History of Database Systems

- Techniques for data storage and processing have evolved over the years:

1950s and early 1960s:

- Magnetic tapes were developed for data storage
- Data processing tasks such as payroll were automated, with data stored on tapes (Processing of data consisted of reading data from one or more tapes and writing data to a new tape)
- Data could also be input from punched card decks, and output to printers
- For example, salary raises were processed by entering the raises on punched cards and reading the punched card deck in synchronization with a tape containing the master salary details; The records had to be in the same sorted order
- The salary raises would be added to the salary read from

the master tape, and written to a new tape; the new tape would become the new master tape

Late 1960s and 1970s:

- Widespread use of hard disks in the late 1960s changed the scenario for data processing greatly, since hard disks allowed direct access to data
- The position of data on disk was immaterial, since any location on disk could be accessed in just tens of milliseconds
- Data were thus freed from the tyranny of sequentiality; with disks, network and hierarchical databases could be created that allowed data structures such as lists and trees to be stored on disk
- Programmers could construct and manipulate these data structures

- In 1970s relational databases were born

1980s:

- Although academically interesting, the relational model was not used in practice initially, because of its perceived performance disadvantages
- Relational databases could not match the performance of existing network and hierarchical databases
- IBM Company developed techniques for the construction of an efficient relational database system
- By the early 1980s, relational databases had become competitive with network and hierarchical database systems even in the area of performance
- Relational databases were so easy to use that they eventually replaced network/hierarchical databases

- In the 1980s, the relational model has reigned supreme among data models
- The 1980s also saw much research on parallel and distributed databases, as well as initial work on object-oriented databases

Early 1990s:

- The SQL language was designed for transaction processing applications of databases in the 1980s
- Querying emerged as a major application area for databases
- Tools for analyzing large amounts of data saw large growths in usage
- Many database vendors introduced parallel database products in this period

- Database vendors also began to add object-relational support to their databases

Late 1990s:

- The major event was the explosive growth of the WorldWideWeb
- Database systems also had to support Web interfaces to data
- Databases were deployed much more extensively than ever before
- Database systems now had to support very high transaction processing rates, as well as very high reliability and 24×7 availability (availability 24 hours a day, 7 days a week, meaning no downtime for scheduled maintenance activities)

Database Design and ER Diagrams

- The database design process can be divided into six steps. The ER model is most relevant to the first three steps:

(1) Requirements Analysis:

- The very first step in designing a database application is to understand what data is to be stored in the database

- We must find out what the users want from the database

- This is usually an informal process that involves discussions with user groups, a study of the current operating environment and how it is expected to change,

analysis of any available documentation on existing applications that are expected to be replaced or complemented by the database, and so on

- Several methodologies have been proposed for organizing and presenting the information gathered in this step, and some automated tools have been developed to support this process

(2) Conceptual Database Design:

- The information gathered in the requirements analysis step is used to develop a high-level description of the data to be stored in the database, along with the constraints that are known to hold over this data
- This step is often carried out using the ER model, or a similar high-level data model

(3) Logical Database Design:

- We must choose a DBMS to implement our database design, and convert the conceptual database design into a database schema in the data model of the chosen DBMS
- The task in the logical design step is to convert an ER schema into a relational database schema
- The result is a conceptual schema, sometimes called the **logical schema**, in the relational data model

Beyond ER Design

- The ER diagram is description of the data, constructed through the information collected during requirements analysis
- A more careful analysis can often refine the logical schema obtained at the end of Step 3

- Once we have a good logical schema, we must consider performance criteria and design the physical schema
- Finally, we must address security issues and ensure that users are able to access the data they need, but not data that we wish to hide from them

The remaining three steps of database design are briefly described below

(4) Schema Refinement:

- The fourth step in database design is to analyze the collection of relations in our relational database schema to identify potential problems, and to refine it
- In contrast to the requirements analysis and conceptual design steps, which are essentially subjective, schema refinement can be guided by some elegant and powerful theory (*normalizing*(restructuring) relations)

(5) Physical Database Design:

- In this step we must consider typical expected workloads that our database must support and further refine the database design to ensure that it meets desired performance criteria
- This step may simply involve building indexes on some tables and clustering some tables, or it may involve a substantial redesign of parts of the database schema obtained from the earlier design steps

(6) Security Design:

- In this step, we identify different user groups and different roles played by various users (e.g., the development team for a product, the customer support representatives, the product manager)

- For each role and user group, we must identify the parts of the database that they must be able to access and the parts of the database that they should *not* be allowed to access, and take steps to ensure that they can access only the necessary parts

ENTITIES, ATTRIBUTES, AND ENTITY SETS

- An **entity** is an object in the real world that is distinguishable from other objects

- Examples: toy, the toy department, the manager of the toy department, the home address of the manager of the toy department

- An **entity set** is a collection of similar entities

- Note that entity sets need not be disjoint; the collection of toy department employees and the collection of appliance department employees may both contain employee John (who happens to work in both departments)

- We could also define an entity set called Employees that contains both the toy and appliance department employee sets

- An entity is described using a set of **attributes**

- All entities in a given entity set have the same attributes

- For example, the Employees entity set could use name, social security number (ssn), and parking lot (lot) as attributes

- For each attribute associated with an entity set, we must identify a **domain** of possible values

- For example, the domain associated with the attribute *name* of Employees might be the set of 20-character strings

- As another example, if the company rates employees on a scale of 1 to 10 and stores ratings in a field called *rating*, the associated domain consists of integers 1 through 10

- Further, for each entity set, we choose a *key*

- A **key** is a minimal set of attributes whose values uniquely identify an entity in the set

- There could be more than one **candidate** key; if so, we designate one of them as the **primary key**

- The Employees entity set with attributes *ssn, name,* and *lot* is shown in Figure 2.1

- An entity set is represented by a rectangle, and an attribute is represented by an ellipse

