What is a DBMS?

- A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data

- The goal of a DBMS is to store and retrieve database information

Database System Applications

- Banking: For customer and accounts information

- Airlines: For reservations and schedule information

- Universities:For student information, course registrations
- Credit card transactions
- Sales: For customer, product, and purchase information
- Manufacturing: For production, inventory, orders
- Human resources: For employees information, salaries, tax deductions

Data base system vs File system

- Consider part of a savings-bank enterprise that keeps information about all customers and savings accounts

- One way to keep the information on a computer is to store it in files

- Users can manipulate the files using application programs, such as

  - *A program to debit or credit an account*
  - *A program to add a new account*
  - *A program to find the balance of an account*
  - *A program to generate monthly statements*

- New application programs are added to the system as the need arises

- The system stores permanent records in *various files*, and it needs different *application programs* to extract records from, and add records to, the appropriate files

- Before database management systems came along, usually information used to store in such systems

- file-processing system has a number of major disadvantages:

- **Data redundancy and inconsistency**
  - Multiple file formats, duplication of information in different files
  - Example: Savings account and Current account

- **Difficulty in accessing data**
  - Need to write a new program to carry out each new task
  - Example: when new query is asked

- **Data isolation**
  - Multiple files and formats

- **Integrity problems**
  - Values in the database must satisfy constraints
  - Hard to add new constraints or change existing ones

- **Atomicity of updates**
  - Failures may leave database in an inconsistent state with partial updates carried out
  - Example: Transfer of funds from one account to another should either complete or not happen at all (atomic)

- **Concurrent access by multiple users**
  - Concurrent access needed for performance
  - Uncontrolled concurrent accesses can lead to inconsistencies
    - Example: Two people reading a balance and updating account at the same time

- **Security problems**
  - Hard to provide user access to some, but not all, data

- Database systems offer solutions to all the above problems

View of Data

- A major purpose of a database system is to provide users with an abstract view of the data. That is the system hides certain details of how the data is stored and maintained

Data Abstraction

- For the system to be usable, it must retrieve data efficiently

- For efficiency designers use complex data structures to represent data in the database

- Developers hide the complexity from users through several levels of abstraction, to simplify users interactions with the system

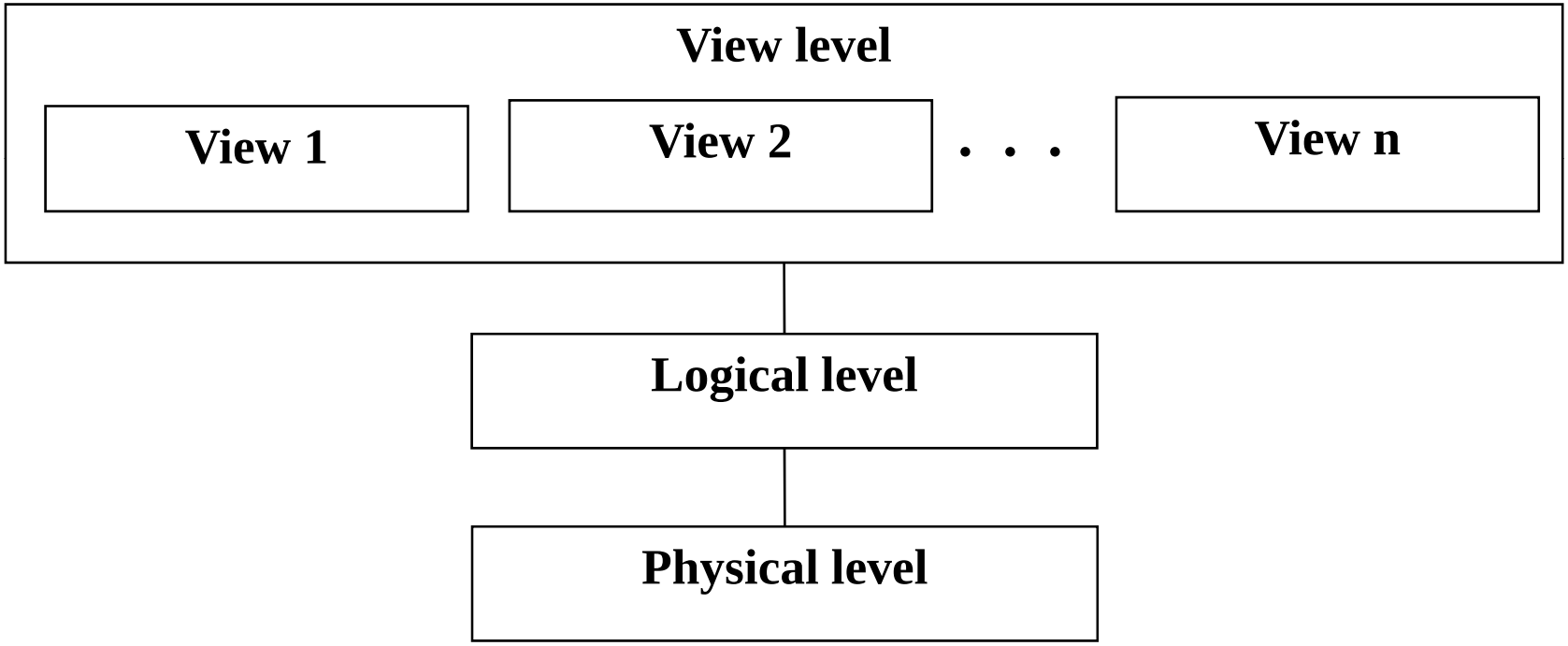


Figure 1.1 The three levels of data abstraction

Physical level

- The lowest level of abstraction describes **how** the data is actually stored

- The physical level describes complex low-level data structures in detail

Logical level

- The next-higher level of abstraction describes **what** data is stored in the database, and what relationships exist among those data

- The logical level describes the entire database in terms of a small number of relatively simple structures

- Implementation of the simple structures at the logical level may involve complex physical-level structures

- The user of the logical level does not need to be aware of this complexity (complex physical-level structures)

- Programmers and Administrators work at this level

View level

- The highest level of abstraction describes only **part of the entire database**

- The view level of abstraction exists to simplify users interaction with the system

Example:

```
struct customer
{
   customer_id: string;
   customer_name: string;
   customer_street: string;
   customer_city: string;
};
```

- A banking enterprise may have several such record types, including

-Account, with fields
account_number and account_balance

-Employee, with fields
employee_name and employee_salary

- At the physical level, a customer, account, or employee record can be described as a block of consecutive storage locations

- At the logical level, each such record is described by a structure definition, as in the previous code segment, and the interrelationship of these record types is defined as well

- At the view level, computer users see a set of application programs that hide details of the data types

- At the view level, several views of the database are defined, and database users see these views

- The views also provide a security mechanism to prevent users from accessing certain parts of the database

For example, students can access only their details, they cannot able to access other students details.

Instances and Schemas

- Databases change over time as information is inserted and deleted

- The collection of information stored in the database at a particular moment is called an instance of the database

- The overall design of the database is called the database schema

- The concept of database schemas and instances can be understood by analogy to a program written in a programming language

- A database schema corresponds to the variable declarations in a program

- The values of the variables in a program at a point in time correspond to an instance of a database schema

- Database systems have several schemas, partitioned according to the levels of abstraction

- The physical schema describes the database design at the physical level

- The logical schema (conceptual schema) describes the database design at the logical level

- A database may also have subschemas at the view level, that describe different views of the database

- Programmers develop applications by using the logical schema

- Physical schema is hidden beneath the logical schema, and can be changed easily without affecting application programs

- Application programs are said to exhibit ***physical data independence*** if they do not depend on the physical schema, and thus need not be rewritten if the physical schema changes

Data Models

- It describes the design of a database at the logical level

- Data model is a collection of tools for describing data, data relationships, data semantics, and consistency constraints

Types of Data Models

- Relational Model
- The Entity-Relationship Model
- Object-based data models (Object-oriented and Object-relational)
- Semistructured data model
- Network model
- Hierarchical model

Relational Model

- The relational model uses a collection of tables to represent both data and the relationships among those data

Example

customer-id	*customer-name*	*customer-street*	*customer-city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

Figure 1.2 Customer table

- The relational model is an example of a record-based model

- Record-based models are so named because the database is structured in fixed-format records of several types

- Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes

- The **columns** of the table correspond to the **attributes** of the record type

- The relational data model is the most widely used data model

Entity-Relationship Model

- The entity-relationship (E-R) data model is a collection of *entities* and *relationships* among these entities

- An *entity* is a "thing" or "object" in the real world that is distinguishable from other objects

- For example, each person is an entity, and bank accounts can be considered as entities

- Entities are **described** in a database by a **set of attributes**

- For example, the attributes ***account-number*** and ***balance*** may describe one particular account in a bank, and they form attributes of the ***account*** **entity set**

- Attributes ***customer-name***, ***customer-street*** and ***customer-city*** may describe a *customer* entity

- A *relationship* is an association among several entities

- For example, a ***depositor*** relationship associates a ***customer*** with each ***account*** that he has

- The set of all entities of the same type is called an ***entity set***

- The set of all relationships of the same type is called ***relationship set***

- The overall logical structure (schema) of a database can be expressed graphically by an ***E-R diagram***, which is built up from the following components:

