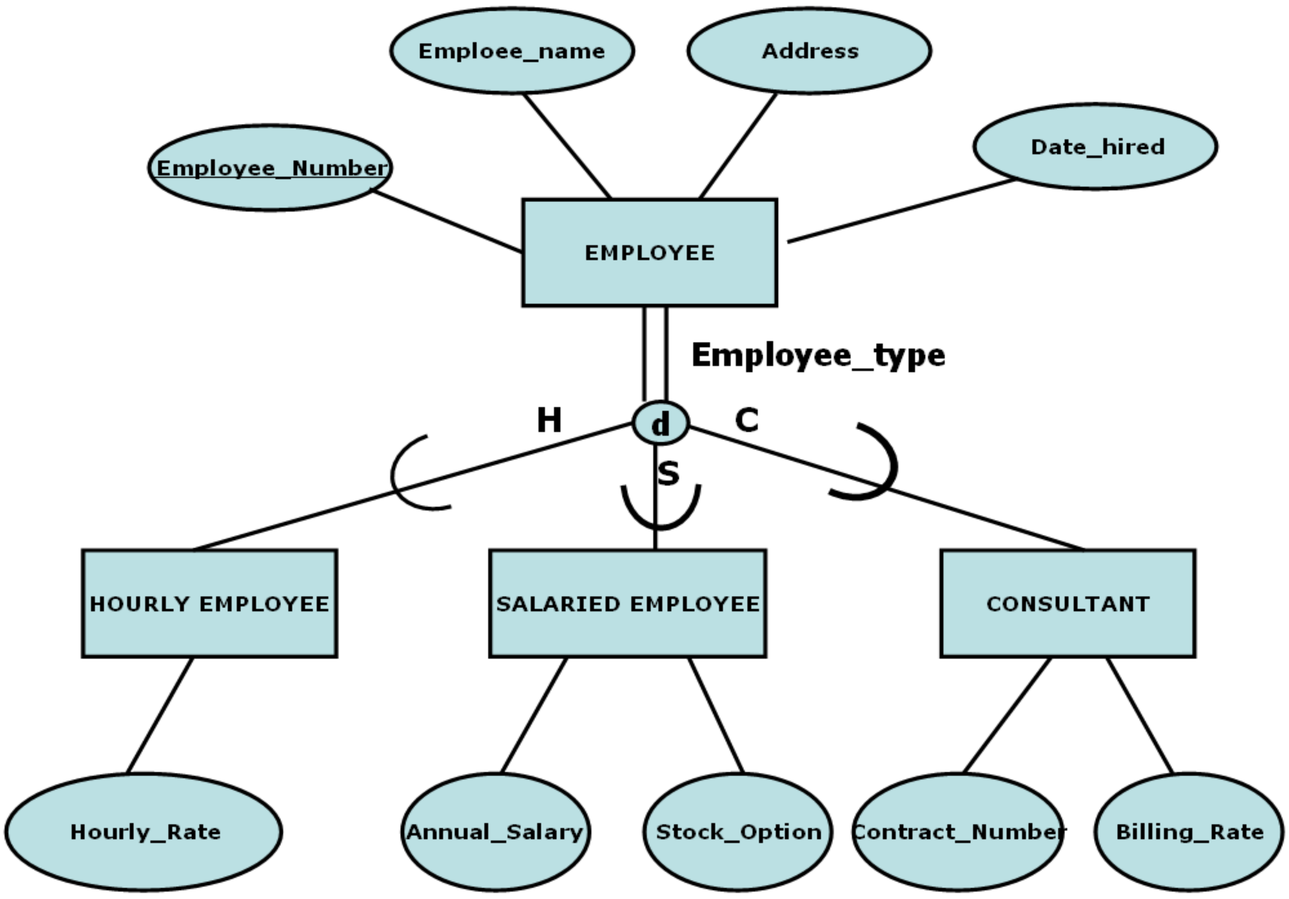
Step 7: Map Supertype/Subtype Relationships

The relational data model does not yet directly support supertype / subtype relationships. Fortunately, there are various strategies that database designers can use to represent these relationships with the relational data model , for our purposes we use the following most used strategy :

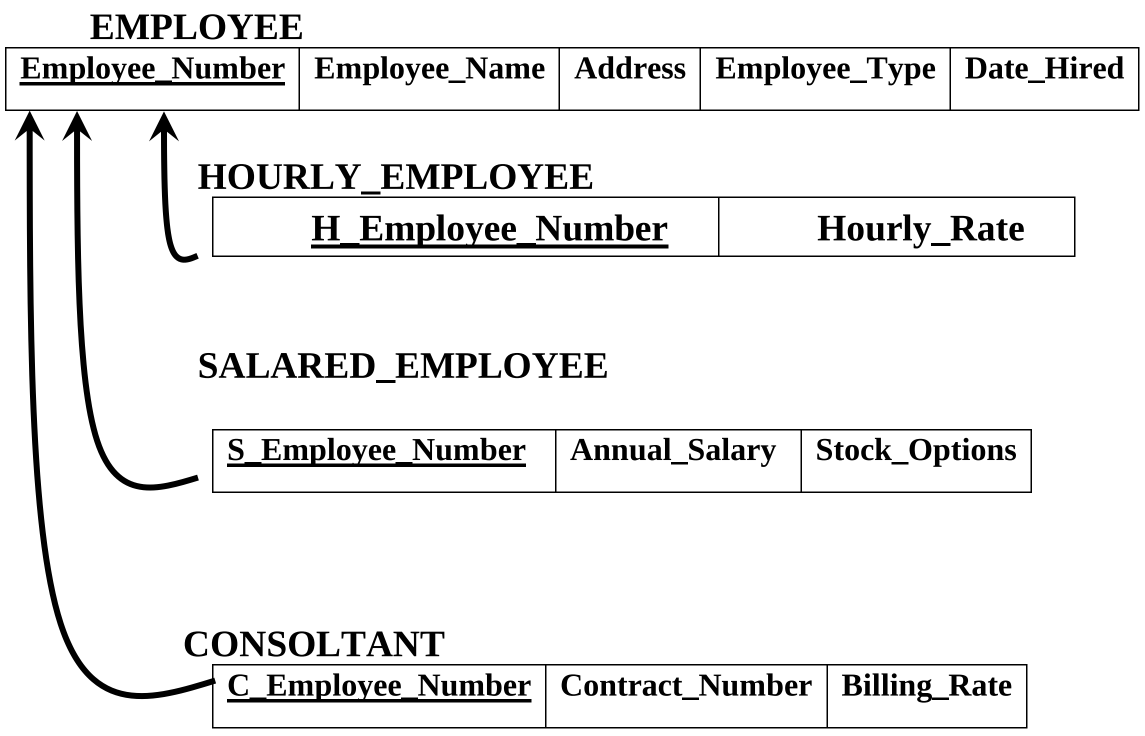
1. Create a separate relation for the supertype and each of its subtypes .

2. Assign to the relation created for the supertype the attributes that are commonly to all members of the supertype, including the primary key.
3. Assign to the relation for each subtype the primary key of the supertype ,and only these attributes that are unique to that subtype type.
4. Assign one (or more) attributes of the supertype to function as the subtype discriminator.

An example of mapping this procedure is shown in figures A and B.
Figure A shows the supertype EMPLOYEE with subtypes HOURLY EMPLOYEE, SALARED EMPLOYEE, and CONSOLTANT.
The primary key of EMPLOYEE is Employee_Number ,and the attribute Employee_Type is the subtype discriminator .



The result of mapping this diagram to relation using the above rule is shown in figure B. There is one relation for the supertype (EMPLOYEE) and one for each of the three subtypes. The primary key for each of the four relations is Employee_Number. A prefix is used to distinguish the name of the primary key for each subtype. For example, S_Employee_Number is the name for primary key of the relation SALARED EMPLOYEE. Each of these attributes is a foreign key that references the supertype primary key, as indicated by arrows in the diagram. Each subtype relation contains only those attributes peculiar to the subtype.



For each subtype, a relation can be produced that contains *all* the attributes of that subtype(both specific and inherited)by using SQL command that joins the subtype with its supertype . For Example, suppose that we want to display a table that contains all of the attributes for SALARED EMPLOYEE. The following command is used :

```
SELECT *
FROM EMPLOYEE, SALARED EMPLOYEE
WHERE Employee_Number = S_Employee_Number;
```

Although you have not yet formally studied such commands, you can intuitively see that this command will join two tables and produce a large table that contains all the attributes from both tables.

13. INTRODUCTION TO NORMALIZATION:

Normalization is a formal process for deciding which attributes should be grouped together in a relation, before we are proceeding with physical design, we need a method to validate the logical design to this point. Normalization is primarily a tool to validate and improve a logical design, so that it satisfies certain constrains that avoid unnecessary duplication of data.

We have presented an intuitive discussion of well-structured relations; however, we need a formal definition of such relations, together with a process for designing them. Normalization is a process of decomposing relations with anomalies to produce smaller, well-structured relations. For example, we use the principles of normalization to convert the EMPLOYEE2 table with its redundancy to (EMPLOYEE1).

13.1. Steps in Normalization:

Normalization can accomplish and understood in stages, each that corresponds to a normal form. see figure below.

A Normal Form is a state of relations that results from applying simple rules regarding functional dependencies (or relationships between attributes) to that relation. We describe these rules briefly in this section and then detail in the following section.

