Chapter 3 – Data Modeling Using the Entity-Relationship Model

Outline of Database Design

The main phases of database design are depicted in Figure 3.1

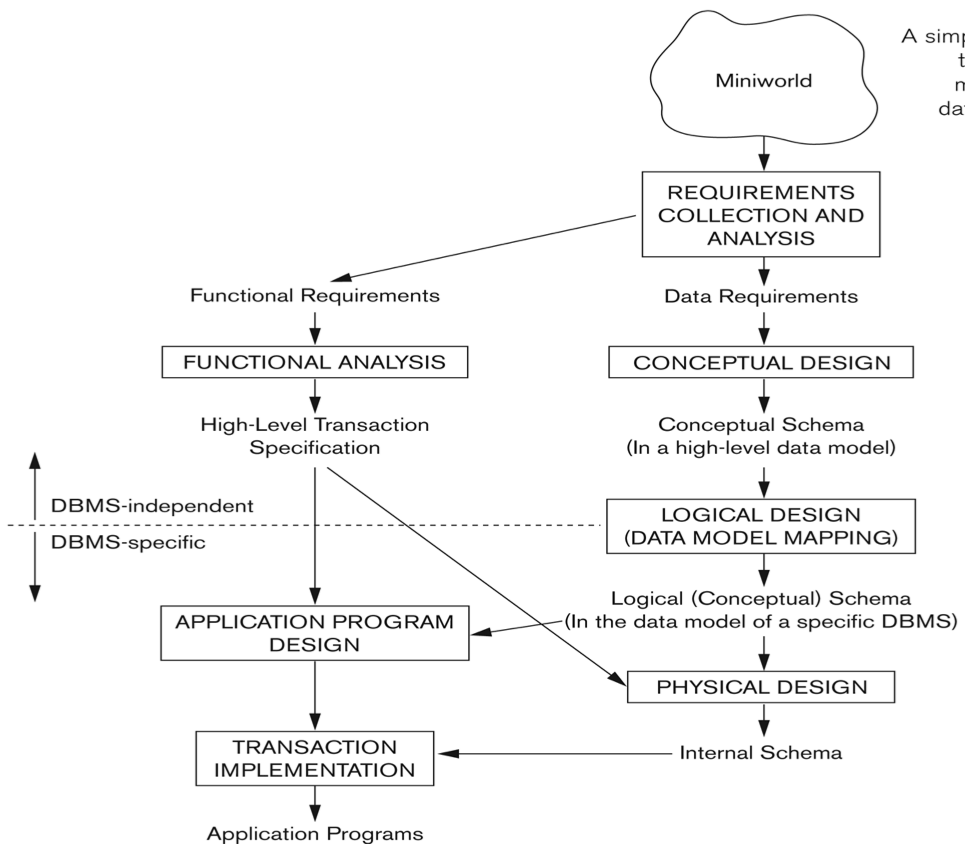

Miniworld
REQUIREMENTS COLLECTION AND ANALYSIS
Functional Requirements
Data Requirements
FUNCTIONAL ANALYSIS
CONCEPTUAL DESIGN
High-Level Transaction Specification
Conceptual Schema (In a high-level data model)
DBMS-independent
DBMS-specific
LOGICAL DESIGN (DATA MODEL MAPPING)
Logical (Conceptual) Schema (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.

- **Requirements Collection and Analysis**: purpose is to produce a description of the users' requirements.
- **Conceptual Design**: purpose is to produce a *conceptual schema* for the database, including detailed descriptions of *entity types*, *relationship types*, and *constraints*. All these are expressed in terms provided by the data model being used.
- **Implementation**: purpose is to transform the conceptual schema (which is at a high/abstract level) into a (lower-level) *representational/implementational* model supported by whatever DBMS is to be used.
- **Physical Design**: purpose is to decide upon the internal storage structures, access paths (indexes), etc., that will be used in realizing the representational model produced in previous phase.