  - ***Rectangles***, which represent entity sets

  - ***Ellipses***, which represent attributes

❑***Diamonds,*** which represent relationships among entity sets

❑***Lines***, which link attributes to entity sets and entity sets to relationships

Example

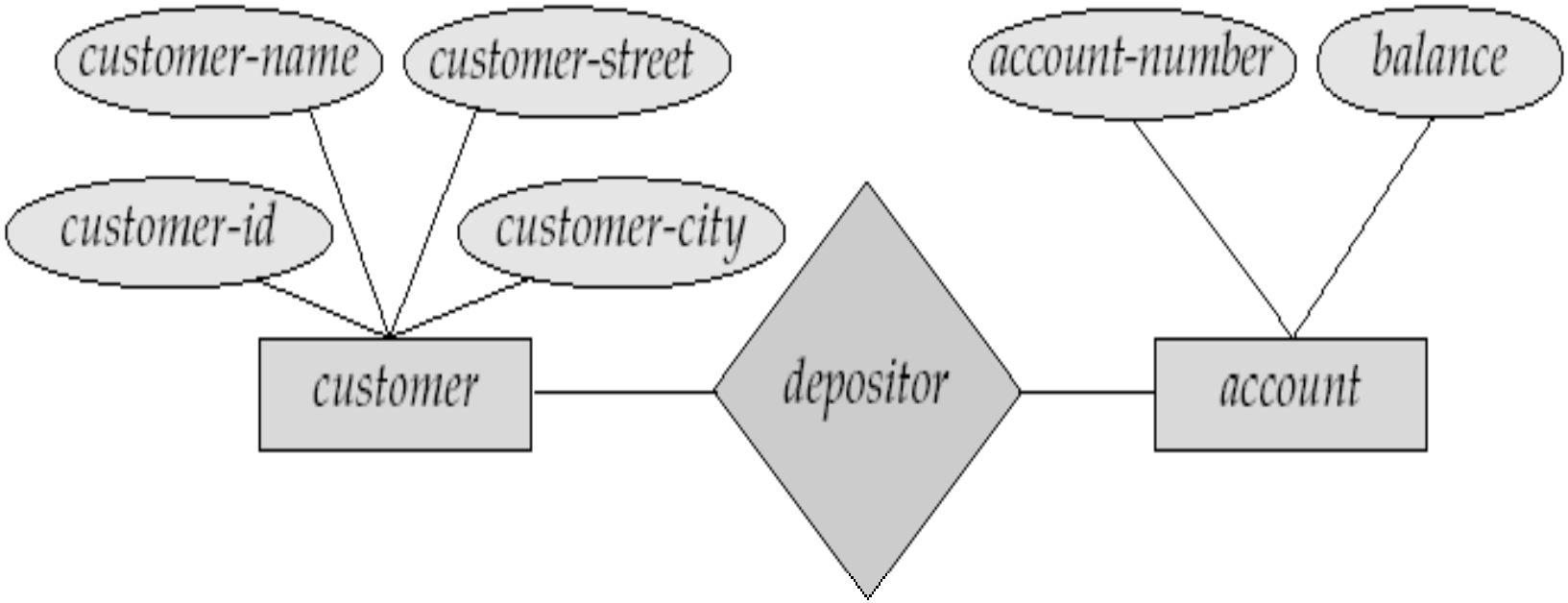


Figure 1.3

Object-based data models (Object-oriented and Object-relational)

- Object-oriented data model can be seen as extending the E-R model with notations of encapsulation, methods (functions), and object identity
- The object-relational data model combines features of the **object-oriented data model** and **relational data model**

Semistructured data model

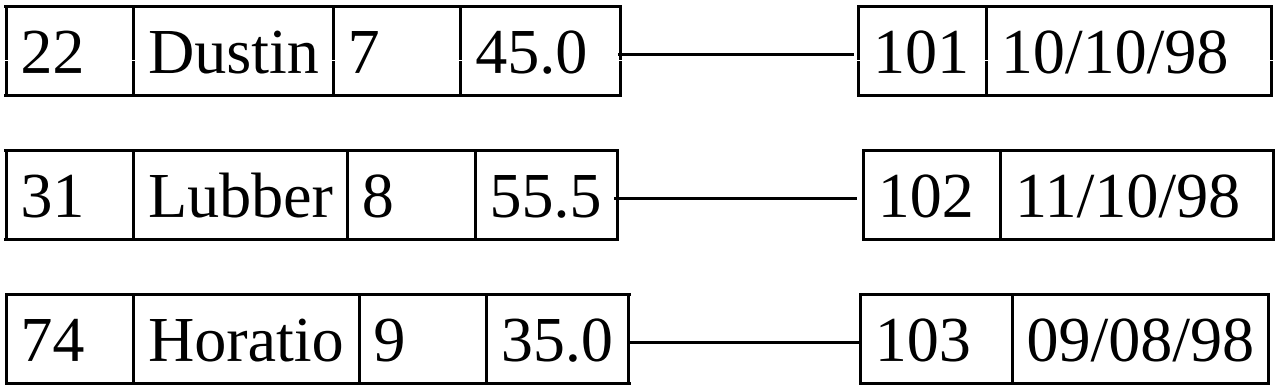
- Semistructured data models permit the specification of data where individual data items of the same type may have different sets of attributes

- This is in contrast with the data models mentioned earlier, where every data item of a particular type must have the same set of attributes

- The extensible markup language (XML) is widely used to represent semistructured data

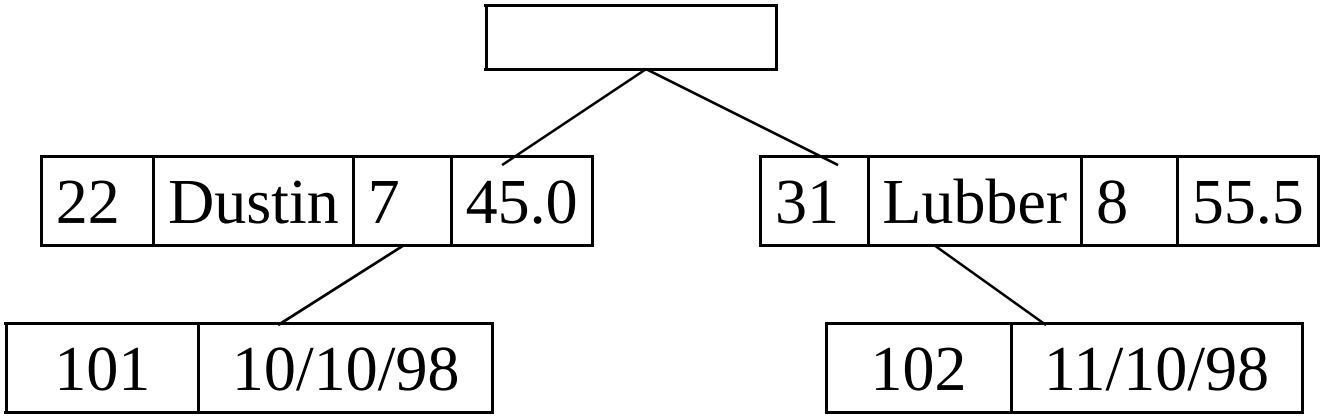
Network model

- Data in this model is represented by collection of records and relationships among data are connected by links



Hierarchical model

- Hierarchical model is same as the network model
- In this model records are represented in the form of tree



Database Languages

- A database system provides

  --data definition language to specify the *database schema*

  --data manipulation language to express *database queries and updates*

  --data control language to *control a database*

- In practice these are not separate languages; instead they simply form parts of a single database language, such as SQL language

Data-Definition Language

- This supports the **creation**, **deletion**, and **modification** of **definitions** for tables, views and indexes

- The following statement in the SQL language creates the *account* table:

create table *account*
(*account-number* ***varchar2*(10),**
balance **number(5)**)

- Execution of the above DDL statement creates the *account* table

- In addition, it updates a special set of tables called the **data dictionary** or **data directory**

- A data dictionary contains **metadata-**that is, data about data

- The schema of a table is an example of metadata

- A database system consults the data dictionary before reading or modifying actual data

- The data stored in the database must satisfy certain **consistency constraints**

- The DDL provides facilities to specify such constraints

- The database systems check these constraints every time the database is updated

Data-Manipulation Language

- Data manipulation is

--The **retrieval** of information stored in the database
--The **insertion** of new information into the database
--The **deletion** of information from the database
--The **modification** of information stored in the database

- A **data-manipulation language (DML)** is a language that enables users to access or manipulate data

- There are of two types:

Procedural DMLs require a user to specify ***what*** data is needed and ***how*** to get those data

Declarative DMLs (also referred to as **nonprocedural DMLs**) require a user to specify ***what*** data is needed ***without*** specifying how to get those data

- This query in the SQL language finds the name of the customer whose customer-id is 192-83-7465:

select *customer-name*
from *customer*
where *customer-id = '192-83-7465'*

Data-Control Language

- This subset of SQL controls a database, including administrative privileges and saving data

- SQL is **nonprocedural Language**

Database Access from Application Programs

- **Application programs** are used to interact with the database

- Application programs are usually written in a language, such as C, C++, or Java

- Examples in a banking system are programs that debit accounts, credit accounts, or transfer funds between accounts