- Each attribute in the primary key is underlined

- The domain information could be listed along with the attribute name, but we omit this to keep the figures compact

- The key is *ssn*

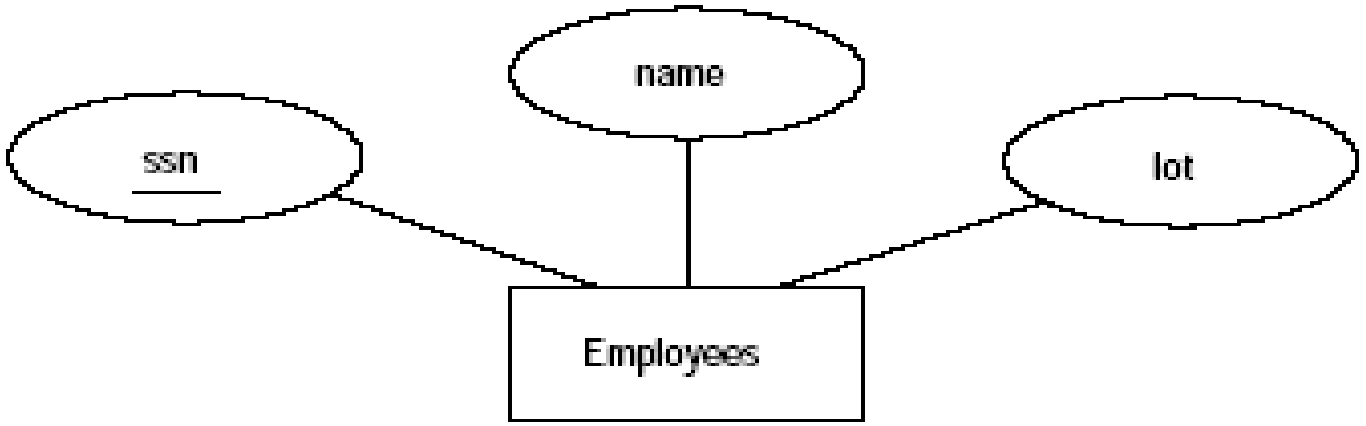

Figure 2.1 The Employees Entity Set

RELATIONSHIPS AND RELATIONSHIP SETS

- A **relationship** is an association among two or more entities

- For example, we may have the relationship that John works in the pharmacy department

- As with entities, we may wish to collect a set of similar relationships into a **relationship set**

- A relationship set can be thought of as a set of *n*-tuples:

$$\{(e_1, \ldots, e_n) \mid e_1 \in E_1, \ldots, e_n \in E_n\}$$

- Each n-tuple denotes a relationship involving n entities e_1 through e_n, where entity e_1 is in entity set E_1

- Figure 2.2 shows the relationship set Works_In, in which each relationship indicates a department in which an employee works

- Note that several relationship sets might involve the same entity sets

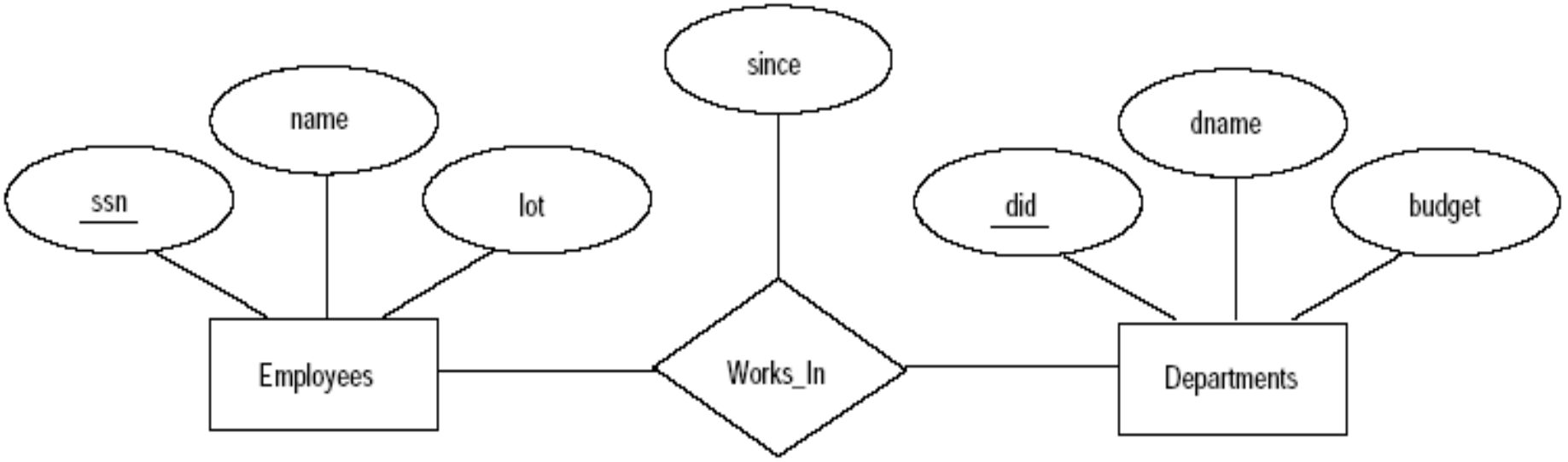

Figure 2.2 The Works_In Relationship Set

- For example, we could also have a Manages relationship set involving Employees and Departments

- A relationship can also have **descriptive attributes**

- Descriptive attributes are used to record information about the relationship, rather than about any one of the participating entities

- For example, we may wish to record that John works in the pharmacy department as of January 1991

- This information is captured in Figure 2.2 By adding an attribute, *since*, to Works_In

- A relationship must be uniquely identified by the participating entities, without reference to the descriptive attributes

- In the Works_In relationship set, for example, each Works_In relationship must be uniquely identified by the combination of employee *ssn* and department *did*

- An **instance** of a relationship set is a set of relationships

- An instance can be thought of as a 'snapshot' of the relationship set at some instant in time

