
UNIT 2 OVERVIEW OF LOGICAL DATABASE DESIGN

Structure

- 2.0 Introduction
- 2.1 Objectives
- 2.1 The Steps of Database Design
 - 2.2.1 Conceptual Design
 - 2.2.2 Schema Refinement
 - 2.2.3 Physical database Design and Tuning
- 2.3 ER Model
- 2.4 ER Model Basics
 - 2.4.1 Entity
 - 2.4.2 Entity Type and Entity Set
- 2.5 Attributes
 - 2.5.1 Attribute
 - 2.5.2 Key Attributes in Entity Types
 - 2.5.3 Composite vs. Simple attributes
 - 2.5.4 Single vs. Multivalued Attributes
 - 2.5.5 Derived vs. Stored Attributes
 - 2.5.6 Null Values
 - 2.5.7 Value Sets of Attributes
- 2.6 Relationship
 - 2.6.1 Relationship
 - 2.6.2 Degree of Relationship Type
 - 2.6.3 Structural Constraints
- 2.7 Weak Entities
- 2.8 Components of an E-R diagram
- 2.9 ER Diagram Development Examples
- 2.10 Summary

2.0 INTRODUCTION

When an organisation decides to use database system then it is required to study the needs of the organisation. In the case of a small organisation with few users, having small volume of data, with no need for online query or update, a database system may not be necessary. In an organisation with a large number of users, and where decision making is distributed, the need for concurrent access to shared data is addressed by a database system.

In an organisation where large number of users and applications exist, the database system provides data independence, insulating these users and applications from changes. For the database to meet its objective, its design must be complete and consistent. All the significant inputs should be used in the design process, including the inputs of the users. The external schema allows multiple views of the data contained in the database. Designing a database system requires gathering details about the applications and transactions that are to be supported and the classes of users that will use the system.

The principal topic of this unit is entity-relationship model, which is basic step in the database design. The entity-relationship (E-R) data model is based on a perception of a real world that consists of a set of basic objects called entities, and of relationships among these objects. The E-R model is one of several semantic data models; the semantic aspect of data model lies in the attempt to represent the meaning of the data. The E-R model is extremely useful in mapping the meanings

and interactions of real-world enterprise onto a conceptual schema. Because of this utility, many database-design tools draw on concepts from the E-R model.

2.1 OBJECTIVES

At the end of this unit, you should be able to:

- recognise and define various entities in the enterprise;
- define relationship among various entities;
- gather information about these entities and relationships that should be stored in the database;
- recognise integrity constraints or business rules that hold;
- make the Entity-Relationship diagram.

2.2 THE STEPS OF DATABASE DESIGN

The three basic steps involved are:

2.2.1 Conceptual Design

It includes the several semantic data models. The Semantic aspect of the model lies in the attempt to represent the meaning of the data. For the conceptual design, the E-R models are used.

Entity Relation diagrams provides you with a graphical methodology for designing the structure of your database. There is a well-defined mapping from E-R diagrams into a relational schema making E-R diagrams a very practical tool for Database design.

2.2.2 Schema Refinement

It is called Normalisation. It checks relational schema for redundancies and related anomalies. Normalisation is a very important component of good database design. Redundancies in data not only makes databases larger, but also creates the potential for update and delete anomalies which can cause a database to become inconsistent. The normalisation will be discussed in the next unit.

2.2.3 Physical database Design and Tuning

It considers typical workloads and further refines the database design. Normally, it is handled by a Database Administrator.

2.3 ER MODEL

The ER model is a:

- High-level conceptual data model.
- Frequently used for conceptual design of database applications.

The E-R diagram (ERD) was originally proposed by Peter Chen for the design of relational database systems. The E-R model is extremely useful in mapping the meaning and interactions of real-world enterprises. Before making E-R diagram, we identify the following:

1. What are entities and relationships in the enterprise?
2. What information about these entities and relationships should we store in the database?
3. What are the integrity constraints or business rules that hold?

So a set of primary components is identified for the ERD; data objects, attributes, relationships and various type indicators. The primary purpose of the ERD is to represent data objects and their relationships.

For example, in general we can represent two entities and relationship between them as follows:

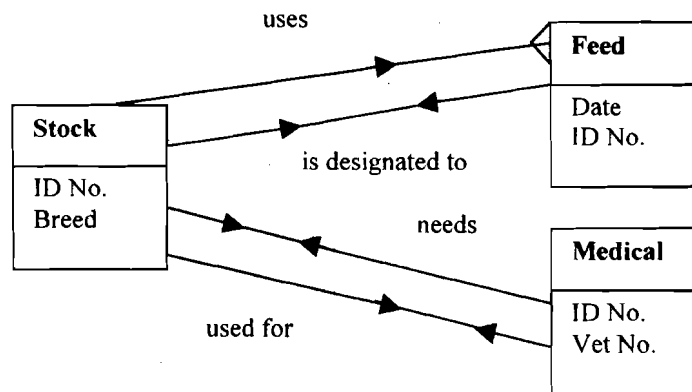


Figure 1: Representing relationship between various entities

Large database management system vendors implements ER model, example Oracle CASE, ERWIN etc.

2.4 ER MODEL BASICS

2.4.1 Entity.

An entity is a data object, a thing, an occurrence or event, a role, an organisational unit, a place or a structure in the real world that is distinguishable from all other objects.

For example, Employee or Department in a Company or an Enterprise is different entities.

The other examples could be:

Data Object - Anything that produces or consumes information.

Thing - A report or a display.

Occurrence - A Telephone Call.

Event - An Alarm.

Role - Salesperson.

Organisational Unit - Accounting Department.

Place - Warehouse.

Structure - File.

An entity may be **concrete** like person or Book, or it may be **abstract** like holiday or concept.

2.4.2 Entity Type and Entity Set

Entity Type: It defines a set of entities that have the same attributes

- Represented as a rectangle in ER diagram.
- Entity type describes the schema for a set of entities that share the same structure.

Entity Set: An entity set is a collection of similar entities, which share same properties or attributes. Each entity set has a key and each attribute has a domain. The individual entities that constitute a set are said to be extension of the entity set.

Example is set of all employees.

We can assume Employee of a university as one of the Entity set and Mr. ABC and Mr. XYZ as extensions of the above entity set.

2.5 ATTRIBUTES

2.5.1 Attribute

Attributes are descriptive properties possessed by each member of entity set. They can be used to—

- name an instance of an entity;
- describe the instance;
- makes reference to another instance in another table.

Attributes Formally

- Defined as function $A | A:E \rightarrow P(V)$, where $P(V)$ represents powerset of value set V .
- Powerset is set of all subsets
- Value of attribute A for entity e is $A(e)$
- $A(e)$ must be a singleton for single-valued attributes
- For a composite attribute A the value set V is the Cartesian product of sets: $P(V_1), P(V_2), \dots, P(V_n)$ where V_1, V_2, \dots, V_n are values sets for the simple components attributes that form A :

$$V = P(V_1) \times P(V_2) \times \dots \times P(V_n)$$

In some cases values for the identifier are unique, although this is not a requirement. For example, the various entities along with their attributes can be listed as follows:

Employee
Employee-Id
Name
Address
Tel No.
Date of Birth
Date of Joining
Department
Designation
Salary

Department
Dept-Id
Dept-name
Budget

ACCOUNT
Name
Account No.
Address
Phone No.
Balance
YTD Deposits
YTD Withdraws

CHEQUE
Amount
Payee
Check No.
Date

2.5.2 Key Attributes in Entity Types

- Key (or uniqueness) constraints can be applied to entity types;
- Key attribute's values are distinct for each individual entity;
- Each key attribute has its name underlined;
- Key must hold for every possible extension of the entity type.