- To access the database, DML statements need to be executed from the host language

- There are **two ways** to do this:

- By providing an **application program interface** (set of procedures) that can be used to send DML and DDL statements to the database, and retrieve the results

  --The Java Database Connectivity (JDBC) with the Java language is a commonly used application program interface standard

- By extending the host language syntax to embed DML calls within the host language program

--Usually, a special character prefaces DML calls, and a preprocessor, called the DML **precompiler,** converts the DML statements to normal procedure calls in the host language

Database Users and Administrator

- People who work with a database can be categorized as database users or database administrator

Database Users and User Interfaces

- There are four different types of database-system users

- Different types of user interfaces have been designed for the different types of users

1.Naive users: users who interact with the system by invoking one of the application programs that have been written previously

(Naive users are users who do not have knowledge about the system)

- They uses **forms interface**, where the users can fill in appropriate fields of the form

- **Ex1:** transfer $50 from account *A* to account *B*

  -user has to enter amount of money to be transferred,
  -account from which the money is to be transferred
  -account to which the money is to be transferred

- **Ex2:** finding account balance over the Internet. user may access a form, where he enters his account number

2.Application programmers: computer professionals who write application programs

- Application programmers can choose from many tools to develop user interfaces

-Rapid Application Development (RAD) tools enable an application programmer to construct forms and reports without writing a program

-Fourth-generation languages, include features to facilitate the generation of forms and the display of data on the screen

-Most major commercial database systems include a fourth-generation language

3.Sophisticated users interact with the system without writing programs

-They form their requests in a database query language.

-Analysts who submit queries to explore data

-Online Analytical Processing (OLAP) tools simplify analysts tasks by letting them view summaries of data in different ways

Ex: an analyst can see **total sales** by **region** (for example, North, South, East, and West), or by **product**, or by a combination of region and product (that is, total sales

of each product in each region)

-**Data mining tools**, help analysts to find certain kinds of patterns in data.

4.Specialized users: users who write specialized database applications

-Computer-aided design systems, knowledge-base and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems

Database Administrator

- A person who has central control over both the data and the programs that access those data is called a **database administrator (DBA)**

- The functions of a DBA include:

-Schema definition. The DBA creates the original database schema by executing a set of data definition statements in the DDL

-Storage structure and access-method definition.

-Schema and physical-organization modification. The DBA carries out changes to the schema to reflect the changing needs of the organization, or to improve the performance

-Granting of authorization for data access. By granting different types of authorization, the DBA can regulate which parts of the database various users can access.

--The authorization information is kept in a file that the database system consults whenever someone attempts to access the data in the system

-Routine maintenance.

Activities such as:

-Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding

-Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required

-Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users

Transaction Management

- A **transaction** is a collection of operations that performs a single logical function in a database application

- **Ex:** Funds transfer in which one account (say *A)* is debited and another account (say *B)* is credited

- It is essential that either both the credit and debit occur, or that neither occur

- Either all operations of the transaction are reflected properly in the database, or none are. This requirement is called **atomicity**

- Execution of the funds transfer must preserve the consistency of the database. That is, the value of the sum $A + B$ must be preserved. This requirement is called **consistency**

- **Isolation:** each transaction is unaware of other transactions executing concurrently in the system

- After the successful execution of a funds transfer, the new values of accounts A and B must persist, despite the possibility of system failure. This requirement is called **durability**

- We require that transactions do not violate any database-consistency constraints

- That is, if the database was consistent when a transaction started, the database must be consistent when the transaction successfully terminates

- It is the programmer's responsibility to define properly the various transactions, so that each preserves the **consistency** of the database

- For example, the transaction to transfer funds from account *A* to account *B* could be defined to be composed of two separate programs: one that debits account *A*, and another that credits account *B*

- The execution of these two programs one after the other will indeed preserve consistency

- Ensuring the **atomicity** and **durability** properties is the responsibility of the database system itself—specifically, of the **transaction-management component**

- If we are to ensure the atomicity property, a failed transaction must have no effect on the state of the database

- Thus, the database must be restored to the state that existed prior to the starting of the transaction

- The database system must therefore perform **failure recovery**, that is, detect system failures and restore the database to the state that existed prior to the occurrence of the failure

- Finally, when several transactions update the database concurrently, the consistency of data may no longer be preserved, even though each individual transaction is correct

- It is the responsibility of the **concurrency-control manager** to control the interaction among the concurrent transactions, to ensure the consistency of the database

Note: each transaction possess **ACID** properties

Database System Structure

- The functional components of a database system can be broadly divided into the ***storage manager*** and the ***query processor*** components

- The ***storage manager*** is important because databases typically require a large amount of storage space

- Corporate databases range in size from hundreds of gigabytes to, for the largest databases, terabytes of data

- A gigabyte is 1000 megabytes and a terabyte is 1000 gigabytes

- Since the main memory of computers cannot store this much information, so this information is stored on disks

- Data is moved between disk storage and main memory as needed

- The ***query processor*** is important because it helps the database system simplify and facilitate access to data

- High-level views help to achieve this goal

- The job of the database system is to translate queries written in a non procedural language, into an efficient sequence of operations at the physical level

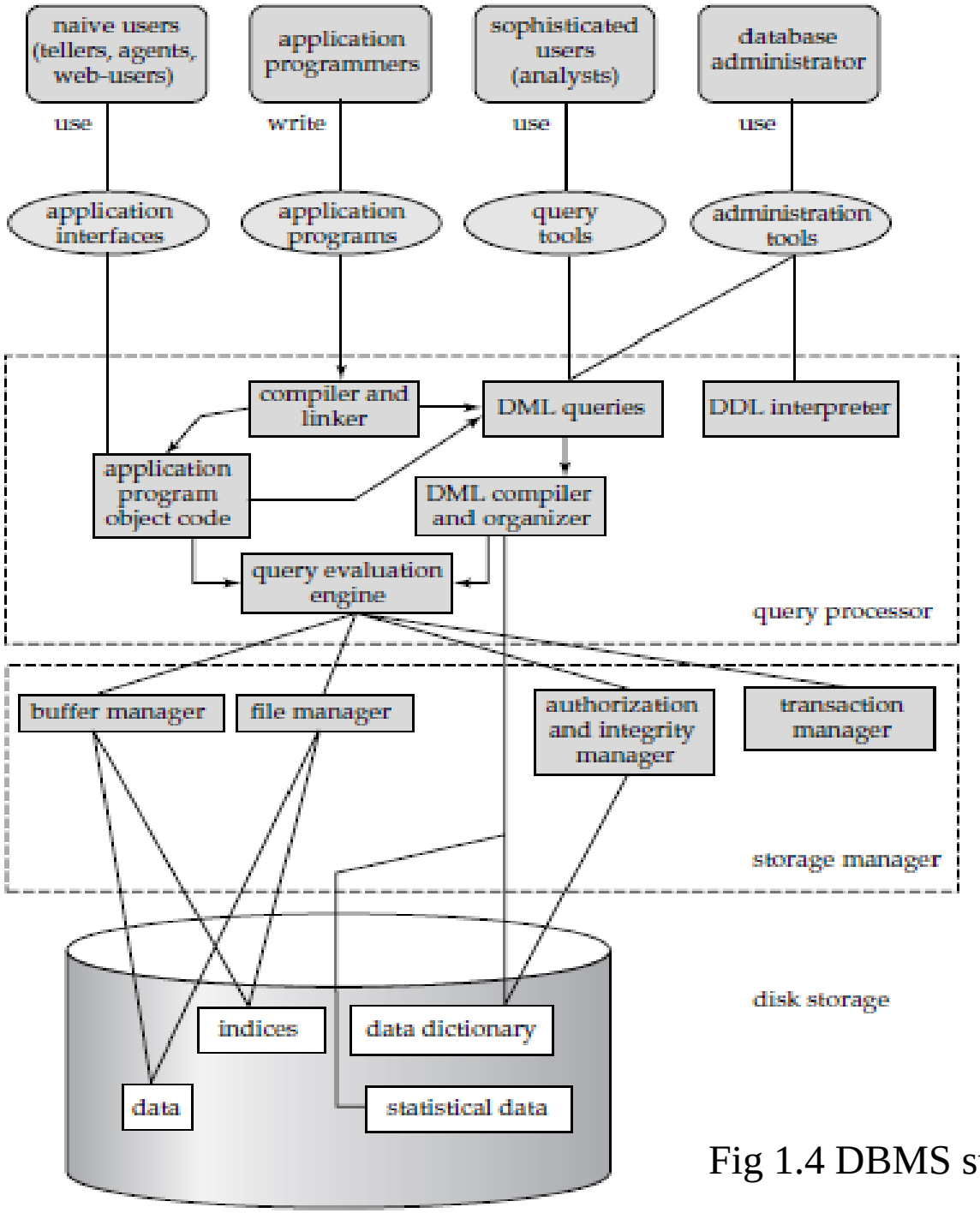


Fig 1.4 DBMS structure

Storage Manager

- A ***storage manager*** is a module that provides the **interface** between the low-level data stored in the database and the application programs and queries submitted to the system

- The storage manager is responsible for the interaction with the file manager

- The raw data is stored on the disk

- The storage manager is **responsible** for storing, retrieving, and updating data in the database

- The storage manager components include:

-Authorization and integrity manager, which tests for the satisfaction of integrity constraints and checks the authority of users to access data

-Transaction manager, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting

-File manager, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk

-Buffer manager, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory

▪The storage manager implements several data structures as part of the physical system implementation:

-Data files, which store the database itself

-Data dictionary, which stores metadata about the structure of the database, in particular the schema of the database

-Indices, which provide fast access to data items that hold particular values

The Query Processor

- The query processor components include

-DDL interpreter, which interprets DDL statements and records the definitions in the data dictionary

-**DML compiler**, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands

--A query can usually be translated into any of a number of alternative evaluation plans that all give the same result

--The DML compiler also performs **query optimization**, that is, it picks the lowest cost evaluation plan from among the alternatives

-Query evaluation engine, which executes low-level instructions generated by the DML compiler

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** consist 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

- **Ex:** 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

- Data sizes were much larger than main memory

- In Tapes only **sequential accessing** is possible

Late 1960s and 1970s:

- Use of **hard disks** in the late 1960s changed the scenario for data processing greatly, since hard disks allowed direct access to data

- Data in any location on disk could be accessed in just tens of milliseconds

- In hard disks **random accessing** is possible

- 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:

- The relational model was not used in practice initially, because of its 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

- 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 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,

--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

- 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

- Schema can be refined using *normalization*

(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 redesign of parts of the database schema obtained from the earlier design steps

(6) Security Design:

- 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

- Complete database design will probably require a subsequent **tuning phase** in which all six kinds of design steps are interleaved and repeated until the design is satisfactory

