Database Management Systems (BCS403) - 2023-24 Module 1

Dr. Narender M
Department of CS&E
The National Institute of Engineering

Topics

Introduction to Databases	• Introduction, Characteristics of database approach, Advantages of using the DBMS approach, History of database applications.
Overview of Database Languages and Architectures	• Data Models, Schemas, and Instances. Three schema architecture and data independence. Database languages, and interfaces, The Database System environment.
Conceptual Data Modelling using Entities and Relationships	• Entity types, Entity sets and structural constraints, Weak entity types, ER diagrams, Specialization and Generalization

Before we start, let us have a look at the

Academic Calendar

Syllabus

Evaluation Pattern

Abridged Lesson Plan

Introduction

- Databases play a critical role in almost all areas where computers are used.
- A database is a collection of related data.
- By data, we mean known facts that can be recorded and that have implicit meaning.
- For example, consider the names, telephone numbers, and addresses of the people.
- Data can also be recorded in an indexed address book or stored on a hard drive, using a personal computer and software such as Microsoft Access or Excel.
- This collection of related data with an implicit meaning is a database.

Introduction

- Database has the following implicit properties:
- A database **represents** some aspect of the **real world**, sometimes called the **miniworld** or the universe of discourse (UoD).
- A database is a **logically coherent collection of data** with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.
- A database is designed, built, and populated with data for a **specific purpose**. It has an intended group of users and some **preconceived applications** in which these users are interested.

Introduction

- For a database to be **accurate and reliable** at all times, it must be a true reflection of the miniworld that it represents.
- Therefore, **changes** must be reflected in the database as soon as possible.
- A database can be of **any size and complexity**.
- A database may be generated and maintained manually, or it may be computerized.
- A **database management system** (DBMS) is a computerized system that enables users to create and maintain a database.

Introduction

- The **DBMS** is a general-purpose software system that facilitates the processes of **defining, constructing, manipulating, and sharing databases** among various users and applications.
- **Defining** a database involves **specifying the data types, structures, and constraints** of the data to be stored in the database.
- The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called **meta-data**.

Introduction

- **Constructing** the database is the process of storing the data on some storage medium that is controlled by the DBMS.
- **Manipulating** a database includes functions such as querying the database to retrieve specific data, **updating** the database to reflect changes in the miniworld, and generating reports from the data.
- **Sharing** a database allows multiple users and programs to access the database simultaneously.

Introduction

- An **application program** accesses the database by sending queries or requests for data to the DBMS.
- A **query** typically causes some data to be retrieved; a **transaction** may cause some data to be **read** and some data to be **written** into the database.
- We will call the database and DBMS software together a database system.

Characteristics of database approach

- In traditional file processing, each user defines and implements the files needed for a specific software application as part of programming the application.
- This redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common up-to-date data.
- In the database approach, a single repository maintains data that is defined once and then accessed by various users repeatedly through queries, transactions, and application programs.

Characteristics of database approach

- The main characteristics of the database approach versus the file-processing approach are the following:
  - Self-describing nature of a database system
  - Insulation between programs and data, and data abstraction
  - Support of multiple views of the data
  - Sharing of data and multiuser transaction processing

Characteristics of database approach

Self-Describing Nature of a Database System

- Database system contains not only the database itself but also a complete definition or description of the database structure and constraints.
- This definition is stored in the DBMS catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data.
- The information stored in the catalog is called meta-data

Characteristics of database approach

Self-Describing Nature of a Database System

- Some newer types of database systems, known as NOSQL systems, do not require meta-data.
- Rather the data is stored as self-describing data that includes the data item names and data values together in one structure.
- The DBMS software must work equally well with any number of database applications. (University, Banking, E-Commerce, etc.)
- In traditional file processing, data definition is typically part of the application programs themselves.
- Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs.

Characteristics of database approach

Insulation between Programs and Data, and Data Abstraction

- In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access that file.
- By contrast, DBMS access programs do not require such changes in most cases.
- The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property program-data independence.

Characteristics of database approach

Insulation between Programs and Data, and Data Abstraction

- In some types of database systems (object-oriented database systems), users can define operations on data as part of the database definitions.
- An operation (a function or method) is specified in two parts. The interface of an operation includes the operation name and the data types of its arguments.
- The implementation of the operation is specified separately and can be changed without affecting the interface.
- User application can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This is termed as program-operation independence.

Characteristics of database approach

Insulation between Programs and Data, and Data Abstraction

- The characteristic that allows program-data independence and program-operation independence is called data abstraction.
- A DBMS provides users with a conceptual representation of data that does not include many of the details of how the data is stored or how the operations are implemented.
- Informally, a data model is a type of data abstraction that is used to provide this conceptual representation.

Characteristics of database approach

Insulation between Programs and Data, and Data Abstraction

- The data model uses logical concepts, such as objects, their properties, and their interrelationships, that may be easier for most users to understand than computer storage concepts.
- Hence, the data model hides storage and implementation details that are not of interest to most database users.

Characteristics of database approach

Support of Multiple Views of the Data

- A database typically has many types of users, each of whom may require a different perspective or view of the database.
- A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored.
- A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views.

Characteristics of database approach

Sharing of Data and Multiuser Transaction Processing

- A multiuser DBMS must allow multiple users to access the database at the same time.
- This is essential if data for multiple applications is to be integrated and maintained in a single database.
- The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.
- **Online transaction processing** (OLTP) applications.

Characteristics of database approach

Sharing of Data and Multiuser Transaction Processing

- A transaction is an executing program or process that includes one or more database accesses, such as reading or updating of database records.
- The isolation property ensures that each transaction appears to execute in isolation from other transactions, even though hundreds of transactions may be executing concurrently.
- The atomicity property ensures that either all the database operations in a transaction are executed or none are.

Advantages of using the DBMS approach

- These capabilities are in addition to the four main characteristics discussed.
- The DBA must utilize these capabilities to accomplish a variety of objectives related to the design, administration, and use of a large multiuser database.

Advantages of using the DBMS approach

1. Controlling Redundancy

- In traditional software development utilizing file processing, every user group maintains its own files for handling its data-processing applications.
- This redundancy in storing the same data multiple times leads to several problems.
- First, there is the need to perform a single logical update multiple times: once for each file where student data is recorded. This leads to **duplication of effort.**
- Second, **storage space** is wasted when the same data is stored repeatedly, and this problem may be serious for large databases.
- Third, files that represent the same data may become **inconsistent.** This may happen because an update is applied to some of the files but not to others.

Advantages of using the DBMS approach

1. Controlling Redundancy

- Ideally, we should have a database design that stores each logical data item in only one place in the database.
- This is known as data normalization, and it ensures consistency and saves storage space.
- It is sometimes necessary to use controlled redundancy to improve the performance of queries.
- By placing all the data together, we do not have to search multiple files to collect this data. This is known as denormalization.
- DBMS should have the capability to control this redundancy to prohibit inconsistencies among the files.