2.5.3 Composite vs. Simple attributes

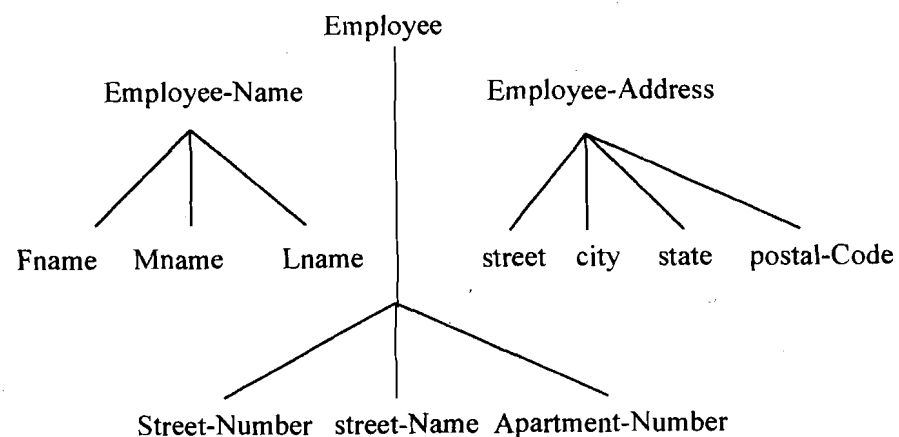
Simple attributes are the one's that can not be divided into sub-parts, for example, Employee-Id in the entity Employee, Account No. in the entity Account.

On the other hand Composite attributes can be divided into subparts. Example, Name (First Name, Middle Name, Last Name), Date (Day, Month, Year) etc.

Entity Set:

Composite Attributes:

Component Attributes:



2.5.4 Single vs. Multivalued Attributes

Single valued attributes have single value for the particular entity, while multivalued have set of values for particular entity.

Example, in the entity Employee, Employee-Id is the single valued attribute. Other examples for single valued attributes can be age, car, color etc.

Telephone Number is the multivalued attribute as an employee can have zero, one or more telephones.

We can represent Attributes as follows:

- Parenthesis (()) for composite attributes; and brackets ({ }) for multivalued attributes;
- Assume a person can have more than one residence and each residence can have multiple telephones, then the attribute AddressPhone can be represented as:

{AddressPhone({Phone(AreaCode,PhoneNum)}),Address(Street(Street-Number, Street-Name, Apartment-Number),City,State,PostalCode)) }

2.5.5 Derived vs. Stored Attributes

Value of derived type of attribute can be derived from the values of other related attributes or entities, for example, the age of an entity Employee can be derived from his Date of Birth.

On the other hand stored attributes are the attributes whose values are directly stored in the entity set, for example, Date of joining, Address etc.

2.5.6 Null Values

A Null value is used in the place when an entity does not have any value for an attribute, for example, in entity Employee, Telephone Number can have the null value as an employee may not have any telephone. Certain attributes like Employee-Id, Name can never take null value.

2.5.7 Value Sets of Attributes

Value Sets: Value sets specify the set of values that may be assigned to a particular attribute of an entity. Value set is the same as domains. They are not displayed on the ER diagram.

For example, age is between 16 and 70. S/he can join the company at the age of 16 years or onwards and at the age of 70 s/he gets retired.

Also Employee Name is the value set, as it is set of strings of alphanumeric characters.

2.6 RELATIONSHIP

2.6.1 Relationship

It is defined as association among two or more entities. Example, Rakesh works in Pharmacy department. Here Rakesh and Pharmacy department, are two different entities. The relationship among them is that one works in another.

In general sense we can say that an employee works in some department.

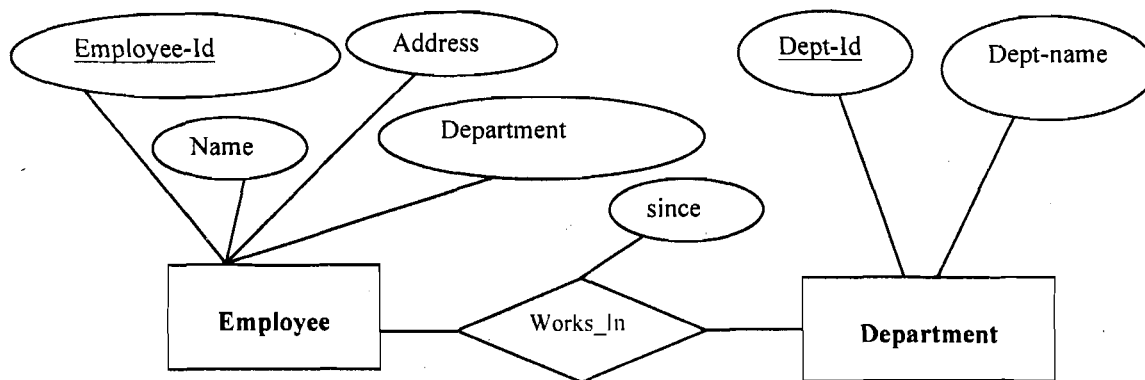


Figure 2: ER Diagram for representing employee works_in department

Relationship Set: Relationship set is defined as the collection of similar relationships. An n-ary relationship set R relates n entity sets E1 ... En; each relationship in R involves entities e1 (in E1), ..., en (in En). Relationship sets can also have descriptive attributes (example, the **since** attribute of Works_In).

In translating a relationship set to a relation, attributes of the relation must include:

- Keys for each participating entity set (as foreign keys). This set of attributes forms superkey for the relation.
- All descriptive attributes.

The following SQL statement will create the table **works_in**

```

CREATE TABLE Works_In (
    Employee-Id INTEGER,
    Dept-Id CHAR(6),
    since DATE,
    PRIMARY KEY (Employee-Id, Dept-Id),
    FOREIGN KEY (Employee-Id) REFERENCES Employee,
    FOREIGN KEY (Dept-Id) REFERENCES Department)
  
```

- Relationship Type defines an association between n entity types E1, ..., En, and also defines the structure of the relationship set
- A relationship set is a subset of the Cartesian product: $E1 \times E2 \times \dots \times En$.
- Each Ei is said to 'participate' in the relationship type.

2.6.2 Degree of Relationship Type

The degree of a relationship type is the number of participating entity types.

For example, in Figure 2, WORKS_IN, is a relationship of degree 2 i.e., it is binary relationship.

The relationship Supply among entities - supplier/part/project is of degree 3 therefore it is ternary relationship.