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

- We could define an entity set called Employees that contains employee entities

- 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 key is *ssn*

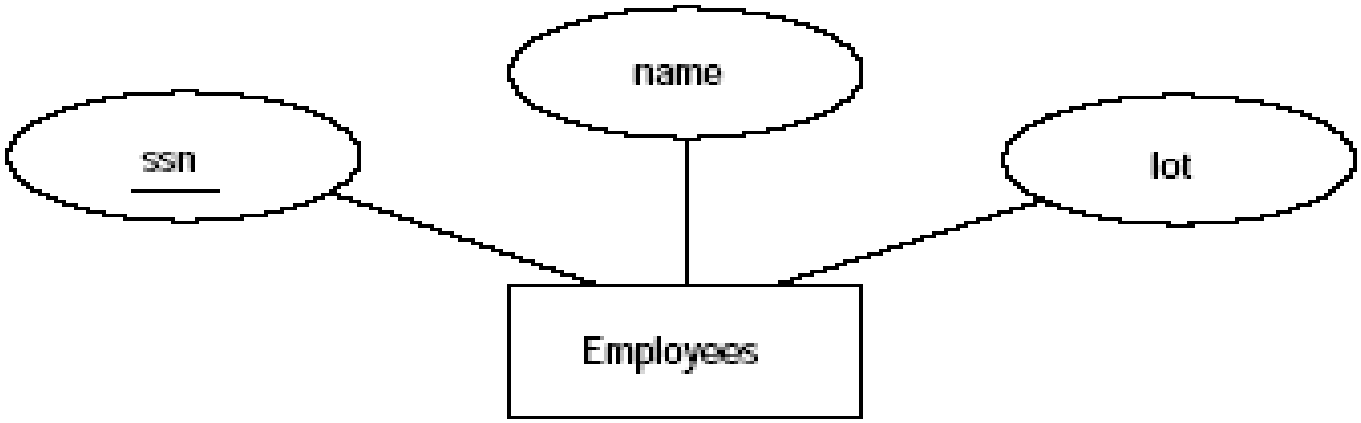


Figure 2.1 The Employees Entity Set

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

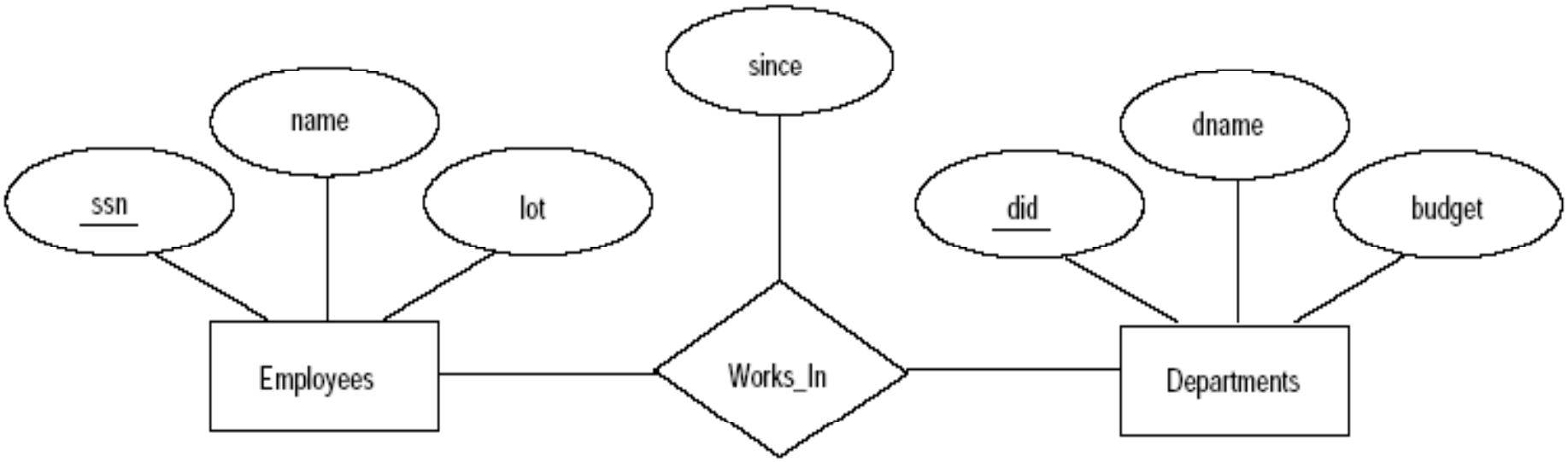


Figure 2.2 The Works_In Relationship Set

- 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

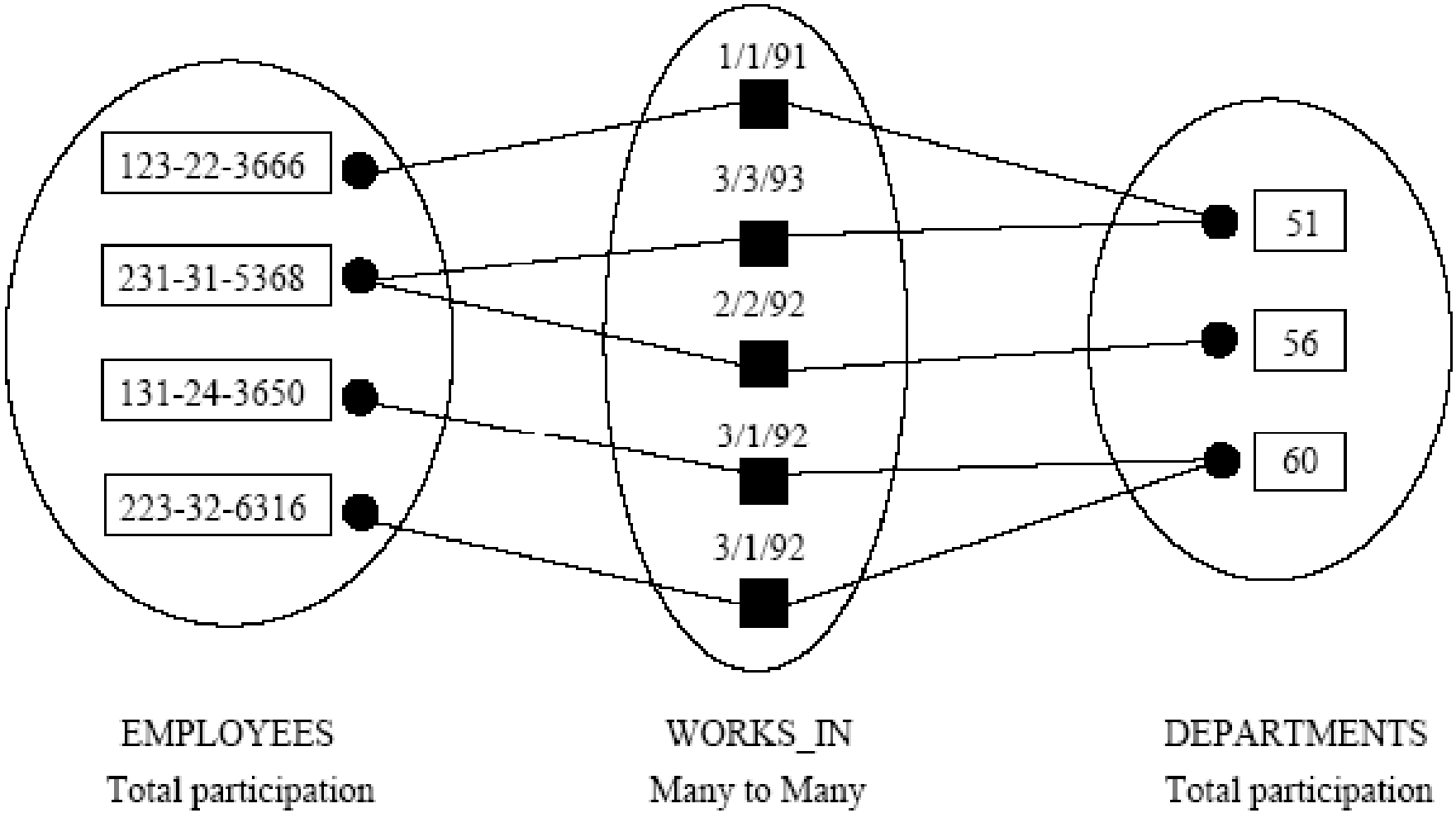


Figure 2.3 An Instance of the Works_In Relationship Set

- 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

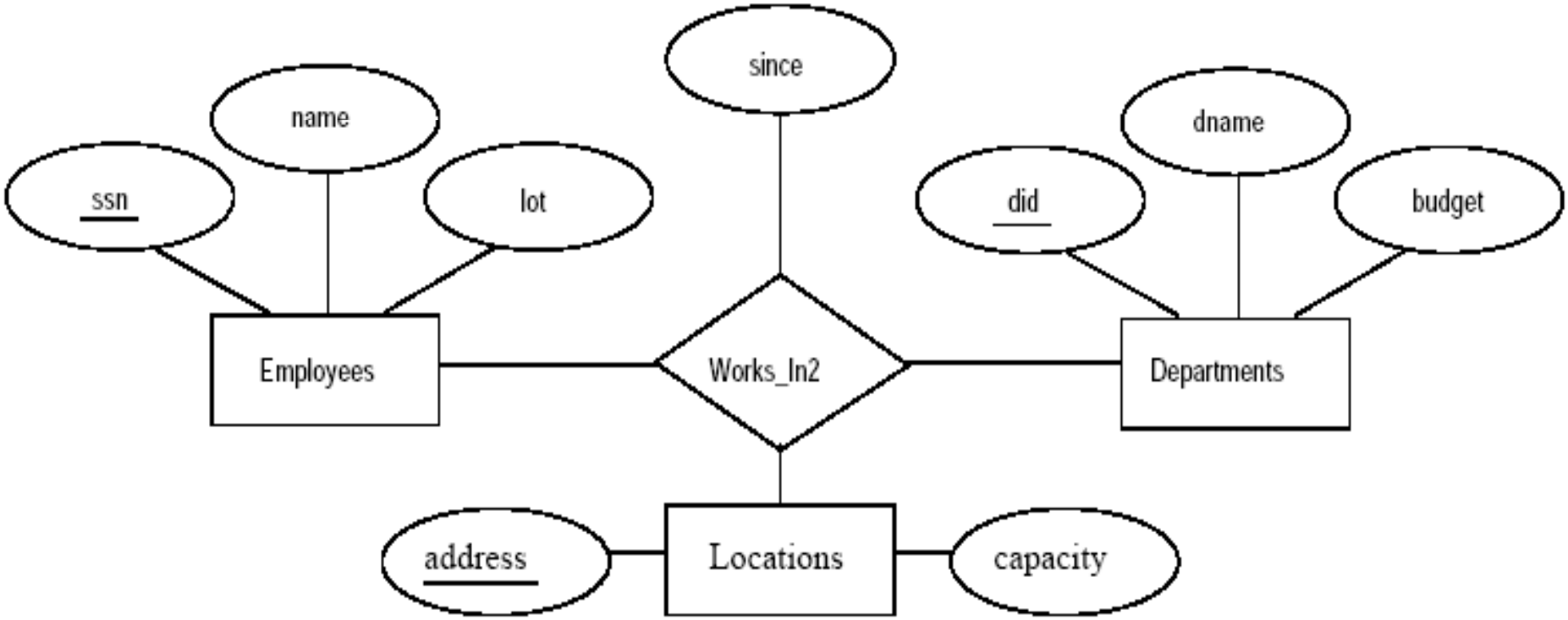


Figure 2.4 A Ternary Relationship Set

- 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

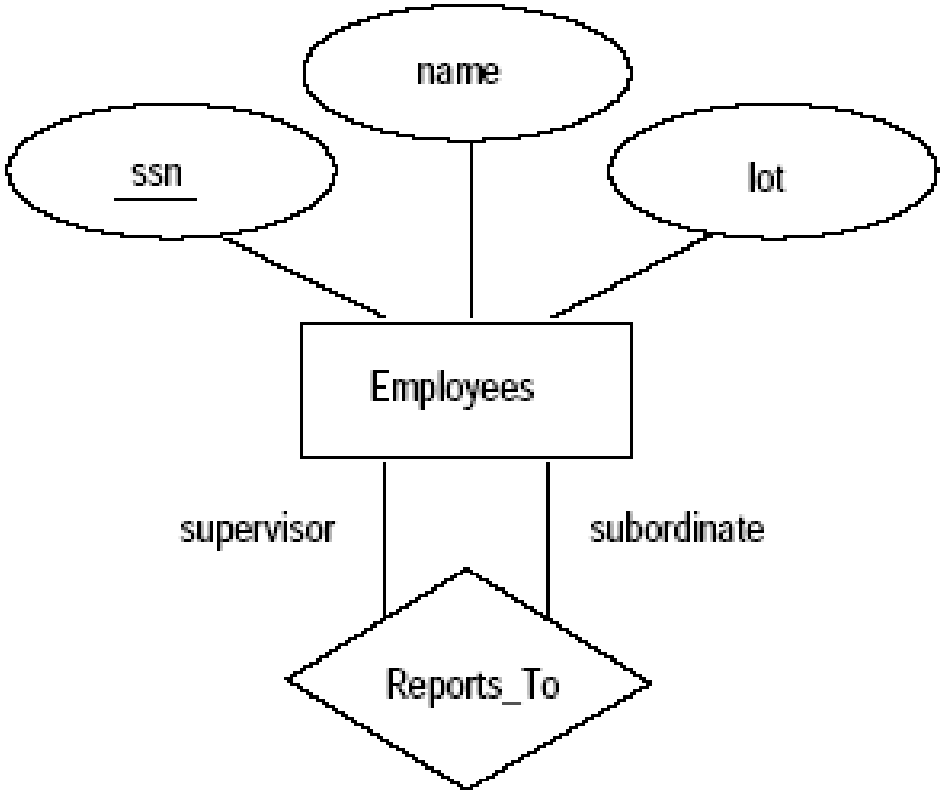


Figure 2.5 The Reports_To Relationship Set

- 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 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