Entity-Relationship (ER) Model

The **Entity-Relationship (ER) Model** is a popular high-level conceptual data model. In the ER model, the main concepts are **entity**, **attribute**, and **relationship.**

Entities and Attributes

Entity: An entity represents some "thing" (in the miniworld) that is of interest to us, i.e., about which we want to maintain some data. An entity could represent a physical object (e.g., house, person, automobile, widget) or a less tangible concept (e.g., company, job, academic course).

Attribute: An entity is described by its attributes, which are properties characterizing it. Each attribute has a **value** drawn from some **domain** (set of meaningful values).

Example: A *PERSON* entity might be described by *Name*, *BirthDate*, *Sex*, etc., attributes, each having a particular value.

We can classify attributes along these dimensions:

- simple/atomic vs. composite
- single-valued vs. multi-valued (or set-valued)
- stored vs. derived (*Note from instructor:* this seems like an implementational detail that ought not be considered at this (high) level of abstraction.)

A **composite attribute** is one that is *composed* of smaller parts. An **atomic** attribute is indivisible or indecomposable.

- **Example 1**: A *BirthDate* attribute can be viewed as being composed of attributes for month, day, and year.
- **Example 2**: An *Address* attribute can be viewed as being composed of attributes for street address, city, state, and zip code. A street address can itself be viewed as being composed of a number, street name, and apartment number. As this suggests, composition can extend to a depth of two (as here) or more.

Single- vs. multi-valued attribute: Consider a *PERSON* entity. The person it represents has (one) *SSN*, (one) *date of birth*, (one, although composite) *name*, etc. But that person may have zero or more academic degrees or dependents. How can we model this via attributes *AcademicDegrees* and *Dependents*? One way is to allow such attributes to be *multi-valued* (perhaps *set-valued* is a better term), which is to say that we assign to them a (possibly empty) *set* of values rather than a single value.

We refer to an attribute that involves some combination of multi-valued ness *and* compositeness as a **complex attribute**.

A more complicated example of a complex attribute is *AddressPhone*. This attribute is for recording data regarding addresses and phone numbers of a business. The structure of this attribute allows for the business to have several offices, each described by an address and a set of phone numbers that ring into that office. Its structure is given by

{AddressPhone ({ Phone(AreaCode, Number) }, Address(StrAddr(StrNum, StrName, AptNum), City, State, Zip)) }

Stored vs. derived attribute: A *derived* attribute is one whose value can be calculated from the values of other attributes, and hence need not be stored. Example: *Age* can be calculated from *BirthDate.*

The Null value: In some cases a particular entity might not have an applicable value for a particular attribute. Or that value may be unknown. *For Example*: The attribute *DateOfDeath* is not applicable to a living person and its correct value may be unknown for some persons who have died. In such cases, we use a special attribute value (non-value?), called **null**.

Entity Types, Entity Sets, Keys, and Value Sets

- An entity type defines a collection (or set) of entities that have the same attributes. It serves as a template for a collection of **entity instances**, all of which are described by the same collection of attributes. That is, an entity type is analogous to a **class** in object-oriented programming and an entity instance is analogous to a particular object (i.e., instance of a class.
- An **entity set** is the collection of all entities of a particular type that exist, in a database, at some moment in time.
- A minimal collection of attributes (often only one) that distinguishes any two (simultaneously-existing) entities of that entity type is called a **Key Attribute**. An entity type could have more than one key.
- An attribute's **Value Set (Domain)** specifies its set of allowable values. The concept is similar to data type.

Example Database Application: COMPANY

Suppose that Requirements Collection and Analysis results in the following (informal) description of the COMPANY miniworld:

The company is organized as a collection of **departments**.

