UNIT-3

SYLLABUS:

Conceptual Data Modeling: High-Level Conceptual Data Models for Database Design, A Sample Database Application, Entity Types, Entity Sets, Attributes and Keys, Relationship Types, Relationship Sets, Roles, and Structural Constraints, Weak Entity Types, Refining the ER Design, ER Diagrams, Naming Conventions and Design Issues, Relationship Types of Degree Higher Than Two. Relational Database Design Using ER-to-Relational Mapping.

1. Conceptual Data Modeling

Introduction

Conceptual modeling is a very important phase in designing a successful database application. Entity-Relationship (ER) model is a popular high-level conceptual data model. This model and its variations are frequently used for the conceptual design of database applications, and many database design tools employ its concepts.

1.1 High-Level Conceptual Data Models for Database Design

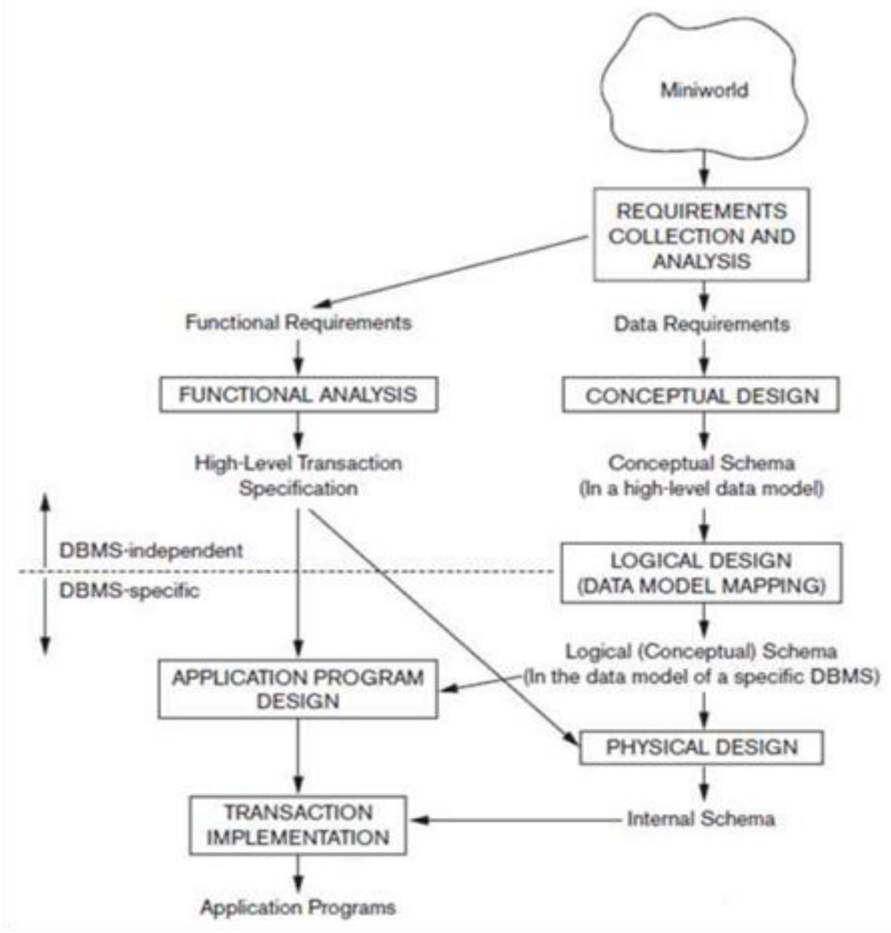

Miniworld
REQUIREMENTS COLLECTION AND ANALYSIS
Functional Requirements
Data Requirements
FUNCTIONAL ANALYSIS
CONCEPTUAL DESIGN
High-Level Transaction Specification
Conceptual Schema (In a high-level data model)
DBMS-independent
DBMS-specific
LOGICAL DESIGN (DATA MODEL MAPPING)
Logical (Conceptual) Schema (In the data model of a specific DBMS)
APPLICATION PROGRAM DESIGN
PHYSICAL DESIGN
TRANSACTION IMPLEMENTATION
Internal Schema
Application Programs

Figure 3.1: A simplified diagram to illustrate the main phases of database design.

The first step shown is requirements collection and analysis. During this step, the database designers interview prospective database users to understand and document their data requirements. The result of this step is a concisely written set of users requirements. These requirements should be specified in as a detailed and complete form as possible. . In parallel with specifying the data requirements, it is useful to specify the known functional requirements of the application. These consist of the user defined operations (or transactions) that will be applied to the database, including both retrievals and updates.

Once the requirements have been collected and analyzed, the next step is to create a conceptual schema for the database, using a high-level conceptual data model. This step is called conceptual design. The conceptual schema is a concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships, and constraints; these are expressed using the concepts provided by the high-level data model.

The next step in database design is the actual implementation of the database, using a commercial DBMS. Most current commercial DBMSs use an implementation data model such as the relational or the object-relational database model so the conceptual schema is transformed from the high-level data model into the implementation data model. This step is called logical design or data model mapping; its result is a database schema in the implementation data model of the DBMS.

The last step is the physical design phase, during which the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified. In parallel with these activities, application programs are designed and implemented as database transactions corresponding to the high level transaction specifications.

2.2 Entity Types, Entity Sets, Attributes and Keys

The ER model describes data as entities, relationships, and attributes.

2.2.1 Entities and Attributes

Entity: A thing in the real world with an independent existence. An entity may be an object with a physical existence (for example, a particular person, car, house, or employee) or it may be an object with a conceptual existence (for instance, a company, a job, or a university course).

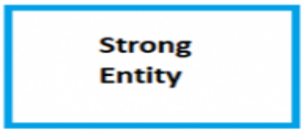
Types of DBMS Entities:

The following are the types of entities in DBMS –

1) Strong Entity
2) Weak Entity

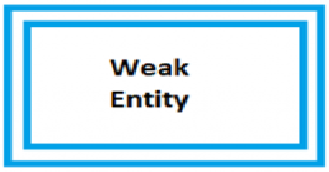
1) Strong Entity:

The strong entity has a primary key. Weak entities are dependent on strong entity. Its existence is not dependent on any other entity.
Strong Entity is represented by a single rectangle.



2) Weak Entity:

The weak entity in DBMS do not have a primary key and are dependent on the parent entity. It mainly depends on other entities.



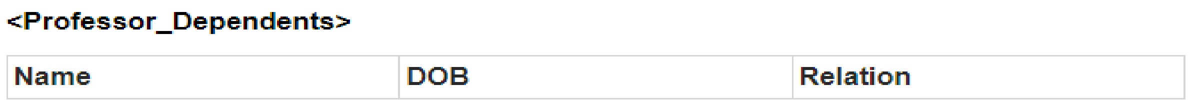
<u>Example of Entity in DBMS:</u>
Let us see an example –
<Professor>

Professor_ID	**Professor_Name**	**Professor_City**	**Professor_Salary**
P01	Tom	Sydney	$7000
P02	David	Brisbane	$4500
P03	Mark	Perth	$5000

Here, **Professor_Name, Professor _Address and Professor _Salary** are attributes. **Professor_ID** is the primary key

Professor is a strong entity here, and the primary key is Professor_ID.

However, another entity is Professor_Dependents, which is our Weak Entity.


<Professor_Dependents>

Name	DOB	Relation

Attributes: Particular properties that describe entity. For example, an EMPLOYEE entity may be described by the Attributes:. For example, an EMPLOYEE entity maybe described by the employee's name, age, address, salary and job.

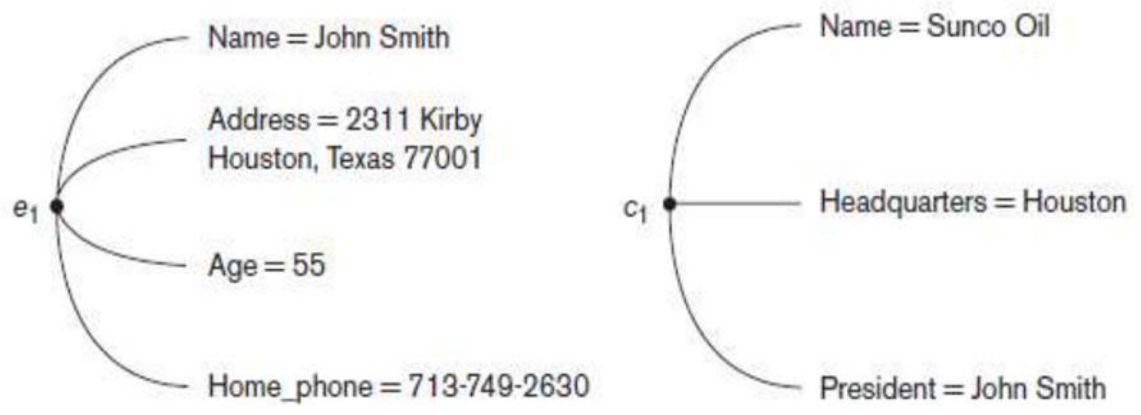


Figure: Two entities, EMPLOYEE e1, and COMPANY c1, and their attributes

Types of attributes:

1) Composite versus Simple (Atomic) Attributes
2) Single-valued versus multivalued
3) Stored versus derived
4) NULL values
5) Complex attributes

1. Composite versus Simple (Atomic) Attributes

Composite Attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings. For example, the Address attribute of the EMPLOYEE entity can be subdivided into Street address, City, State, and Zip.

Composite attributes can form a hierarchy. For example, Street address can be further subdivided into three simple component attributes: Number, Street, and Apartment number. The value of a composite attribute is the concatenation of the values of its component simple attributes.

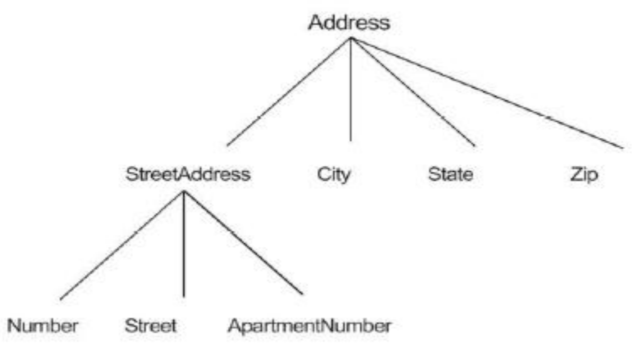


Figure: A hierarchy of composite attributes.

Attributes that are not divisible are called simple or atomic attributes. Example SSN of an employee.

2. Single-Valued versus Multivalued Attributes

Attributes that have a single value for a particular entity are called **single-valued**. For example, Age is a single-valued attribute of a person.

Attributes that can have a set of values for a particular entity are called **Multivalued Attributes.** For example Colors attribute for a car, or a College_degrees attribute for a person. A multivalued attribute may have lower and upper bounds to constrain the number of values allowed for each individual entity. For example, the Colors attribute of a car may be restricted to have between one and three values, if we assume that a car can have three colors at most.

3. Stored versus Derived Attributes

An attribute, which cannot be derived from other attribute are called **stored attribute.** For example, Birth_Date of an employee Attributes derived from other stored attribute are called derived attribute. For example age of an employee can be determined from the current (today's) date and Date of Birth.

4. Null Value Attribute (Optional Attribute)
In some cases, a particular entity may not have an applicable value for an attribute. For example, the Apartment_number attribute of an address applies only to addresses that are in apartment buildings and not to other types of residences, such as single-family homes. Similarly, a College_degrees attribute applies only to people with college degrees. For such situations, a special value called **NULL** is created. An address of a single-family home would have NULL for its Apartment_number attribute, and a person with no college degree would have NULL for College_degrees. NULL can also be used if we do not know the value of an attribute for a particular entity.
5. Complex Attributes
If an attribute for an entity, is built using composite and multivalued attributes, then these attributes are called complex attributes. For example, a person can have more than one residence and each residence can have multiple phones, an address phone for a person entity can be specified as:

{ Address phone (phone {(Area Code, Phone Number)},

Address (Sector Address (Sector Number, House Number), City, State, Pin))

}

Here {} are used to enclose multivalued attributes and () are used to enclose composite attributes with comma separating individual attributes

2.2.2 Entity Types, Entity Sets, Keys, and Value Sets

Entity Types: An entity type defines a collection (or set) of entities that have the same attributes. Each entity type in the database is described by its name and attributes. For example, a company employing hundreds of employees may want to store similar information concerning each of the employees. These employee entities share the same attributes, but each entity has its own value(s) for each attribute.