Advantages of using the DBMS approach

2. Restricting Unauthorized Access

- When multiple users share a large database, it is likely that most users will not be authorized to access all information in the database.
- Some users may only be permitted to retrieve data, whereas others are allowed to retrieve and update.
- Typically, users or user groups are given account numbers protected by passwords, which they can use to gain access to the database.
- A DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts and to specify account restrictions.

Advantages of using the DBMS approach

3. Providing Persistent Storage for Program Objects

- The values of program variables or objects are discarded once a program terminates.
- The programmer must explicitly store them in permanent files, which often involves converting these complex structures into a format suitable for file storage.
- When the need arises to read this data, the programmer must convert from the file format to the program variable or object structure.

Advantages of using the DBMS approach

3. Providing Persistent Storage for Program Objects

- Object-oriented database systems are compatible with programming languages such as C++ and Java, and the DBMS software automatically performs any necessary conversions.
- A complex object in C++ can be stored permanently in an object-oriented DBMS.
- Such an object is said to be persistent, since it survives the termination of program execution and can later be directly retrieved by another program.

Advantages of using the DBMS approach

3. Providing Persistent Storage for Program Objects

- Traditional database systems often suffered from the impedance mismatch problem.
- Data structures provided by the DBMS were incompatible with the programming language's data structures.

Advantages of using the DBMS approach

4. Providing Storage Structures and Search Techniques for Efficient Query Processing

- The database is typically stored on disk, the DBMS must provide specialized data structures and search techniques to speed up disk search for the desired records.
- Auxiliary files called indexes are often used for this purpose. Indexes are typically based on tree data structures or hash data structures that are suitably modified for disk search.
- To process the database records needed by a particular query, those records must be copied from disk to main memory.
- Therefore, the DBMS often has a buffering or caching module that maintains parts of the database in main memory buffers.

Advantages of using the DBMS approach

4. Providing Storage Structures and Search Techniques for Efficient Query Processing

- The query processing and optimization module of the DBMS is responsible for choosing an efficient query execution plan for each query based on the existing storage structures.
- The choice of which indexes to create and maintain is part of physical database design and tuning, which is one of the responsibilities of the DBA staff.

Advantages of using the DBMS approach

5. Providing Backup and Recovery

- A DBMS must provide facilities for recovering from hardware or software failures.
- The backup and recovery subsystem of the DBMS is responsible for recovery.
- If the computer system fails in the middle of a complex update transaction, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the transaction started executing.
- Disk backup is also necessary in case of a catastrophic disk failure.

Advantages of using the DBMS approach

6. Providing Multiple User Interfaces

- Many types of users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces.
- These include apps for mobile users, query languages for casual users, programming language interfaces for application programmers, forms and command codes for parametric users, and menu-driven interfaces and natural language interfaces for standalone users.

Advantages of using the DBMS approach

6. Providing Multiple User Interfaces

- Many specialized languages and environments exist for specifying graphical user interfaces (GUIs).
- Capabilities for providing Web GUI interfaces to a database are common.

Advantages of using the DBMS approach

7. Representing Complex Relationships among Data

- A database may include numerous varieties of data that are interrelated in many ways.
- DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise.

Advantages of using the DBMS approach

8. Enforcing Integrity Constraints

- Most database applications have certain integrity constraints that must hold for the data.
- A DBMS should provide capabilities for defining and enforcing these constraints.
- The simplest type of integrity constraint involves specifying a data type for each data item.
- A more complex type of constraint that frequently occurs involves specifying that a record in one file must be related to records in other files - referential integrity constraint

Advantages of using the DBMS approach

8. Enforcing Integrity Constraints

- Another type of constraint specifies uniqueness on data item values – Key or Uniqueness constraint.
- It is the responsibility of the database designers to identify integrity constraints during database design.
- A data item may be entered erroneously and still satisfy the specified integrity constraints.

Advantages of using the DBMS approach

9. Permitting Inferencing and Actions Using Rules and Triggers

- Some database systems provide capabilities for defining deduction rules for inferencing new information from the stored database facts.
- Such systems are called deductive database systems.
- It is possible to associate triggers with tables. A trigger is a form of a rule activated by updates to the table, which results in performing some additional operations.

Advantages of using the DBMS approach

9. Permitting Inferencing and Actions Using Rules and Triggers

- Stored procedures become a part of the overall database definition and are invoked appropriately when certain conditions are met.
- Active database systems provide active rules that can automatically initiate actions when certain events and conditions occur.

Advantages of using the DBMS approach

10. Additional Implications of Using the Database Approach

- Potential for Enforcing Standards
  - The database approach permits the DBA to define and enforce standards among database users in a large organization.
  - Standards can be defined for names and formats of data elements, display formats, report structures, terminology, and so on.
  - The DBA can enforce standards in a centralized database environment more easily.

Advantages of using the DBMS approach

10. Additional Implications of Using the Database Approach

- Reduced Application Development Time
  - Once a database is up and running, substantially less time is generally required to create new applications using DBMS facilities.
  - Development time using a DBMS is estimated to be one sixth to one-fourth of that for a file system.

Advantages of using the DBMS approach

10. Additional Implications of Using the Database Approach

- Flexibility
  - It may be necessary to change the structure of a database as requirements change.
  - Modern DBMSs allow certain types of evolutionary changes to the structure of the database without affecting the stored data.
- Availability of Up-to-Date Information.
  - As soon as one user's update is applied to the database, all other users can immediately see this update.
  - This availability of up-to-date information is essential for many transaction-processing applications, such as reservation systems or banking databases.

Advantages of using the DBMS approach

10. Additional Implications of Using the Database Approach

- Economies of Scale
  - The DBMS approach permits consolidation of data and applications, thus reducing the amount of wasteful overlap between activities of data-processing personnel in different projects or departments as well as redundancies among applications.
  - This enables to invest in more powerful processors, storage devices, or networking gear, rather than having each department purchase its own (lower performance) equipment.

History of database applications

Early Database Applications Using Hierarchical and Network Systems

- Many early database applications maintained records in large organizations.
- In many of these applications, there were large numbers of records of similar structure and interrelationships among them.
- **Intermixing of conceptual relationships with the physical** storage and placement of records on disk.
- Hence, these systems **did not provide sufficient data abstraction** and program-data independence capabilities.

History of database applications

Early Database Applications Using Hierarchical and Network Systems

- It did **not provide enough flexibility** to access records efficiently when new queries and transactions were identified.
- They provided only programming language interfaces. This made it **time-consuming and expensive to implement new queries** and transactions, since new programs had to be written, tested, and debugged.
- Early systems were based on three main paradigms: **hierarchical systems, network model–based systems, and inverted file systems.**

History of database applications

Providing Data Abstraction and Application Flexibility with Relational Databases

- Relational databases were originally proposed **to separate the physical storage of data from its conceptual representation** and to provide a **mathematical foundation** for data representation and querying.
- The relational data model also introduced high-level query languages that provided an alternative to programming language interfaces, making it much faster to write new queries.