1. First Normal Form.
2. Second Normal Form.
3. Third Normal Form.
4. Boyce/ Codd Normal Form.
5. Fourth Normal Form.
6. Fifth Normal Form.

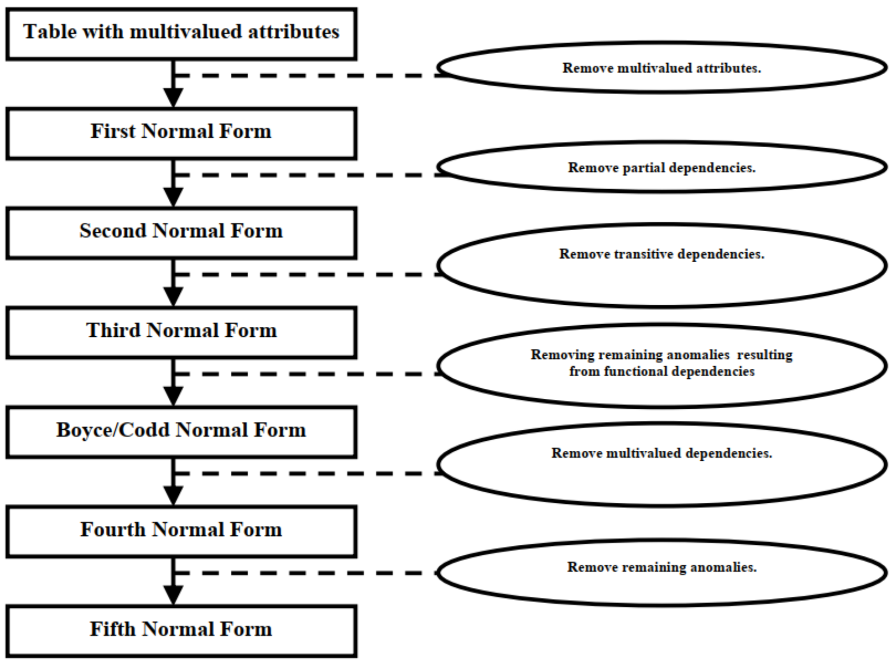
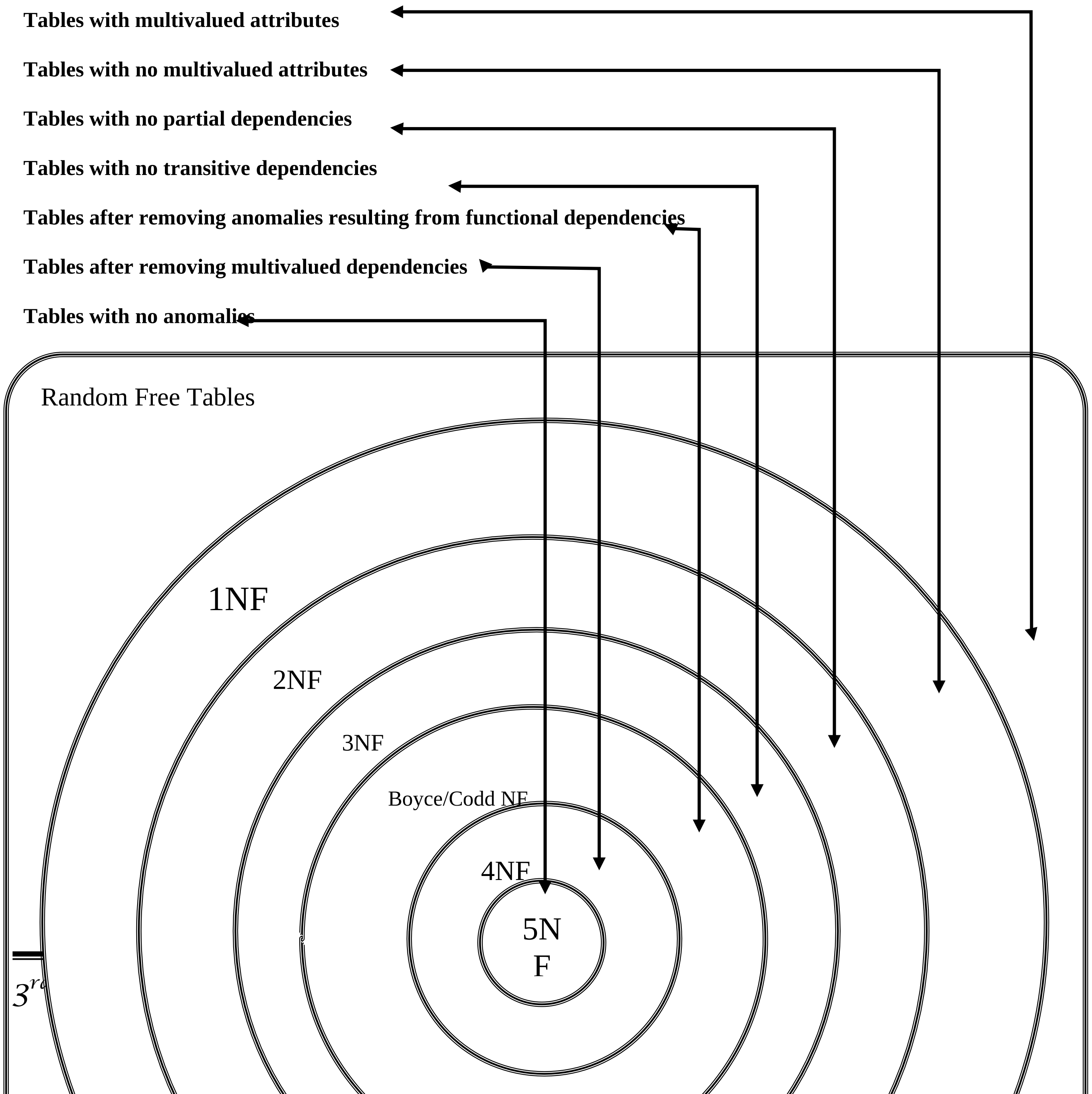


Table with multivalued attributes
Remove multivalued attributes.
First Normal Form
Remove partial dependencies.
Second Normal Form
Remove transitive dependencies.
Third Normal Form
Removing remaining anomalies resulting from functional dependencies
Boyce/Codd Normal Form
Remove multivalued dependencies.
Fourth Normal Form
Remove remaining anomalies.
Fifth Normal Form


Tables with multivalued attributes
Tables with no multivalued attributes
Tables with no partial dependencies
Tables with no transitive dependencies
Tables after removing anomalies resulting from functional dependencies
Tables after removing multivalued dependencies
Tables with no anomalies
Random Free Tables
1NF
2NF
3NF
Boyce/Codd NF
4NF
5NF

13.2. Functional Dependencies and Keys:

Normalization is based on the analysis of functional dependencies.

Functional Dependency: is a constraint between two attributes or two sets of attributes.

For any relation R , attribute B is functionally dependent on attribute A if , for every valid Instance of A ,that value of A uniquely determines the value of B .

The functional dependency of B on A represented by an arrow, as follows: A→ B . An attribute may be functionally dependent on two (or more) attributes , rather than on a single attribute . For Example, consider the relation

EMP_COURSE (Emp_ID, Course_Name,Date_Completed) shown in figure below :

EMP_COURSE

Emp_ID	**Course_Title**	**Date_Completed**
100	SPSS	6/19/2016
100	Cisco	10/7/2016
140	CNC	12/8/2015
110	Tax Acc	1/12/2016
110	SPSS	4/22/2015
150	SPSS	6/16/2015
150	Java	8/12/2016

We represent the functional dependencies in this relation as follows:

Emp_ID, Course_Name→Date_Completed

The functional dependency in this statement implies that the date of a course is completed is completely determined by the identity of the employee and the name of the course . Common examples of functional dependences are the following:

1. SSN → Name, Address, Birthdate A person's name, address, and birthrate are functionally dependent on the person Social Security Number.

2. VIN → Make, Model, Color The make; model, and color of a vehicle are functionally dependent on the Vehicle Identification Number.

3. ISBN → Title, First_Author_Name, The title of a book and the name of the first author are functionally dependent of the book's International Standard Book Number(ISBN).

Determinates: the attribute on the left-hand side of the arrow in a functional dependency is called a determinant. SSN,VIN ,and ISBN are determinates (respectively) in the preceding three examples . In the EMP_COURSE relation, the combination of the Emp_ID and Course_Name is a determinate.

Candidate Keys: A candidate key is an attribute or combination of attributes, that uniquely identifies a row in a relation. A candidate key must satisfy the following properties, which are subset of the six properties of a primary key:

a) ***Unique identification***: for every row, the value of the key must uniquely identify that row *.This property implies that each nonekey attribute is functionally dependent on that key.*

b) ***None-Redundancy*** **:** No attribute in the key can be deleted without destroying the property of unique identification.

By applying the preceding definition to identify candidate keys in two of the relations described in this section. The EMPLOYEE relation has the following schema: EMPLOYEE1 (Emp_ID,Name,Dept_Name,Salary). Emp_ID is the only Determinant in this relation. all of the other attributes are functionally dependent on Emp_ID . Therefor Emp_ID is a candidate key and(since there are no other candidate keys) also is the primary key.

We represent the functional dependences for a relation using the notation shown in figure below ,in part(a) Shows the representation for EMPLOYEE1.

The horizontal line in the figure portrays the functional dependences. A vertical line drops from the primary key(Emp_ID) and connects to this line.