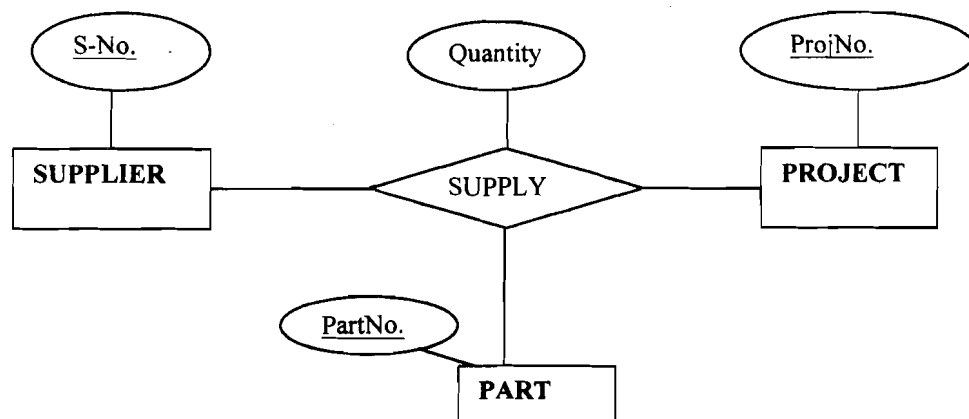


Figure 3: ER Diagram for the relationship SUPPLY among entities SUPPLIER, PROJECT and PART

Same entity type could participate in different relationship types, or in different 'roles' in same relationship type. Each entity type that participates in a relationship type plays a particular role in the relationship type. The role name signifies the role that a participating entity from the entity type plays in each relationship instance.

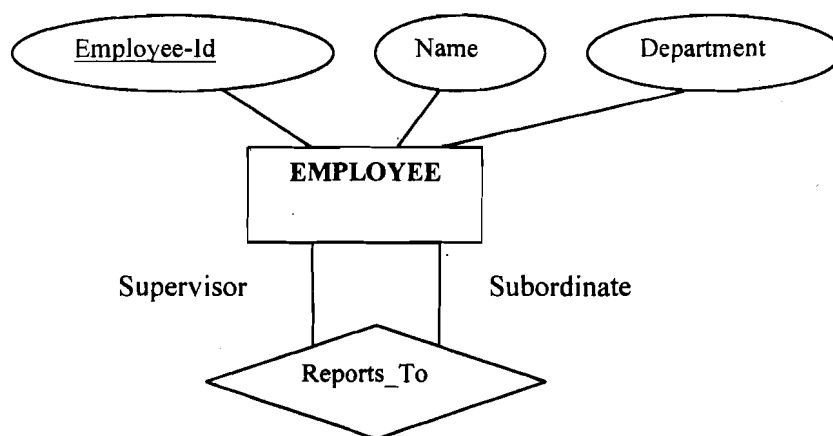


Figure 4: ER Diagram for the relationship Reports_To representing various roles of employee in an organisation

2.6.3 Structural Constraints

Two kinds of structural constraints can be placed on relationship types. These are:

- Cardinality Ratio;
- Participation Constraints.

Cardinality Ratio

It defines the number of entities to which another entity can be associated via a relationship set. It is sometimes referred to as Mapping Cardinalities.

For a binary Relationship set R between entity sets A and B, the mapping Cardinalities must be one of the following:

One-to-one: An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

For example, in an ideal society, A relationship between husband and wife is One-to-one.

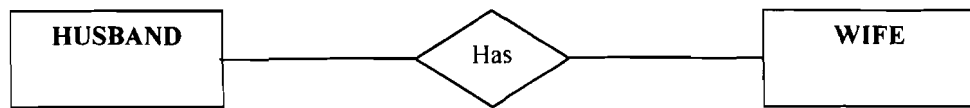


Figure 5: ER Diagram to explain one-to-one relationship

One-to-Many: An entity in A is associated with any number of entities in B. An entity in B, however, can be associated with at most one entity in A.

For example, each department has at most one manager, according to the key constraint on the relation **Manages**, but a manager can manage one or more departments. Thus this is an example of One-to-many relationship.

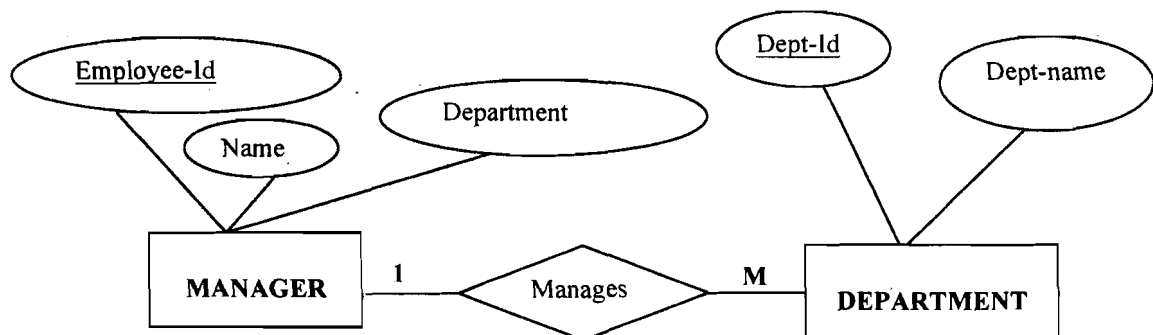


Figure 6: ER Diagram to explain one-to-many relationship

Many-to-one: An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number of entities in A.

For example, two sisters have a common father, thus, forming an example of many-to-one relationship.

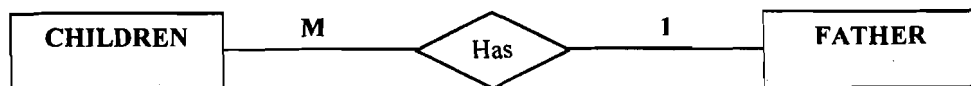


Figure 7: ER Diagram to explain many-to-one relationship

Many-to-Many: An entity in A is associated with any number of entities in B and an entity in B is associated with any number of entities in A.

For example, to understand Many-to-many relationship we can again consider the example of Works_In relationship in Figure 2. An employee can work in many departments and a department can have many employees.

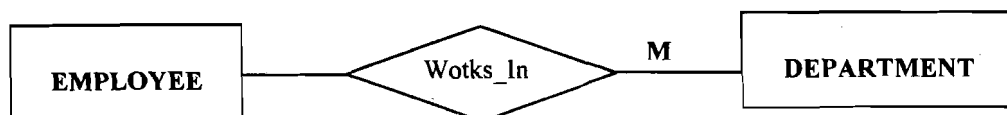


Figure 8: ER Diagram to explain many-to-many relationship

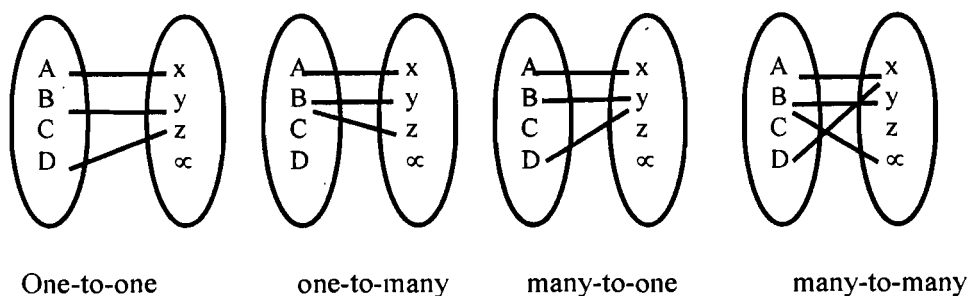


Figure 9: Relationship Cardinality

Translating ER Diagrams with Key Constraints

Before translating ER Diagram, we need to discuss that why do we need key constraints?