Entity Sets: The collection of all entities of a particular entity type in the database at any point in time is called an **entity set;** the entity set is usually referred to using the same name as the entity type. For example, EMPLOYEE refers to both a type of entity as well as the current set of all employee entities in the database.

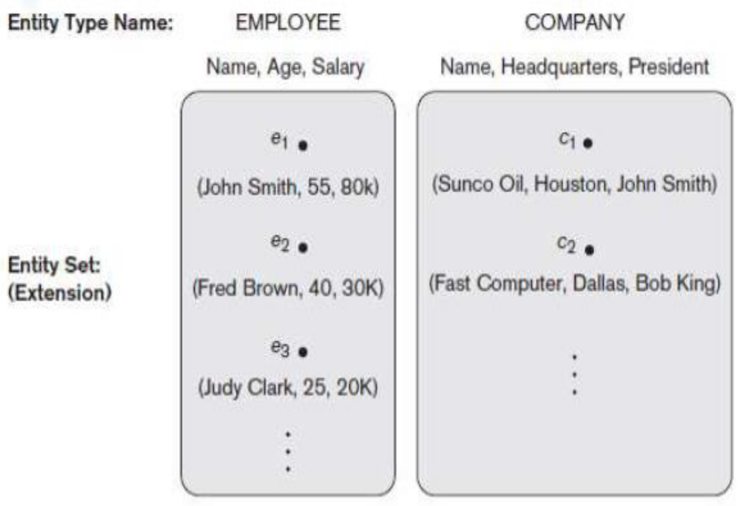


Figure: Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

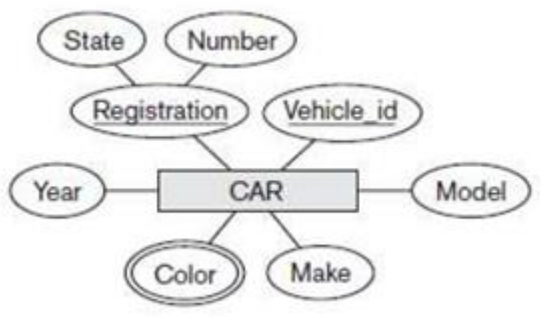
An entity type describes the **schema** or **intension** for a set of entities that share the same structure. The collection of entities of a particular entity type is grouped into an entity set, which is also called the **extension** of the entity type.

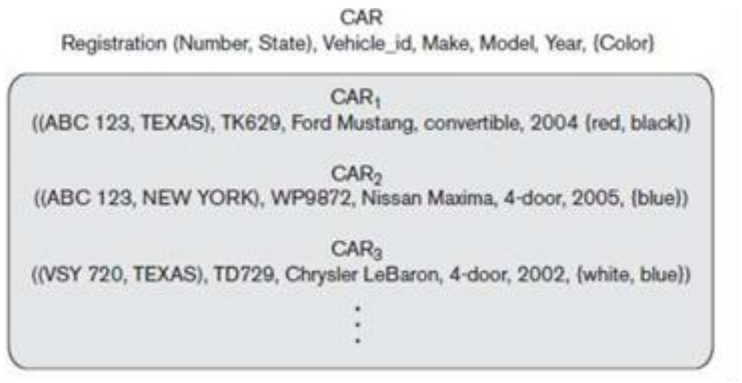
An **entity type** is represented in ER diagrams a **rectangular box** enclosing the entity type name. **Attribute names** are enclosed in **ovals** and are attached to their entity type by straight lines. **Composite attributes** are attached to their component attributes by straight lines. **Multivalued attributes** are displayed in **double ovals.**

Key Attributes of an Entity Type

An entity type usually has one or more attributes whose values are distinct for each individual entity in the entity set. Such an attribute is called a key attribute, and its values can be used to identify each entity uniquely. For example, the Name attribute is a key of the COMPANY entity because no two companies are allowed to have the same name. In ER diagrammatic notation, each key attribute has its name underlined inside the oval. Some entity types have more than one key attribute. For example, each of the Vehicle_id and Registration attributes of the entity type CAR is a key in its own right.

Example: The CAR entity type with two key attributes, Registration and Vehicle_id.





CAR
Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}

CAR_1
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR_2
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR_3
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

⋮

Figure: ER diagram notation *Entity set with three entities.*

Value Sets (Domains) of Attributes

Each simple attribute of an entity type is associated with a value set (or domain of values), which specifies the set of values that may be assigned to that attribute for each individual entity. For example, if the range of ages allowed for employees is between 16 and 70, we can specify the value set of the Age attribute of EMPLOYEE to be the set of integer numbers between 16 and 70.

Value sets are not displayed in ER diagrams, and are specified using the basic data types available in most programming languages, such as integer, string, Boolean, float, enumerated type, subrange, and so on.

Mathematically, an attribute A of entity set E whose value set is V can be defined as a **function from E to the power set P(V) of V**: $A: E \rightarrow P(V)$.We refer to the value of attribute A for entity e as A(e).A NULL value is represented by the empty set.

3.3 A Sample Database Application

Miniworld: COMPANY database keeps track of a company's employees, departments, and projects.

- ➢ After the requirements collection and analysis phase, the database designers provide the following description of the miniworld:
- The company is organized into departments.
- Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department.
- A department may have several locations.
- A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
- We store each employee's name, social security number, address, salary, gender and birth date.
- An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department.

- We keep track of the current number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee (who is another employee).
- We want to keep track of the dependents of each employee for instance purpose. We keep each dependent's first name, gender, birth date and relationship of employee.

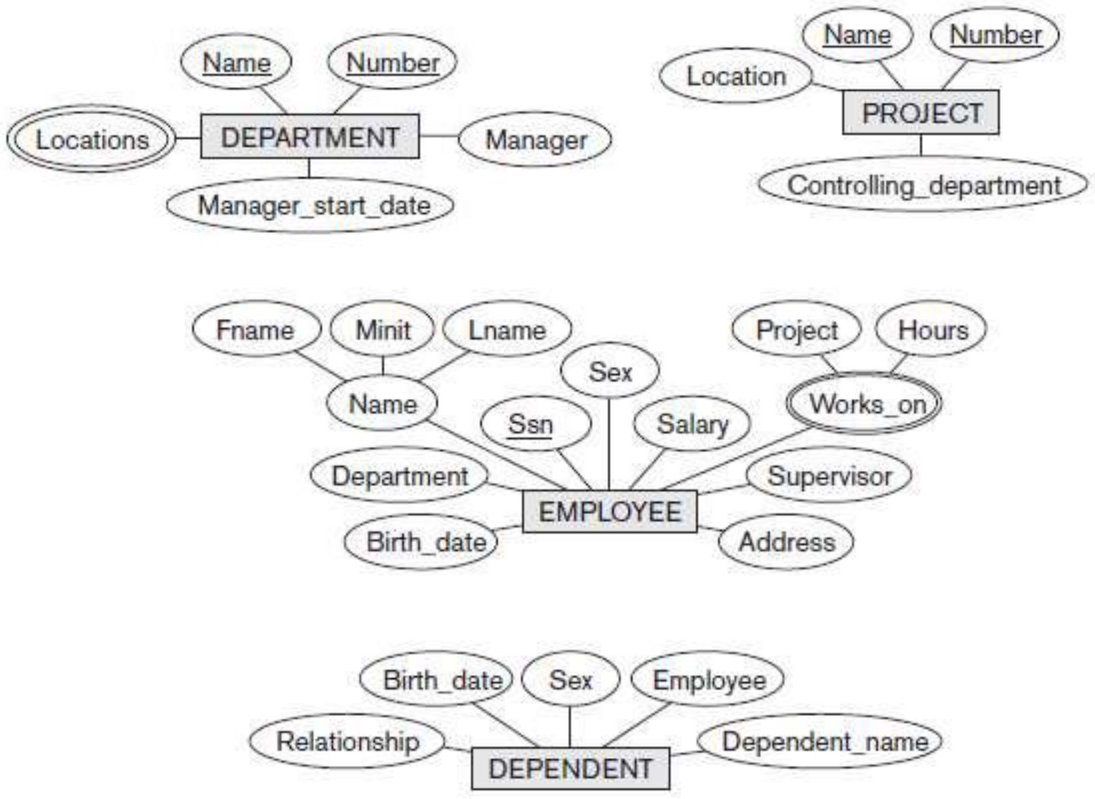


Figure: Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

3.4 Relationship Types

3.4.1 Relationship Types, Relationship Sets, Roles, and Structural Constraints

There are several implicit relationships among the various entity types. Whenever an attribute of one entity type refers to another entity type, some relationship exists. For example

- The attribute Manager of DEPARTMENT refers to an employee who manages the department.
- The attribute Controlling department of PROJECT refers to the department that controls the project.
- The attribute Supervisor of EMPLOYEE refers to another employee -the one who supervises this employee.
- The attribute Department of EMPLOYEE refers to the department for which the employee works.

In the ER model, these references should not be represented as attributes but as relationships.

3.4.2 Relationship Types, Sets, and Instances

A **relationship type R** among n entity types E1, E2, ..., En defines a set of associations or a **relationship set** among entities from these entity types. Entity types and Entity sets, a Relationship type and its corresponding Relationship set are usually referred to by the same name, R.

Mathematically, the relationship set R is a set of relationship instances r_i , where each r_i associates n individual entities (e_1, e_2, ..., e_n), and each entity e_i in r_i is a member of entity set E_j , $1 \leq j \leq n$. Each of the entity types E_1, E_2, ..., E_n is said to participate in the relationship type R. similarly, each of the individual entities e_1, e_2, ..., e_n is said to participate in the relationship instance $r_i = (e_1, e_2, ..., e_n)$

Informally, each relationship instance r_i in R is an association of entities, where the association includes exactly one entity from each participating entity type. For example, consider a relationship type WORKS_FOR between the two entity types EMPLOYEE and DEPARTMENT, which associates each employee with the department for which the employee works in the corresponding entity set. Each relationship instance in the relationship set WORKS_FOR associates one EMPLOYEE entity and one DEPARTMENT entity.

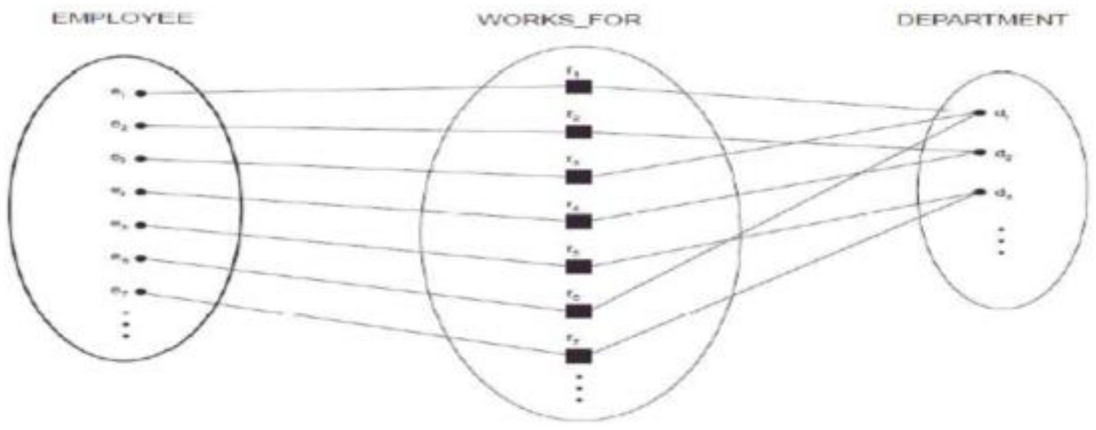


Figure: Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

employes e1 , e3, and e6 work for department d1. employs e2 and e4 work for department d2 and employees e5 and e7 work for department d3.

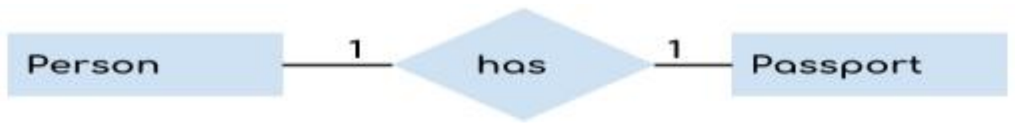
In ER diagrams, relationship types are displayed as diamond-shaped boxes, which are connected by straight lines to the rectangular boxes representing the participating entity types. The relationship name is displayed in the diamond-shaped box.