History of database applications

Providing Data Abstraction and Application Flexibility with Relational Databases

- **Data abstraction and program-data independence** were much improved when compared to earlier systems.
- **Early relational** database management systems (RDBMS) introduced in the early 1980s were quite **slow**.
- With the **development** of new storage and indexing techniques and better query processing and optimization, their **performance improved**.
- Eventually, relational databases **became the dominant** type of database system for traditional database applications.

History of database applications

Object-Oriented Applications and the Need for More Complex Databases

- The **emergence of object-oriented programming** languages in the 1980s and the need to store and share complex, structured objects led to the development of object-oriented databases (OODBs).
- The complexity of the model and the **lack of an early standard** contributed to their limited use. They are now mainly used in specialized applications, such as engineering design, multimedia publishing, and manufacturing systems.

History of database applications

Object-Oriented Applications and the Need for More Complex Databases

- Many object-oriented concepts were incorporated into the newer versions of relational DBMSs, leading to **object-relational database management systems**, known as ORDBMSs.

History of database applications

Interchanging Data on the Web for E-Commerce Using XML

- The **World Wide Web** provides a large network of interconnected computers.
- **Electronic commerce** (e-commerce) emerged as a major application on the Web.
- Much of the critical information on e-commerce Web pages is dynamically extracted data from DBMSs.
- A variety of techniques were developed to allow the interchange of dynamically extracted data on the Web for display on Web pages.
- The **eXtended Markup Language (XML)** is one standard for interchanging data among various types of databases and Web pages.

History of database applications

Extending Database Capabilities for New Applications

- The success of database systems in traditional applications encouraged developers of **other types of applications** to attempt to use them.
- Such applications traditionally used their own **specialized software and file and data structures.**
- Examples are:
  - Scientific applications, Storage and retrieval of images, Storage and retrieval of videos, Data mining applications, Spatial applications, Time series applications.

History of database applications

Extending Database Capabilities for New Applications

- It was apparent that **basic relational systems were not very suitable** for many of these applications, usually for one or more of the following reasons:
- **More complex data structures** were needed for modeling the application than the simple relational representation.
- **New data types** were needed in addition to the basic numeric and character string types.
- **New operations and query language** constructs were necessary to manipulate the new data types.
- **New storage and indexing structures** were needed for efficient searching on the new data types.

History of database applications

Emergence of Big Data Storage Systems and NOSQL Databases

- Proliferation of applications and platforms, web search indexes, and cloud storage/backup led to a **surge in the amount of data** stored on large databases and massive servers.
- **New types of database systems** were necessary to manage these huge databases, systems that would **provide fast search** and **retrieval** as well as reliable and safe storage.
- Some of the requirements of these new systems were **not compatible** with SQL relational DBMSs.

History of database applications

Emergence of Big Data Storage Systems and NOSQL Databases

- The term NOSQL is generally interpreted as **Not Only SQL**.
- Some of the data is stored using SQL systems, whereas other data would be stored using NOSQL, depending on the application requirements.

Data Models, Schemas, and Instances

- **Data abstraction**: Refers to the **suppression of details** of data organization and storage and the **highlighting of the essential features** for an improved understanding of data.
- **Data Model**: A set of concepts to describe
  - the structure of a database,
  - the operations for manipulating these structures,
  - and certain constraints that the database should obey.

Data Models, Schemas, and Instances

Types of Data Models:

1. High Level or Conceptual data models.
2. Low Level or Physical data models.
3. Representational or Implementation data models.

Data Models, Schemas, and Instances

High-Level or Conceptual Data Models:

- These models focus on the **overall structure and organization** of data without going into the specifics of how data is stored physically.
- They provide a conceptual framework for understanding the **entities, attributes, relationships, and constraints** within a system.
- Examples include **Entity-Relationship (ER)** models and Unified Modeling Language (**UML**) diagrams.

Data Models, Schemas, and Instances

Low-Level or Physical Data Models:

- These models deal with the **physical storage aspects** of data, including details such as file structures, indexing methods, and access paths.
- They are closely **tied to the implementation** of the database on a specific database management system (DBMS).
- Examples include **Relational Data Model** (relational tables, indexes) and **Object-Oriented Data Model** (object classes, inheritance).

Data Models, Schemas, and Instances

Representational or Implementation Data Models:

- These models **bridge the gap** between high-level conceptual models and low-level physical models.
- They represent how the **conceptual model will be implemented** in a specific database management system.
- Representational models often include elements such as **data types, storage structures, and access methods.**
- Examples include the **relational model with specific SQL** implementations (e.g., MySQL, PostgreSQL) and **object-relational mapping** (ORM) frameworks.

Data Models, Schemas, and Instances

- **Database Schema**
  - The description of a database.
  - Includes descriptions of the database structure, data types, and the constraints on the database.
- **Schema Diagram**
  - An illustrative display of (most aspects of) a database schema.
- **Schema Construct**
  - A component of the schema or an object within the schema, e.g., STUDENT, COURSE.