Vertical arrows then point to each of the nonekey attributes that are functionally dependent on the primary key.

For the relation EMPLOYEE2 notice that(unlike EMPLOYEE1),the Emp_ID does not uniquely identify a row in the relation. For Example, there are two rows in the table for Emp_ID 100. There are two functional dependences in this relation.

1.Emp_ID → Name,Dept_Name,Salary

2. Emp_ID,Course_Title→ Date_Completed

The function of dependences indicates that the combination of Emp_ID and Course_Title is the only candidate key(and therefore the primary key) forEMPLOYEE2 in other words the primary key ofEMPLOYEE2 is a composite key . Neither Emp_ID nor Course_Title uniquely identifies a row in this relation ,and therefore by property 1 cannot by itself be a candidate key .

Examine the data in table below to verify that the combination of Emp_ID and Course_Title Does uniquely identify each row of EMPLOYE2.

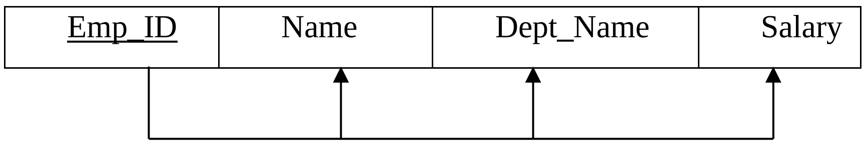
EMPLOYEE2

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Mohammad zaki	Marketing	530.000	SPSS	6/19/2016
100	Mohammad zaki	Marketing	530.000	Cisco	10/7/2016
140	Sufyan salim	Accounting	480.000	CNC	12/8/2015
110	Sinan zohair	Info systems	225.000	Tax Acc	1/12/2016
110	Sinan zohair	Info systems	225.000	SPSS	4/22/2015
190	Muthanna Mohammad	Finance	400.000		
150	Mahmmod khalid	Marketing	75.000	SPSS	6/16/2015
150	Mahmmod khalid	Marketing	75.000	Java	8/12/2016

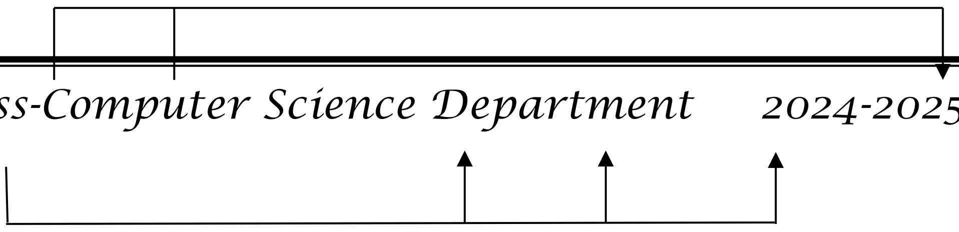
We represent the functional dependencies in this relation in Figure(1) below . Notice that Date Completed is the only attribute that is functionally dependent on the full primary key consisting of the attributes Emp_ID and Course_Title

Part(a) Functional Dependencies in EMPLOYEE1

EMPLOYEE1



Part(b) Functional Dependencies in EMPLOYEE2



EMPLOYEE2

<u>Emp_ID</u>	<u>Course_Title</u>	Name	Dept_Name	Salary	Date_Completed

We can summarize the relationship between the determinants and candidate keys as follows .

A candidate key is always a determinant, while a determinant may or may not be a candidate key. For example, in EMPLOYEE2, Emp_ID is a determinant but not a candidate key.

A candidate key is a determinant that uniquely identifies the remaining (nonekey) attributes in a relation.

A determinant may be candidate key (such as Emp_Id in EMPLOYEE1), part of a composite candidate key such as Emp_ID in EMPLOYEE2)or a nonekey attribute. We will describe examples of this shortly.

13.3. THE BASIC NORMAL FORMS:

1. First Normal Form:

A relation is in first normal form (1NF) if it contains no multivalued attributes.

2. Second Normal Form:

A relation is in second normal form (2NF) if it is in first normal form and every nonekey attribute is fully functionally dependent on the primary key. Thus, no nonekey attribute is functionally dependent on part (but not all) of the primary key: A relation that is in first normal form will be in second normal form if any one of the following conditions applies :

- The primary key consists of only one attribute.
- No nonekey attributes exist in the relation (thus all of the attributes in the relation are components of the primary key)
- Every nonekey attribute is functionally dependent of the full set of primary key attributes.

EMPLOYEE2 is an example of a relation that is not in second normal form . The primary key for this relation is the composite key Emp_ID ,Course_Title. Therefore, the nonekey attribute Name ,dept_Name, Salary are functionally dependent on the part of the primary key (Emp_ID) but not on Course_Title .(Shown before)

A partial functional dependency is a functional dependency in which one or more nonekey attributes (such as Name) are functionally dependent on part (but not all) of the primary key.

The partial functional dependency in EMPLOYEE2 creates redundancy in that relation. Which results in anomalies when the table is updated as we noted before.

To convert a relation to second normal form ,we decompose the relation into new relations that satisfy one (or more) of the conditions described above . EMPLOYEE2 is decomposed into the following two relations:

First : - EMPLOYEE1(Emp_id,Name,Dept_Name,Salary).

This relation satisfies condition 1 above and is in second normal form.

EMPLOYEE1

Emp_ID	**Name**	**Dept_Name**	**Salary**
100	Mohammad zaki	Marketing	530.000
140	Sufyan salim	Accounting	480.000
110	Sinan zohair	Info systems	225.000
190	Muthanna Mohammad	Finance	400.000
150	Mahmmod khalid	Marketing	75.000

Second : - EMP_COURSE(Emp_ID, Course_Title, Date_Completed) this relation satisfies property 3 above and also in second normal form

EMP_COURSE

Emp_ID	**Course_Title**	**Date_Completed**
100	SPSS	6/19/2016
100	Cisco	10/7/2016
140	CNC	12/8/2015
110	Tax Acc	1/12/2016
110	SPSS	4/22/2015
150	SPSS	6/16/2015
150	Java	8/12/2016

3. Third Normal Form:

A relation is in third normal form (3NF) if it is in second normal form and no transitive dependences exist. A transitive dependency in a relation is a functional dependency between two (or more) nonekey attributes, for example consider the relation.

SALES (Cust_ID , Name ,Salesperson, Region)

The sample data of this relation appear in part (a) of figure below.

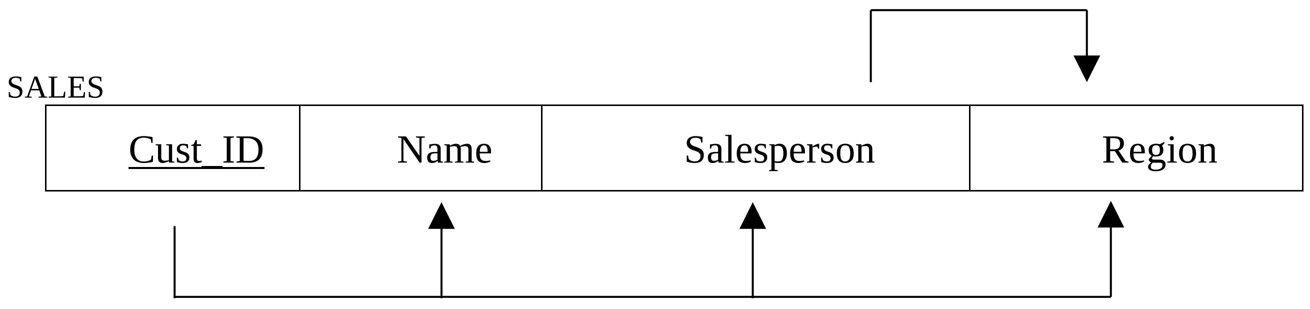
Part (a) SALES relation with sample data:

SALES

<u>**Cust_ID**</u>	**Name**	**Salesperson**	**Region**
8023	Nawzat	Sami	South
9167	Bassam	Jassam	West
7924	Ahmed	Sami	South
6837	Talal	Hadi	East
8596	Emad	Jassam	West
7018	Arkan	Fuaad	North

The functional dependences in SALES are shown graphically in part (b) of figure below :

Part (b) Transitive dependency in SALES relation :



Cust_ID is the primary key so that for all the remaining attributes are functional dependent on this attribute.

However, there is a transitive dependency (Region is functional dependent on Salesperson and Salesperson is functional dependent of Cust_ID as a result there are update anomalies in SALES:

a) ***Insertion Anomaly***: a new salesperson (Riad) assigned the north region cannot be entered until a customer has been assigned to that salesperson (since a value for Cust_ID must be provide to insert a row in the table) .

b) ***Deletion Anomaly***: if a customer number 6837 is deleted from the table , we lose the information that salesperson (Hadi) is assigned to the east region .

c) ***Modification Anomaly*:** if a salesperson (Sami) is assigned to the east region , several rows must be changed to reflect that fact (two rows are shown in Part(a)

These anomalies arise because of a transitive dependency.

The transitive dependency can be removed by decomposing SALES into two relations as shown in figure below Part (a), note that salesperson which is a determinant in the transitive dependency in SALES becomes the primary key in the SPERSON, Salesperson becomes a foreign key in SALES1.

As shown in figure Part(b) the new relations are now in third normal form since no transitive dependencies exists. you should verify that the anomalies that exists in SALES are not presented in SALES1 and SPERSON .