- Each department
    - has a unique name
    - has a unique number
    - is associated with a set of locations
    - has a particular employee who acts as its manager (and who assumed that position on some date)

  - has a set of employees assigned to it
  - controls a set of projects
- Each project
  - has a unique name
  - has a unique number
  - has a single location
  - has a set of employees who work on it
  - is controlled by a single department
- Each employee
  - has a name
  - has a SSN that uniquely identifies her/him
  - has an address
  - has a salary
  - has a sex
  - has a birthdate
  - has a direct supervisor
  - has a set of dependents
  - is assigned to one department
  - works some number of hours per week on each of a set of projects (which need not all be controlled by the same department)
- Each dependent
  - has first name
  - has a sex
  - has a birthdate
  - is related to a particular employee in a particular way (e.g., child, spouse, pet)
  - is uniquely identified by the combination of her/his first name and the employee of which (s)he is a dependent

Initial Conceptual Design of COMPANY database

Using the above structured description as a guide, we get the following preliminary design for entity types and their attributes in the COMPANY database:

- DEPARTMENT(Name, Number, { Locations }, Manager, ManagerStartDate, { Employees }, { Projects })
- PROJECT(Name, Number, Location, { Workers }, ControllingDept)
- EMPLOYEE(Name(FName, MInit, LName), SSN, Sex, Address, Salary, BirthDate, Dept, Supervisor, { Dependents }, { WorksOn(Project, Hours) })
- DEPENDENT(Employee, FirstName, Sex, BirthDate, Relationship)

Remarks: Note that the attribute *WorksOn* of EMPLOYEE (which records on which projects the employee works) is not only multi-valued (because there may be several such projects) but also composite, because we want to record, for each such project, the number of hours per week that the employee works on it. Also, each *candidate key* has been indicated by underlining.

For similar reasons, the attributes *Manager* and *ManagerStartDate* of DEPARTMENT really ought to be combined into a single composite attribute. Not doing so causes little or no harm, however, because these are single-valued attributes. Multi-valued attributes would pose some difficulties, on the other hand. Suppose, for example, that a department could have two or more managers, and that some department had managers Mary and Harry, whose start dates were 10-4-1999 and 1-13-2001, respectively. Then the values of the *Manager* and *ManagerStartDate* attributes should be *{ Mary, Harry }* and *{ 10-4-1999, 1-13-2001 }*. But from these two attribute values, there is no way to determine which manager started on which date. On the other hand, by recording this data as a set of ordered pairs, in which each pair identifies a manager and her/his starting date, this deficiency is eliminated. *End of Remarks*