Data Models, Schemas, and Instances

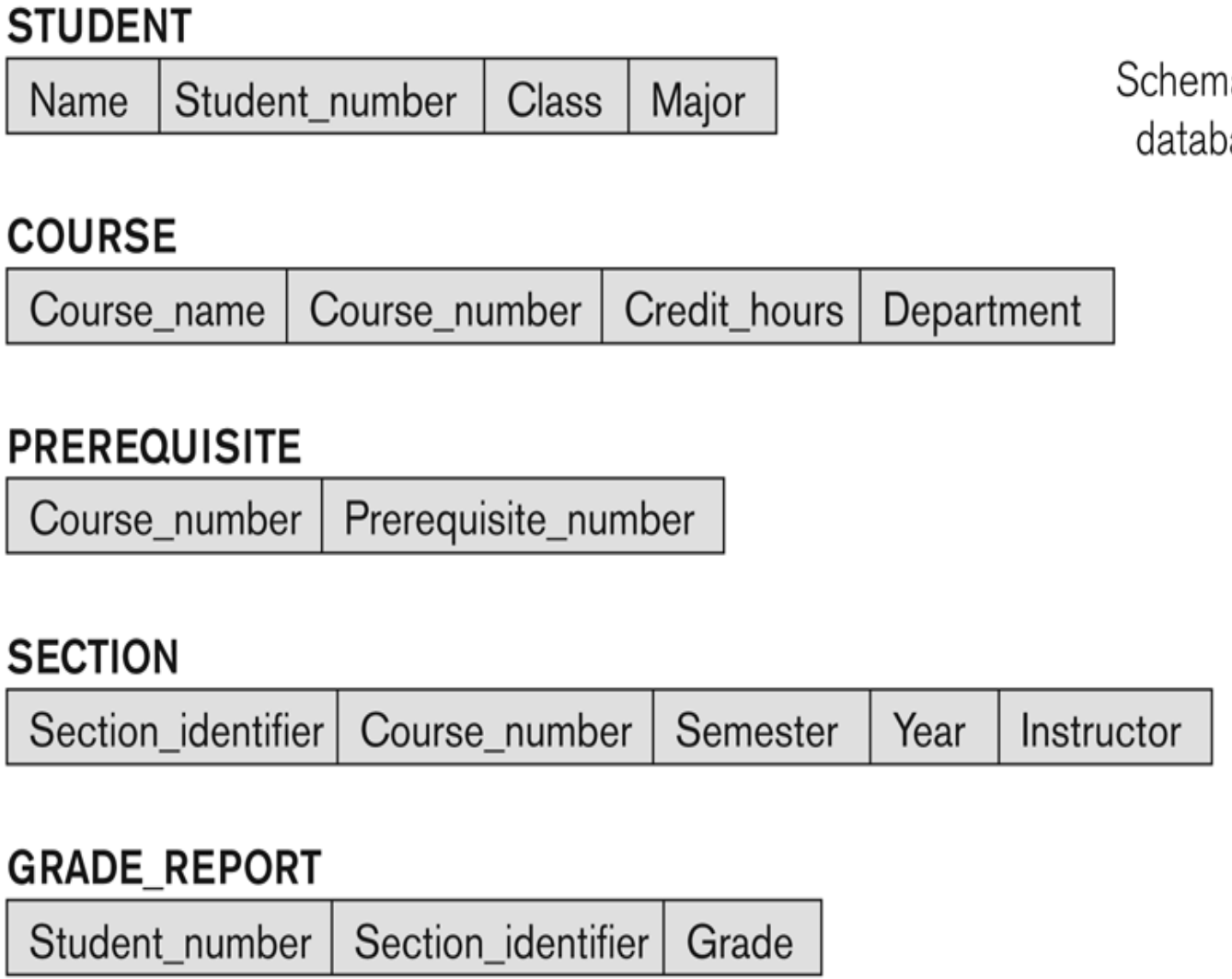


Figure 2.1
Schema diagram for the database in Figure 1.2.

Data Models, Schemas, and Instances

Database State

- The **actual data stored** in a database at a particular moment in time.
- This includes the collection of **all the data** in the database.
- Also called database **instance** (or **occurrence** or **snapshot**).
- The term instance is also **applied to individual database components**, e.g. record instance, table instance, entity instance.

Data Models, Schemas, and Instances

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

Data Models, Schemas, and Instances

- **Distinction** between database **schema** & database **state**
  - The database schema changes very infrequently.
  - The database state changes every time the database is updated.
- The DBMS is partly responsible for ensuring that every state of the database is a **valid state**—that is, a state that satisfies the structure and constraints specified in the schema.
- Hence, specifying a correct schema to the DBMS is extremely important and the schema must be designed with utmost care.

Data Models, Schemas, and Instances

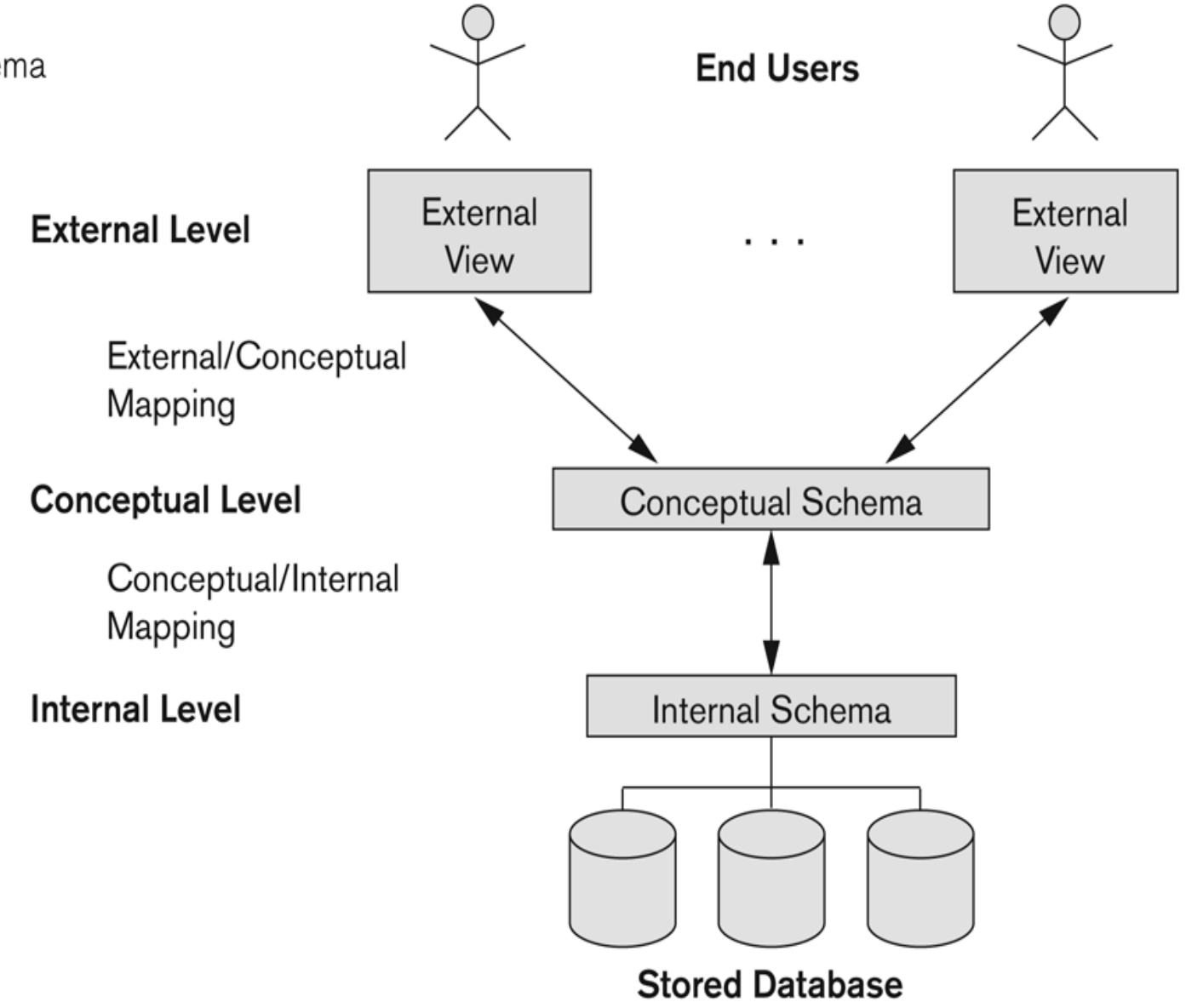
- The DBMS stores the descriptions of the schema constructs and constraints—also called the **meta-data**—in the DBMS catalog so that DBMS software can refer to the schema whenever it needs to.
- **Schema** is also called **intension**.
- **State** is also called **extension**.