- An instance of the Works_In relationship set is shown in Figure 2.3

- Each Employees entity is denoted by its *ssn*, and each Departments entity is denoted by its *did*, for simplicity

- The *since* value is shown beside each relationship

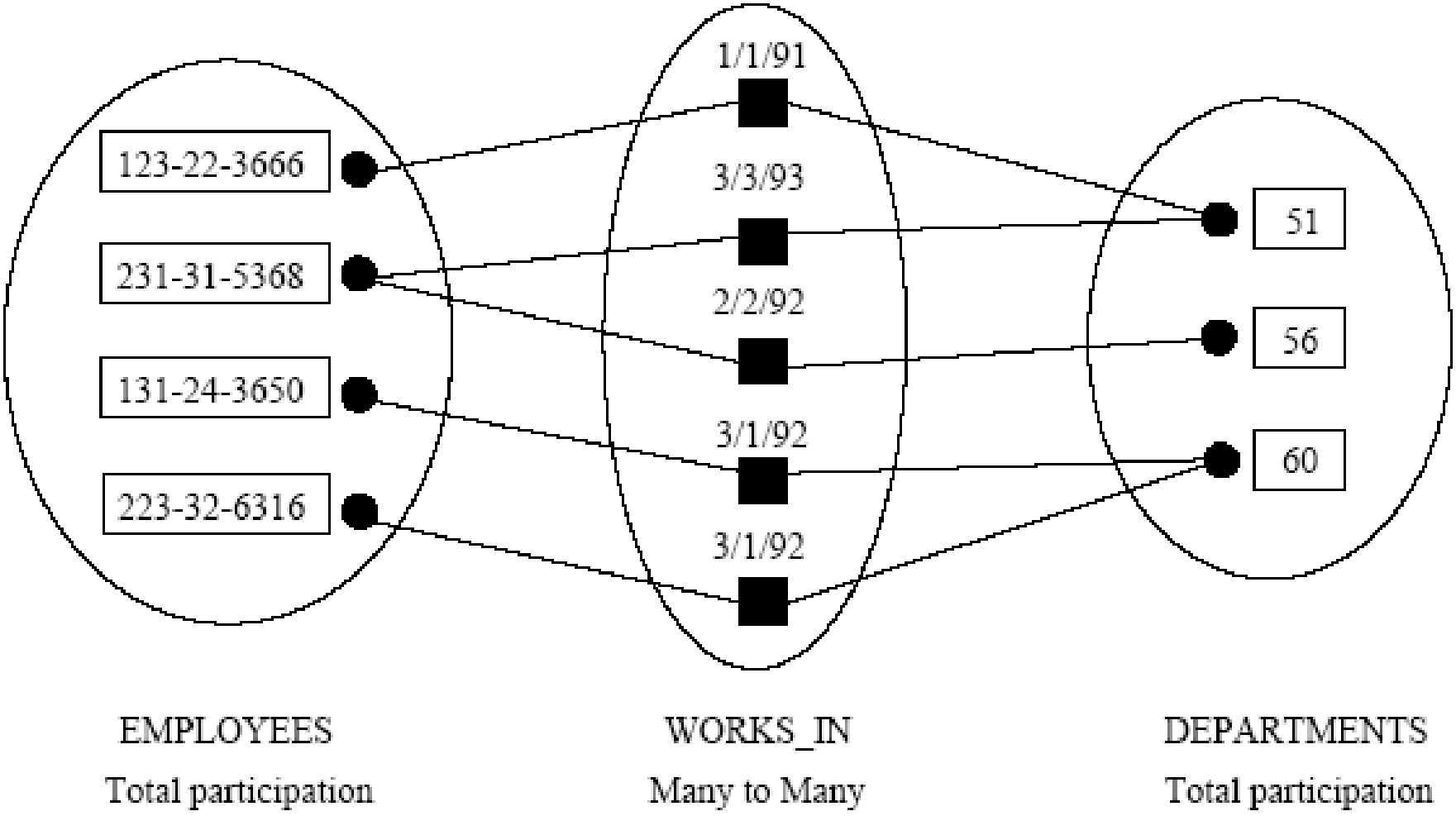

Figure 2.3 An Instance of the Works_In Relationship Set

- As another example of an ER diagram, suppose that each department has offices in several locations and we want to record the locations at which each employee works

- This relationship is **ternary** because we must record an association between an employee, a department, and a location

- The ER diagram for this variant of Works_In, which we call Works_In2, is shown in Figure 2.4