Consider the situation: A father has two daughters, now there has to be some way for father to identify his daughters uniquely. Thus he gives them a unique name, thus in this case name becomes the key.

Thus we can very well observe that in **Many** side of the relationships there has to be some key for the unique identification.

We can see it in the following Diagram:

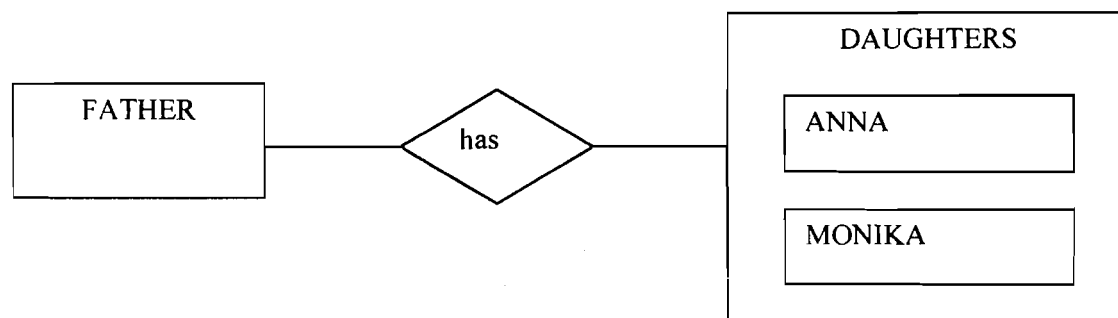


Figure 10: Importance of key in one-to-many relationship

Our next step is to understand how do we translate ER Diagrams with key constraints?

Map relationship to a table

Note that in the example of Works_In relationship, on assuming the constraint that an employee can work only in one department, Dept-Id becomes the key and we will have to make separate tables for Employees and Departments. (Please refer to figure 2) These tables look like follows:

The table **Employee**

Employee-Id	Name	Address	Department
1918	Amit	GH-9, ND-87	SND
2223	Mini	2, LIC Colony, ND-86	NMG
1672	Vineet	303, Bhera Enclave, PV, ND	VOIP

The table **Department**

Dept-Id	Dept-name
SND	Satellite Networking Division
NMG	Network Management Group
VOIP	Voice over Internet Protocol

Since each employee works in a unique department, we can combine Employee and Departments, thus, we have the table **Works_In**.

The table **Works_In**

Employee-Id	Dept-Id	Since
1918	SND	02-10-98
2223	NMG	07-05-95
1672	VOIP	22-07-96

The SQL for creating the tables **Employee**, **Department** and **Works_In** will be like as follows:

```
CREATE TABLE Employee(
Employee-Id INTEGER,
Name CHAR(25),
Address CHAR(50),
Department CHAR(6))
```

```
CREATE TABLE Department(
Dept_Id CHAR(6),
Dept-name CHAR(25) )
```

```
CREATE TABLE Works_In(
Employee-Id INTEGER,
Dept_Id CHAR(6),
since DATE,
PRIMARY KEY (Employee-Id, Dept_Id),
FOREIGN KEY (Employee-Id) REFERENCES Employee,
FOREIGN KEY (Dept_Id) REFERENCES Department)
```

Participation Constraints

The participation of an entity set E in a relationship set R is said to be total if every entity in E participates in at least one relationship in R. If only some entities in E participate in relationships in R, the participation of entity set E in relationship R is said to be partial.

Let us try to understand total relationship. Let us consider each department has a unique manager, we could instead combine Manages and Departments and have the table **Manages**.

The table **Manages**

Manager-Name	Employee-Id	Dept-Id	Since
Pratyush DasGupta	1002	SND	02-10-98
Mamta Bansal	1076	NMG	07-05-95
Mohan Hebbar	1115	VOIP	22-07-96

The SQL for creating the table will be like as follows:

```
CREATE TABLE Manages(
Manager-Name CHAR(25),
Employee-Id INTEGER,
Dept-Id CHAR(6),
since DATE,
```

```
PRIMARY KEY (Employee-Id, Dept-Id),
FOREIGN KEY (Employee-Id) REFERENCES Employees,
FOREIGN KEY (Dept-Id) REFERENCES Departments)
```

Let's try to find, does every department has a manager?

If answer is yes i.e., Every department has a manager, then this is a participation constraint: the participation of Departments in Manages is said to be total as every Dept-Id value in Departments table must appear in a row of the **Manages** table (with a non-null Employee-Id value).

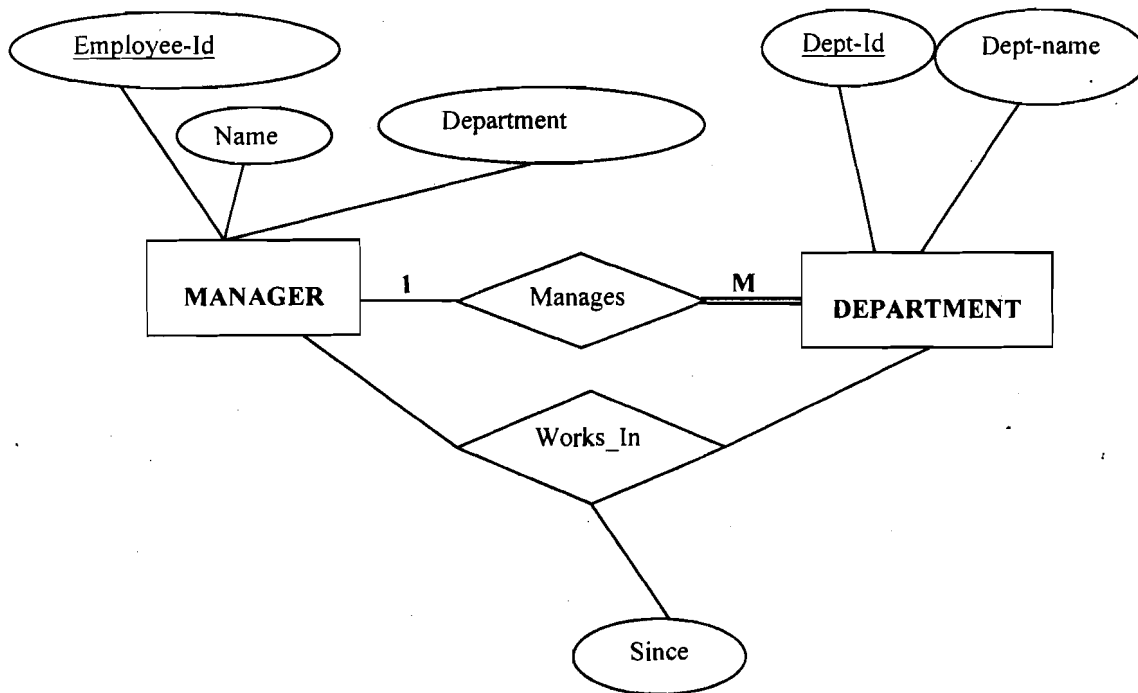


Figure 11: ER Diagram to explain total participation

Example of Partial Relationship

As we know that there are some schemes like LTC (Leave Travel Certificate) etc. These schemes can be called as Flexible Benefit Schemes. An employee can claim these FBSs if his or her spouse is not claiming it. Thus, out of husband and wife only one is eligible for FBP. Thus, we can define a relationship is eligible for between set of employees and FBP. Out of set all the employees in the company all employees may or may not be eligible for FBP, thus, this relationship is a partial relationship.