Three schema architecture and data independence

- **Proposed to support DBMS characteristics** of:
  - Program-data independence.
  - Support of multiple views of the data.
  - Use of catalog to store the DB description
- Its goal is to **separate** the user **applications** and the physical **database**.
- Not explicitly used in commercial DBMS products but has been useful in explaining database system organization.

Three schema architecture and data independence

Figure 2.2
The three-schema architecture.



Three schema architecture and data independence

Defines DBMS schemas at three levels:

- **Internal schema** at the internal level to describe physical storage structures.
  - Lowest level of abstraction.
  - Uses physical data model.
  - Describes the complete details of data storage and access paths for the database.

Three schema architecture and data independence

- **Conceptual schema** at the conceptual level to describe the structure and constraints for the whole database for a community of users.
  - Hides the details of physical storage structures.
  - Concentrates on describing entities, data types, relationships, user operations and constraints.
  - Uses an implementation/Representational data model to describe conceptual schema.

Three schema architecture and data independence

- **External schemas** at the external level (or view level) to describe the various user views.
  - Each External schemas describes the part of the DB that a particular user group is interested in and hides the rest of the DB from that user group.
  - Uses representational model to implement external schema.
- The three-schema architecture is a convenient tool with which the user can visualize the schema levels in a DB system.

Three schema architecture and data independence

- **Most DBMSs do not separate the three levels explicitly** but support the three-schema architecture to some extent.
- In most DBMSs **external schemas** are specified in the same data model that describes the **conceptual–level** information. Ex: Oracle-SQL.
- Some DBMSs allow different data model for external and Conceptual level.
- Ex: Universal Data Base –IBMs DBMS
  - Uses Relational model for Conceptual schema
  - May use Object-oriented model for external schema

Three schema architecture and data independence

- Mappings: The processes of **transforming requests and results** between levels is called mappings.
- Programs refer to an external schema and are mapped by the DBMS to the internal schema for execution.
- Data extracted from the internal DBMS level is reformatted to match the user's external view (e.g. formatting the results of an SQL query for display in a Web page)

Three schema architecture and data independence

- **Data independence** can be defined as the **capacity to change the schema** at one level without changing the schema at next higher level.
- There are two types of data Independence. They are:

1. Logical data independence.
2. Physical data independence.

Three schema architecture and data independence

Logical data independence

- Is the capacity to change the **conceptual schema** without having to change the **external schema** or application programs.
- Only the **view definition and mapping** need to be changed in a DBMS that support logical data independence.
- After the conceptual schema undergoes a logical reorganization, application programs that reference the external schema constructs must work as before.

Three schema architecture and data independence

Physical data independence:

- Is the capacity to change the **internal schema** without changing the **conceptual schema**.
- Hence, the external schemas need not be changed as well.
- Modification at the physical level are occasionally necessary to **improve performance**.

Three schema architecture and data independence

- By creating **additional access structures**—to improve the performance of retrieval or update.
- If the same data as before remains in the database, we should not have to change the conceptual schema.
- For example, providing an access path to improve retrieval speed of section records.

Three schema architecture and data independence

- **Logical data independence is more difficult** to achieve than physical data independence, because it allows structural and constraint changes without affecting application programs
- When a schema at a lower level is changed, **only the mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence.

Database languages and interfaces

Types of **DBMS languages**:

- Data Definition Language (**DDL**)
- Data Manipulation Language (**DML**)
- Data Control Language (**DCL**)

Database languages and interfaces

Data Definition Language (DDL)

- Used by the DBA and database designers to specify the conceptual schema of a database.
- In many DBMSs, the **DDL** is also used to define internal and external schemas (views).
- In some DBMSs, separate **storage definition language** (SDL) and **view definition language** (VDL) are used to define internal and external schemas.
- **Ex: Create, Alter, Drop, Truncate**

Database languages and interfaces

Data Manipulation Language (DML)

- Used to specify database **retrievals and updates**.
- DML commands **can be embedded in a general-purpose programming language** (host language), such as COBOL, C, C++, or Java.
- Alternatively, stand-alone DML commands can be applied directly (called a query language).
- **Ex: Insert, Delete, Select, Update.**

Database languages and interfaces

Data Control Language (DCL)

- The DCL language is used for **controlling the access** to the table and hence securing the database.
- DCL is used **to provide certain privileges** to a particular user. Privileges are rights to be allocated.
- The various privileges that can be **granted or revoked** are: Select, Insert, Delete, Update, References, Execute All.

Database languages and interfaces

Some of the DCL commands

- COMMIT - save work done
- SAVEPOINT - identify a point in a transaction to which you can later roll back
- ROLLBACK - restore database to original since the last COMMIT
- SET TRANSACTION - Change transaction options like what rollback segment to us.

Database languages and interfaces

Database Interfaces

- User friendly interfaces provided by DBMS may include the following:
- **Menu-based, for web clients or browsing :**
  - Present the user with list of options (called menus) that lead the user through the formulations of a request.
  - No need to memorize the specific commands & syntax of a query language.
  - Pull-down menus are a very popular technique in web-based user interfaces.
  - Used in browsing interfaces, which allow a user to look through the content of DB in an exploratory & unstructured manner.

Database languages and interfaces

Form-based interfaces:

- Displays a form to each user.
- User can fill out all of the form entries to insert data.
- Usually designed and programmed for naive users as interfaces to canned transactions.
- Many DBMSs have form specification languages, which are special languages that help programmers specify such forms.
- Ex: SQL forms, Oracle forms, etc.,

Database languages and interfaces

- **Graphical User Interfaces(GUI):**
  - Displays a schema to the user in diagrammatic form. Then user can specify a query by manipulating the diagram.
  - In many cases GUI utilize both menu & forms.
  - Most GUIs use a pointing devices such as a mouse, to pick certain parts of the displayed schema diagram.

Database languages and interfaces

- **Natural language interface**
  - Accept request written in English or some other language and attempt to understand them.
  - Usually has its own schema, which is similar to the DB conceptual schema, as well as a dictionary of important words.
  - Refers to the words in its schema, as well as to the set of standard words in its dictionary, to interpret the request.
  - The capabilities of natural language interfaces have not advanced rapidly.
  - Ex: Search engine that accept strings of natural language.

Database languages and interfaces

- **Speech input & output:**
  - Uses speech as an input query & speech as an answer to a question or result of a request.
  - Ex: enquires for telephone directory, flight arrival/departure, back account information
  - Speech i/p is detected using a library of predefined words and used to set up the parameters that are supplied to the queries. For o/p, a similar conversion from text or numbers into speech takes place.