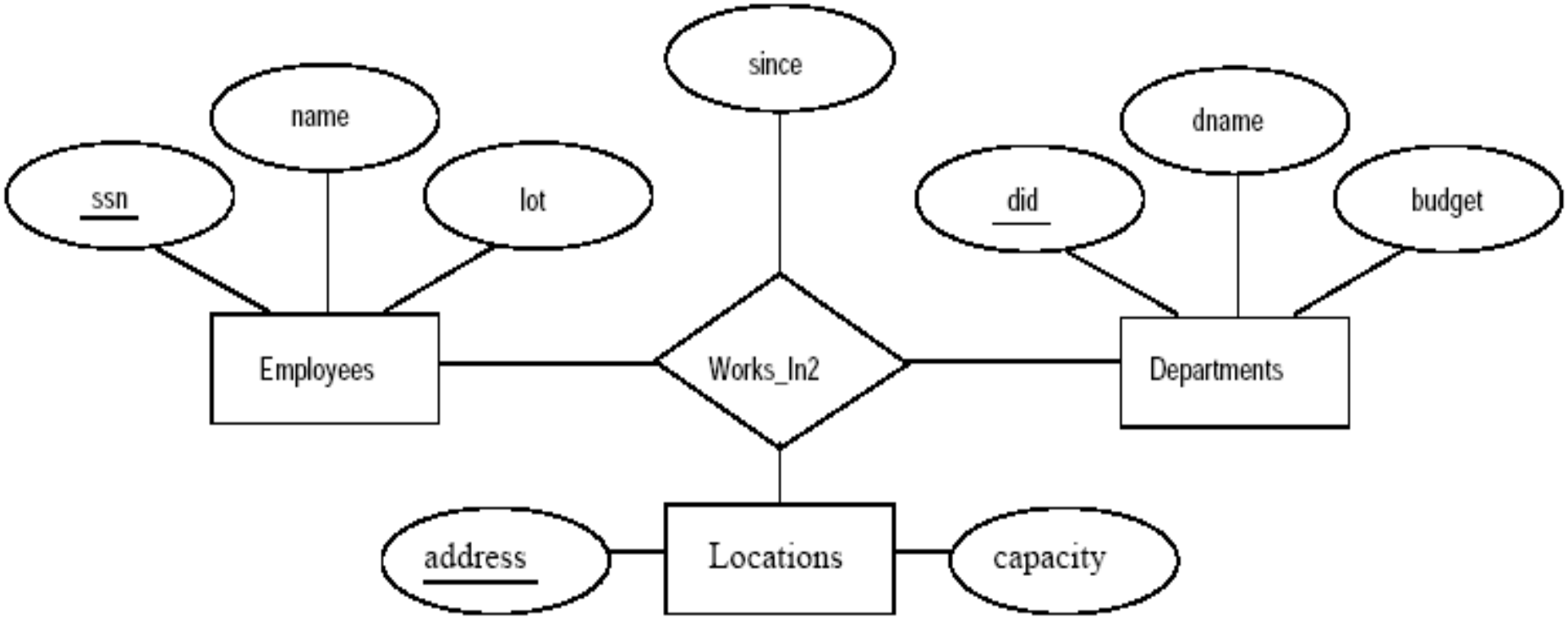

Figure 2.4 A Ternary Relationship Set

- The entity sets that participate in a relationship set need not be distinct

- Sometimes a relationship might involve two entities in the same entity set

- For example, consider the Reports_To relationship set that is shown in Figure 2.5

- Since employees report to other employees, every relationship in Reports_To is of the form (emp_1,emp_2), where both emp_1 and emp_2 are entities in Employees

- However, they play different **roles**: emp_1 reports to the managing employee emp_2, which is reflected in the **role indicators** supervisor and subordinate in Figure 2.5

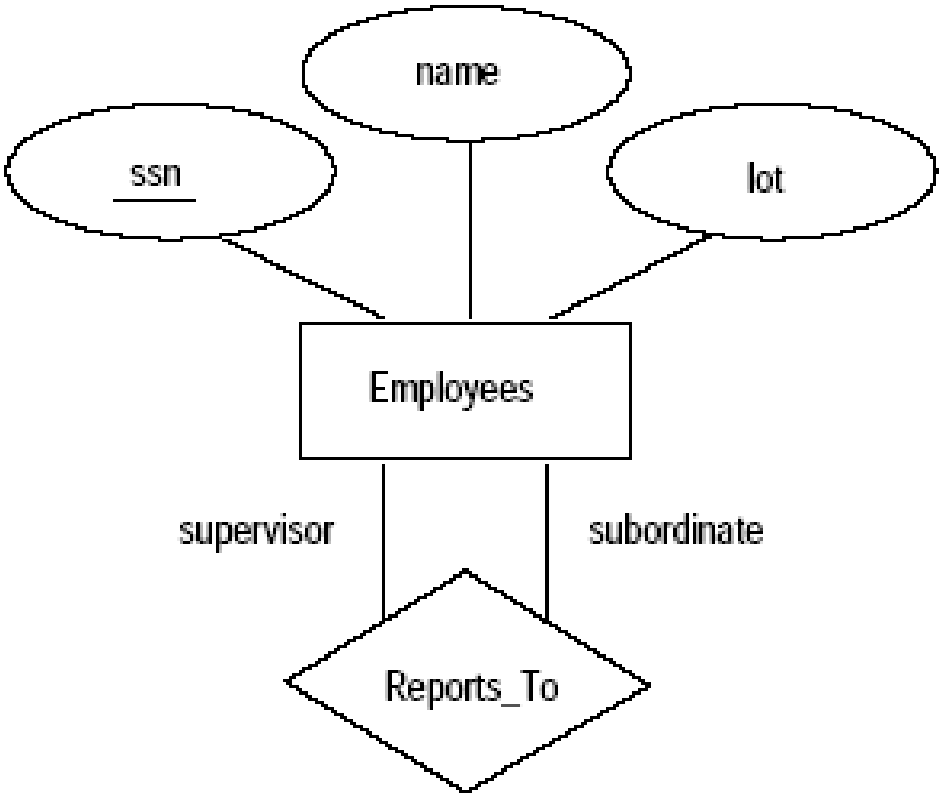

Figure 2.5 The Reports_To Relationship Set

- If an entity set plays more than one role, the role indicator concatenated with an attribute name from the entity set gives us a unique name for each attribute in the relationship set

- For example, the Reports_To relationship set has attributes corresponding to the *ssn* of the supervisor and the *ssn* of the subordinate, and the names of these attributes are *supervisor_ssn* and *subordinate_ssn*

ADDITIONAL FEATURES OF THE ER MODEL

Key Constraints:

- Consider the Works_In relationship shown in Figure 2.2

- An employee can work in several departments, and a department can have several employees

- As illustrated in the Works_In instance shown in Figure 2.3, Employee 231-31-5368 has worked in Department 51 since 3/3/93 and in Department 56 since 2/2/92.Department 51 has two employees

- Now consider another relationship set called Manages between the Employees and Departments entity sets such

that each department has at most one manager, although a single employee is allowed to manage more than one department

- The restriction that each department has at most one manager is an example of a **key constraint**

- This restriction is indicated in the ER diagram of Figure 2.6 by using an arrow from Departments to Manages