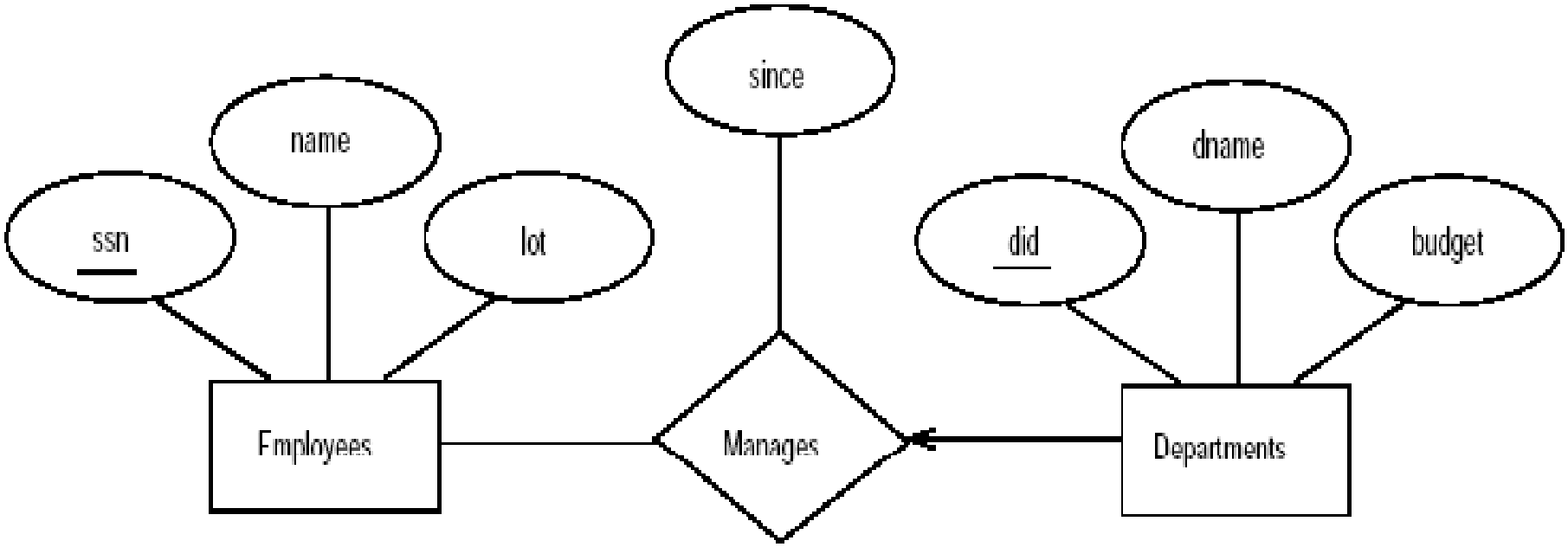


Figure 2.6 Key Constraint on Manages

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

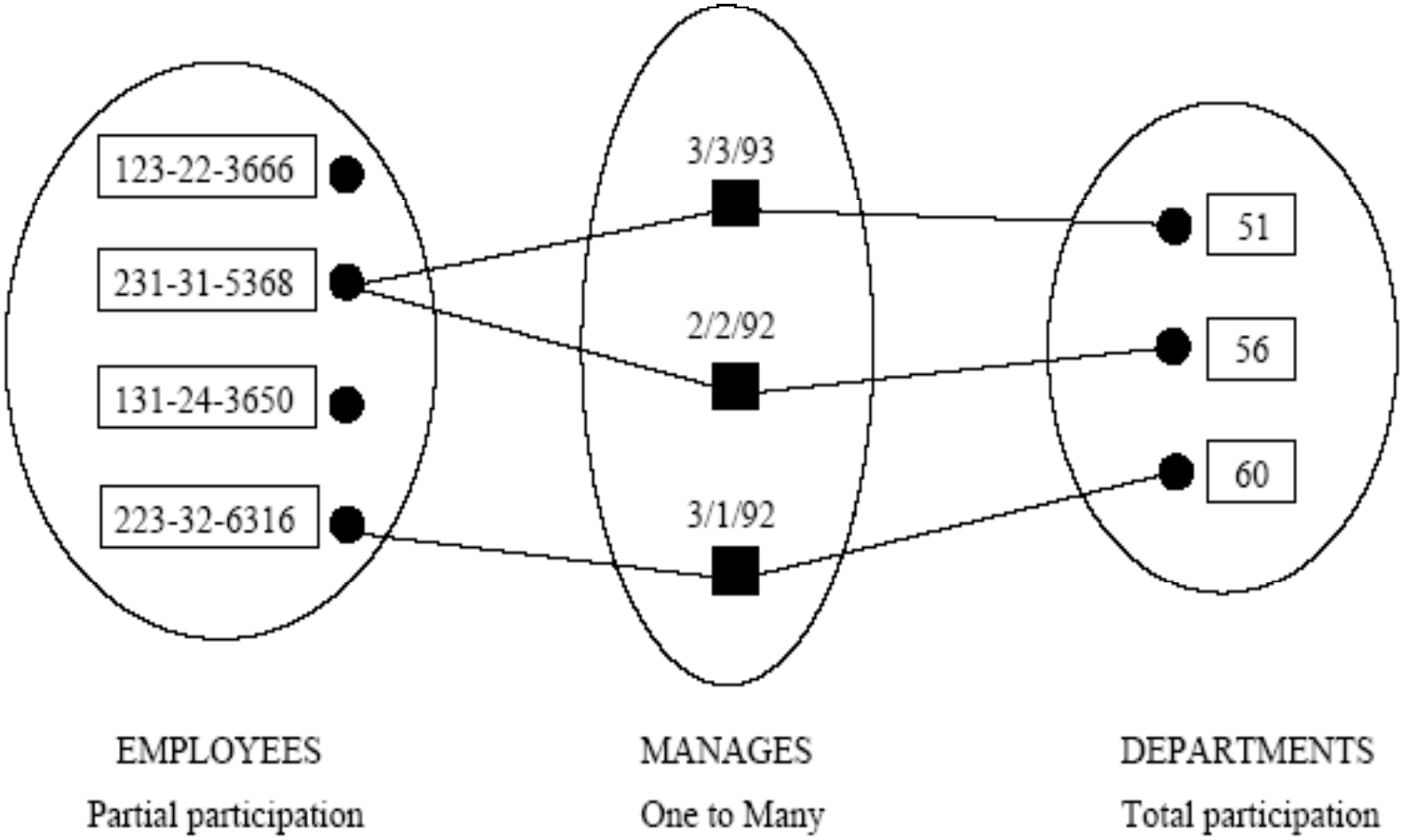


Figure 2.7 An Instance of the Manages Relationship Set

- 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

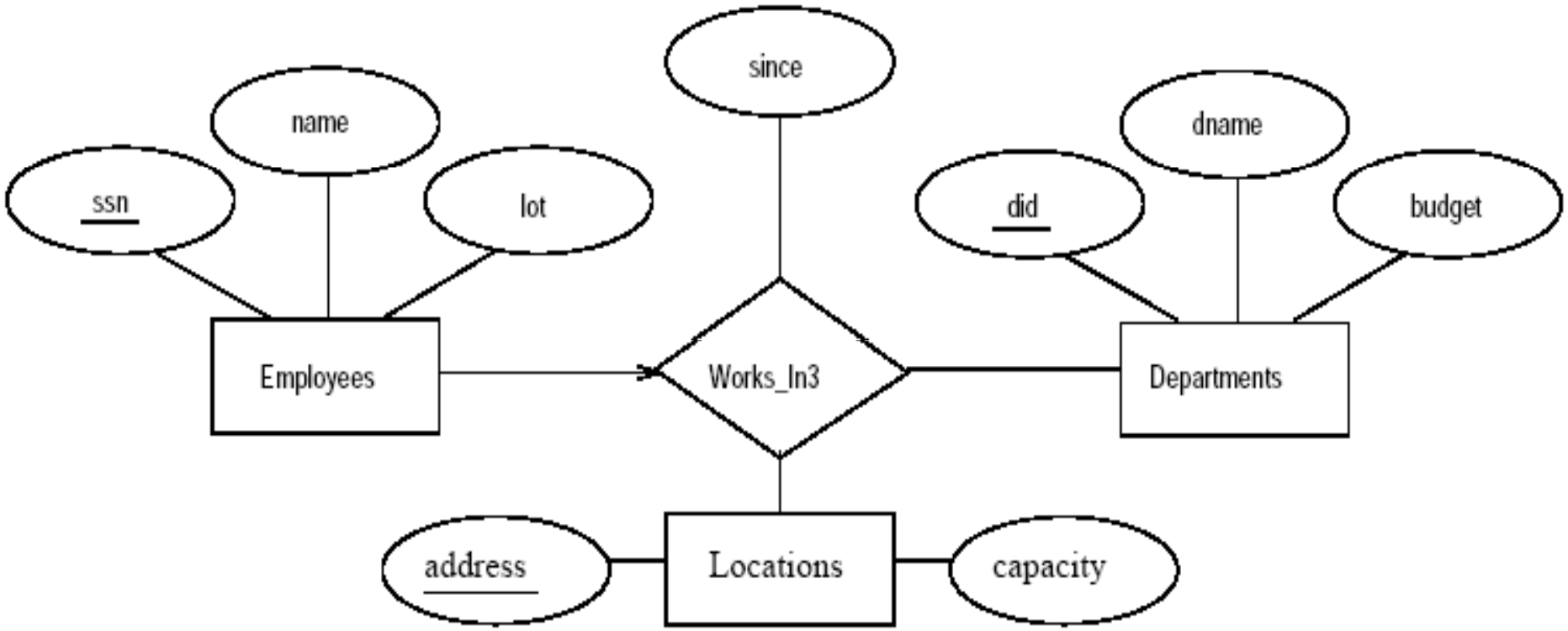


Figure 2.8 A Ternary Relationship Set with Key Constraints

- 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

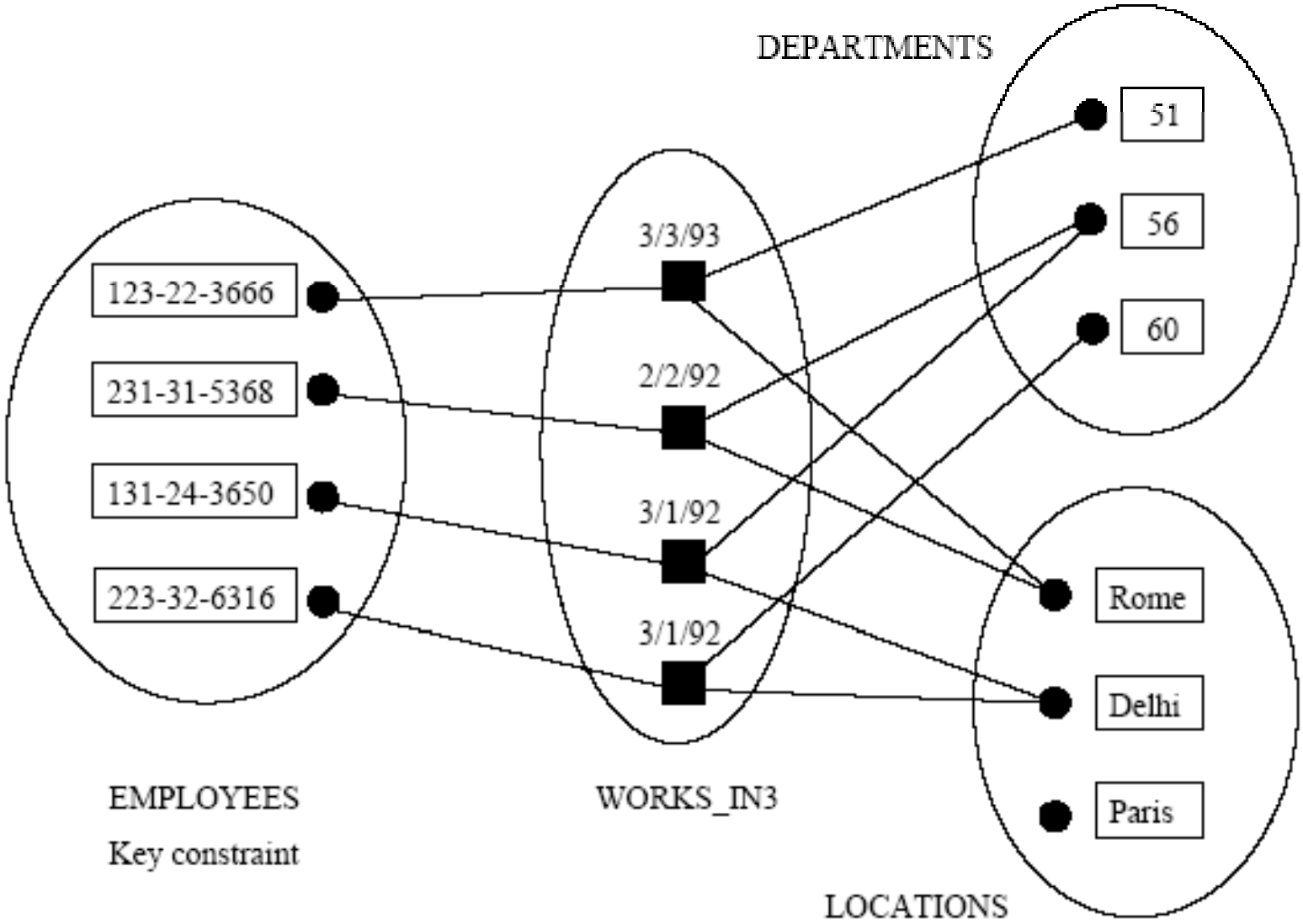


Figure 2.9 An Instance of Works_In3

Participation Constraints

- 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

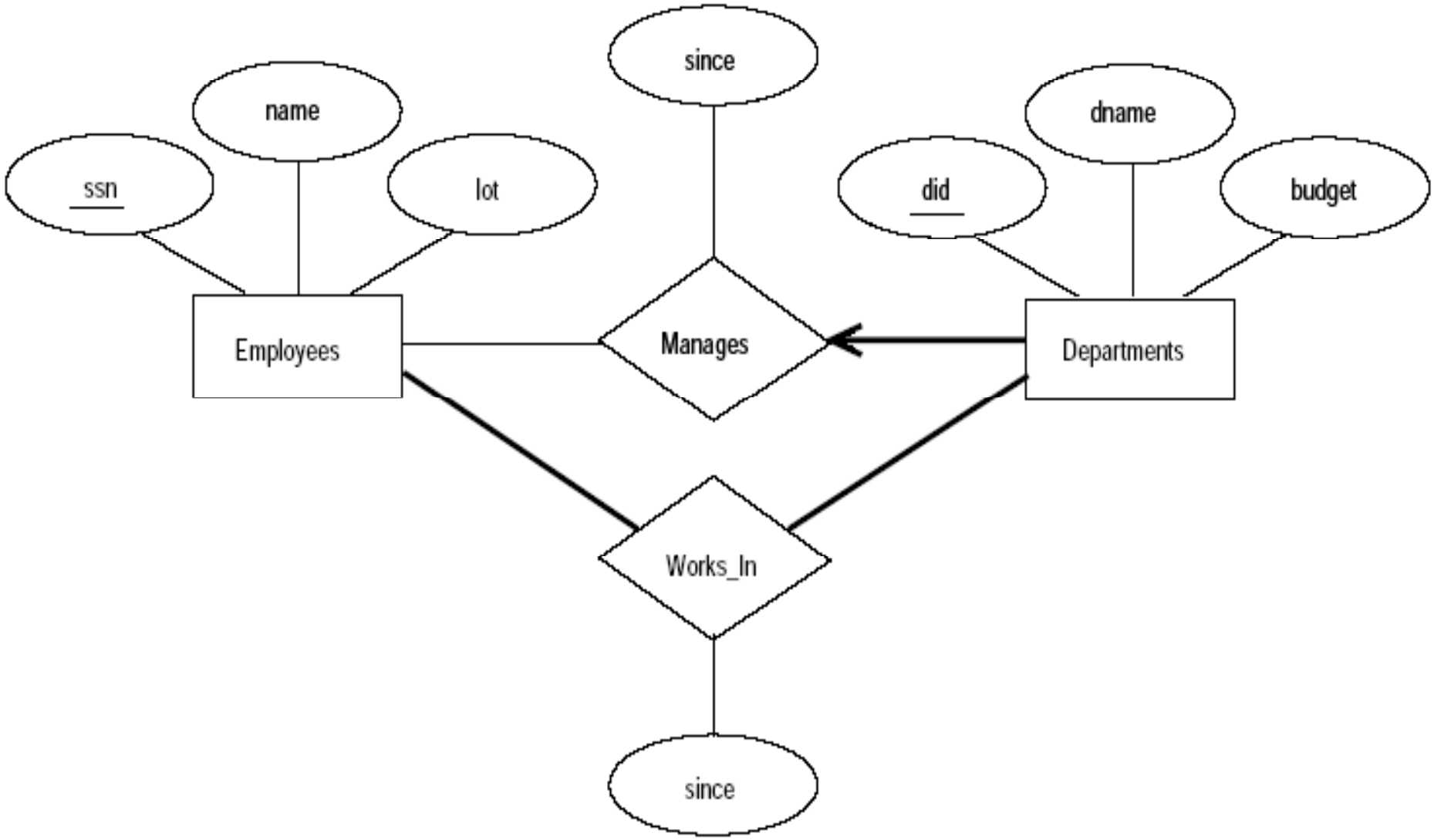


Figure 2.10 Manages and Works_In

Weak Entities

- Entity set which does not have a key is called weak entity set

- 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 set, 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