Database languages and interfaces

- **Interfaces for parametric users:**
  - System analysts and programmers design and implement a special interfaces for each class of naive users. Ex: bank tellers
  - Usually, a small set of abbreviated commands is included, with the goal of minimizing the no. of keystrokes required for each request.
  - Ex: Function keys in a terminal can be programmed to initiate various commands.

Database languages and interfaces

- **Interfaces for the DBA:**
  - Most database systems contain privileged commands that can be used only by DBA's staff.
  - Ex: Commands for creating accounts, setting system parameters, granting a/c authorization, changing a schema and reorganizing the storage structure of database.

The Database System environment

- A DBMS is a complex software system. In this section we discuss the types of **s/w components** that constitute a DBMS and the types of computer system s/w with which the DBMS interacts.
- **DBMS Component Modules:** The following Figure illustrates the typical DBMS components in a simplified form. Figure is divided into two halves.
- Top half - refers various users of DB & their interfaces
- Lower half- components responsible for storage of data and processing of transactions.

The Database System environment

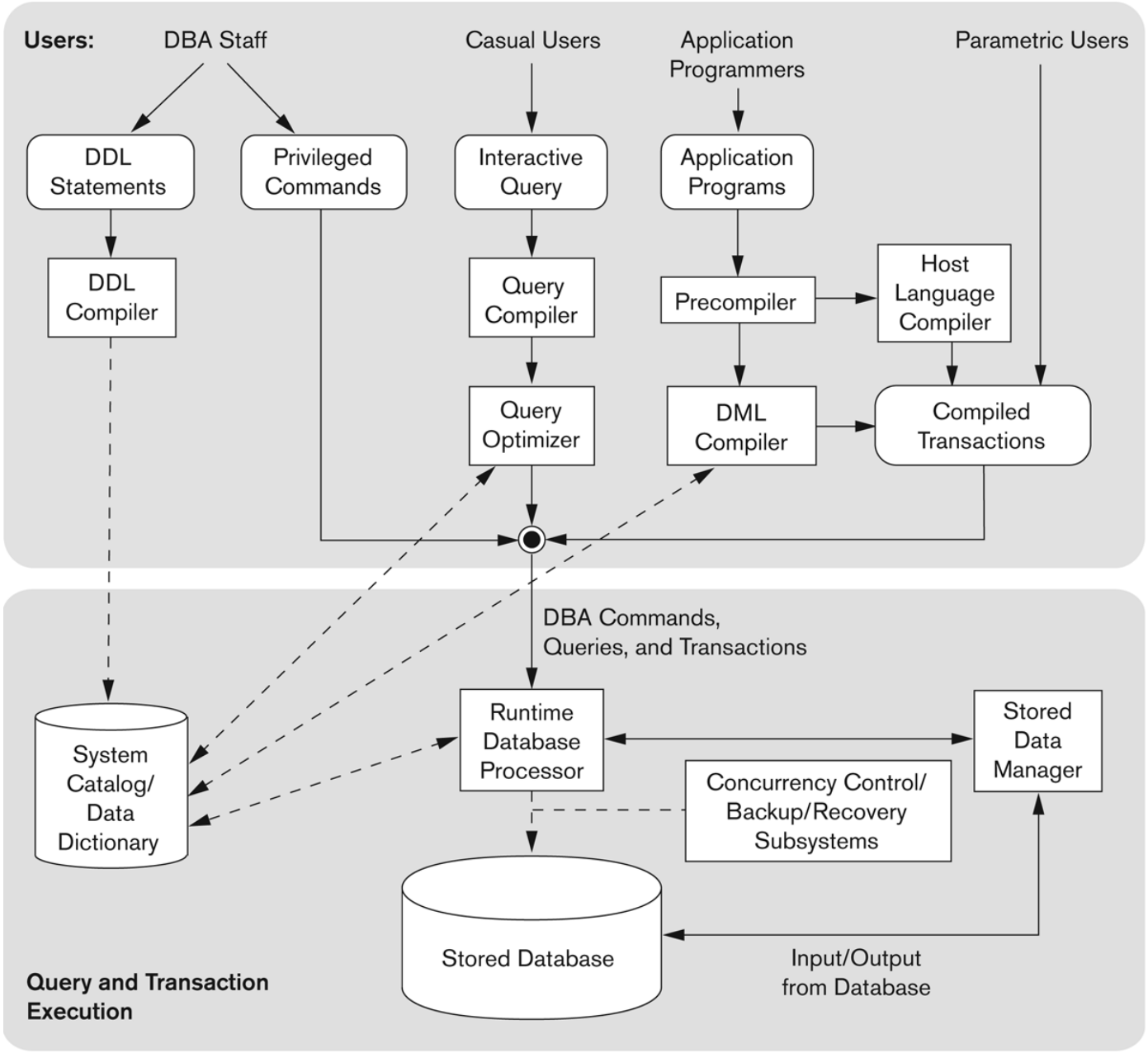


Figure 2.3
Component modules of a DBMS and their interactions.