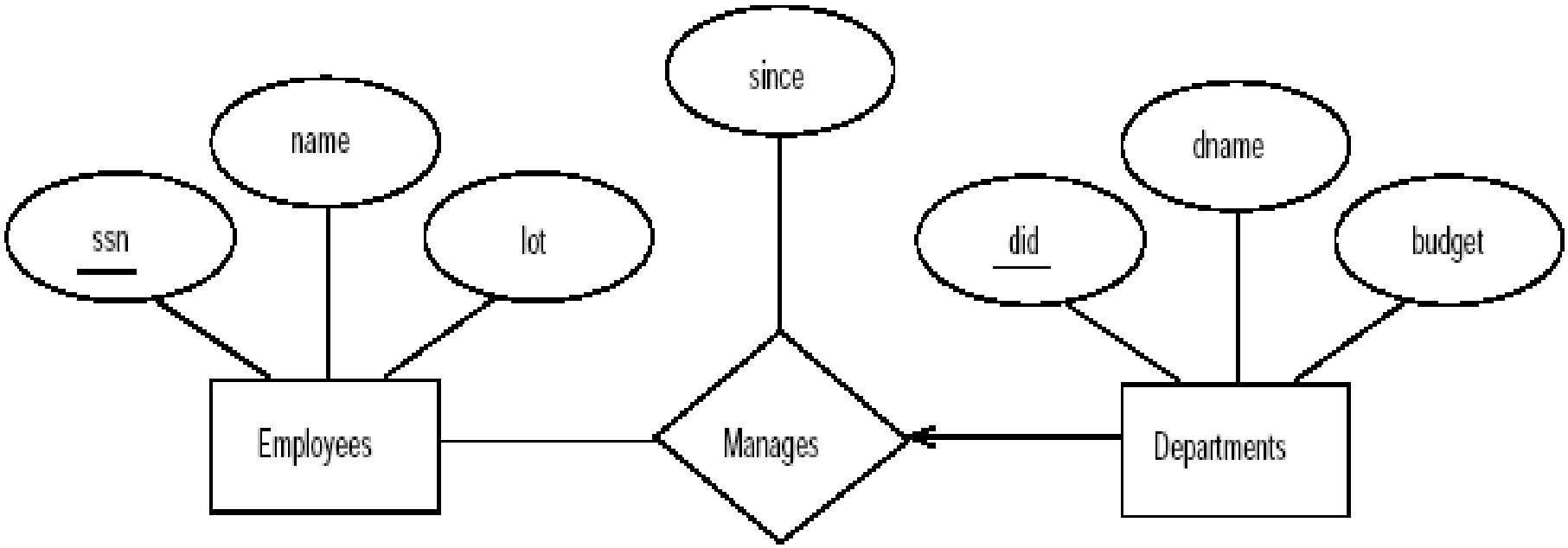

Figure 2.6 Key Constraint on Manages

- An instance of the Manages relationship set is shown in Figure 2.7

- The instance of Works_In shown in Figure 2.3 violates the key constraint on Manages