Part (a) decomposing SALES relation

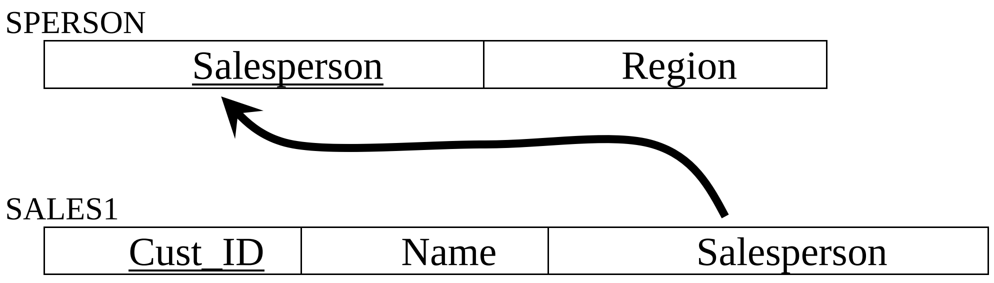
SALES1

<u>Cust_ID</u>	**Name**	**Salesperson**
8023	Nawzat	Sami
9167	Bassam	Jassam
7924	Ahmed	Sami
6837	Talal	Hadi
8596	Emad	Jassam
7018	Arkan	Fuaad

SPERSON

<u>Salesperson</u>	**Region**
Sami	South
Jassam	West
Hadi	East
Fuaad	North

Part (b) Relation in 3NF :



Transitive dependencies may also occur between sets of attributes in relation for example the relation SHIPMENT (Snum, Origin, Destination, Distance) could be used to record shipments according to origin, the destination , and distance . Sample data for this relation appear in figure below Part (a). the functional dependencies in the shipment relation are shown in Part(b). The primary key of SHIPMENT relation is the attribute Snum(or shipment number) .

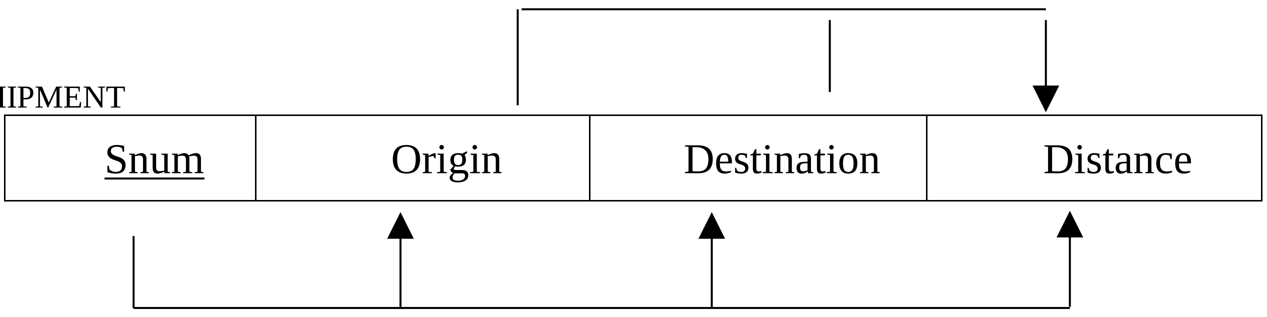
Part (a)SHIPMENT relation with sample data:

SHIPMENT

<u>Snum</u>	Origin	Destination	Distance
49	Seattle	Denver	1.537
618	Chicago	Dallas	1.053
723	Boston	Atlanta	1.214
824	Denver	Los angeles	1.150
629	Minneapolis	St Louis	587

Part (b) Functional dependence in SHIPMENT relation:

SHIPMENT



As a result, we now that the relation is in second normal form, (Why?). However, there is a transitive dependency in this relation: the distance attribute is a functional dependent on a pair of nonekey attributes origin and destination. as a result there are anomalies in SHIPMENT (as an example you should examine figure Part(a) and identify insertion , deletion , and modification anomalies). We can the transitive dependency in SHIPMENT by decomposing it into two relations in third normal form (3NF).

SHIPTO (<u>Snum</u>, Origin, Destination)

DISTANCES (<u>Origin</u>, <u>Destination</u>, Distance)

The first of these relations provides the origin and destination for any given shipment, the second provides distance between an origin and destination pair.

Sample data for this relation appear in figure Part (c)

Part (C) Relation in 3NF :

SHIPMENT

<u>Snum</u>	Origin	Destination

DISTANCES

<u>Origin</u>	<u>Destination</u>	Distance
Seattle	Denver	1.537
Chicago	Dallas	1.053
Boston	Atlanta	1.214
Denver	Los Angeles	1.150
Minneapoli	St Louis	587

49	Seattle	Denver
618	Chicago	Dallas
723	Boston	Atlanta
824	Denver	Los Angeles
629	Minneapolis	St Louis

4. BOYCE-CODD NORMAL FORM

When a relation has more than one candidate key, anomalies may result even though that relation is in 3NF .

For example, consider the STUDENT ADVISOR relation shown in Figure B-1.

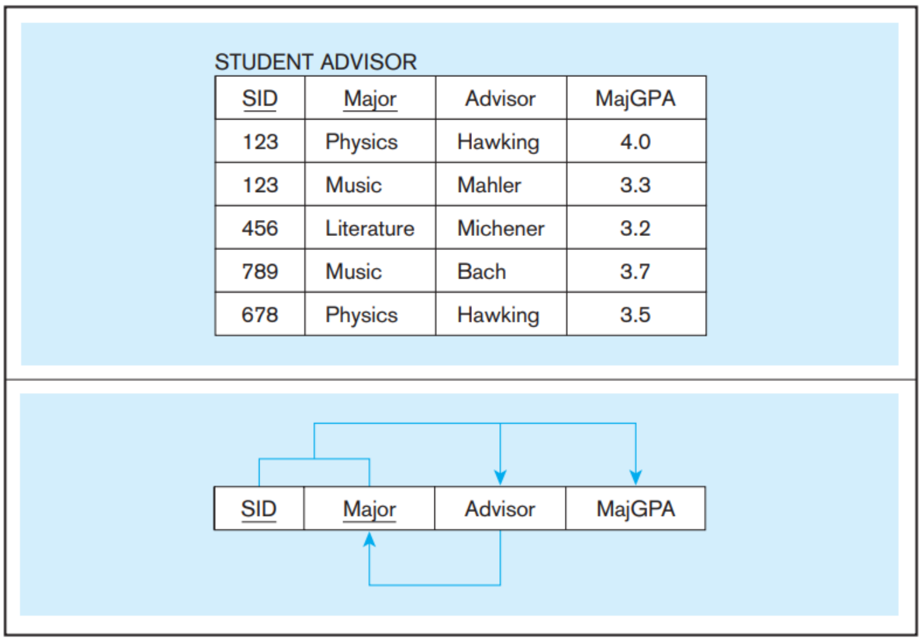


FIGURE B-1 Relation in 3NF but not in BCNF
(a) Relation with sample data

(b) Functional dependencies in STUDENT ADVISOR

This relation has the following attributes: SID (student ID), Major, Advisor, and MajGPA .

Sample data for this relation are shown in Figure B-1a, and the functional dependencies are shown in Figure B-1b.

- As shown in Figure B-1b, the primary key for this relation is the composite key consisting of SID and Major.
- Thus, the two attributes Advisor and MajGPA are functionally dependent on this key.

This reflects the constraint that although a given student may have more than one major, for each major a student has exactly one advisor and one GPA.

- There is a second functional dependency in this relation: Major is functionally dependent on Advisor.
- That is, each advisor advises in exactly one major.
- Notice that this is not a transitive dependency, where a transitive dependency is a functional dependency between two nonekey attributes.

In contrast, in this example a key attribute (Major) is functionally dependent on a nonekey attribute (Advisor)

Anomalies in STUDENT ADVISOR

The STUDENT ADVISOR relation is clearly in 3NF because there are no partial functional dependencies and no transitive dependencies.

Nevertheless, because of the functional dependency between Major and Advisor, there are anomalies in this relation.

Consider the following examples:

1. Suppose that in Physics, the advisor Hawking is replaced by Einstein. This change must be made in two (or more) rows in the table (update anomaly).

2. Suppose we want to insert a row with the information that Babbage advises in Computer Science. This, of course, cannot be done until at least one student majoring in Computer Science is assigned Babbage as an advisor (insertion anomaly).

3. Finally, if student number 789 withdraws from school, we lose the information that Bach advises in Music (deletion anomaly).