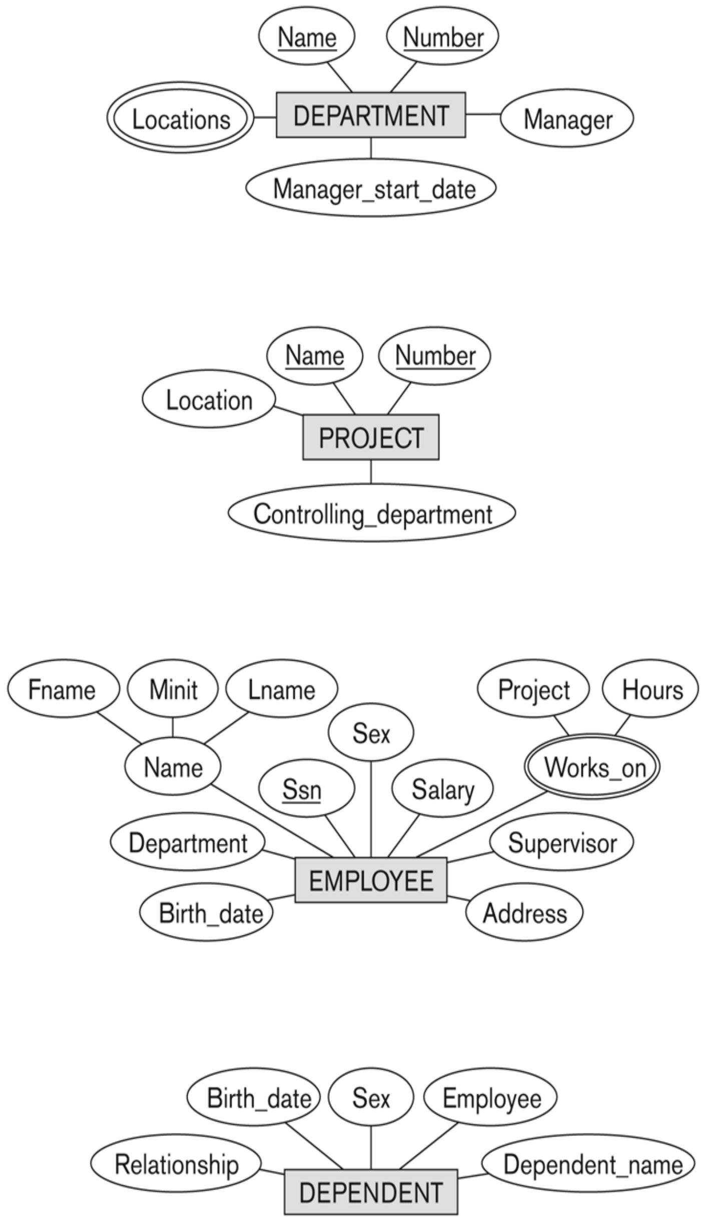


Figure 3.8
Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

Relationship Types, Sets, Roles, and Structural Constraints

Relationship: This is an association between two entities. As an example, one can imagine a STUDENT entity being associated to an ACADEMIC_COURSE entity via, say, an ENROLLED_IN relationship.

Whenever an attribute of one entity type refers to an entity (of the same or different entity type), we say that a relationship exists between the two entity types.

From our preliminary COMPANY schema, we identify the following **relationship types** (using descriptive names and ordering the participating entity types so that the resulting phrase will be in active voice rather than passive):

- EMPLOYEE MANAGES DEPARTMENT (arising from *Manager* attribute in DEPARTMENT)
- DEPARTMENT CONTROLS PROJECT (arising from *ControllingDept* attribute in PROJECT and the *Projects* attribute in DEPARTMENT)
- EMPLOYEE WORKS_FOR DEPARTMENT (arising from *Dept* attribute in EMPLOYEE and the *Employees* attribute in DEPARTMENT)
- EMPLOYEE SUPERVISES EMPLOYEE (arising from *Supervisor* attribute in EMPLOYEE)
- EMPLOYEE WORKS_ON PROJECT (arising from *WorksOn* attribute in EMPLOYEE and the *Workers* attribute in PROJECT)
- DEPENDENT DEPENDS_ON EMPLOYEE (arising from *Employee* attribute in DEPENDENT and the *Dependents* attribute in EMPLOYEE)