A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities. There are four types of relationships:

1. One to One

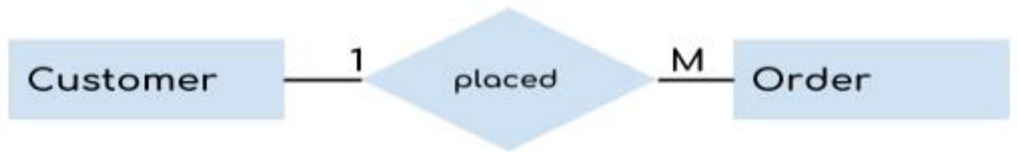
2. One to Many
3. Many to One
4. Many to Many

1. One to One Relationship



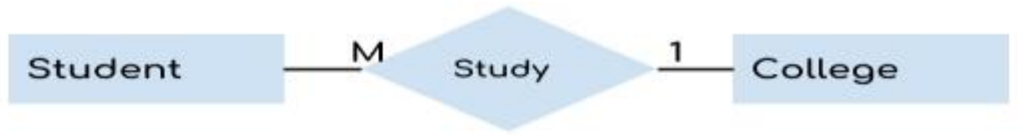
When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship. For example, a person has only one passport and a passport is given to one person.

2. One to Many Relationship



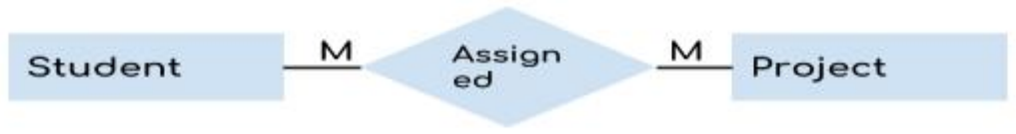
When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationship. For example – a customer can place many orders but a order cannot be placed by many customers.

3. Many to One Relationship



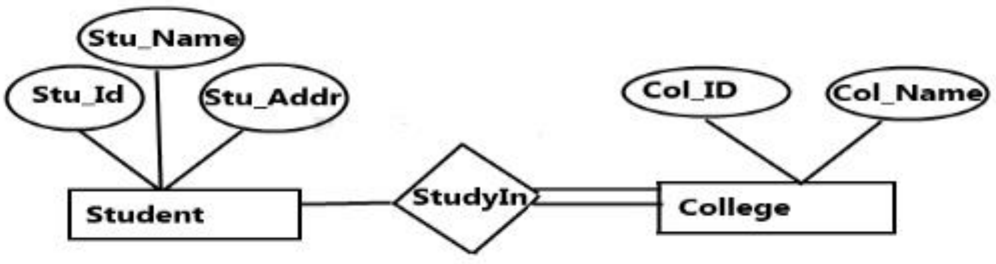
When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship. For example – many students can study in a single college but a student cannot study in many colleges at the same time.

4. Many to Many Relationship



When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationship. For example, a can be assigned to many projects and a project can be assigned to many students.

Total Participation of an Entity set



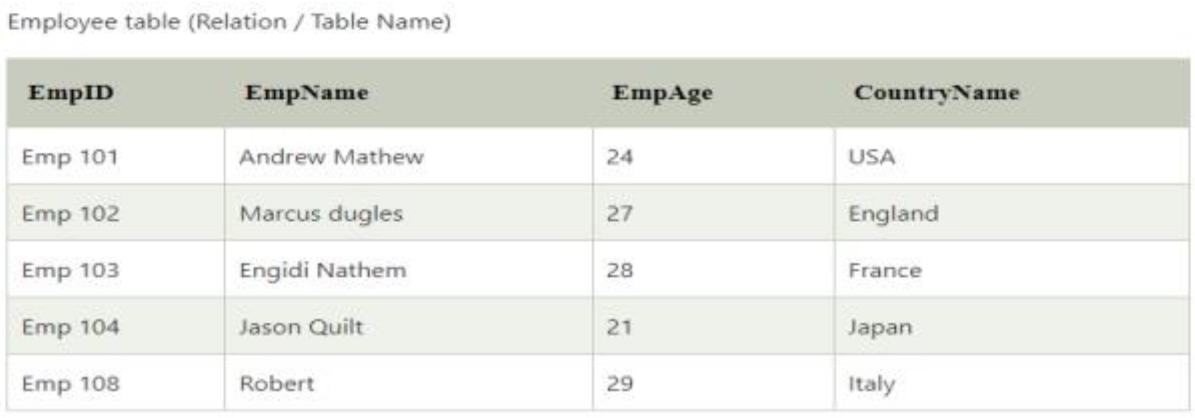
E-R Digram with total participation of College entity set in StudyIn relationship Set - This indicates that each college must have atleast one associated Student.

EXAMPLE:

A relational database collects different types of data sets that use tables, records, and columns. It is used to create a well-defined relationship between database tables so that relational databases can be easily stored. For example of relational databases such as Microsoft SQL Server, Oracle Database, MYSQL, etc.

There are some important parameters of the relational database:

- It is based on a relational model (Data in tables).
- Each row in the table with a unique id, key.
- Columns of the table hold attributes of data.


Employee table (Relation / Table Name)

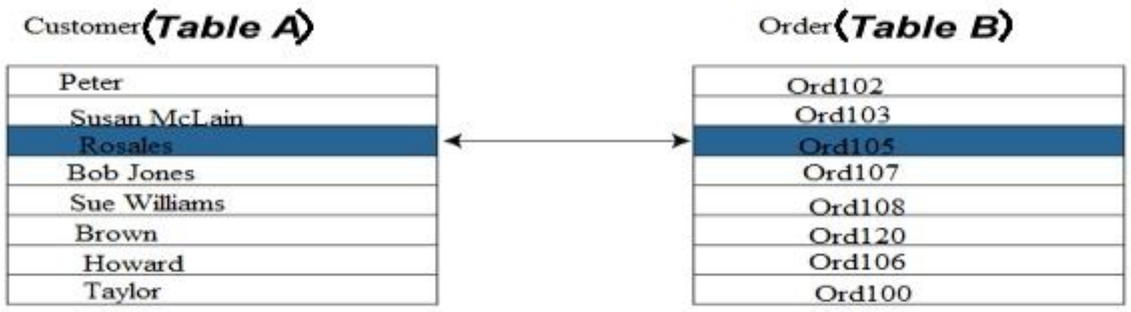
EmpID	EmpName	EmpAge	CountryName
Emp 101	Andrew Mathew	24	USA
Emp 102	Marcus dugles	27	England
Emp 103	Engidi Nathem	28	France
Emp 104	Jason Quilt	21	Japan
Emp 108	Robert	29	Italy

Following are the different types of relational database tables.

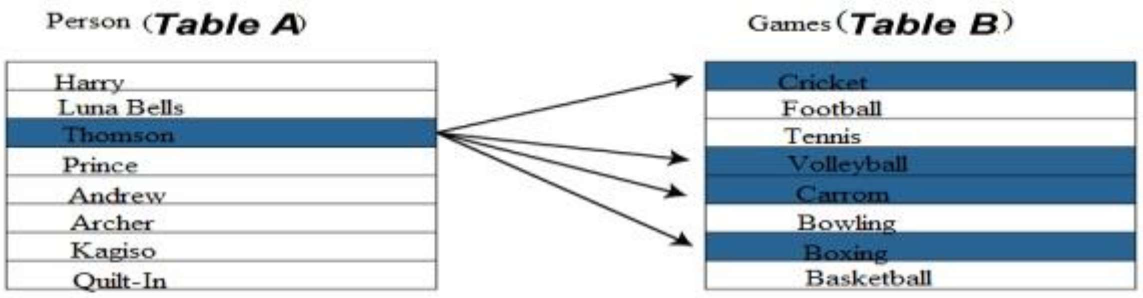
1) One to One relationship
2) One to many or many to one relationship
3) Many to many relationships

1) **One to One Relationship (1:1):** It is used to create a relationship between two tables in which a single row of the first table can only be related to one and only one records

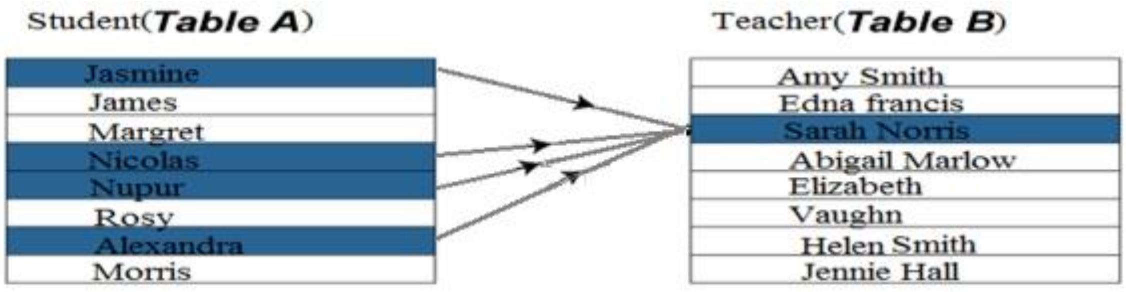
of a second table. Similarly, the row of a second table can also be related to anyone row of the first table.
Following is the example to show a relational database, as shown below.



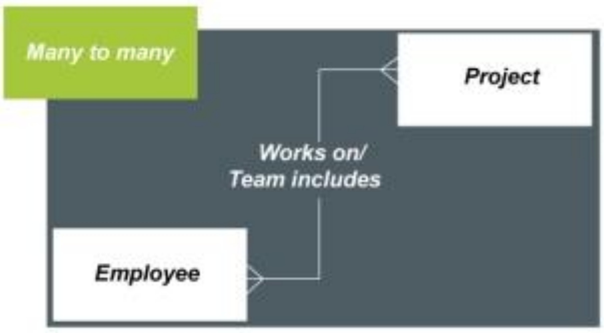
2) **One to Many Relationship:** It is used to create a relationship between two tables. Any single rows of the first table can be related to one or more rows of the second tables, but the rows of second tables can only relate to the only row in the first table. It is also known as a **many to one** relationship.
Representation of **One to Many** relational databases:



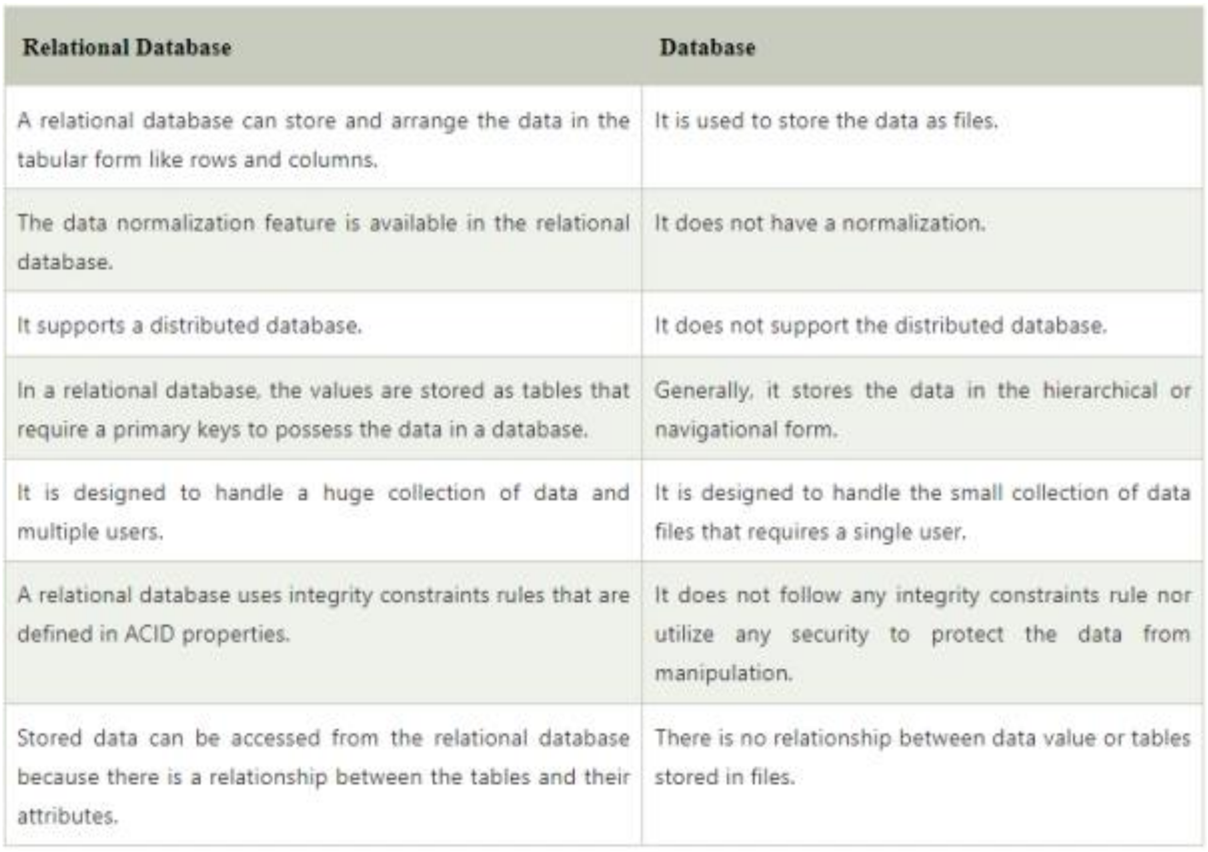
Representation of **many to one** relational database:



3) **Many to Many Relationship**: It is many to many relationships that create a relationship between two tables. Each record of the first table can relate to any records (or no records) in the second table. Similarly, each record of the second table can also relate to more than one record of the first table. It is also represented an N:N relationship.
For example, there are many people involved in each project, and every person can involve more than one project.