Definition of Boyce-Codd Normal Form (BCNF)

- The anomalies in STUDENT ADVISOR result from the fact that there is a determinant (Advisor) that is not a candidate key in the relation. R. F. Boyce and

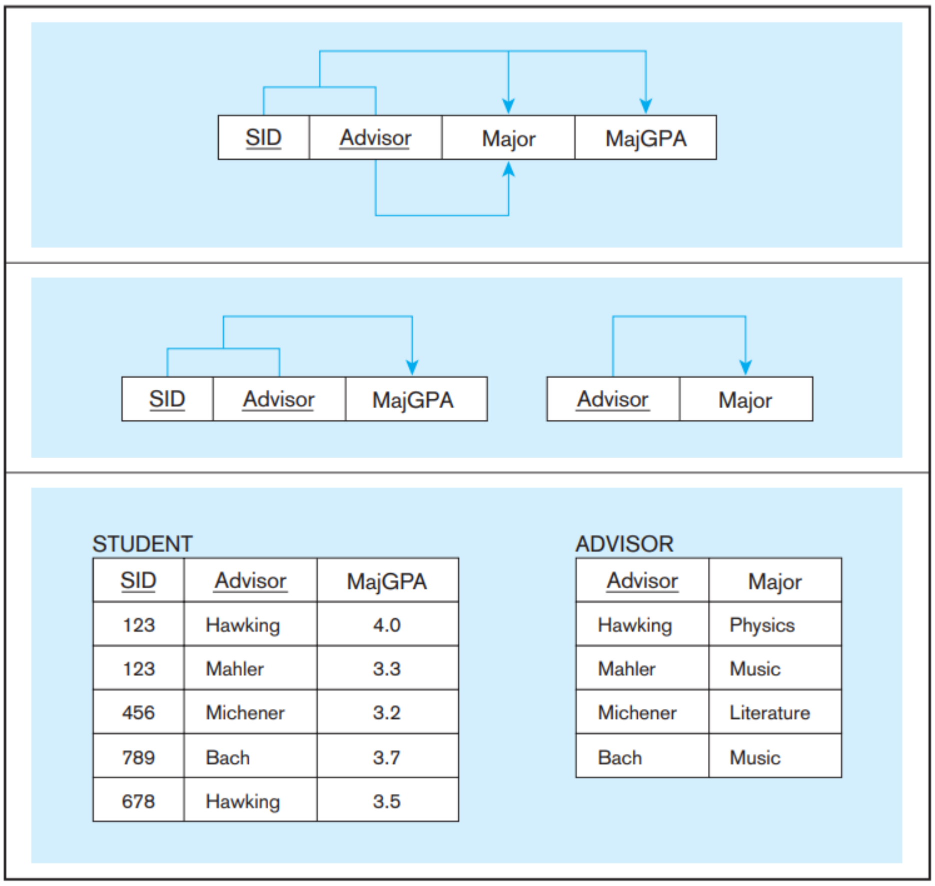
E. F. Codd identified this deficiency and proposed a stronger definition of 3NF that remedies the problem.

- We say a relation is in Boyce-Codd normal form (BCNF) if and only if every determinant in the relation is a candidate key. STUDENT ADVISOR is not in BCNF because although the attribute Advisor is a determinant, it is not a candidate key. (Only Major is functionally dependent on Advisor.)

Boyce-Codd normal form (BCNF): A normal form of a relation in which every determinant is a candidate key.

Converting a Relation to BCNF:

- A relation that is in 3NF (but not BCNF) can be converted to relations in BCNF using a simple two-step process. This process is shown in Figure B-2.



STUDENT

SID	Advisor	MajGPA
123	Hawking	4.0
123	Mahler	3.3
456	Michener	3.2
789	Bach	3.7
678	Hawking	3.5

ADVISOR

Advisor	Major
Hawking	Physics
Mahler	Music
Michener	Literature
Bach	Music

FIGURE B-2 Converting a relation to BCNF relations
(a) Revised STUDENT ADVISOR relations (1NF)
(b) Two relations in BCNF
(c) Relations with sample data

A. In the first step, the relation is modified so that the **determinant** in the relation that is not a candidate key becomes a **component of the primary** key of the revised relation.

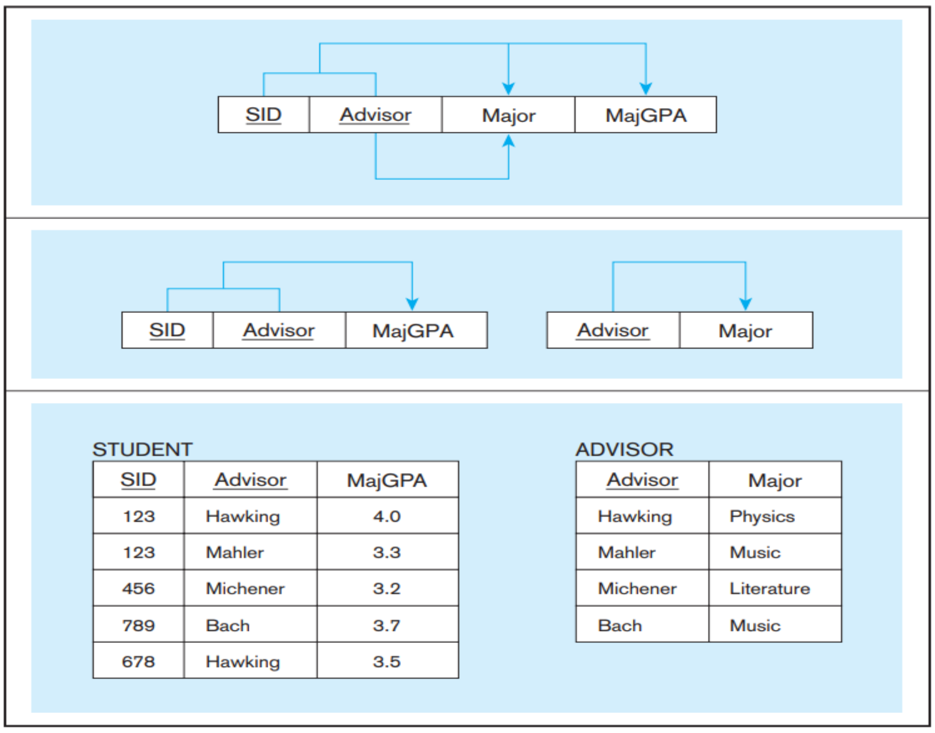
The attribute that is functionally dependent on that determinant becomes a nonekey attribute.

- This is a legitimate restructuring of the original relation because of the functional dependency.
  The result of applying this rule to STUDENT ADVISOR is shown in Figure B-2a. **<u>The determinant Advisor</u>** becomes part of the **<u>composite primary key</u>**. The attribute Major, which is functionally dependent on Advisor, becomes a nonekey attribute.

- If you examine Figure B-2a, you will discover that the new relation has a partial functional dependency. (Major is functionally dependent on Advisor, which is just one component of the primary key.) Thus, the new relation is in first (but not second) normal form.

B. The second step in the conversion process is to decompose the relation to eliminate the partial functional dependency, as we learned earlier.

This results in two relations, as shown in Figure B-2b. These relations are in 3NF. In fact, the relations are also in BCNF because there is only one candidate key (the primary key) in each relation.



STUDENT

SID	Advisor	MajGPA
123	Hawking	4.0
123	Mahler	3.3
456	Michener	3.2
789	Bach	3.7
678	Hawking	3.5

ADVISOR

Advisor	Major
Hawking	Physics
Mahler	Music
Michener	Literature
Bach	Music

FIGURE B-2 Converting a relation to BCNF relations
(a) Revised STUDENT ADVISOR relations (1NF)
(b) Two relations in BCNF
(c) Relations with sample data

Thus, we see that if a relation has only one candidate key (which therefore becomes the primary key), 3NF and BCNF are equivalent.

The two relations (now named STUDENT and ADVISOR) with sample data are shown in Figure B-2c.

5. The Fourth Normal form

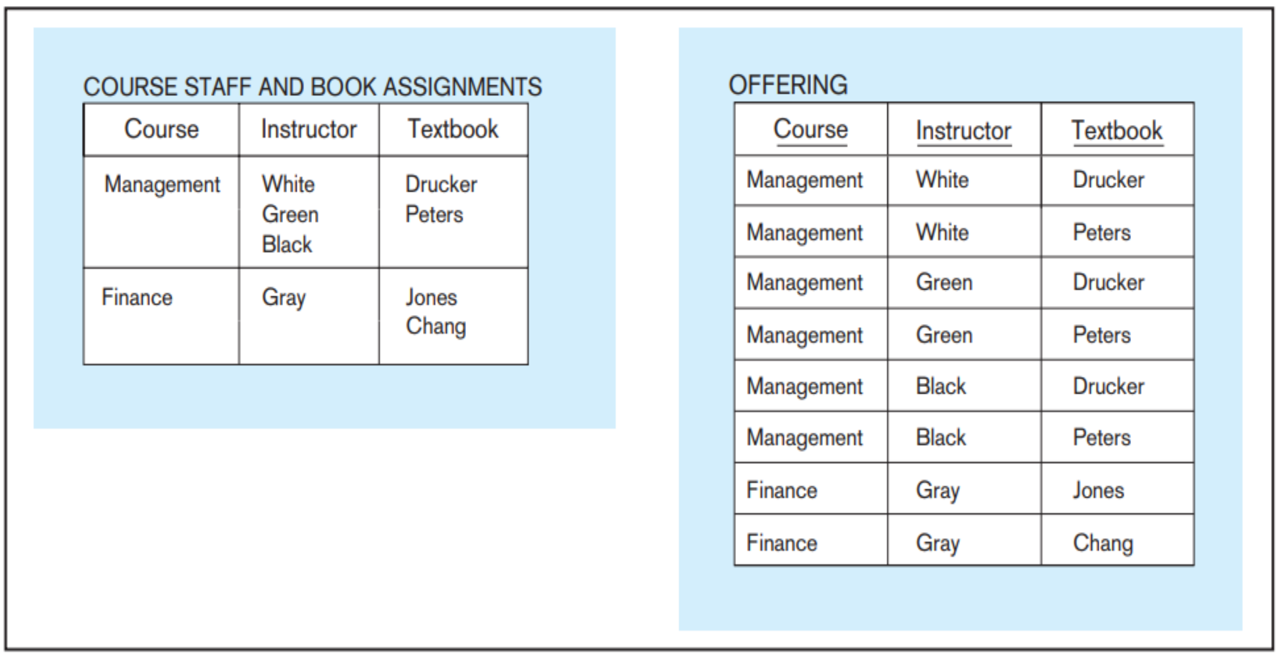
The study of fourth normal form 4NF is focusing on what so called (Multivalued Dependency)

When a relation is in BCNF, there are no longer any anomalies that result from functional dependencies. However, there may still be anomalies that result from multivalued dependencies.