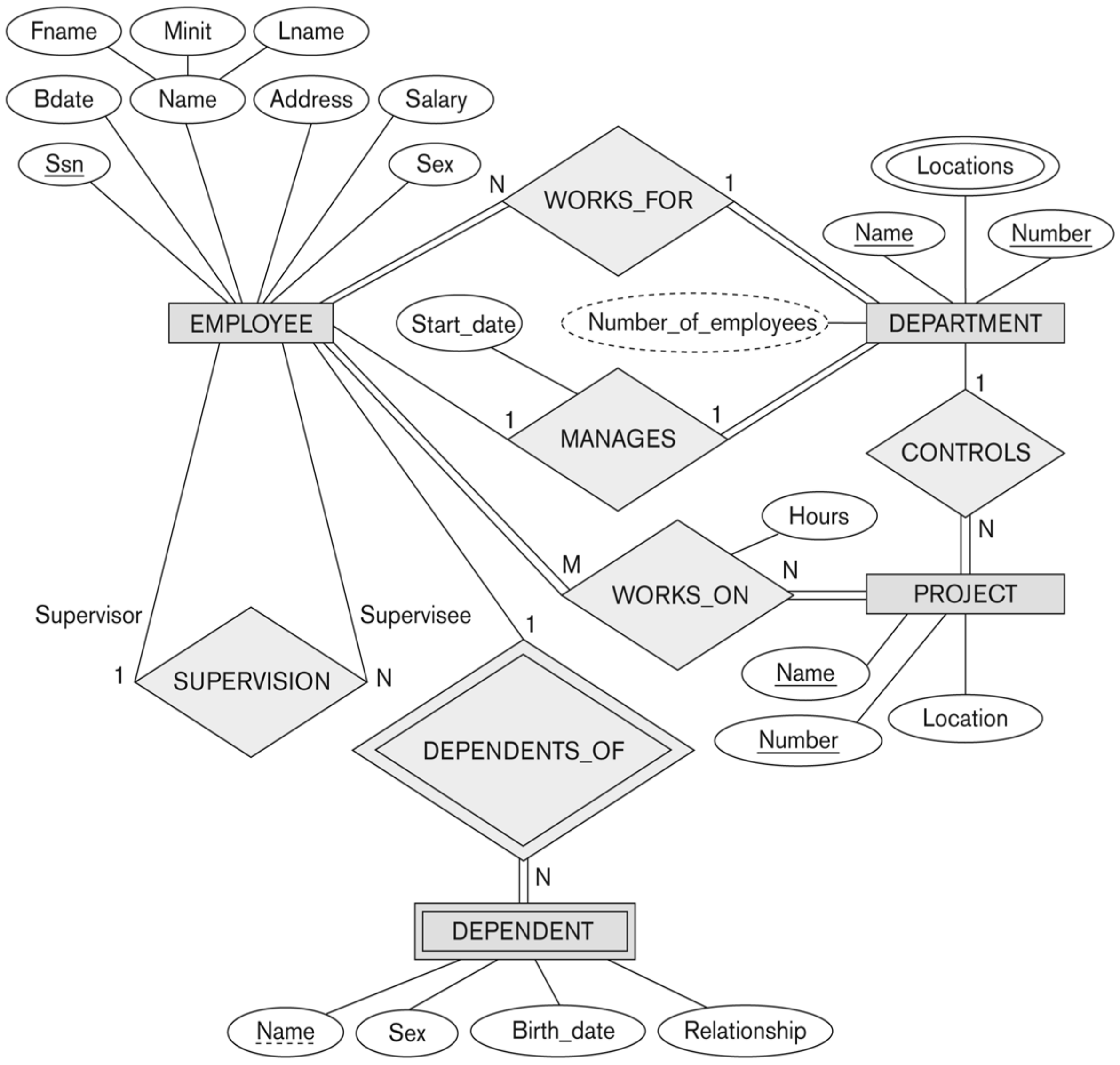


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

A **relationship set** is a set of instances of a relationship type. If, say, R is a relationship type that relates entity types A and B, then, at any moment in time, the relationship set of R will be a set of ordered pairs *(x,y)*, where *x* is an instance of A and *y* is an instance of B. What this means is that, for example, if our COMPANY miniworld is, at some moment, such that employees e_1, e_3, and e_6 work for department d_1, employees e_2 and e_4 work for department d_2, and employees e_5 and e_7 work for department d_3, then the WORKS_FOR **relationship set** will include as **instances** the ordered pairs (e_1, d_1), (e_2, d_2), (e_3, d_1), (e_4, d_2), (e_5, d_3), (e_6, d_1), and (e_7, d_3).

Degree of a relationship type is the number of participating entity types.

Roles in relationships: Each entity that participates in a relationship plays a particular *role* in that relationship, and it is often convenient to refer to that role using an appropriate name. For example, in each instance of a WORKS_FOR relationship set, the employee entity plays the role of *worker* or employee and each department plays the role of *employer* or *department*. Whenever the same entity type participates more than once in a relationship type in different roles, then the role name becomes essential for distinguishing the meaning of each participation. Such relationship types are called **Recursive Relationships**. For example, in each instance of a SUPERVISES relationship set, one employee plays the role of *supervisor* and the other plays the role of *supervisee*.