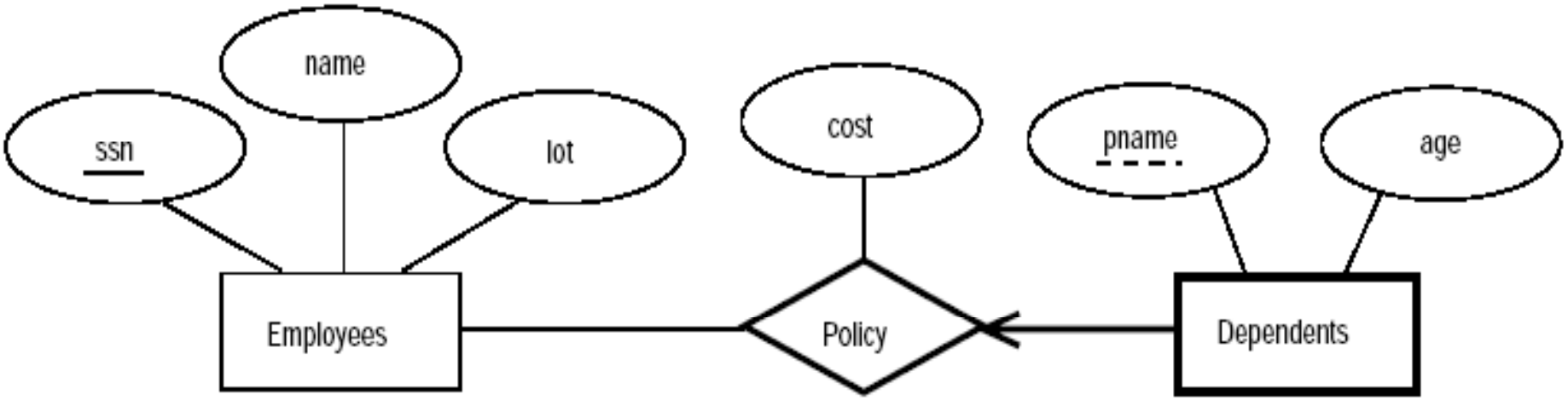


Figure 2.11 A Weak Entity Set

- 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

Class Hierarchies

- Sometimes it is natural to classify the entities in an entity set into subclasses

- For example, we might want to talk about an Hourly_Emps entity set and a Contract_Emps entity set to distinguish the basis on which they are paid

- We might have attributes *hours_worked* and *hourly_wages* defined for Hourly_Emps and an attribute *contractid* defined for Contract_Emps

- The attributes defined for an Hourly_Emps entity set are the attributes for Employees plus Hourly_Emps

- The attributes for the entity set Employees are **Inherited** by the entity set Hourly_Emps, and that Hourly_ Emps **ISA** (read *is a)* Employees