For example, consider the user view shown in Figure B-4a.

FIGURE B-4 Data with multivalued dependencies

(a) View of courses, instructors, and textbooks (b) Relation in BCNF



COURSE STAFF AND BOOK ASSIGNMENTS

Course	Instructor	Textbook
Management	White Green Black	Drucker Peters
Finance	Gray	Jones Chang

OFFERING

Course	Instructor	Textbook
Management	White	Drucker
Management	White	Peters
Management	Green	Drucker
Management	Green	Peters
Management	Black	Drucker
Management	Black	Peters
Finance	Gray	Jones
Finance	Gray	Chang

This user view shows for each course the instructors who teach that course and the textbooks that are used. (These appear as repeating groups in the view.)

In that table view, the following assumptions hold:

a) Each course has a well-defined set of instructors (e.g., Management has three instructors).

b) Each course has a well-defined set of textbooks that are used (e.g., Finance has two textbooks).

c) The textbooks that are used for a given course are independent of the instructor for that course (e.g., the same two textbooks are used for Management regardless of which of the three instructors is teaching Management).

In Figure B-4b, this table view has been converted to a relation by filling in all of the empty cells.

This relation (named OFFERING) is in 1NF. Thus, for each course, all possible combinations of instructor and text appear in OFFERING.

Notice that the primary key of this relation consists of all three attributes (Course, Instructor, and Textbook). Because there are no determinants other than the primary key, the relation is in BCNF.

Yet it does contain much redundant data that can easily lead to update anomalies.

For example, suppose that we want to add a third textbook (author: Middleton) to the Management course.

This change would require the addition of three new rows to the relation in Figure B-4b, one for each Instructor (otherwise that text would apply to only certain instructors)

Multivalued Dependencies

The type of dependency that called a multivalued dependency, it exists when there are at least three attributes (e.g., A, B, and C) in a relation, and for each value of A there is a well-defined set of values of B and a well-defined set of values of C.

However, the set of values of B is independent of set C,and vice versa.

To remove the multivalued dependency from a relation, we divide the relation into two new relations. Each of these tables contains two attributes that have a multivalued relationship in the original relation.

Removing Multivalued Dependency

To remove the multivalued dependency from a relation, we divide the relation into two new relations.

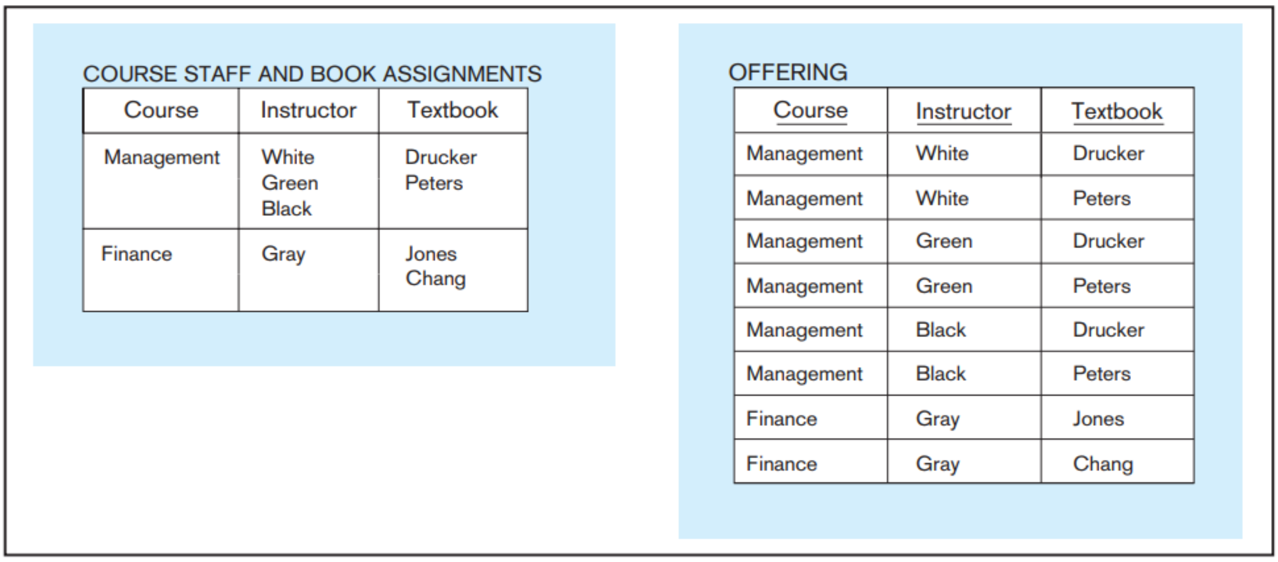
Each of these tables contains two attributes that have a multivalued relationship in the original relation.

Figure B-5 shows the result of this decomposition for the OFFERING relation of Figure B-4b. Notice that the relation called TEACHER contains the Course and Instructor attributes, because for each course there is a well-defined set of instructors.

Also, for the same reason, TEXT contains the attributes Course and Textbook.

However, there is no relation containing the attributes Instructor and Course because these attributes are independent.

FIGURE B-4 Data with multivalued dependencies



(a) View of courses, instructors, and textbooks

COURSE STAFF AND BOOK ASSIGNMENTS

Course	Instructor	Textbook
Management	White Green Black	Drucker Peters
Finance	Gray	Jones Chang

(b) Relation in BCNF

OFFERING

Course	Instructor	Textbook
Management	White	Drucker
Management	White	Peters
Management	Green	Drucker
Management	Green	Peters
Management	Black	Drucker
Management	Black	Peters
Finance	Gray	Jones
Finance	Gray	Chang

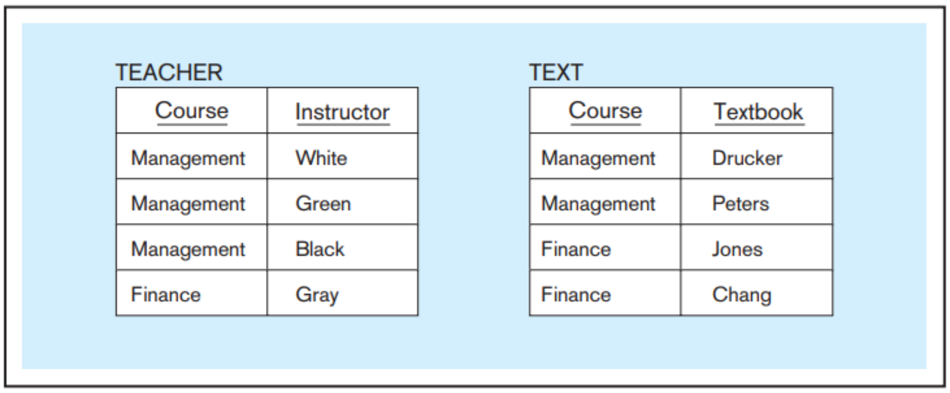


FIGURE B-5 Relations in 4NF

TEACHER

Course	Instructor
Management	White
Management	Green
Management	Black
Finance	Gray

TEXT

Course	Textbook
Management	Drucker
Management	Peters
Finance	Jones
Finance	Chang

Definition of Fourth Normal Form

We can define 4NF as : A normal form of a relation in which the relation is in BCNF and contains no multivalued dependencies.

14.MERGING RELATIONS:

As a part of logical design process , normalized relations may have been created from number of separated E-R Diagrams and (possibly) other users' views , some of the relations may redundant as ; that is ,they may refer to the same entities . If so, we should merge those relations to remove redundancy. This section describes merging relations also called (*view integration*). An understanding of how merge relations is important for three reasons:

1. On large projects the work of several sub teams becomes together during logical design , so there is a need to merge relations .

2. Integrating existing databases with new information requirements often leads to the need to integrate different views.

3. New data requirements may arise during life cycle, so there is a need to merge any new relations with what has already been developed.

An example:

Suppose that modeling user view results in the following 3NF relation:

EMPLOYEE1 (<u>Employee_ID</u>, Name , Address, Phone)

Modeling a second user view might result in the following relation:

EMPLOYEE2 (<u>Employee_ID</u>, Name , Address, Jobcode,No_years)

Since this two relations have the same primary key (Employee_ID) ,they likely describe the same entity and may be merged into one relation , the resulting of merging the relations is in the following relation :

EMPLOYEE(<u>Employee_ID</u>, Name , Address, Phone, Jobcode,No_years)

Notice that the attributes in both relations (such as Name in this example) appear only once in the merged relations.