Constraints on Relationship Types:

There are two main kinds of relationship constraints (on binary relationships).

- **Cardinality ratio/Mapping Cardinality:** Specifies the maximum number of relationship instances that an entity can participate in.
  - **1:1 (one-to-one)**: Under this constraint, no instance of *A* may participate in more than one instance of *R*; similarly for instances of *B*. **Example**: Our informal description of COMPANY says that every department has one employee who manages it. If we also stipulate that an employee may not (simultaneously) play the role of manager for more than one department, it follows that MANAGES is 1:1.
  - **1:N (one-to-many)**: Under this constraint, no instance of *B* may participate in more than one instance of *R*, but instances of *A* are under no such restriction. In other words, if (a_1, b_1) and (a_2, b_2) are (distinct) instances of *R*, then it cannot be the case that $b_1 = b_2$. **Example**: CONTROLS is 1:N because no project may be controlled by more than one department. On the other hand, a department may control any number of projects, so there is no restriction on the number of relationship instances in which a particular department instance may participate. For similar reasons, SUPERVISES is also 1:N.
  - **N:1 (many-to-one)**: This is just the same as 1:N but with roles of the two entity types reversed. **Example**: WORKS_FOR and DEPENDS_ON are N: 1.
  - **M:N (many-to-many)**: Under this constraint, there are no restrictions. (Hence, the term applies to the absence of a constraint!) **Example**: WORKS_ON is M:N, because an employee may work on any number of projects and a project may have any number of employees who work on it.
- **Participation:** specifies whether or not the existence of an entity depends upon its being related to another entity via the relationship.
  - **Total participation (or existence dependency)**: To say that entity type *A* is constrained to **participate totally** in relationship *R* is to say that there can be no member of *A*'s instance set that does not participate in at least one instance of *R*. According to our informal description of COMPANY, every employee must be assigned to some department. That is, every

employee instance must participate in at least one instance of WORKS_FOR, which is to say that *EMPLOYEE* satisfies the total participation constraint with respect to the WORKS_FOR relationship

- **Partial participation**: the absence of the total participation constraint! (E.g., not every employee has to participate in MANAGES; hence we say that, with respect to MANAGES, *EMPLOYEE* participates partially. This is not to say that for all employees to be managers is not allowed; it only says that it need not be the case that all employees are managers.

Attributes of Relationship Types: Relationship types, like entity types, can have attributes. A good example is WORKS_ON, each instance of which identifies an employee and a project on which (s)he works. In order to record (as the specifications indicate) how many hours are worked by each employee on each project, we include *Hours* as an attribute of WORKS_ON.

Weak Entity Types: An entity type that has no key attribute of its own is called a **weak entity type**. (Ones that do are **strong**.)