Difference between a database and a relational database:



Relational Database	Database
A relational database can store and arrange the data in the tabular form like rows and columns.	It is used to store the data as files.
The data normalization feature is available in the relational database.	It does not have a normalization.
It supports a distributed database.	It does not support the distributed database.
In a relational database, the values are stored as tables that require a primary keys to possess the data in a database.	Generally, it stores the data in the hierarchical or navigational form.
It is designed to handle a huge collection of data and multiple users.	It is designed to handle the small collection of data files that requires a single user.
A relational database uses integrity constraints rules that are defined in ACID properties.	It does not follow any integrity constraints rule nor utilize any security to protect the data from manipulation.
Stored data can be accessed from the relational database because there is a relationship between the tables and their attributes.	There is no relationship between data value or tables stored in files.

<u>Advantages of relational databases:</u>

1) **Simple Model:** The simplest model of the relational database does not require any complex structure or query to process the databases. It has a simple architectural process as compared to a hierarchical database structure. Its simple architecture can be handled with simple SQL queries to access and design the relational database.
2) **Data Accuracy**: Relational databases can have multiples tables related to each other through primary and foreign keys. There are fewer chances for duplication of data fields. Therefore the accuracy of data in relational database tables is greater than in any other database system.

3) **Easy to access Data:** The data can be easily accessed from the relational database, and it does not follow any pattern or way to access the data. One can access any data from a database table using SQL queries. Each table in the associated database is joined through any relational queries such as join and conditional descriptions to concatenate all tables to get the required data.
4) **Security:** It sets a limit that allows specific users to use relational data in RDBMS.
5) **Collaborate:** It allows multiple users to access the same database at a time.

3.4.3 Relationship Degree, Role Names, and Recursive Relationships

Degree of a Relationship Set

The number of entity sets that participate in a relationship set is termed as the degree of that relationship set. Thus,

Degree of a relationship set = Number of entity sets participating in a relationship set

Degree of a Relationship Type

The degree of a relationship type is the number of participating entity types. A relationship type of degree two is called **binary**, and one of degree three is called **ternary** an example of a binary relationship WORKS_FOR and ternary relationship is SUPPLY

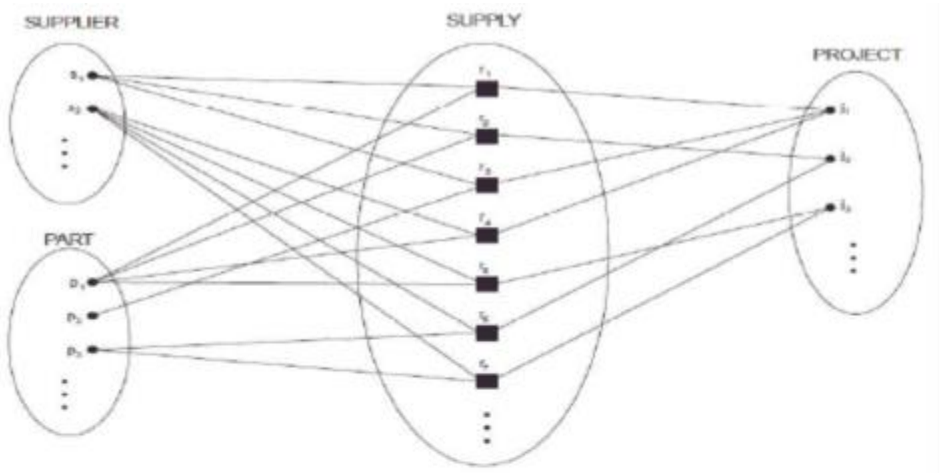


Figure: Some relationship instances in the SUPPLY ternary relationship set.

Each relationship instance r_i associates three entities a supplier s, a part p and a project j whenever s supplies part p to project j.

Relationships as Attributes

It is sometimes convenient to think of a binary relationship type in terms of attributes. Consider the WORKS_FOR relationship type. One can think of an attribute called Department of the EMPLOYEE entity type, where the value of Department for each EMPLOYEE entity is a reference to the DEPARTMENT entity for which that employee works. This concept of representing relationship types as attributes is used in a class of data models called functional data models.

In relational databases, foreign keys are a type of reference attribute used to represent relationships

Role Names and Recursive Relationships

Each entity type that participates in a relationship type plays a particular role in the relationship.

The role name signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means. For example, in the WORKS_FOR relationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the role of department or employer.

Role names are not technically necessary in relationship types where all the participating entity types are distinct, since each participating entity type name can be used as the role name. However, in some cases the same entity type participates more than once in a relationship type in different roles.

In such cases the role name becomes essential for distinguishing the meaning of the role that each participating entity plays. Such relationship types are called recursive relationships. Example of recursive relationships: SUPERVISION relationship type

The SUPERVISION relationship type relates an employee to a supervisor, where both employee and supervisor entities are members of the same EMPLOYEE entity set. Hence, the EMPLOYEE entity type participates twice in SUPERVISION: once in the role of supervisor (or boss), and once in the role of supervisee (or subordinate). Each relationship instance ri in SUPERVISION associates two employee entities ej and ek , one of which plays the role of supervisor and the other the role of supervisee.

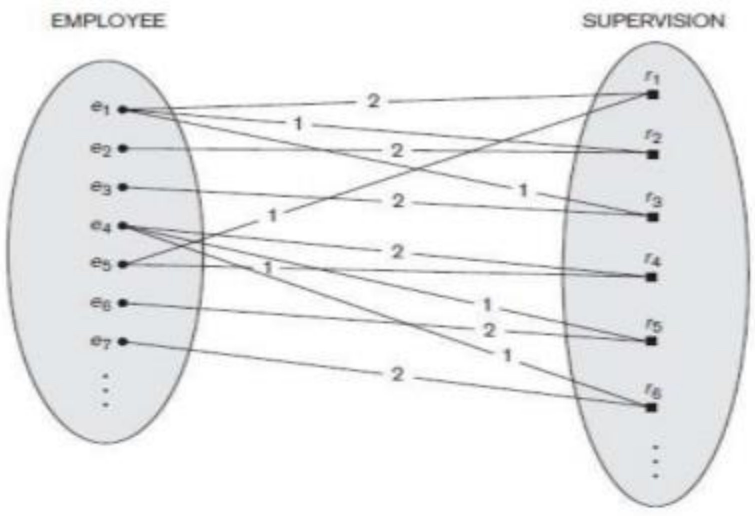


Figure: A recursive relationship SUPERVISION between EMPLOYEE in the supervisor role (1) and EMPLOYEE in the subordinate role (2)

3.4.4 Constraints on Binary Relationship Types

Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set. These constraints are determined from the miniworld situation that the relationships represent. For example, if the company has a rule that each employee must work for exactly one department, then we would like to describe this constraint in the schema. Two main types of binary relationship constraints:

1) Cardinality ratio
2) Participation.

1) Cardinality Ratios for Binary Relationships

The cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in. For example, in the WORKS_FOR binary relationship type, DEPARTMENT:EMPLOYEE is of cardinality ratio 1:N, meaning that each department can be related to any number of employees, but an employee can be related to (work for) only one department. The possible cardinality ratios for binary relationship types are 1:1, 1:N, N:1, and M:N.

Types of Cardinality :

There can be 4 types of cardinality –

1) One-to-one (1:1) –

When one entity in each entity set takes part at most once in the relationship, the cardinality is one-to-one.

2) One-to-many (1: N) –

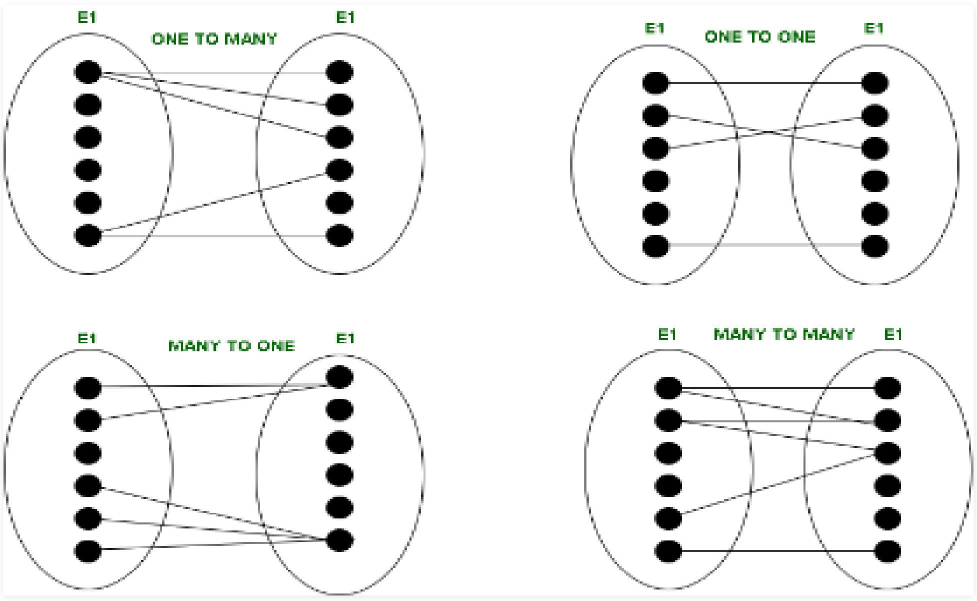
If entities in the first entity set take part in the relationship set at most once and entities in the second entity set take part many times (at least twice), the cardinality is said to be one-to-many.

3) Many-to-one (N:1) –

If entities in the first entity set take part in the relationship set many times (at least twice), while entities in the second entity set take part at most once, the cardinality is said to be many-to-one.

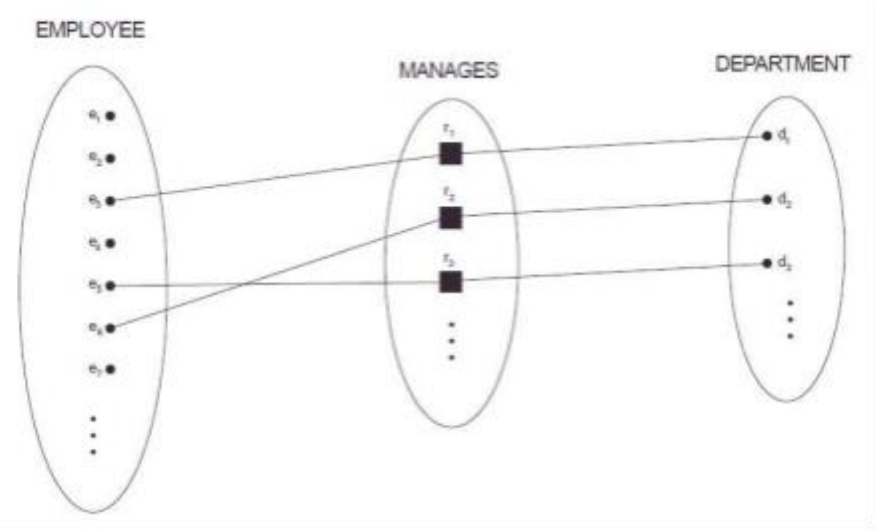
4) Many-to-many (N: N) –

The cardinality is said to be many to many if entities in both the entity sets take part many times (at least twice) in the relationship set.