14.1. View Integration Problems:

When integration relations as in the preceding example ,the database analyst must understand the meaning of the data must be prepared resolve any problems that may arise in that process. In this section we describe a briefly illustrate for problems that arise in view integration synonyms , homonyms, transitive dependences and supertype/subtype relationship,

<u>1. Synonyms:</u> in some situations, two(or more) attributes may have different names but the same meaning ,as when they describe the same characteristics of an entity . Such attributes called synonyms for example Employee_ID and Employee_Number may be synonyms when merging relations that contains synonyms you should obtain agreement of (if possible) from user on a single, standardized name for the attribute and eliminate others synonyms .Another alternative is to choose a third name to address the synonyms , for example consider the following two relations :

STUDENT1 (<u>Student_ID</u>,Name)

STUDENT2 (<u>Matriculation_Number</u> ,Name, Address)

In this case the analyst realized that both Student_ID and Matriculation_Number are synonyms for Social Security Number (SSN) are identical attributes .

One possible solution would be standardized on the two attributes names such as Student_ID .

Another option is to use a new attribute name such as SSN, to replace both synonyms . Assuming the later approach merging the two relations would produce the following results :

STUDENT (<u>SSN</u>, Name, Address)

Often when there are synonyms there is a need to allows some database users to refer to the same data by different names , users may need to use familiar names that are consistent with terminology of there part of the organization . an alias is an alternative name used for an attribute . Many database management systems allow the definition of alias that may be used interchangeably with primary attribute label.

<u>2. Homonyms</u> : an attribute that may have more than one meaning is called a Homonyms , for example the term account might refers to bank's checking account , saving account , loan account , or other type of account, (therefore account may refer to the different data , depending on how it is used) .

You should be lockout for homonyms when merging relations , consider the following example.

STUDENT1 (<u>Student_ID</u>,Name, Address)

STUDENT2 (<u>Student_ID</u> ,Name, Phone_No, Address)

In discussions with user the analyst discovered that the attribute Address in STUDEN1 refers to student campus address , while in STUDENT2 the same attribute refers to student permanent address(or home address) to resolve this conflict we would probably need to create new attributes name , so that merged new relation will become :

STUDENT (Student_ID ,Name, Phone_No, Campus_Address , Permanent_Address)

<u>3. Transitive dependencies</u> : when two 3NF relations are merged to form a single relation , transitive dependency may result . For example, consider the following two relations :

STUDENT1 (<u>Student_ID</u> , Major)

STUDENT2 (<u>Student</u>_ID , Advisor)

Since STUDENT1 and STUDENT2 have the same primary key , the two relations may be merged :

STUDENT (Student_ID , Major, Advisor)

However, suppose that each major has exactly one advisor . in this case Advisor is functionally dependent on Major.

Major → Advisor

If the preceding functional dependency exist ,the student is in 2NF but not in 3NF since it contains a transitive dependency , the analyst can create 3NF relations by removing the transitive dependency (Major becomes foreign key in STUDENT) ;

STUDENT (<u>Student_ID</u> , Major)

MAJOR ADVISOR (<u>Major</u>, Advisor)

<u>4. Supertype/ subtype relationships</u>: these relationships may be hidden in user views or relations . Suppose that we have the following two hospital relations:

PATIENT1 (<u>Patient_ID</u>, Name, Address) ,

PATIENT2 (<u>Patient_ID</u>, Room_No)

Initially it appears that these two relations can be merged into a single PATIENT relation. However the analyst correctly suspects that there are two different types of patients inpatients and outpatients . Patient1 contains attributes that common to all patients. PATIENT2 contains an attributes (Roon_No) that is a characteristic only for inpatients . In this situation the analyst should create a supertype' subtype relationship for these entities :

PATIENT (<u>Patient_ID</u>, Name, Address) ,

INPATIENT (<u>Patient_ID</u>, Room_No)

OUTPATIENT (<u>Patient_ID</u>, Date_Treated)

15. PHYSICAL DATABASE DESIGN:

The primary goal of physical database design is data processing efficiency. Today, with ever decreasing costs for computer technology per unit of measure (both speed and space measures), it is typically very important for you to design the physical system. Thus, we concentrate on how to make processing of physical files and databases efficient, with less attention on efficient use of space.

Designing physical files and databases requires certain information that should have been collected and produced during prior systems development phases. The information needed for physical files and data base design includes these requirements:

1. Normalized relations, including volume estimates.
2. Definitions of each attribute.
3. Descriptions of where and when data are used: entered , retrieved , deleted, and updated (including frequencies)
4. Expectations or requirements for response time and data security, backup, recovery, retention , and integrity .
5. Describing of the technologies (database management systems) used for implementing the database)

Physical database design requires several critical decisions that will affect the integrity and performance of the application system. These key decisions include the following:

1. Choosing the storage format (*called data type*) for each attribute from the logical data model. The format is chosen to minimize storage space and to maximize data integrity.

2. Grouping attributes from the logical data model into *physical records* .You will discover that although the columns of a relational table are a natural definition for the contents of a physical record, this is not always the most desirable grouping of attributes.

3. Arranging similarly structured records in secondary memory (primarily hard disks) so that individual and groups of records (*called file organization)* can be stored, retrieved, and updated rapidly. Consideration must be given also to protecting data and recovering data after errors are found.

4. Selecting structures (*called indexes and database architectures*) for storing and connecting files, to make retrieving related data more efficient.

5. Preparing strategies for handling queries against the database that will optimize performance and take advantage of the file organizations and indexes that you have specified. Efficient database structures will be of benefit only if queries and the database management systems that handle those queries are turned to intelligently use those structures.

15.1. Designing Fields:

A field is the smallest unit of application data recognized by system software, such as a programming language or database management system. A field corresponds to a simple attribute from the logical data model, so a field, represents each component of a composite attributes.

The basic decisions you must take in specifying each field concern this type of data (are storage type) used to represent values of this field , data integrity controls built into the database , and how the DBMS should handle missing values for the field . There are other field specifications, such as display format, which must be made as part of the total specification of the information system.

15.2. Choosing Data Types

A **Data type** is a detailed coding scheme recognize by system software, such as DBMS, for representing organizational data.

That bit pattern of the coding scheme is usually transparent to you, but the space to store data and the speed required to access data that are of consequence in physical database design. The specific DBMS you will use when dictate which choices are available to you. A typical DBMS that uses the SQL data definition and manipulation language. Additional data types might be available for currency, voice, image, and user defined for some DBMS’s.

Selecting the data that involves the objectives that will have different relative importance is for different applications.

1. Minimize Storage space.
2. Represents all possible values.
3. Improved date integrity.
4. Support all data manipulations.

The correct data type to choose from a field that can, in minimal space, represent every possible value (but eliminate illegal values) for the associated attributes .and can support the required data manipulation. (e.g., numeric data types for arithmetic and character data types for string manipulation). Any attribute to domain constraint from the conceptual data model are helpful in selecting a good data type of that attribute.

Achieving these four objectives can be subtle. , For example consider a DBMS for which the SMALLINT data type has a maximum width of two bytes. Suppose SMALLINT is sufficient to represent a quantity sold field. When quantity Sold field are summed, the sum may require a number of larger than 2 bytes. If the DBMS uses the fields data type for results of any mathematics on that failed, SMALLINT will not work. . Some data types have a special manipulation capability; for example, only the DATE data type allows to true date arithmetic.

15.3. Coding and Compression Techniques:

Some attributes have spares set of values or are so large that given data volumes , considering a storage space will be consumed (large data fields mean that data are further apart, which yields slower data processing) a field with a limited number of possible values can be translated into a code that requires less space.

A form of a code table is used by data compression techniques. Such as file zip routines. The data compression technique looks for patterns in data and then codes frequently appearing patterns with fewer bits. Although such routines are typically used to compress a whole file, such a routine could be used with some DBMSs to compress specific fields. Related to data compression techniques are encrypted techniques, which translate a field into a rescue format. In both cases, for a user to see the actual field value, software must know that the reverse translation process.

15.4. Controlling Data Integrity:

For many DBMS's data, integrity controls can be built into the physical structure of the fields and controls enforced by the DBMS on those fields. The data type enforces in one form of data integrity control since it may limit by the type of data (numeric or character) and length of a field value.

Some of those typically integrated controls that a DBMS may support are the following:

a) ***Default value***: a default value is the value of a field will assume unless a user enters an explicit value for an instance of the field. Assigning a default value to a field can reduce data entry time since entry of a value can be skipped and it can also help to reduce data entry errors for the most common values.

b) ***Range control:*** a range control limits is the set of permissible values a field may assume. The range may be numeric lower to upper bound to a set of specific values. Range controls must be used with caution since the limits of the range may change over time. A combination of range controls and coding has led to the year 2000 problem phased by many organizations, in which a field for a year is represented by only the numbers 00- 99. It is better to implement any range controls through a DBMS since range controls in programs may be inconsistently enforced and it is more difficult to find and change them.

c) ***Null value control:*** a null value was defined earlier as an entity value each primary key must have an integrity control that prohibits a null value.

   Any other required field may also have a null value control placed on it if that is the policy of the organization. For example, a university may prohibit adding a course to its database unless that course has a title as well as a value of the primary key , Course_ID. Many fields legitimately may have null values, so this control should be used only when truly required by business rules.

d) ***Referential Integrity:*** the term referential integrity was defined earlier, referential integrity on field is a form of range control in which the value of that field must exist as a value in some field in another row of the same or different table. That is, the range of legitimate values comes from the dynamic contents of field in a database table, not from some pre-specified set of the values. Note that referential integrity guarantees that only some existing cross-referencing value is used, not that it is the correct one.