ER Introduction

- Overview of Database Design Process

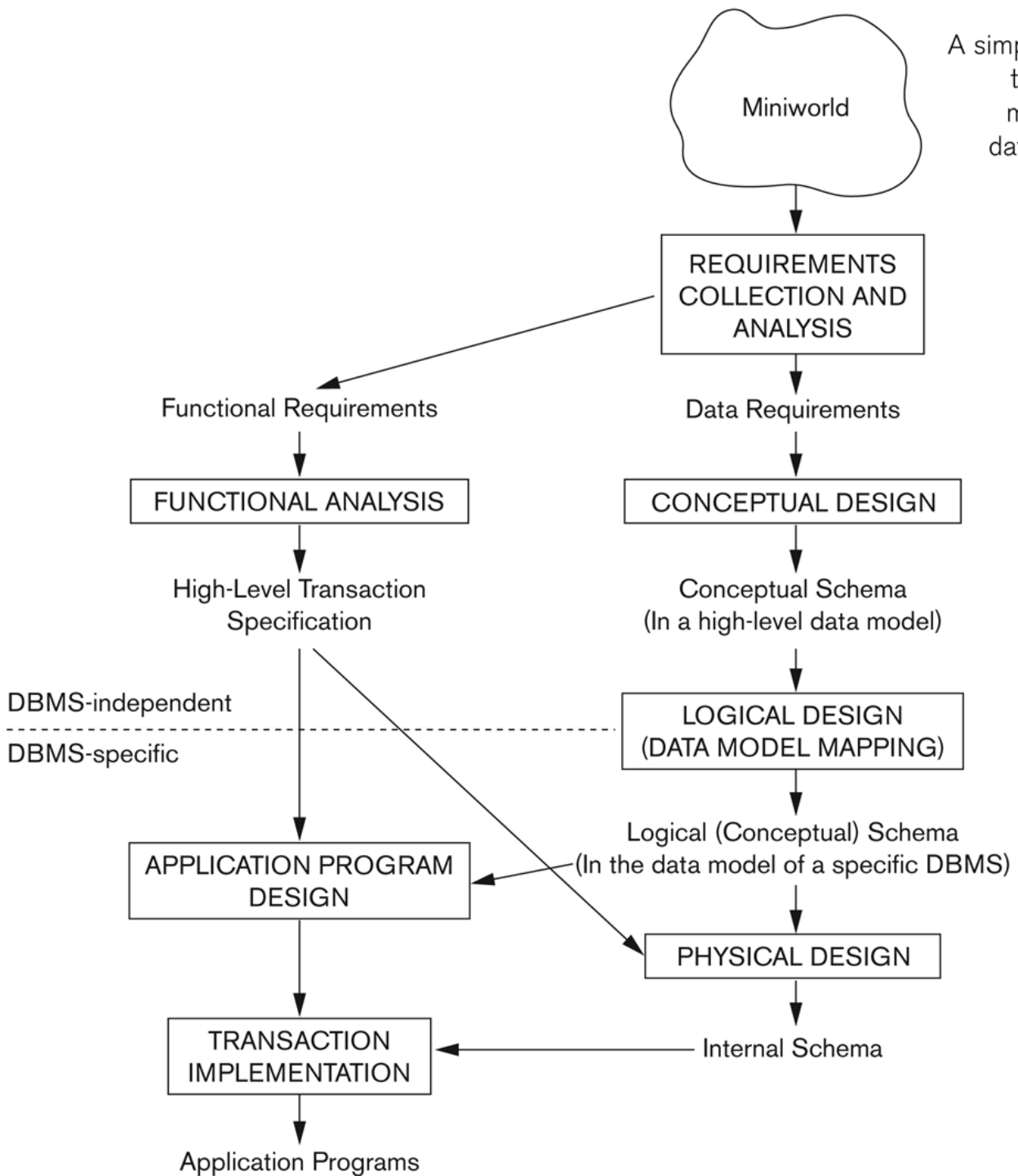


Figure 3.1
A simplified diagram to illustrate the main phases of database design.

ER Example

- Example: **Company Database Design**
- We need to create a database schema design based on the following (simplified) **requirements** of the COMPANY Database:
  - The company is organized into DEPARTMENTs. Each department has a name, number and an employee who *manages* the department. We keep track of the start date of the department manager. A department may have several locations.
  - Each department *controls* a number of PROJECTs. Each project has a unique name, unique number and is located at a single location.

ER Example

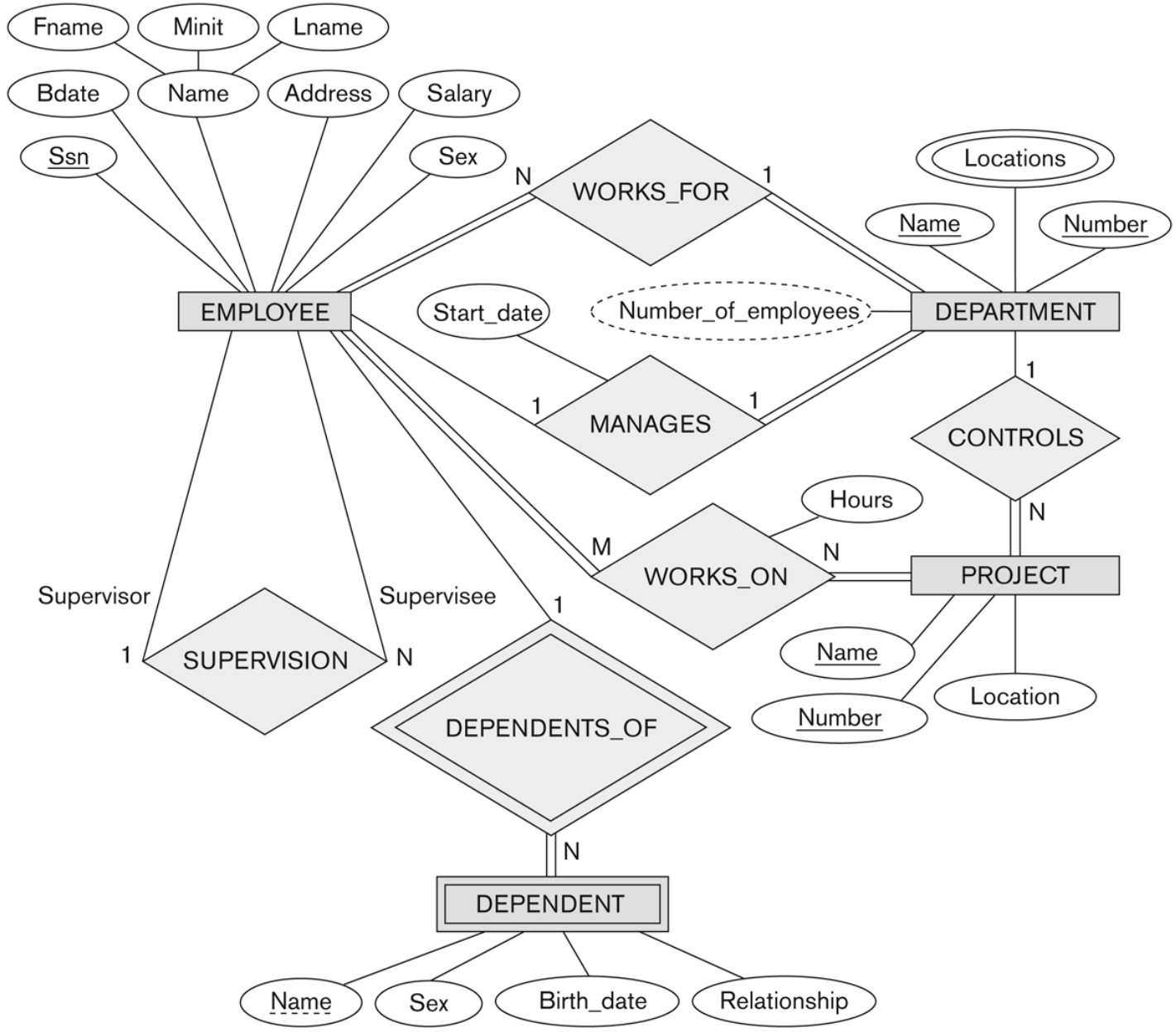


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Entity, Attributes and Relationship

- **Entity**
  - Anything that may have an independent existence and about which we intend to collect data.
  - Ex: Person, car, house or employee.
- **Attributes**
  - Properties / characteristics that describe entities
  - The set of possible values for an attribute is called the domain of the attribute.
  - Ex:- The domain of attribute marital status is just the four values: single, married, divorced, widowed
  - For month - 12 values ranging from Jan to Dec
- **Key attribute**
  - The attribute (or combination of attributes) that is unique for every entity instance
  - Ex:- the account number of an account, the employee id of an employee etc.