Example of a 1:1 binary relationship

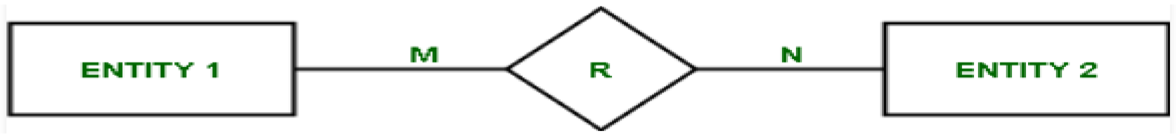
- MANAGES which relates a department entity to the employee who manages that department.
- This represents the miniworld constraints that at any point in time an employee can manage one department only and a department can have one manager only.

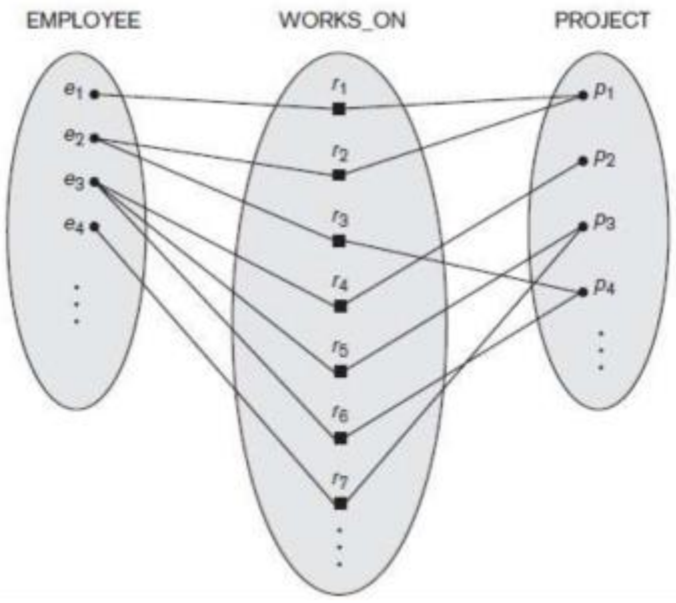


Example of a M:N binary relationship

- The relationship type WORKS_ON is of cardinality ratio M:N, because the mini-world rule is that an employee can work on several projects and a project can have several employees.

- Cardinality ratios for binary relationships are represented on ER diagrams by displaying 1, M, and N on the diamonds





There are numbers (represented by M and N) written above the lines which connect relationships and entities. These are called cardinality ratios. These represent the maximum number of entities that can be associated with each other through relationship, R.

2) Participation Constraints and Existence Dependencies

The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type. This constraint specifies the minimum number of relationship instances that each entity can participate in, and is sometimes called the **minimum cardinality constraint.** There are two types of participation constraints:

1) Total
2) Partial

1) Total participation

If a company policy states that every employee must work for a department, then an employee entity can exist only if it participates in at least one WORKS_FOR relationship Instance. Thus, the participation of EMPLOYEE in WORKS_FOR is called total participation, meaning that every entity in the total set of employee entities must be related to a department entity via WORKS_FOR. Total participation is also called **existence dependency**

2) Partial participation

We do not expect every employee to manage a department .So the participation of EMPLOYEE in the MANAGES relationship type is partial, meaning that some or part of the set of employee entities are related to some department entity via MANAGES, but not necessarily all.

In ER diagrams, total participation is displayed as a double line connecting the participating entity type to the relationship, whereas partial participation is represented by a single line.

cardinality ratio** + **participation constraints** = **structural constraints of a relationship type.

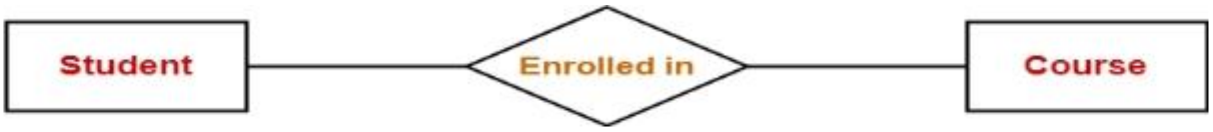
3.4.5 Attributes of Relationship Types

Relationship types can also have attributes, similar to those of entity types. For example, to record the number of hours per week that an employee works on a particular project, we can include an attribute Hours for the WORKS_ON relationship type. Another example is to include the date on which a manager started managing a department via an attribute Start_date for the MANAGES relationship type.

Attributes of 1:1 or 1:N relationship types can be migrated to one of the participating entity types. For a 1:N relationship type, a relationship attribute can be migrated only to the entity type on the N-side of the relationship. For M:N relationship types, some attributes may be determined by the combination of participating entities in a relationship instance, not by any single entity. Such attributes must be specified as relationship attributes.

4.1 Relationship Sets:

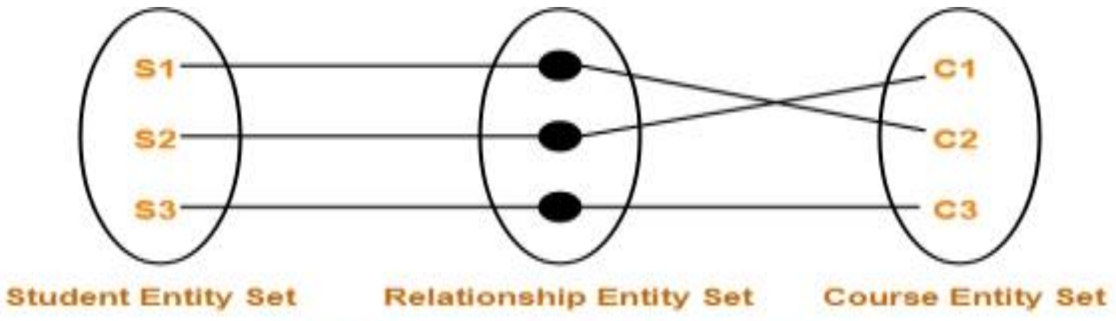
'Enrolled in' is a relationship that exists between entities **Student** and **Course**.



A relationship set is a set of relationships of same type.

Example-

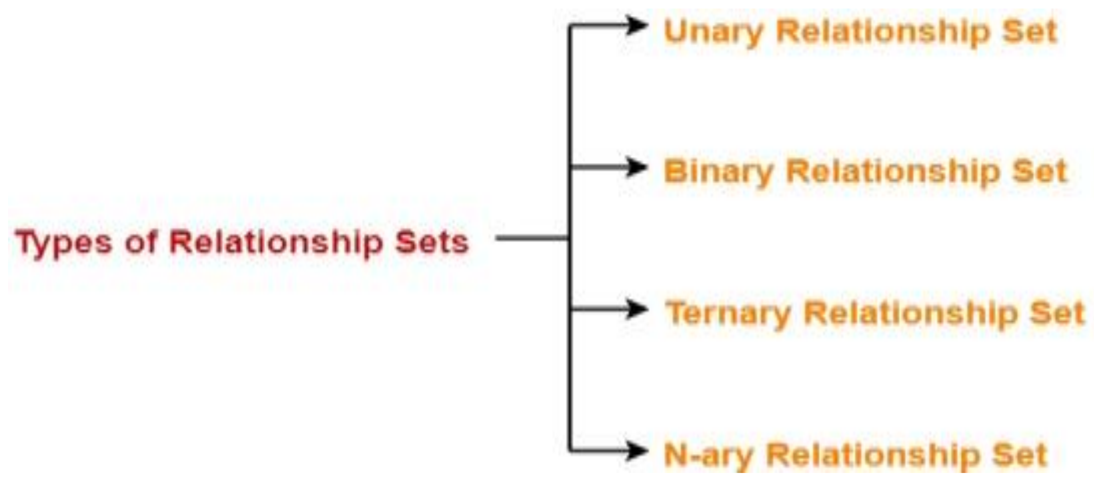
Set representation of above ER diagram is-



Set Representation of ER Diagram

4.2 Types of Relationship Sets-

On the basis of degree of a relationship set, a relationship set can be classified into the following types-



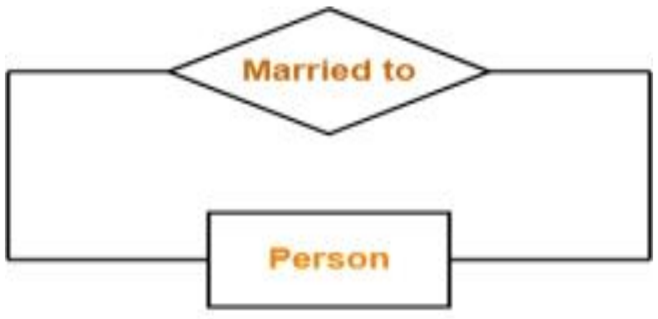
1) Unary relationship set
2) Binary relationship set
3) Ternary relationship set
4) N-ary relationship set

1. Unary Relationship Set-

Unary relationship set is a relationship set where only one entity set participates in a relationship set.

Example-

One person is married to only one person



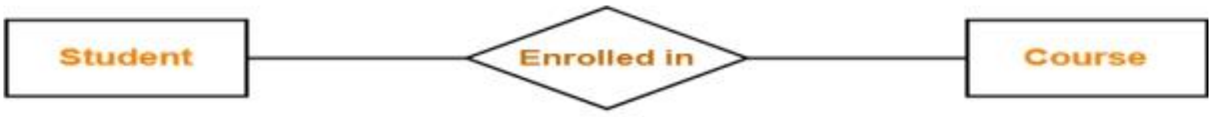
Unary Relationship Set

2. Binary Relationship Set-

Binary relationship set is a relationship set where two entity sets participate in a relationship set.

Example-

Student is enrolled in a Course

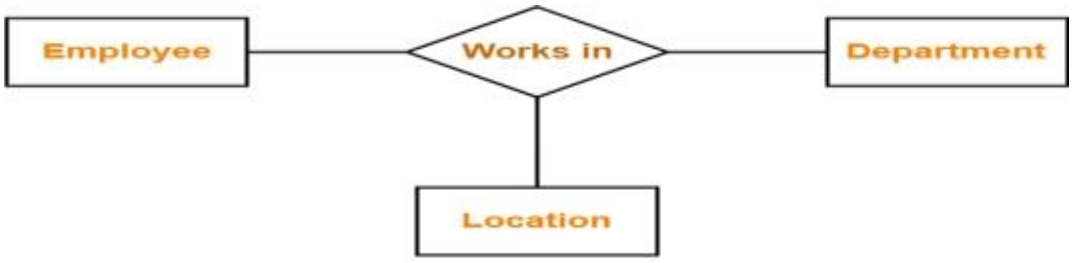


Binary Relationship Set

3. Ternary Relationship Set-

Ternary relationship set is a relationship set where three entity sets participate in a relationship set.

Example-



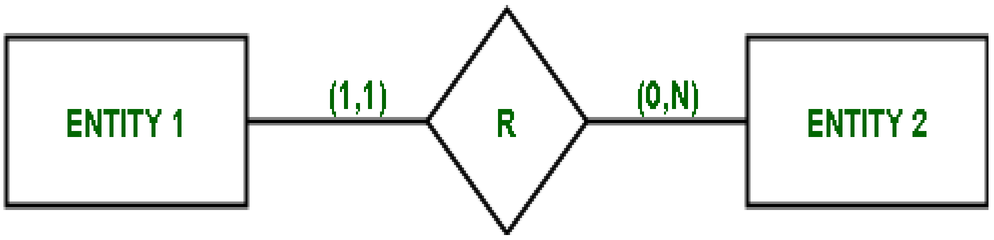
Ternary Relationship Set

4. N-ary Relationship Set-

N-ary relationship set is a relationship set where 'n' entity sets participate in a relationship set.

5.1 Roles and Structural Constraints

Structural Constraints are also called Structural properties of a database management system (DBMS). **Cardinality Ratios and Participation Constraints taken together are called Structural Constraints.** The name constraints refer to the fact that such limitations must be imposed on the data, for the DBMS system to be consistent with the requirements.



The Structural constraints are represented by Min-Max notation. This is a pair of numbers(m, n) that appear on the connecting line between the entities and their relationships. The minimum number of times an entity can appear in a relation is represented by m whereas, the maximum time it is available is denoted by n. If m is 0 it signifies that the entity is participating in the relation partially, whereas, if m is either greater than or equal to 1, it denotes total participation of the entity.

Note –

Number of times an entity participates in a relationship is same as the number appearance of the entity in the tuples.

5.2 Weak Entity Types

Entity types that do not have key attributes of their own are called **weak entity** types. Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. We call this other entity type the identifying or **owner entity type**. We call the relationship type that relates a weak entity type to its owner the **identifying relationship** of the weak entity type.