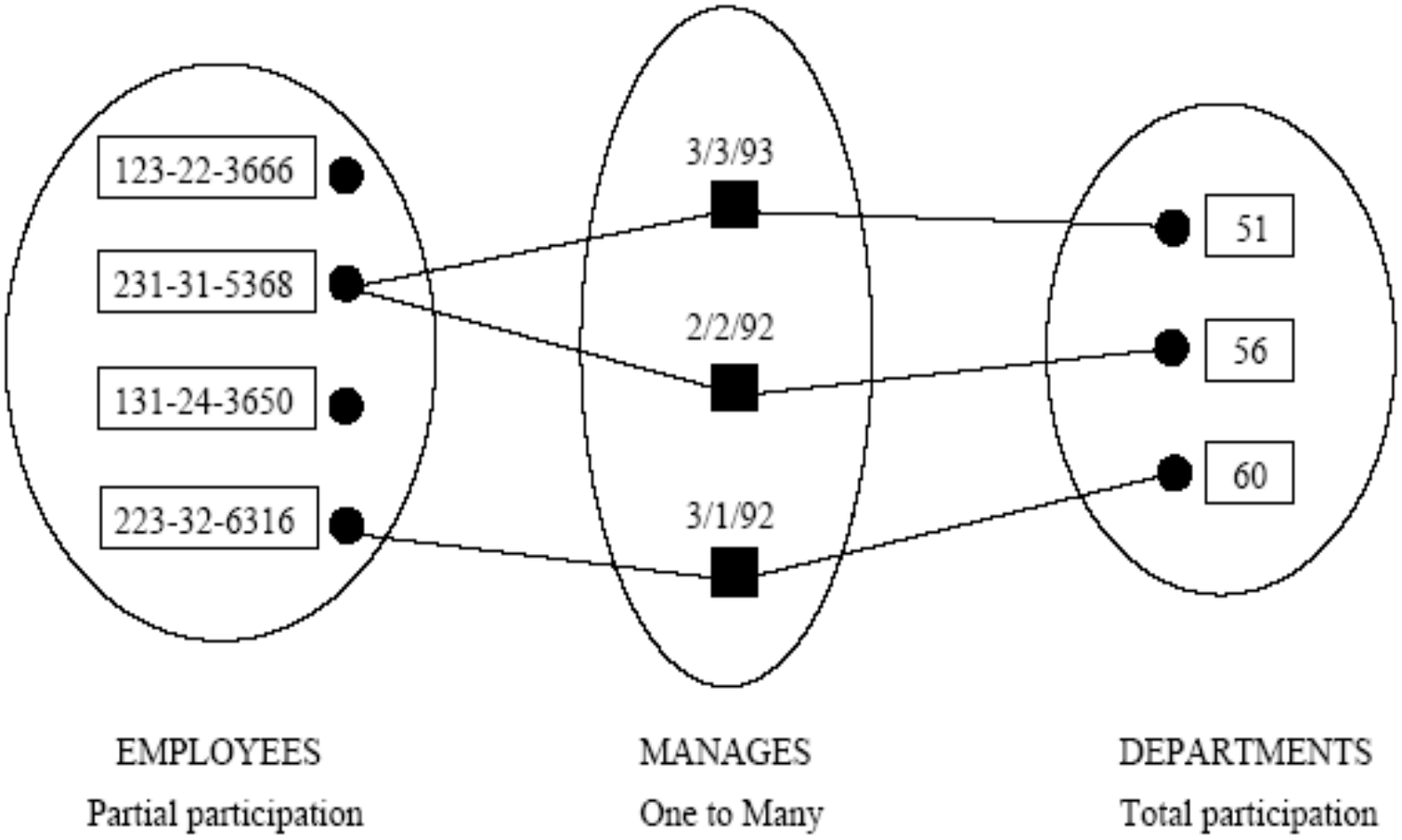

Figure 2.7 An Instance of the Manages Relationship Set

- A relationship set like Manages is sometimes said to be **one-to-many**

- In contrast, the Works_In relationship set is said to be **many-to-many**

- If we add the restriction that each employee can manage at most one department to the Manages relationship set, which would be indicated by adding an arrow from Employees to Manages in Figure 2.6, we have a **one-to-one** relationship set

Key Constraints for Ternary Relationships

- To indicate a key constraint on entity set E in relationship set R, we draw an arrow from E to R

- In Figure 2.8, we show a ternary relationship with key constraints

- Here key constraint specifies that, each employee works in at most one department, and at a single location

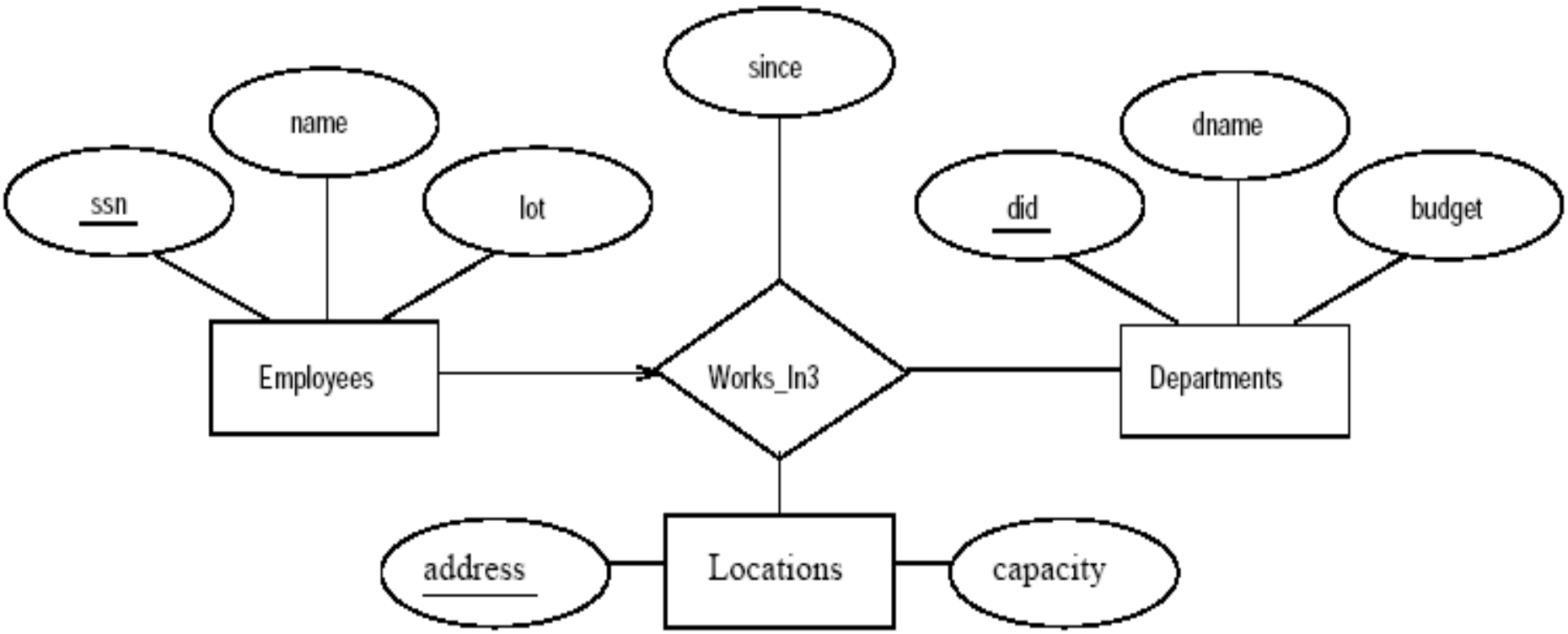

Figure 2.8 A Ternary Relationship Set with Key Constraints

- An instance of the Works_In3 relationship set is shown in Figure 2.9

- Notice that each department can be associated with several employees and locations, and each location can be associated with several departments and employees; however, each employee is associated with a single department and location