15.5. **Handling Missing Data:**

When field may be null, simply entering no value may be sufficient. For example, suppose a customer ZIP code field is null and a report summarizes total sales by month and zip code, how should sales to customers with unknown zip codes be handled? Two options for handling preventing missing data have already been mentioned: using a default value and not permitting missing (null values) . Missing data are inevitable. Other possible methods for handling missing data are the following:

a) **Substitute and estimate of the missing value**. For example , for a missing sales value when computing monthly product sales, use a formula involving the mean of an existing monthly sales values for that product indexed by total sales for that month across all products. Such estimates must be marked so that users know that these are not actual values.

b) **Track missing data**: So that special reports and other system elements cause people to quickly resolve unknown values. This can be done by setting up a trigger in the database definition. A trigger is a routine that will automatically execute when some events occur, or time passes. On trigger could lock a missing entry to a file when a null or other missing value is stored and another trigger vane periodically to create a report of the contents of this log file.

c) **Perform Sensitivity Testing**: so that missing data are ignored unless knowing a value might significantly change results; if, for example, total monthly sales for a particular salesperson are almost over a threshold that makes a difference in that person compensation. This is the most complex of the methods mentioned and hence required the sophisticated programming which must be written in application programs since DBMS's do not have the sophistication to handle this method.

16.DESIGNING PHYSICAL RECORDS AND DENORMALIZATION:

In a Logical data model, you grouped into a relation those attributes that are determined by the same primary key. In contrast, a physical record is a group of fields stored in adjacent memory locations and retrieved together as a unit.

The design of a physical record involves choosing the sequencing of fields into adjacent storage locations to achieve two goals efficient use of secondary storage and data processing speed.

The efficient use of secondary storage is influenced by both the size of the physical record and the structure of secondary storage. Computer operating systems read data from hard disks in units called pages, not physical records. A page is the amount of data read or written in a secondary memory input or output operation .

The page size is fixed by system programmers and is selected to use RAM most efficiently across all applications. Depending on the computer system a physical record may or may not be allowed to span two pages.

Thus, if page length is not an integer Multiple of the physical record length, wasted space may occur at the end of a page.

The number of physical records per page is called the blocking factor. If storage space is scarce and physical records cannot span ages, creating multiple physical records from one logical relation will minimize wasted storage space.

16.1. Denormalization

The preceding discussion of physical record design concentrated an efficient use of storage space. In most cases the second goal of physical record design (efficient data processing) dominates the design process. Efficient processing of data, just like efficient accessing at books in a library, depends on how close together related data (or book) are. Often all the attributes that appear with a relation are not used together and data from different relations are needed together to answer a query or produce a report. Thus, although normalized relations solve data maintenance anomalies normalized relations, if implemented one for one as physical record, may not yield efficient data processing.

Denormalization: is the process transforming normalized physical record specifications.

In general, denormalization may partition a relation into several physical records, may combine attributes from several relations together into one physical record or may do a combination of both.

Denormalization can increase the chance of errors and inconsistencies and can force reprogramming systems if business rules changed. Further, Denormalization optimizes certain data processing at the expense of others, so if the frequencies of different processing activities change, the benefits of Denormalization may no longer exists.

16.2. Designing Physical Files

A physical file is a named portion of secondary (a magnetic or hard disks) allocated for the purpose storing physical records. Some computer operating systems allows a physical file to be split into separate pieces. If this occurs, it typically be transparent to you as the database designer.

16.3. Pointer

All files are organized by using two basic constructs to link one piece of data with another piece of data: sequential storage and pointers. With sequential storage, one field or record is stored right after another 'field or record. Although simple to imple-ment and use, sometimes sequential storage is not the most efficient way to organize data.

A **pointer** is a field of data that can be used to locate a related field or record of data.

In most cases, a pointer contains the address, or location, of the associated data. Pointers are used in a wide variety of data storage structures.

16.4. Access Methods

All Input-Output operations are ultimately handled by the data management portion of the computer’s operating system. Each operating systems supports one or more algorithms for storing and retrieving data; these algorithms are called access methods.

There are two basic types of access methods relative and direct.

A) **Relative access method:** support accessing data as an offset from the most recently referenced point in secondary memory . A sequential access method is a special case of this type since the "next" record begins the distance one record from the beginning of the current record. In general, a relative access method supports finding the *n*th record from the current position or from the beginning of the file.
B) **Direct Access Method:** uses a calculation to generate the beginning address of a record. The simplest form of a direct method is to tell the access method to go to a particular disk address. Another variation is to provide a record's primary-key and the direct access method determines where this record should be located.

17.File Organizations

A file organization is a technique for physically arranging the records of a file on secondary storage devices. In choosing a file organization for a particular file in a database, you should consider seven important factors:

1. Fast data retrieval

2. High throughput for processing data input and maintenance transactions.

3. Efficient use of storage space.

4. Protection from failures or data loss.

5. Minimizing need for reorganization.

6. Accommodating growth.

7. Security from unauthorized access.

A) Sequential File Organizations:

In sequential file organization, the records in the file are stored in sequence according to a primary key value.

To locate a particular record, a program must normally scan the file from the beginning until the desired record is located. A common example of a sequential file is the alphabetical list of persons in the white pages of a telephone directory.

B) Indexed File Organizations:

In the indexed file organization, the records are stored either sequentially or non-sequentially and an index is created that allows the application software to locate individual records.

Like a card catalog in A library in index is a table that u used to determine the location of rows in a file that satisfy some condition.

Each index entry matches a key value with one or more records. An index can point to unique records (a primary key index, such as on the Product_ID field of a product record) or to potentially more than one record.

An index that allows each early to point to more than one record is called a secondary key index.

Secondary key indexes are important for supporting many reporting requirements and for providing rapid ad hoc data retrieval.

C) Hashed File Organizations

In a hashed file organization, the address of each record' is determined using a hashing algorithm.

A hashing algorithm is a routine that converts a primary key value to a record address. Although there are several variations of hashed files , in most cases the records are located non sequentially as dictated by the hashing algorithm. Thus, sequential data processing is impractical.

A typical hashing algorithm uses the technique of dividing each primary key value by a suitable prime number and then using the remainder of the division as the relative storage location.

For example, suppose that an organization has a set of approximately 1,000 employee records to be stored on magnetic disk. A suitable prime number would be 997, since it is close to 1,000. Now consider the record for employee 12396, when divide this number be 997, the reminder is 432. Thus, this record is stored at location 432 in the file.

18. DESIGNING DATABASE

Most modern information systems utilize database technologies, either database management systems or data warehouse systems, for data storage and retrieval. Recall that a database is a collection of logically related data, designed to meet the information needs of multiple users in an organization.

The relationship between files in a database is due to relationship identified in the conceptual and logical data models.

The relationships imply access paths between data.

Each type of database technology allows different types of access paths. So, the process of choosing the appropriate type of DBMS or data warehousing technology is one of matching the needed access path, with the capabilities of the database technology.

1. Hierarchical Database Model:

In this model, files are arranged in a top-down structure that resembles a tree or genealogy chart. Data are related in a nested, one-to-many set of relationships.

The top file is called the root, the bottom files are called leaves, and intermediate files have one parent, or owner, file and one or several children files.

Among the oldest of the database architectures, many hierarchical databases exist in larger organizations today.

This technology is best applied when the conceptual data model also resembles a tree and when most data access begins with the same (root) file.

Hierarchical database technology is used for high-volume transaction processing and MIS (Management Information System) applications. Few new databases are developed with hierarchical DBMSs since newer applications tend to have broader needs than simply transaction processing or summarization of transaction data. .

2. Network Database Model:

In this model, each file may be associated with an arbitrary number of files. Although very flexible because any, relationships can be implemented (a hierarchy is a special case of a network), the form of implementation, usually using pointers; between related records in different files, creates significant overhead in storage space and maintenance time.

Typically network model systems support only one-to-many relationships along each arc in the network, but some support many-to-many relationships.

Network model systems are still popular on powerful mainframes and for high-volume transaction processing applications and used to design highly optimized databases with network systems.

Network systems support a wider variety of processing requirements than do hierarchical database systems, but network systems still require significant programming and database design knowledge and time, and hence are used primarily in those organizations with significant expertise with such technologies.

3. Relational Database Model:

The most common database model for new systems defines simple tables for each relation and many-to-many relationships.

Cross-reference keys link the tables together, representing the relationships between entities. Primary and secondary key indexes provide rapid access to data based upon qualifications. Most new applications are built using relational DBMSs, and many relational DBMS products exist.

4. Object-Oriented Database Model:

In this model, attributes and methods that operate on those attributes are encapsulated in structures called object classes.

Relationships between object classes are shown, in part, by nesting or encapsulating one object class within another object classes are defined from more general object classes.

A major advantage of this data model is that multimedia data types like graphics, video, and sound are supported as easily as simpler data types.

5. Multidimensional Database Model:

This database model is used in data warehousing applications, two ways of viewing this model exist:

A) The first views data as a multidimensional table in which each cell contains one or more simple attributes and the dimensions are ways to categorize the raw data.

These categories, or dimensions, are the factors on which users want to summarize or segment the data, such as period, geography, lines of business, or people. A cell contains data relevant to the intersection of all its dimension values. For example, a cell might hold the number-of-units-sold attribute for a given period, location, line of business, and salesperson.

B) The second (equivalent) view called a star schema. At the center is a fact table, equivalent to the cell in the multidimensional view.

This table contains all the raw attributes and a composite key made up of the primary keys of all the surrounding dimension tables. The surrounding dimension tables define each of the ways to categorize data, such as all the description data about each salesperson.