Participation Constraints in SQL

We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints).

```
CREATE TABLE Dept_Mgr ( Dept-Id CHAR(6), Dept-name CHAR(25), Employee-Id INTEGER
NOT NULL, since DATE, PRIMARY KEY (Dept-Id), FOREIGN KEY (Employee-Id)
REFERENCES Employees, ON DELETE NO ACTION)
```

Existence Dependencies

Another class of constraint is **existence dependencies**. It can be defined as - 'If the existence of entity X depends on the existence of entity Y, then X is said to be existence dependent on Y. Practically, if Y is deleted, so is X.'

Let us consider the relationship CCI between the entities CANDIDATE, COMPANY and INTERVIEW. A candidate is offered a job in the company if he appears in the interview and

qualifies it. Thus JOB_OFFER is an entity, which is dependent on the existence of the entity INTERVIEW. If there is no INTERVIEW then there is no JOB_OFFER. Thus, entity JOB_OFFER is existentially dependent on entity INTERVIEW.

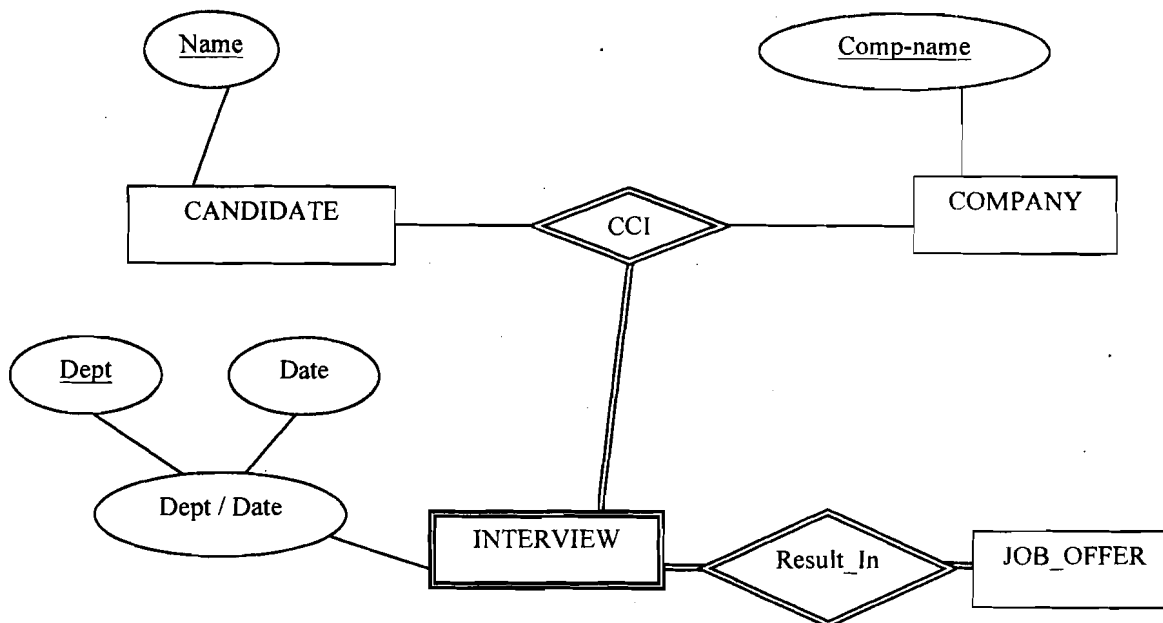


Figure 12: ER Diagram to explain existential dependency

2.7 WEAK ENTITIES

An entity set may not have sufficient attributes to form a primary key. Such an entity set is termed as weak entity set. An entity that has a primary key is termed as a strong entity set. A weak entity can be identified uniquely only by considering the primary key of another (owner) entity.

- Owner entity set and weak entity set must participate in a one-to-many relationship set (i.e., one owner, many weak entities).
- Weak entity set must have total participation in this identifying relationship set ('existence dependency').

Entities may not be distinguished by their attributes but by their relationship to another entity. Let us establish a relationship, DEDUCTIONS, between the entity EMPLOYEE and DEPENDENTS as shown in Figure 13. In this case the instances of the entity from the set DEPENDENTS are distinguishable only by their relationship with an instance of an entity from the entity set EMPLOYEE. The relationship set DEDUCTIONS is an example of an **identifying relationship** and the entity set DEPENDENTS is an example of a **weak entity**.

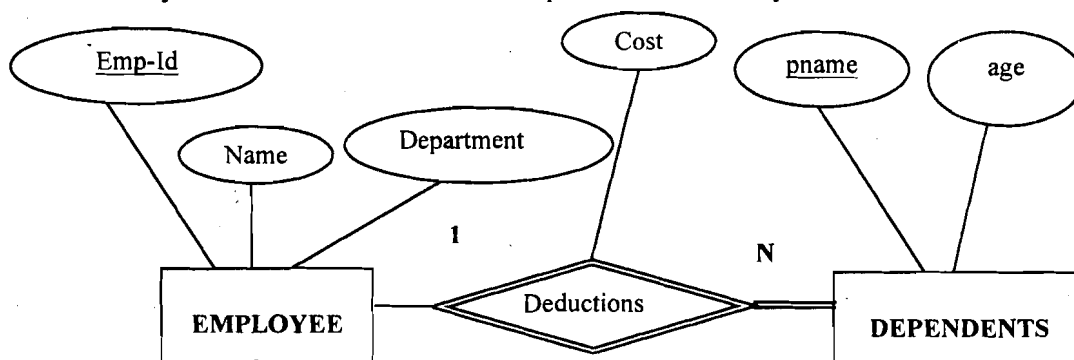


Figure 13: Converting an attribute association to a relationship and explaining weak entity set.

Identifying Relationship

- Identifying relationship is the relation type that relates a weak entity to its 'identifying owner'.
- Not every existence dependency results in a weak entity.
- Although a weak entity set does not have a primary key, we nevertheless need a means to distinguish among all those entities in the entity set that depend on one particular strong entity. The discriminator of a weak entity set is a set of attributes that allows this distinction to be made. The discriminator of a weak entity set is also called the partial key of the entity set.
- Primary Key is set of attributes that uniquely identify weak entities related to the same owner entity. It is formed by the Primary Key of the strong entity set on which the weak entity set is dependant, plus the weak entity set's discriminator.

For instance, let us assume The EMPLOYEE 1672(Vineet Anand) has two DEPENDENTS, Daksha Anand, his spouse and Swastik Anand, his son. These are distinct and can be distinguished from each other. The organisation can have another Anand in its employ (say Bharat Anand with Employee-Id as 1888), who has dependents Meena Anand, spouse and Sumit Anand, son.

Now note that the two instance (Vineet Anand, son) of the weak entity set DEPENDENTS associated with different instances of the strong entity set EMPLOYEE are not distinguishable from each other. They are nonetheless distinct because they are associated with different instances of the strong entity set EMPLOYEE. The primary key of a weak entity set is thus formed by using the strong entity set to which it is related, along with the discriminator of the weak entity. We rule out the case where a dependent such as Sumit Anand is the son of two different employees, namely his mother and father, since only one of them will claim him a deduction. However, if we allow this possibility, the relationship between EMPLOYEE and DEPENDENTS becomes many-to-many.