Consider the entity type DEPENDENT, related to EMPLOYEE, which is used to keep track of the dependents of each employee via a 1: N relationship. In our example, the attributes of DEPENDENT are Name, Birth_ date, gender, and Relationship (to the employee). Two dependents of two distinct employees may, by chance, have the same values for Name, Birth_date, gender, and Relationship, but they are still distinct entities. They are identified as distinct entities only after determining the particular employee entity to which each dependent is related. Each employee entity is said to own the dependent entities that are related to it.

A weak entity type always has a total participation constraint (existence dependency) with respect to its identifying relationship because a weak entity cannot be identified without an owner entity. A weak entity type normally has a **partial key,** which is the attribute that can uniquely identify weak entities that are related to the same owner entity. In our example, if we assume that no two dependents of the same employee ever have the same first name, the attribute Name of DEPENDENT is the partial key.

In ER diagrams, both a weak entity type and its identifying relationship are distinguished by surrounding their boxes and diamonds with double lines. The partial key attribute is underlined with a dashed or dotted line.

5.3 Refining the ER Design

Refining the ER Design for the COMPANY Database

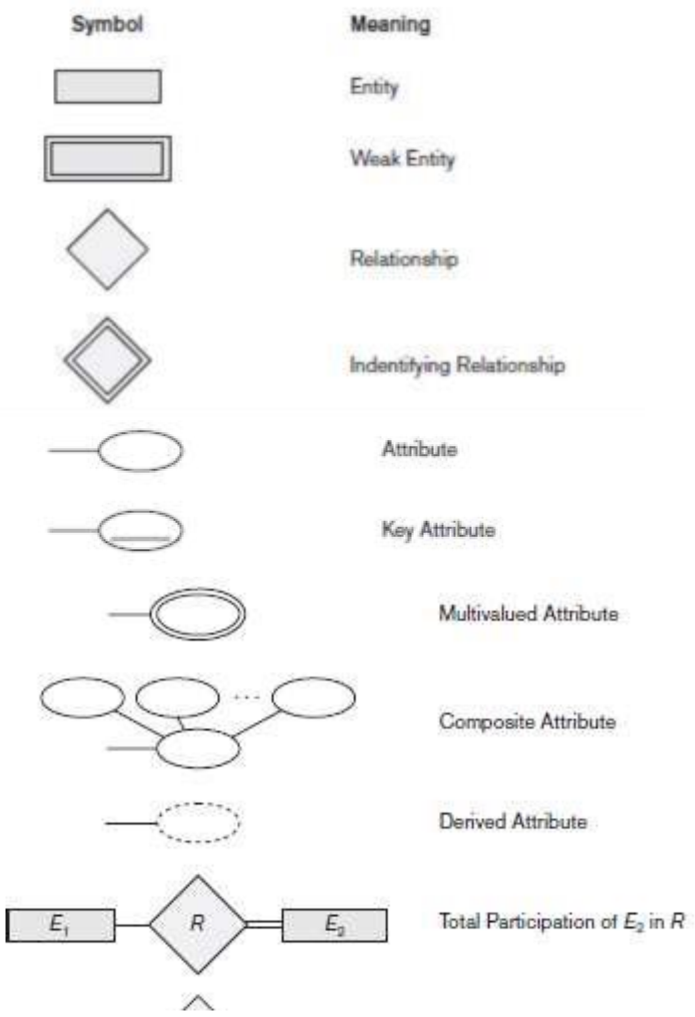
In our example, we specify the following relationship types:

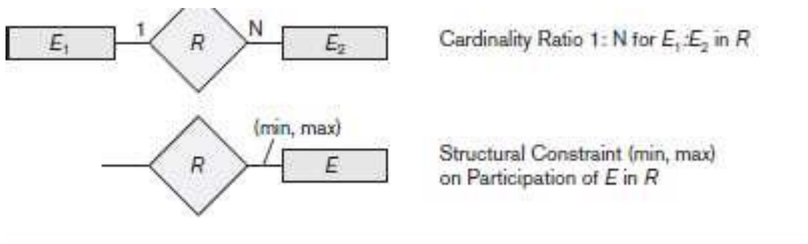
- MANAGES, a 1:1 relationship type between EMPLOYEE and DEPARTMENT. EMPLOYEE participation is partial. DEPARTMENT participation is not clear from the requirements. We question the users, who say that a department must have a manager at all times, which implies total participation. The attribute Start_date is assigned to this relationship type.
- WORKS_FOR, a 1:N relationship type between DEPARTMENT and EMPLOYEE. Both participations are total.
- CONTROLS, a 1:N relationship type between DEPARTMENT and PROJECT. The participation of PROJECT is total, whereas that of DEPARTMENT is determined to be

partial, after consultation with the users indicates that some departments may control no projects.

- SUPERVISION, a 1:N relationship type between EMPLOYEE (in the supervi-sor role) and EMPLOYEE (in the supervisee role). Both participations are determined to be partial, after the users indicate that not every employee is a supervisor and not every employee has a supervisor.
- WORKS_ON, determined to be an M:N relationship type with attribute Hours, after the users indicate that a project can have several employees working on it. Both participations are determined to be total.
- DEPENDENTS_OF, a 1:N relationship type between EMPLOYEE and DEPENDENT, which is also the identifying relationship for the weak entity type DEPENDENT. The participation of EMPLOYEE is partial, whereas that of DEPENDENT is total.

5.4 ER Diagrams, Naming Conventions and Design Issues




E1
1
R
N
E2
Cardinality Ratio 1: N for E1:E2 in R
(min, max)
R
E
Structural Constraint (min, max) on Participation of E in R

Proper Naming of Schema Constructs

Choose names that convey, as much as possible, the meanings attached to the different constructs in the schema.

- Use singular names for entity types, rather than plural ones, because the entity type name applies to each individual entity belonging to that entity type.
- In ER diagrams, entity type and relationship type names are uppercase letters, attribute names have their initial letter capitalized, and role names are lowercase letters.
- As a general practice, given a narrative description of the database requirements, the nouns appearing in the narrative tend to give rise to entity type names, and the verbs tend to indicate names of relationship types. Attribute names generally arise from additional nouns that describe the nouns corresponding to entity types.
- Another naming consideration involves choosing binary relationship names to make the ER diagram of the schema readable from left to right and from top to bottom.

5.4.1 Design Choices for ER Conceptual Design

In general, the schema design process should be considered an iterative refinement process, where an initial design is created and then iteratively refined until the most suitable design is reached. Some of the refinements that are often used include the following:

- A concept may be first modeled as an attribute and then refined into a relationship because it is determined that the attribute is a reference to another entity type. It is often the case that a pair of such attributes that are inverses of one another are refined into a binary relationship.
- Similarly, an attribute that exists in several entity types may be elevated or promoted to an independent entity type. For example, suppose that several entity types in a UNIVERSITY database, such as STUDENT, INSTRUCTOR, and COURSE, each has an attribute Department in the initial design; the designer may then choose to create an entity type DEPARTMENT with a single attribute Dept_name and relate it to the three entity types (STUDENT, INSTRUCTOR, and COURSE) via appropriate relationships.
- An inverse refinement to the previous case may be applied for example, if an entity type DEPARTMENT exists in the initial design with a single attribute Dept_name and is related to only one other entity type, STUDENT. In this case, DEPARTMENT may be reduced or demoted to an attribute of STUDENT.

3.6.4 Alternative Notations for ER Diagrams

There are many alternative diagrammatic notations for displaying ER diagrams. One alternative ER notation for specifying structural constraints on relationships, which replaces the cardinality ratio (1:1, 1:N, M:N) and single/double line notation for participation constraints. This notation involves associating a pair of integer numbers (min, max) with each participation of an entity type E in a relationship type R where $0 \leq \text{min} \leq \text{max}$ and $\text{max} \geq 1$.

The numbers mean that for each entity e in E, e must participate in at least min and at most max relationship instances in R at any point in time. In this method, min = 0 implies partial participation, whereas min > 0 implies total participation.