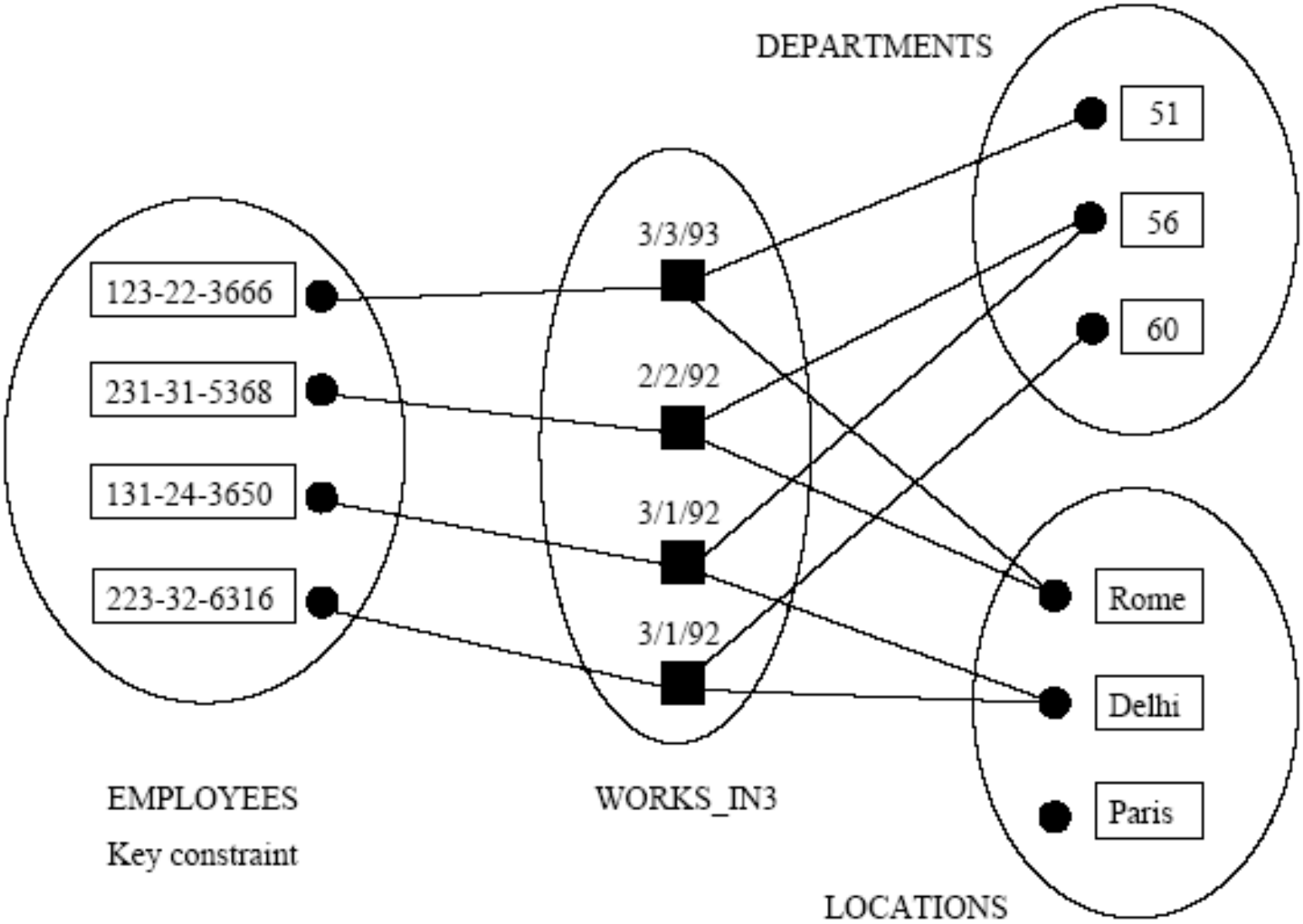

Figure 2.9 An Instance of Works_In3

Participation Constraints

- The key constraint on Manages tells us that a department has at most one manager

- A natural question to ask is whether every department has a manager

- Let us say that every department is required to have a manager. This requirement is an example of a **participation constraint**

- The participation of the entity set Departments in the relationship set Manages (Fig 2.7) is said to be **total**

- A participation that is not total is said to be **partial**

- As an example, the participation of the entity set Employees in Manages (Fig 2.7) is **partial**, since not every employee gets to manage a department

- In the Works_In relationship set (Fig 2.3), the participation of both Employees and Departments is total

- The ER diagram in Figure 2.10 shows both the Manages and Works_In relationship sets and all the given constraints

- If the participation of an entity set in a relationship set is total, the two are connected by a thick line; the presence of an arrow indicates a key constraint

- The instances of Works_In and Manages shown in Figures 2.3 and 2.7 satisfy all the constraints in Figure 2.10