Translating Weak Entity Sets

Weak entity set and identifying relationship set is translated into a single table, and when the owner entity is deleted, all owned weak entities must also be deleted.



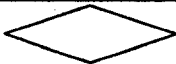
The SQL query for relationship with weak entity can be given as follows:


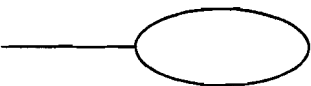
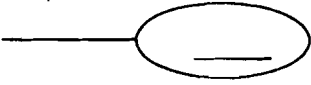

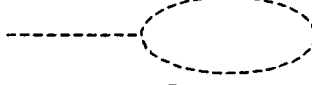
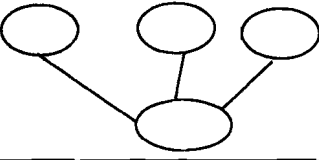



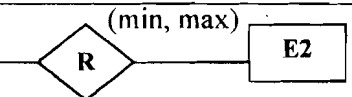
```
CREATE TABLE Deduction (pname CHAR(25), age INTEGER, cost REAL, Employee-Id
INTEGER NOT NULL, PRIMARY KEY (pname, Employee-Id), FOREIGN KEY (Employee-
Id) REFERENCES Employees, ON DELETE CASCADE)
```

2.8 COMPONENTS OF AN E-R DIAGRAM

Components of an E-R diagram

Following are the various components of an E-R diagram:

Symbol Name	Symbol	Meaning
Rectangles		It represents entity sets
Double Rectangles		It represents weak entity sets
Diamonds		It represents relationship sets

Symbol Name	Symbol	Meaning
Double Diamonds		It represents identifying relationship sets or types
Ellipses		It represents attributes
Underlined Ellipses		It represents key attributes
Double Ellipses		It represents multivalued attributes
Dashed ellipses		It represents derived attributes
		It represents composite attributes
Lines		It represents link attributes to entity sets and entity set to relationship sets
Double lines		It represents total participation of an entity in a relationship set i.e., total participation of E2 in R
Numbers 0,1,N		It represents Cardinality ratio 1:N for E1:E2 in R
(min, max)		It represents structural constraints (min, max) on participation of E in R.

2.9 ER DIAGRAM DEVELOPMENT EXAMPLES

Example 1: A Company ABC develops applications for the clients. Same ideas apply when a company develops an application for their own personal use. Thus, the company can work in "user mode" and "developer mode". Try to find out all possible entities and relationships among the entities. Give a step-by-step procedure to make an E-R diagram for the process of the company when they develop an application for the client and use the diagram to derive an appropriate base table.

Solution: We are going to design a database to hold information about the clients and the applications of a system. We will call this CAPO, for Client Application Package Organiser.

First let us define the entities of the system. At present we have two objects—namely **Client** and **Application**. Each of these is a noun, and thus eligible to be entities in the database.



Figure 14: Drawing Entities as Boxes

Taking these in turn, let's see if we can determine the correct entity types. Client is **independent** of anything else, and so is Application. So the entities clients and applications form an entity set.

Let us now consider what the relationship is between these two entities, if any. Obviously, the relationship among the entities depends on precise interpretation of spoken (or written) requirements, and we must pay meticulous attention to the contextual semantics - for example, just what do we mean by an **application**?

Is **Accounts Receivable** an application, or is the AR system installed at each client site regarded as a different application? Can the same application be installed more than once at a particular client site?

Before we answer these questions, have you noticed that another entity is trying to get into the act here? The client **site** seems to be another candidate entity. This is the kind of thing you need to be sensitive to at this stage of the process.

Let's deal with the relationship between Client and Site before coming back to Application. It is often easier to tackle what seems likely to prove simple before trying to resolve the apparently complex.

Each Client can have many sites, but each site belongs to one and only one client: This is a good example of a existential dependency and one-to-many relationship, and illustrates the precision and clarity of expression we employ when we read the following diagram.

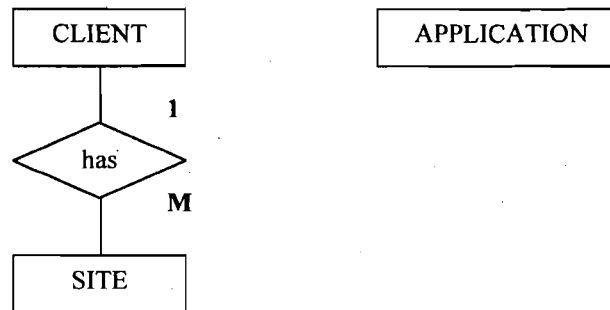


Figure 15: Adding a One-to-Many Relationship

Now the question arises that what entity type is Site?

We cannot have a site without a client, if there exist any site without a client then who would pay the company? Obviously a company gets the payment from the clients.

Now let's try and relate Application to the others: **we would like to install the same application at many client sites. We also hope to install more than one application at some of these sites.** That describes a many-to-many relationship between Site and Application:

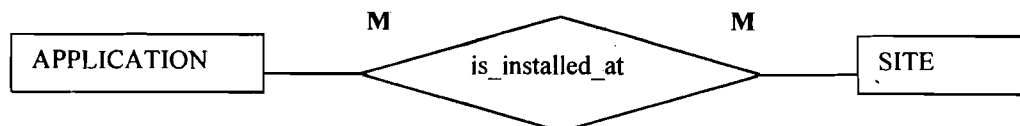


Figure 16: A Many-to-Many Relationship

To decompose the M:M relationship into 1:M relationship, we add a new entity called Installation in the above figure and by doing so we get the following figure:

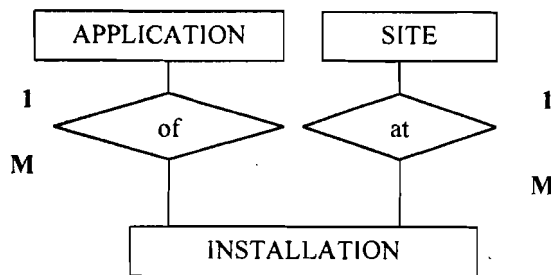


Figure 17: Resolving The Many-to-Many Relationship

Defining relationships in above figure:

of - relationship describing installation of applications, and

at - relationship describing installation at sites.

Note that entities can arise in one of two ways. Either we draw them because they were named in the specification, or as a result of resolving a M:M, as in this case. When we create a new entity in this way, we have to find a suitable name for it. This can often be based on the verb used to describe the M:M. Thus, from the statement **we can install the application at many sites** we get the name **Installation**.

Now we need to determine the correct type for the new entity. Always get the user to do this. Use the non-technical statement for each entity type to help decide which one describes it correctly. What we are concerned with here is the most effective way of uniquely identifying each record of this type. How do we want to identify each Installation?

Is an Installation independent of any other entity? i.e., can an entity Installation exist without being associated with the entities Client, Site and Application?

In this case, there cannot be an Installation until we can specify the Client, Site and Application. But since Site is existentially dependent on Client or we can say that Site is subordinate to Client, that means that Installation is a Combination entity type, identified by (Client) Site (it means Client or Site) and Application. We do not want to allow more than one record for the same combination of site and application.