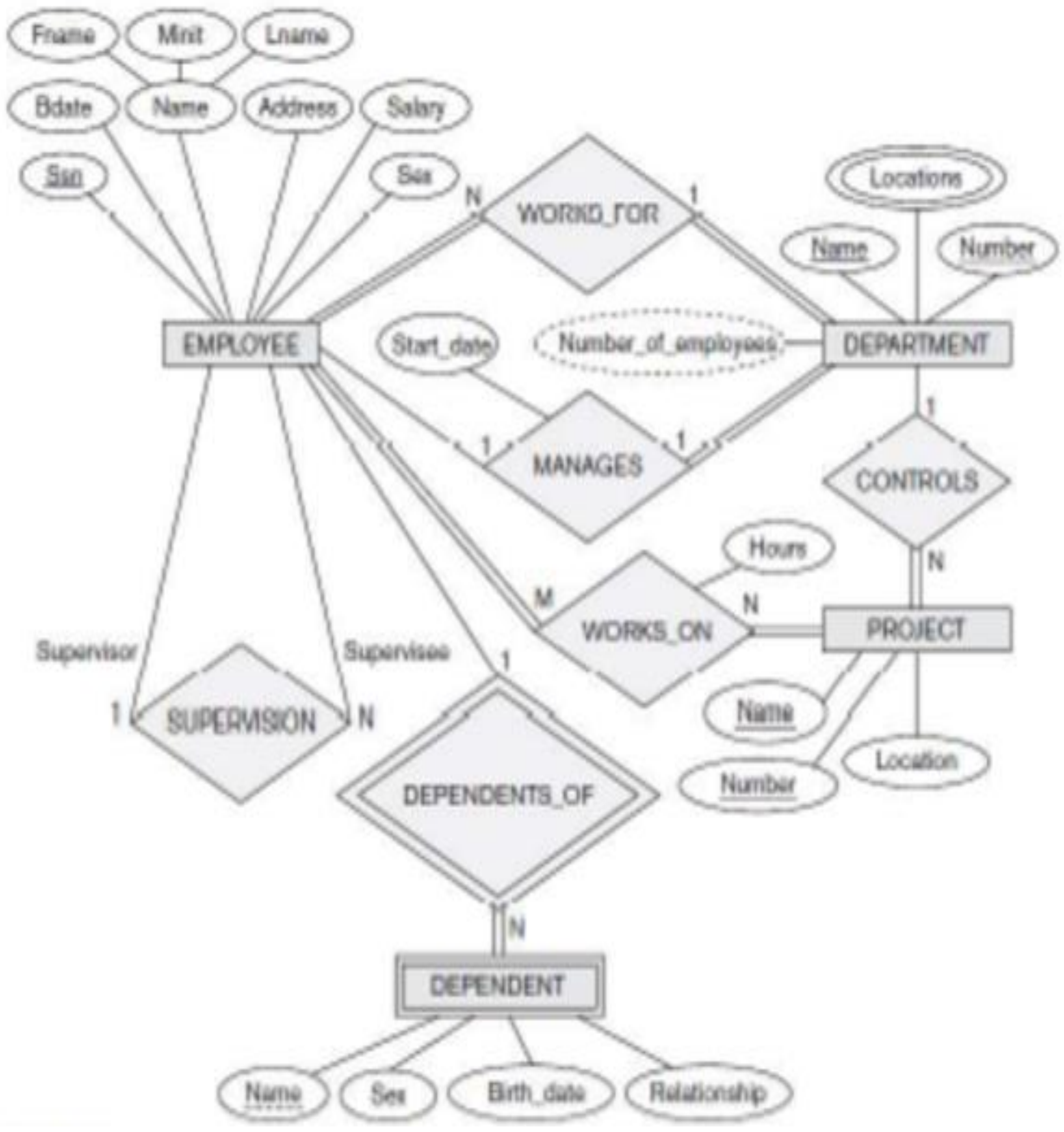


Figure: ER diagram for Company Database

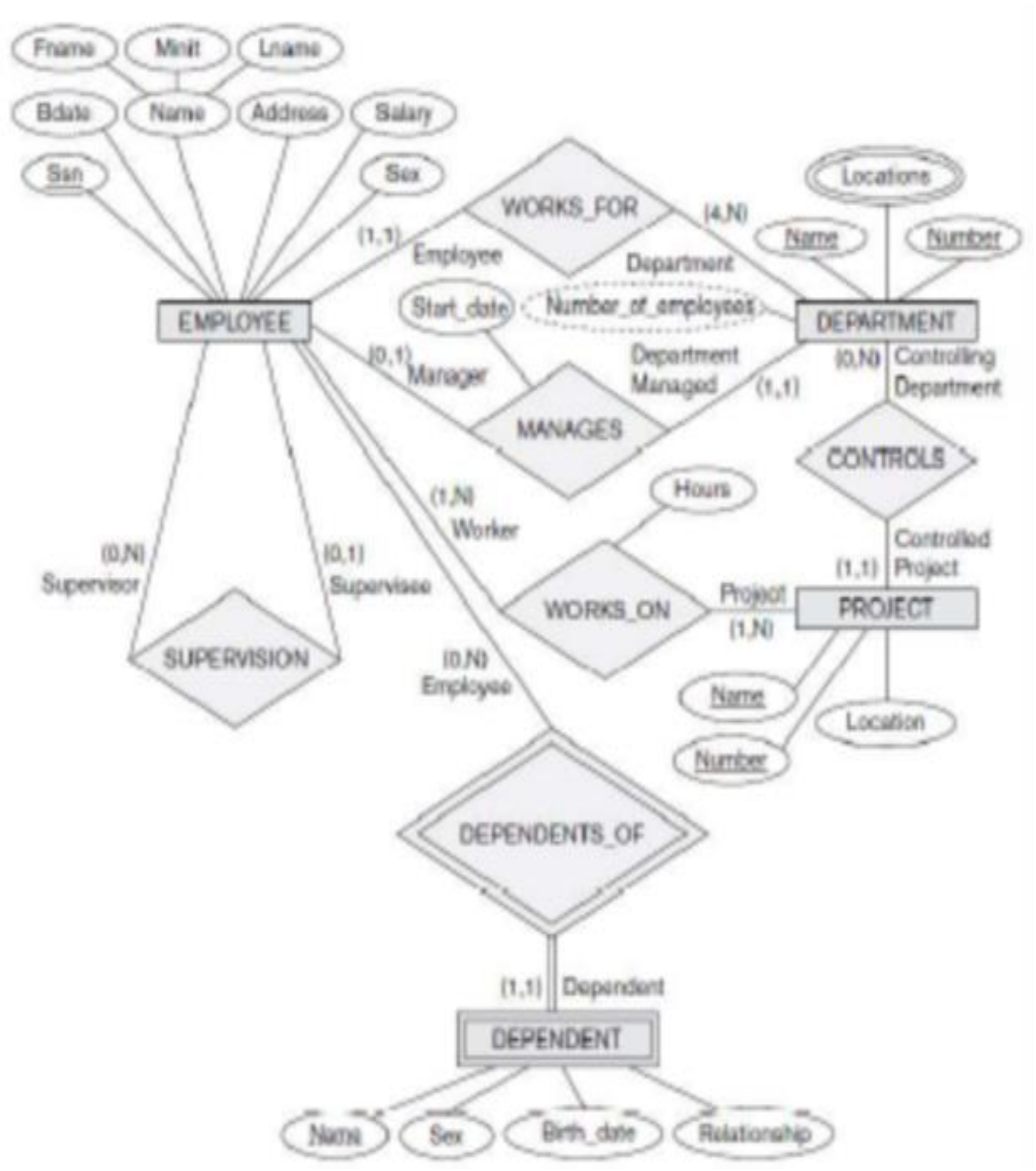


Figure: ER diagram for Company Database (using alternative notation)

5.5 Relationship Types of Degree Higher Than Two

5.5.1 Choosing between Binary and Ternary (or Higher-Degree) Relationships

A relationship type R of degree n will have n edges in an ER diagram, one connecting R to each participating entity type.

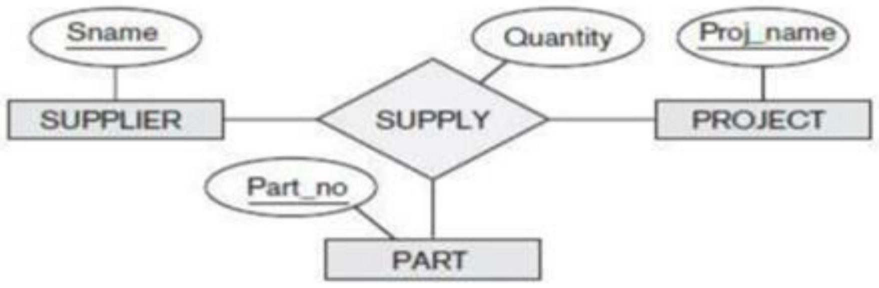


Fig: The SUPPLY relationship

Figure shows the ER diagram notation for a ternary relationship. SUPPLY is a set of relationship instances (s, j, p), where s is a SUPPLIER who is currently supplying a PART p to a PROJECT j.

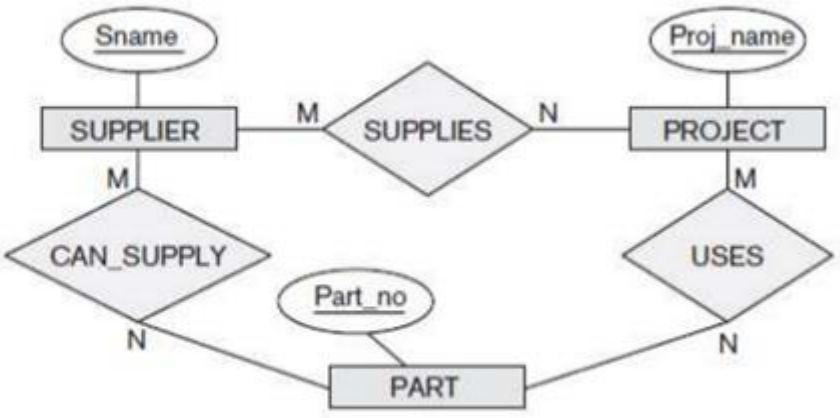
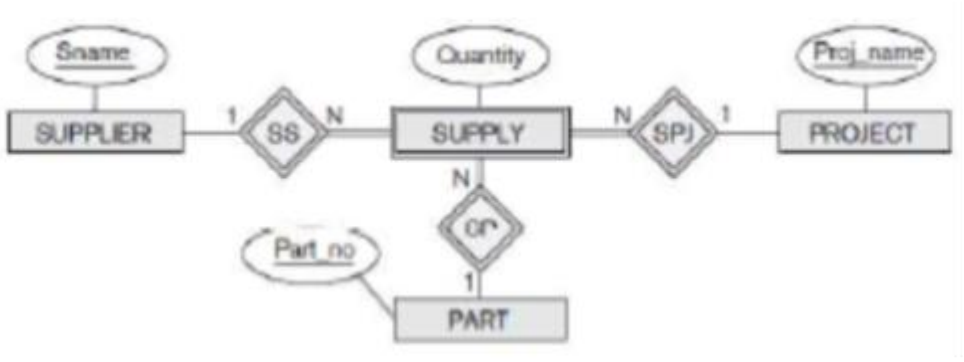


Fig: ER diagram for three binary relationship and SUPPLIES

Figure shows an ER diagram for three binary relationship types CAN_SUPPLY, USES, and SUPPLIES. CAN_SUPPLY between SUPPLIER and PART, includes an instance (s, p) whenever supplier s can supply part p (to any project). USES between PROJECT and PART, includes an instance (j , p) whenever project j uses part p. SUPPLIES between SUPPLIER and PROJECT, includes an instance (s, j) whenever supplier s supplies some part to project j.

Some database design tools are based on variations of the ER model that permit only binary relationships. In this case, a ternary relationship such as SUPPLY must be represented as a weak entity type, with no partial key and with three identifying relationships.



The three participating entity types SUPPLIER, PART, and PROJECT are together the owner entity types. Hence, an entity in the weak entity type is identified by the combination of its three owner entities from SUPPLIER, PART, and PROJECT.

5.5.2 Constraints on Ternary (or Higher-Degree) Relationships

There are two notations for specifying structural constraints on n-ary relationships.

1) based on the cardinality ratio notation of binary relationships displayed
   - 1, M, or N is specified on each participation arc (both M and N symbols stand for many or any number)
2) based on the (min, max) notation
   - specifies that each entity is related to at least min and at most max relationship instances in the relationship set

5.6 Relational Database Design Using ER-to-Relational Mapping.

ER-to-Relational Mapping Algorithm

Step 1: Mapping of Regular Entity Types.

Step 2: Mapping of Weak Entity Types.

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

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

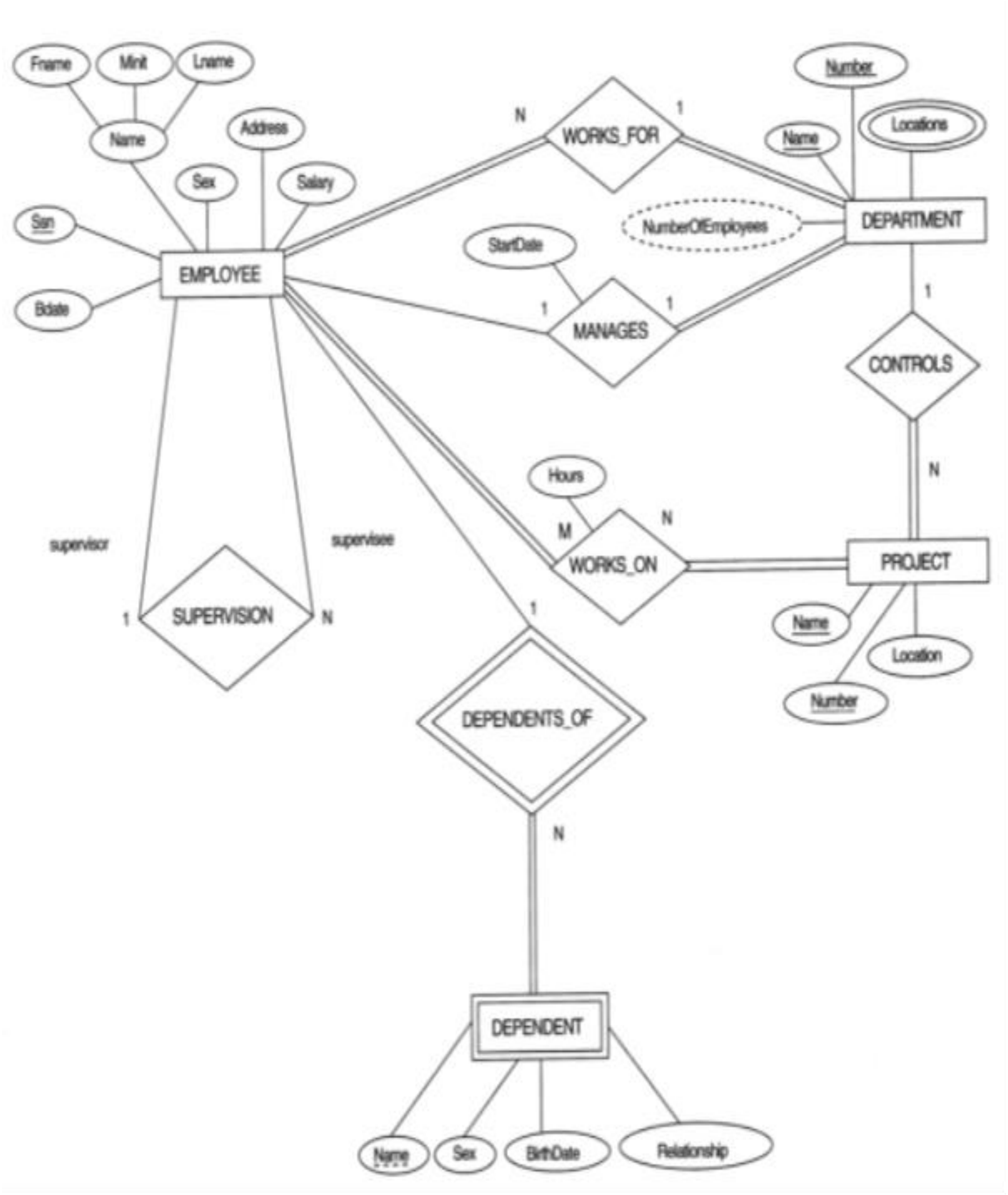
Step 5: Mapping of Binary M:N Relationship Types.

Step 6: Mapping of Multivalued attributes.

Step 7: Mapping of N-ary Relationship Types.

Step 1: Mapping of Regular Entity Types. For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E. Choose one of the key attributes of E as the primary key for R. If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R.

Example: We create the relations EMPLOYEE, DEPARTMENT, and PROJECT in the relational schema corresponding to the regular entities in the ER diagram. SSN, DNUMBER, and PNUMBER are the primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT as shown. The ER conceptual schema diagram for the COMPANY database.