- Figure 2.12 illustrates the class hierarchy

- The entity set Employees may also be classified using a different criterion

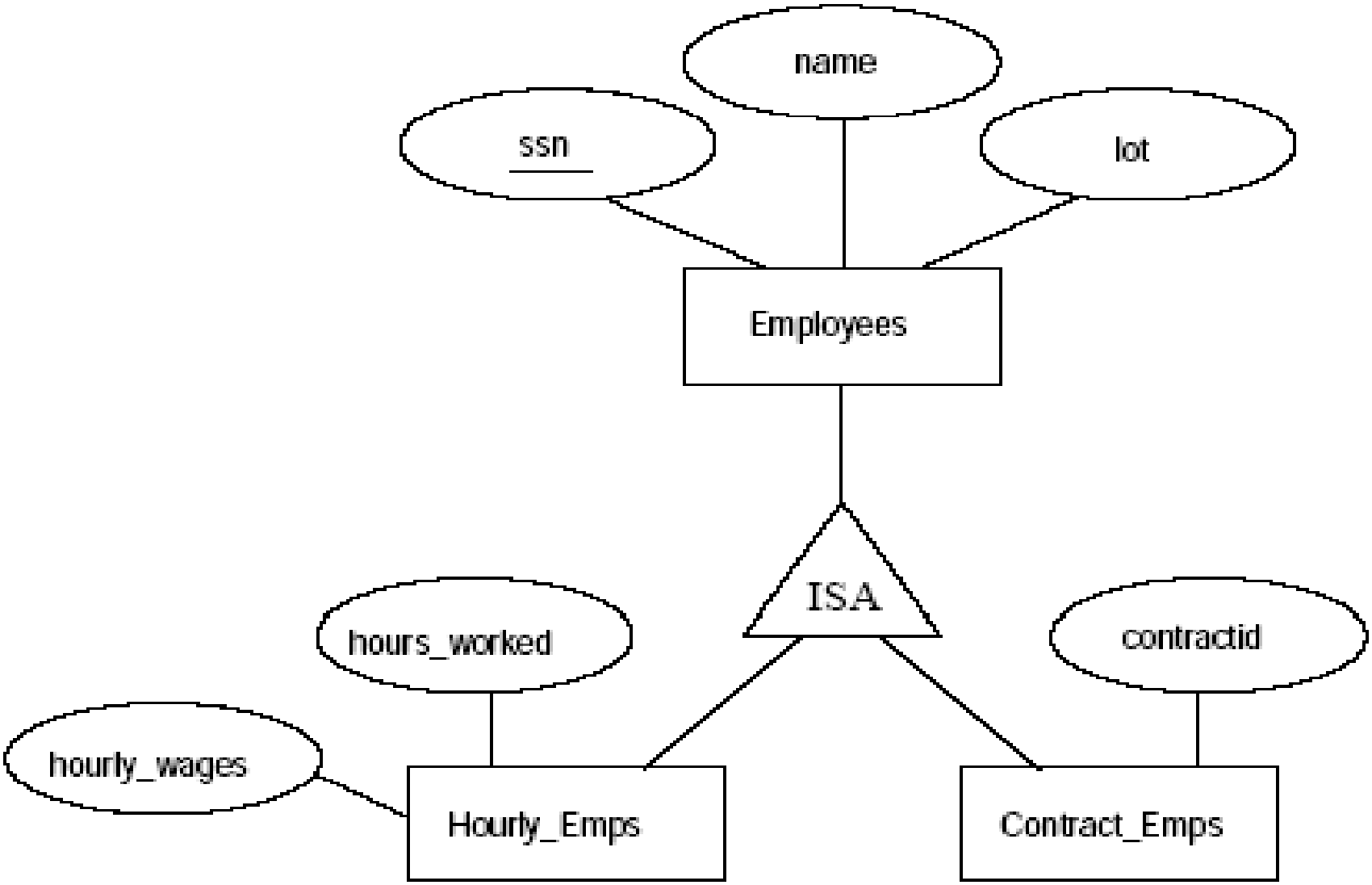


Figure 2.12 Class Hierarchy

▪For example, we might identify a subset of employees as Senior_Emps

▪A class hierarchy can be viewed in one of two ways:
--Employees (the **superclass)** is **specialized** into **subclasses**. subclass-specific attributes are then added
--Hourly_Emps and Contract_Emps are **generalized** by Employees. As another example, two entity sets Motorboats and Cars may be generalized into an entity set Motor_ Vehicles

▪We can specify two kinds of constraints with respect to ISA hierarchies, namely, *overlap* and *covering constraints*

- **Overlap constraints** determine whether two subclasses are allowed to contain the same entity

- For example, can John be both an Hourly_Emps entity and a Contract_Emps entity? …no

- Can he be both a Contract_Emps entity and a Senior_Emps entity? …yes

- We denote this by writing 'Contract_Emps OVERLAPS Senior_Emps'

- **Covering constraints** determine whether the entities in the subclasses collectively include all entities in the superclass

▪For example, does every Employees entity have to belong to one of its subclasses? …no

▪Does every Motor_Vehicles entity have to be either a Motorboats entity or a Cars entity? …yes

▪We denote this by writing 'Motorboats AND Cars COVER Motor_Vehicles'

Aggregation

▪Sometimes we have to model a relationship between a collection of entities and relationships

▪**Aggregation** allows us to indicate that a relationship set (identified through a dashed box) participates in another relationship set.This is illustrated in Figure 2.13

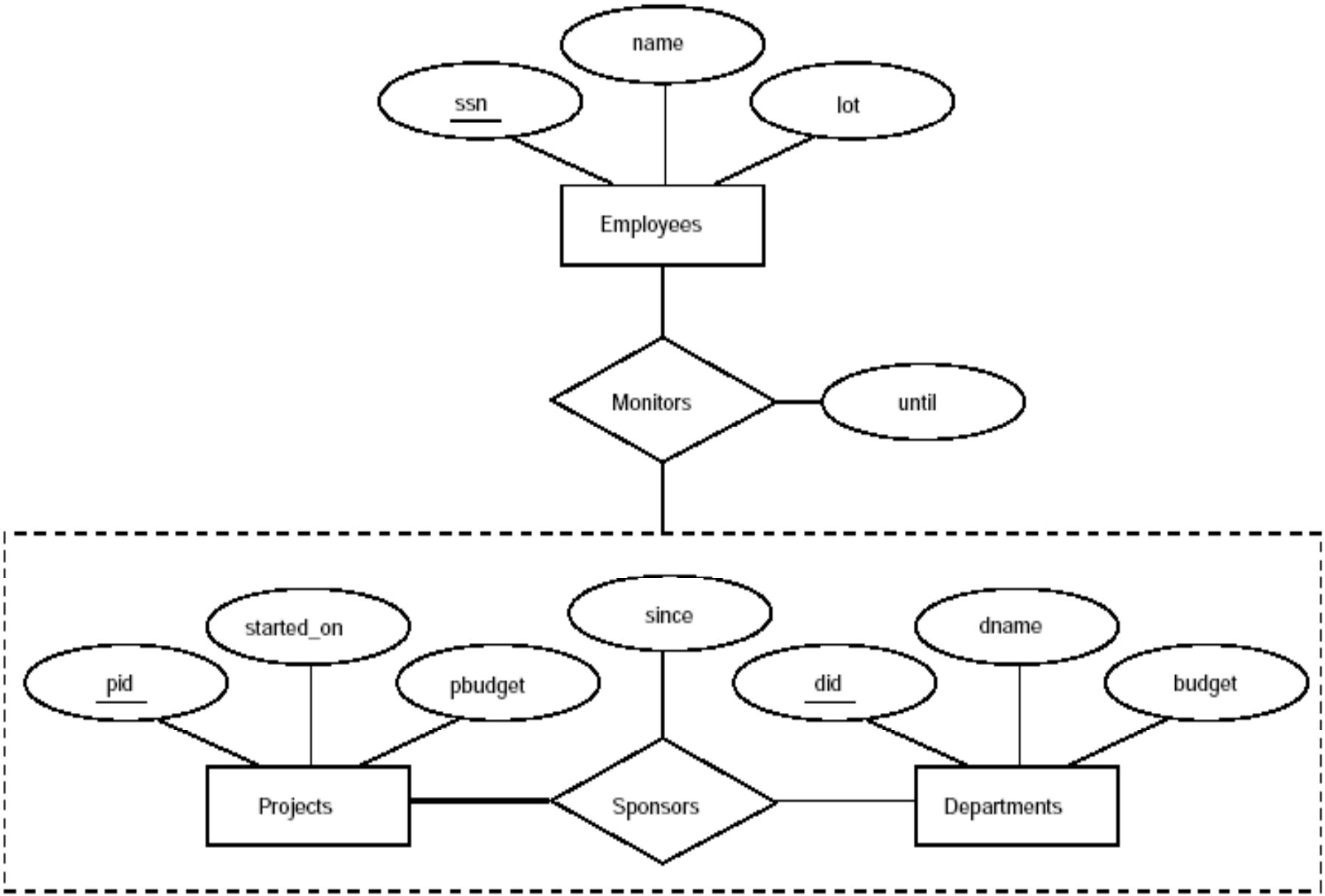


Figure 2.13 Aggregation

- Use aggregation when we need to express a relationship among relationships

CONCEPTUAL DESIGN FOR LARGE ENTERPRISES

- Designing database for large enterprise takes efforts of more than one designer

- It diagrammatically represents the complete database and enables the user who provide inputs to database, to understand the complete functionality of database

- Large databases are modeled in two methodologies:

(I) -The requirements of all the users are collected

-The conflicting requirements are resolved and a final conceptual view is generated that satisfies the requirements of all users

(II)-The user provides his requirements, the designer generates a conceptual view from these requirements

-Likewise all the conceptual views from all user requirements are generated

-Comprehensive conceptual view that satisfies all the requirements is generated