But what if we also sell multiple copies of packages, and need to keep track of each individual copy (license number) at each site? The answer is that we need another entity. We may even find that we need separate entities for Application and Package. This will depend on what attributes we want to keep in each - how far our requirements differ in respect of each of these.

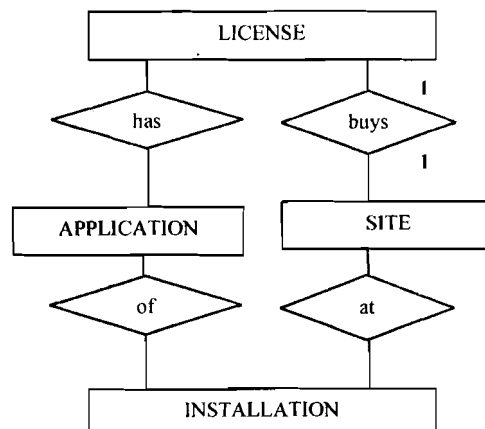


Figure 18: Adding More Entities

has - relationship describing that each application has one or more licenses,

buys - relationship describing each site buys the license copies.

We might decide that License should be subordinate to Package: the best unique identifier for the entity could be Package ID and license serial number. We could also define License as identified by Client Site, Application and License Copy (serial number).

The only objection to the subordinate identification is that one of the key elements has an externally assigned value - we do not issue license numbers, and thus have no control over their length and data type (up to 15 mixed alpha-numeric characters?) nor even over their uniqueness and unchangeability. It is always safer to base primary keys on internally assigned values.

What if we make License subordinate to Package, but substitute our own Copy serial number for the software manufacturer's License number? It seems that the Client Site is not an essential part of the identifier. Come to think of it, we sometimes buy copies of packages for stock, and later sell them. And the client is quite free to move the copy of the package to another site. For both of these reasons, we should definitely not make Client/Site part of the primary key of License.

The above discussion is trying to emulate the kind of mental acrobatics you have to be prepared to undertake when designing the database, and particularly when typing the entities.

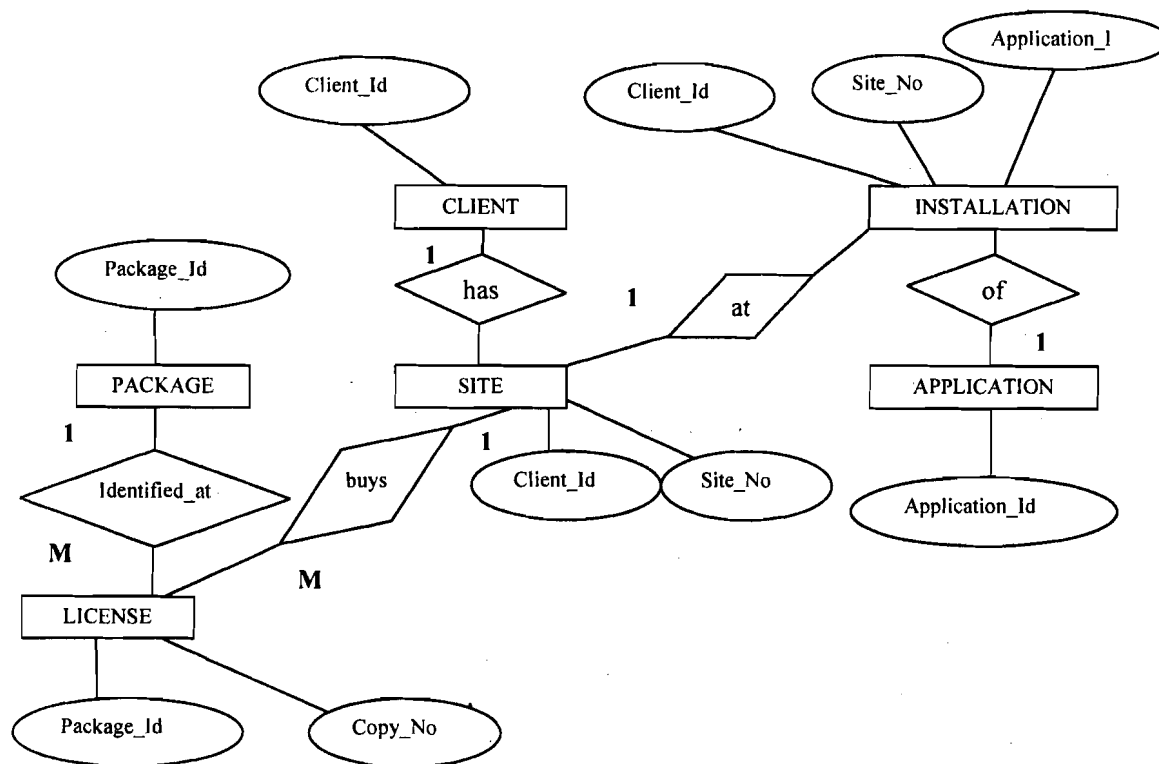


Figure 19: The Entity Types along with the attributes and keys

Let us list each entity identified so far, together with its entity type, primary keys, and any foreign keys.

Entity	Type	Primary Key	Foreign Keys
Client	Independent	Client ID	
Site	Subordinate	Client ID, Site No	
Application	Independent	Application ID	
Package	Independent	Package ID	
Installation	Combination	Client ID, Site No, Application ID	
License	Subordinate	Package ID, Copy No	Client ID, Site No

Now we are ready to switch on the computer, fire up Windows and any RDBMS say MS-Access or Oracle, and start defining the tables.

Example 2: An employee works for a department. If the employee is a manager then he manages the department. As an employee the person works for the project, and the various departments of a company control those projects. An employee can have many dependents. Draw an E-R diagram for the above company. Try to find out all possible entities and relationships among them.