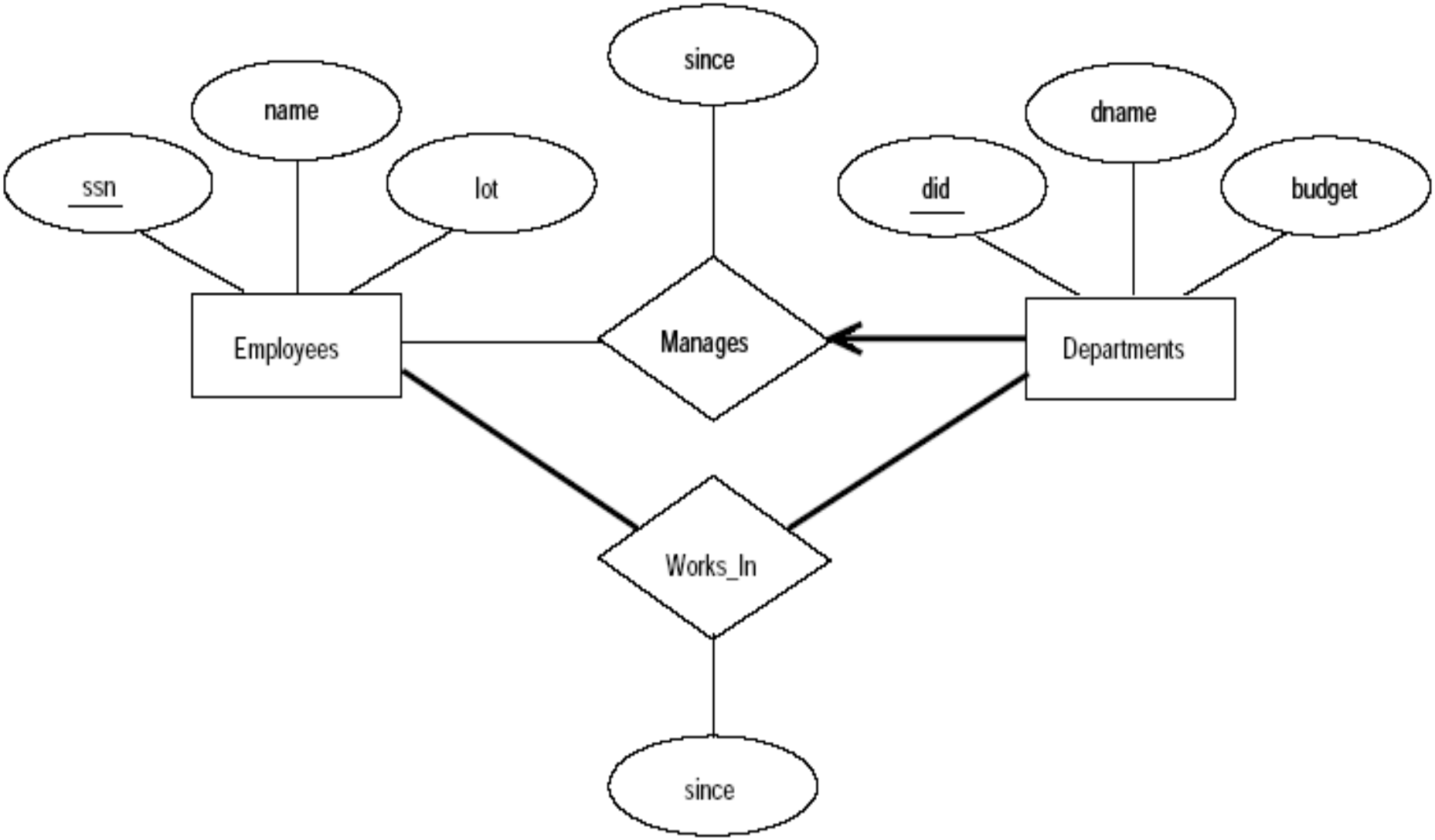

Figure 2.10 Manages and Works_In

Weak Entities

- An entity set include a key. This assumption does not always hold

- For example, suppose that employees can purchase insurance policies to cover their dependents

- We wish to record information about policies, including who is covered by each policy

- If an employee quits, any policy owned by the employee is terminated and we want to delete all the relevant policy and dependent information from the database

- When we want to identify a dependent by name alone in this situation, the dependents of a given employee have different names

- Thus the attributes of the Dependents entity set might be *pname* and *age*

- The attribute *pname* does *not* identify a dependent uniquely

- Recall that the key for Employees is ssn; thus we might have two employees called Smith, and each might have a son called Andy

- Dependents is an example of a **weak entity set**

- A weak entity can be identified uniquely only by considering some of its attributes in conjunction with the primary key of another entity, which is called the **identifying owner**

- The following restrictions must hold:

-The owner entity set and the weak entity set must participate in a one-to-many relationship set. This relationship set is called the **identifying relationship set** of the weak entity set

-The weak entity set must have total participation in the identifying relationship set

- For example, a Dependents entity can be identified uniquely by key *ssn* of Employees entity set and the attribute *pname* of the Dependents entity set

- The set of attributes of a weak entity set that uniquely identify a weak entity for a given owner entity is called a ***partial key*** of the weak entity set. In our example *pname* is a partial key for Dependents

- The Dependents weak entity set and its relationship to Employees is shown in Figure 2.11

- The total participation of Dependents in Policy is indicated by linking them with a dark line

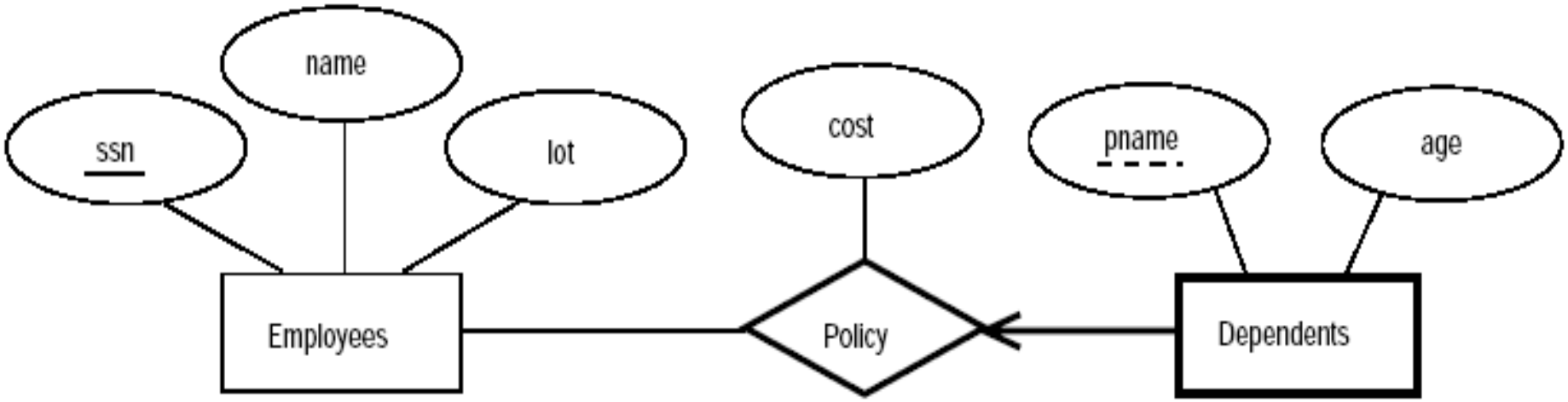

Figure 2.11 A Weak Entity Set

- The arrow from Dependents to Policy indicates that each Dependents entity appears in at most one Policy relationship

- To identify Dependents is a weak entity and Policy is its identifying relationship, we draw both with **dark lines**

- To indicate that *pname* is a partial key for Dependents, we underline it using a broken line

- This means that there may be two dependents with the same *pname* value