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

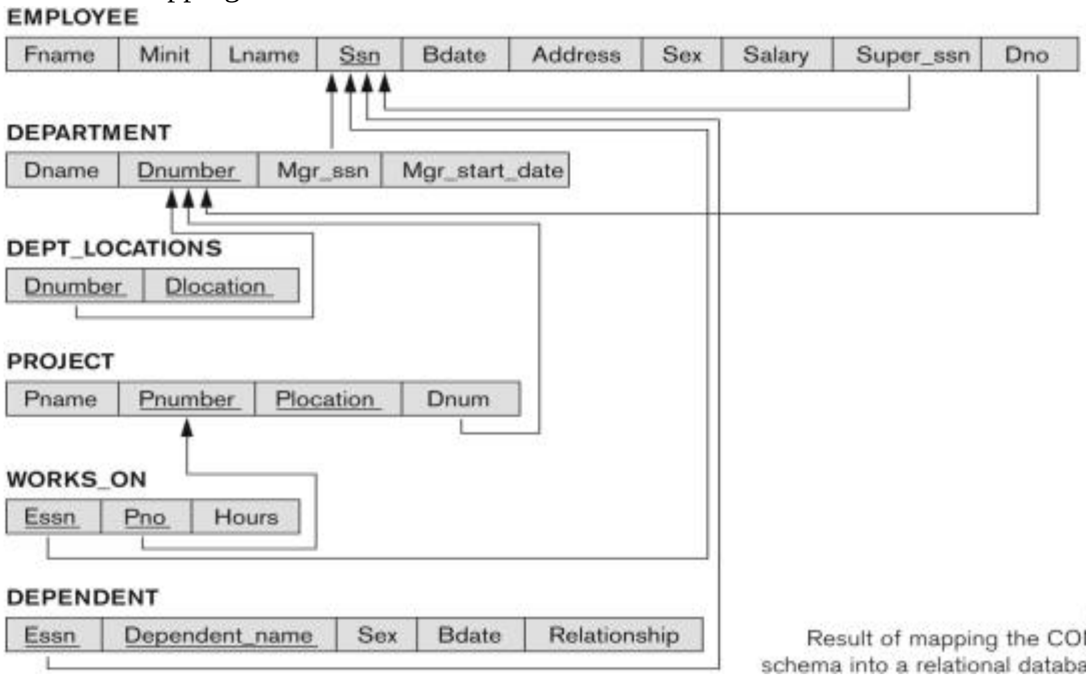


Figure 7.2
Result of mapping the COMPANY ER schema into a relational database schema.

Step 2: Mapping of Weak Entity Types For each weak entity type W in the ER schema with owner entity type E, create a relation R & include all simple attributes (or simple components of composite attributes) of W as attributes of R. Also, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any.

Example: Create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT. Include the primary key SSN of the EMPLOYEE relation as a foreign key attribute of DEPENDENT (renamed to ESSN). The primary key of the DEPENDENT relation is the combination {ESSN, DEPENDENT_NAME} because DEPENDENT_NAME is the partial key of DEPENDENT.

Step 3: Mapping of Binary 1:1 Relation Types For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. There are three possible approaches:

1) Foreign Key approach: Choose one of the relations-say S-and include a foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role of S.

Example: 1:1 relation MANAGES is mapped by choosing the participating entity type DEPARTMENT to serve in the role of S, because its participation in the MANAGES relationship type is total.

2) Merged relation option: An alternate mapping of a 1:1 relationship type is possible by merging the two entity types and the relationship into a single relation. This may be appropriate when both participations are total.

3) Cross-reference or relationship relation option: The third alternative is to set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types.

Step 4: Mapping of Binary 1:N Relationship Types. For each regular binary 1:N relationship type R, identify the relation S that represent the participating entity type at the N-side of the relationship type. Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R. Include any simple attributes of the 1:N relation type as attributes of S.

Example: 1:N relationship types WORKS_FOR, CONTROLS, and SUPERVISION in the figure. For WORKS_FOR we include the primary key DNUMBER of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it DNO.

Step 5: Mapping of Binary M:N Relationship Types. For each regular binary M:N relationship type R, create a new relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S. Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S.

Example: The M:N relationship type WORKS_ON from the ER diagram is mapped by creating a relation WORKS_ON in the relational database schema. The primary keys of the PROJECT and EMPLOYEE relations are included as foreign keys in WORKS_ON and renamed PNO and ESSN, respectively. Attribute HOURS in WORKS_ON represents the HOURS attribute of the relation type. The primary key of the WORKS_ON relation is the combination of the foreign key attributes {ESSN, PNO}.

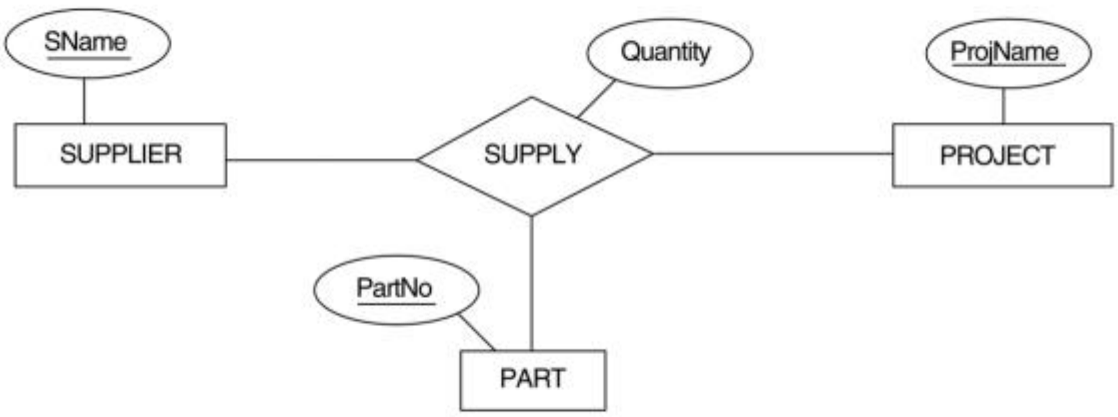
Step 6: Mapping of Multivalued attributes. For each multivalued attribute A, create a new relation R. This relation R will include an attribute corresponding to A, plus the primary key attribute K-as a foreign key in R-of the relation that represents the entity type of relationship type that has A as an attribute. The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.

Example: The relation DEPT_LOCATIONS is created. The attribute DLOCATION represents the multivalued attribute LOCATIONS of DEPARTMENT, while DNUMBER-as foreign key-represents the primary key of the DEPARTMENT relation. The primary key of R is the combination of {DNUMBER, DLOCATION}.

Step 7: Mapping of N-ary Relationship Types. For each n-ary relationship type R, where n>2, create a new relationship S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types. Also include any simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes of S.

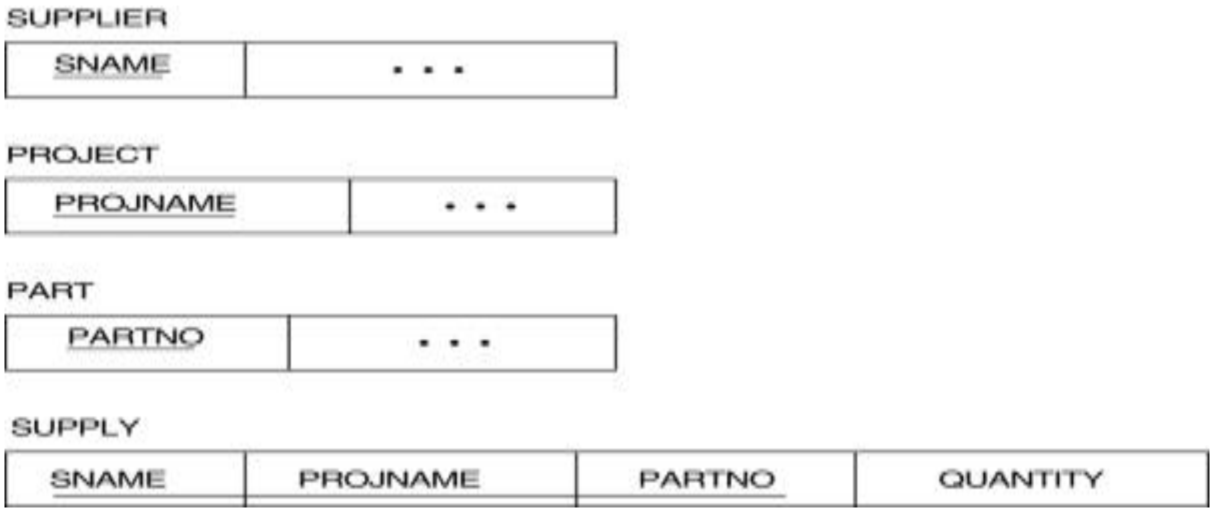
Example: The relationship type SUPPY in the ER on the next slide. This can be mapped to the relation SUPPLY shown in the relational schema, whose primary key is the combination of the three foreign keys {SNAME, PARTNO, PROJNAME}

(a)



Ternary relationship types. (a) The SUPPLY relationship.

Mapping the n-ary relationship type SUPPLY from Figure



SUPPLIER

SNAME	. . .

PROJECT

PROJNAME	. . .

PART

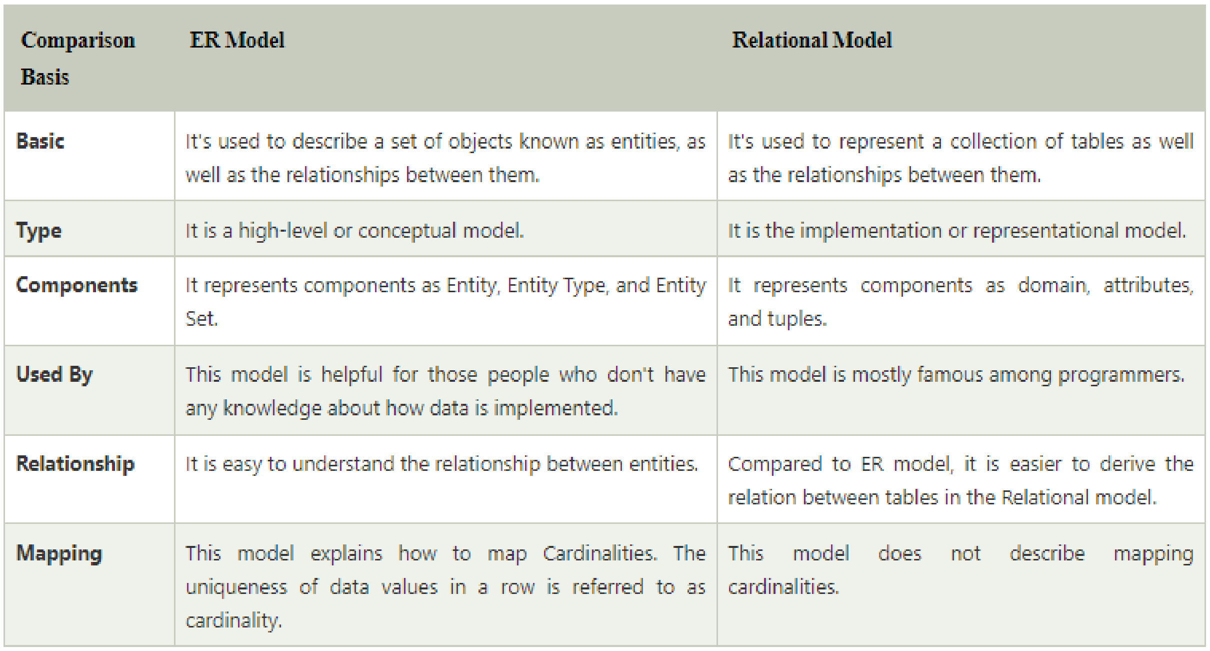
PARTNO	. . .

SUPPLY

SNAME	PROJNAME	PARTNO	QUANTITY

<u>Summary of Mapping constructs and constraints</u>

Comparision between ER Model and Relational Models



Comparison Basis	ER Model	Relational Model
Basic	It's used to describe a set of objects known as entities, as well as the relationships between them.	It's used to represent a collection of tables as well as the relationships between them.
Type	It is a high-level or conceptual model.	It is the implementation or representational model.
Components	It represents components as Entity, Entity Type, and Entity Set.	It represents components as domain, attributes, and tuples.
Used By	This model is helpful for those people who don't have any knowledge about how data is implemented.	This model is mostly famous among programmers.
Relationship	It is easy to understand the relationship between entities.	Compared to ER model, it is easier to derive the relation between tables in the Relational model.
Mapping	This model explains how to map Cardinalities. The uniqueness of data values in a row is referred to as cardinality.	This model does not describe mapping cardinalities.