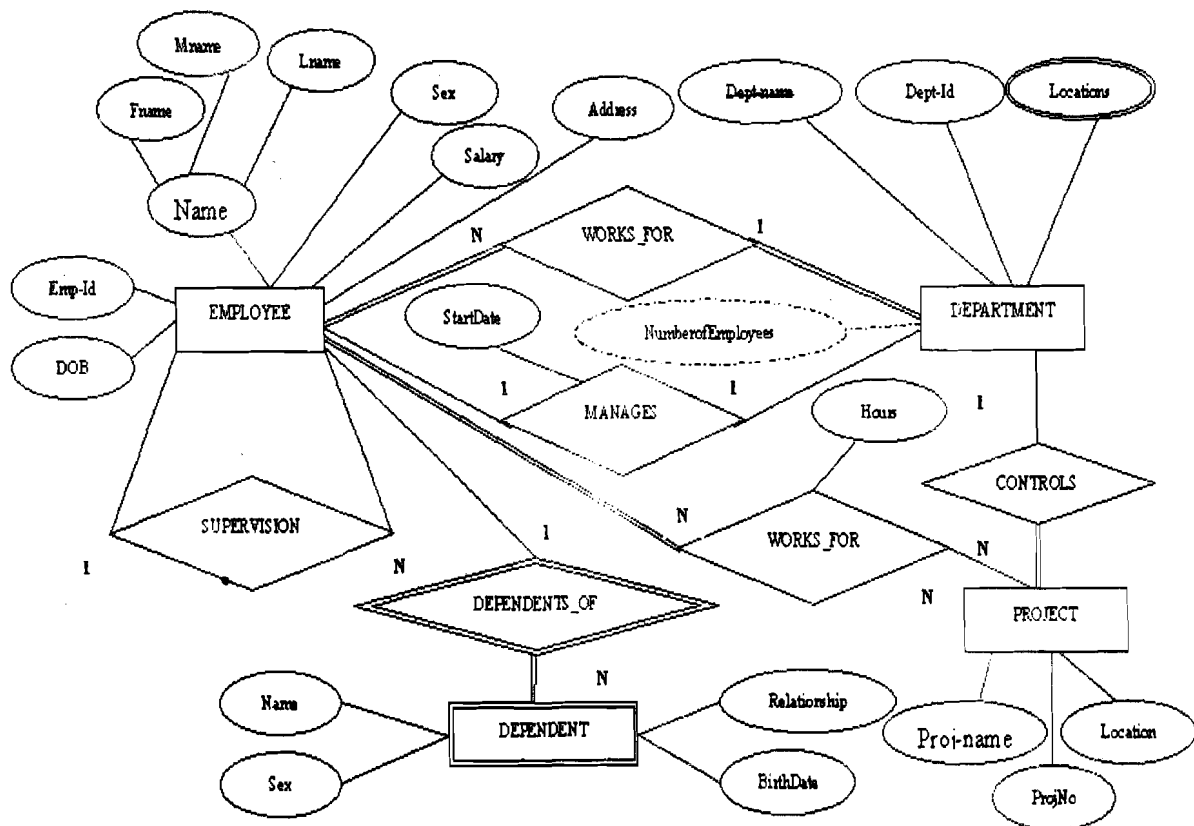


Figure 20: The E-R diagram for EMPLOYEE-DEPARTMENT database

In the above E-R diagram, EMPLOYEE is an entity, who works for the department i.e., entity DEPARTMENT, thus **works_for** is many-to-one relationship, as many employees work for one department. Only one employee (i.e., Manager) manages the department, thus **manages** is the one-to-one relationship. The attribute **Emp_Id** is the primary key for the entity EMPLOYEE, thus **Emp-Id** is unique and not-null. The primary key for the entity DEPT are **Dept-name** and **Dept_Id**. Along with other attributes, **NumberofEmployees** is the derived attribute on the entity DEPT, which could be recognised the number of employees working for that particular department. Both the entities EMPLOYEE and DEPARTMENT participates totally in the relationship **works_for** as at least one employee works for the department, similarly an employee works for at least one department.

The entity EMPLOYEES work for the entity PROJECTS. Since many employees can work for one or more than one projects simultaneously, thus works_for is the N:N relationship. The entity DEPARTMENT controls the entity PROJECT, and since one department controls many projects, thus, controls in a 1:N relationship. The entity EMPLOYEE participates totally in the relationship works_for as at least one employee works for the project. A project has no meaning if no employee is working in a project.

The employees can have many dependents, but the entity DEPENDENTS can not exist without the existence of the entity EMPLOYEE, thus, DEPENDENT is a weak entity.

We can very well see the primary keys for all the entities. The underlined attributes in the eclipses represent the primary key.

Example 3: A supplier located in only one city, supplies various parts for the projects to different companies located in various cities. Let's call this database as supplier-and-parts database. Draw the E-R diagram for the **supplier-and-parts** database.

The E-R diagram for supplier-and-parts database is given as follows:

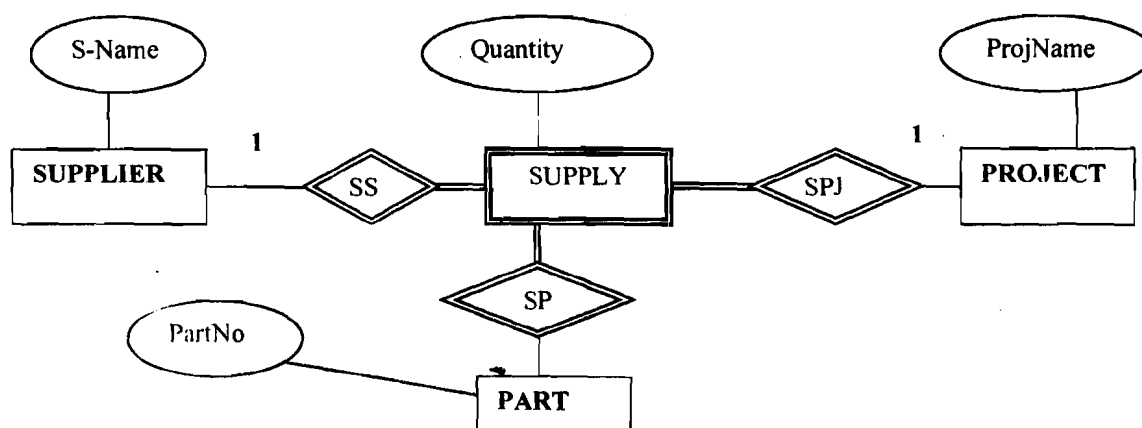


Figure 21: The E-R Diagram for supplier-and-parts database

2.10 SUMMARY

In this unit we discussed that **entity-relationship (E-R) data model** is based on a perception of a real world that consists of a set of basic objects called entities, and of relationships among these objects. E-R modeling is a **semantic modeling technique**, which is used primarily for the database-design process. It was developed to facilitate database design by allowing the specification of an enterprise schema. Such a schema represents the overall logical structure of the database. This overall structure can be expressed graphically by an E-R diagram.

An **entity** is a data object, a thing, an occurrence or event, a role, an organisational unit, a place or a structure in the real world that exists and is distinguishable from other objects. We accomplished the distinction by associating with each entity a set of attributes that describes the object. The **attributes** could be of various type— composite or simple, single or multivalued, derived or stored. A **null** value is used in the place when an entity does not have any value for an attribute.

A relationship is an association among several entities. The collection of all entities of the same type is an entity set, and the collection of all relationships of the same type is a relationship set. There are two kind of structural constraints that can be placed on relationship types. The cardinality ratio or mapping cardinalities express the number of entities to which another entity can be associated via a relationship set. They could be one-to-one, one-to-many, many-to-one and many-to-many.

Participation constraint on a relationship set is either total or partial depending on the fact whether all entities in the entity set participates in at least one relationship in the relationship set or only few entities in the entity set participates in the relationship in relationship set. Another form of constraint is existence dependency, which specifies that the existence of entity x depends on the existence of entity y.

An important task in database modeling is to specify how entities and relationships are distinguished. Conceptually, individual entities and relationships are distinct; from a database perspective, however, their difference must be expressed in terms of their attributes. To make such distinctions, we assign a primary key to each entity set. The primary key is a set of one or more attributes that, taken collectively, allows us to identify uniquely an entity in the entity set and a relationship in a relationship set. An entity set that does not have sufficient attributes to form a primary key is termed a weak entity set. An entity set that has a primary key is termed a strong entity set.

A database that conforms to an E-R diagram can be represented by a collection of tables. For each entity set and for each relationship set in the database, there is a unique table that is assigned the name of the corresponding entity set or relationship set. Each table has a number of columns, each of which has a unique name. Converting a database representation from an E-R diagram to a table format is the basis for driving a relational-database design from an E-R diagram.