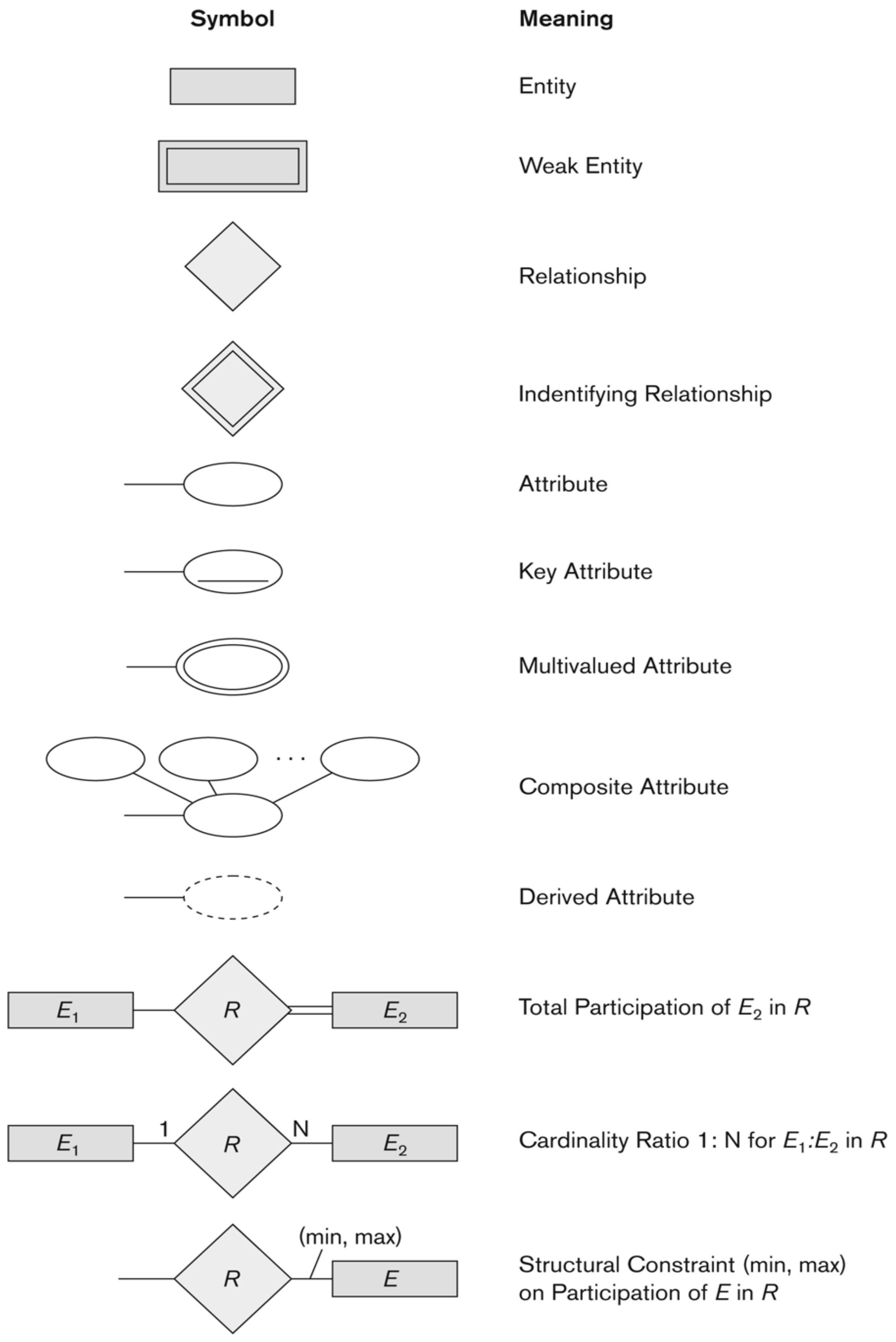
An entity of a weak identity type is uniquely identified by the specific entity to which it is related (by a so-called **identifying relationship** that relates the weak entity type with its so-called **identifying** or **owner entity type**) in combination with some set of its own attributes (called a ***partial key***).

Example: A *DEPENDENT* entity is identified by its first name (**partial key** of dependent) together with the *EMPLOYEE* entity to which it is related via DEPENDS_ON.

Because an entity of a weak entity type cannot be identified otherwise, that entity type has a **total participation constraint** (i.e., **existence dependency**) with respect to the identifying relationship.

Summary of notation for ER diagrams

Figure 3.14
Summary of the notation for ER diagrams.


Symbol
Meaning
Entity
Weak Entity
Relationship
Indentifying Relationship
Attribute
Key Attribute
Multivalued Attribute
. . .
Composite Attribute
Derived Attribute
E_1
R
E_2
Total Participation of E_2 in R
E_1
1
R
N
E_2
Cardinality Ratio 1: N for E_1:E_2 in R
R
(min, max)
E
Structural Constraint (min, max) on Participation of E in R