Entity, Attributes and Relationship

Types of Attributes

1. Simple Vs Composite attribute

- Simple attribute: cannot be divided into simpler components
- Ex: Emp ID
- Composite attribute: can be split into components
- Ex: Address, Name

Entity, Attributes and Relationship

2. Single Vs Multi-valued Attributes

- Single valued : can take on only a single value for each entity instance.
- Ex: Age
- Multi-valued: can take many values.
- Ex: Color, Degree

Entity, Attributes and Relationship

3. Stored Vs Derived attribute

- Stored Attribute: Attribute that need to be stored permanently.
- Ex: name of an employee, Birth date
- Derived Attribute: Attribute that can be calculated based on other attributes
- Ex: Age

Entity, Attributes and Relationship

NULL values

- Used when particular entity may not have an
  - Applicable value
  - Do not know the value
  - Value is missing

Entity, Attributes and Relationship

Complex Attributes

- Arbitrary Nesting of Composite () and Multivalued attributes { }
- Ex: If a person can have more than one residence and each residence can have a single address and multiple phones, an attribute Address_phone for a person can be specified as
- Address_phone ({Phone (Area_code, Phone_number)}, Address (Street_address (Number, Street, Apartment_number), City, State, Zip)) }

Entity, Attributes and Relationship

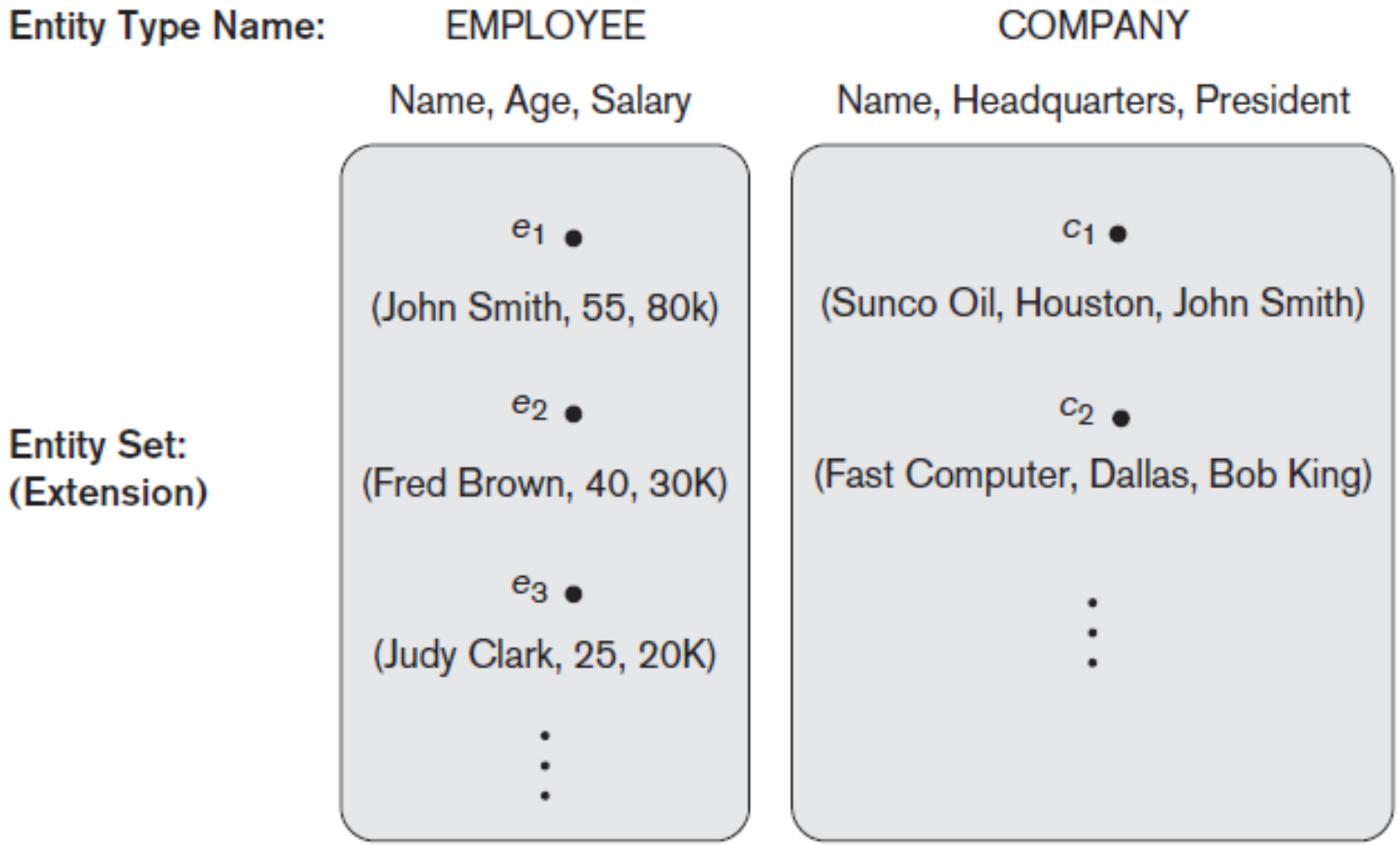
Entity type

- A collection of entities that have the same attributes.
- Ex: employee, student, etc.,

Entity set

- Is a collection of entities of a same type that share the same properties or attributes.
- The entity set is usually referred to using the same name as the entity type.
- Ex: Employee – refers to both entity type as well as the current set of all employee entities in the DB.

Entity, Attributes and Relationship



ER Diagrams

- Based on the requirements, we can identify four initial entity types in the COMPANY database:
  - DEPARTMENT
  - PROJECT
  - EMPLOYEE
  - DEPENDENT
- Their initial design is shown on the following slide
- The initial attributes shown are derived from the requirements description

ER Diagrams

- We need to create a database schema design based on the following (simplified) requirements of the COMPANY Database:
  - The company is organized into DEPARTMENTs. Each department has a name, number and an employee who manages the department. We keep track of the start date of the department manager. A department may have several locations.
  - Each department controls a number of PROJECTs. Each project has a unique name, unique number and is located at a single location.

ER Diagrams

- We store each EMPLOYEE's social security number, address, salary, sex, and birthdate.
  - Each employee *works for* one department but may *work on* several projects.
  - We keep track of the number of hours per week that an employee currently works on each project.
  - We also keep track of the *direct supervisor* of each employee.
  - Each employee may *have* a number of DEPENDENTs.
- For each dependent, we keep track of their name, sex, birthdate, and relationship to the employee.

ER Diagrams

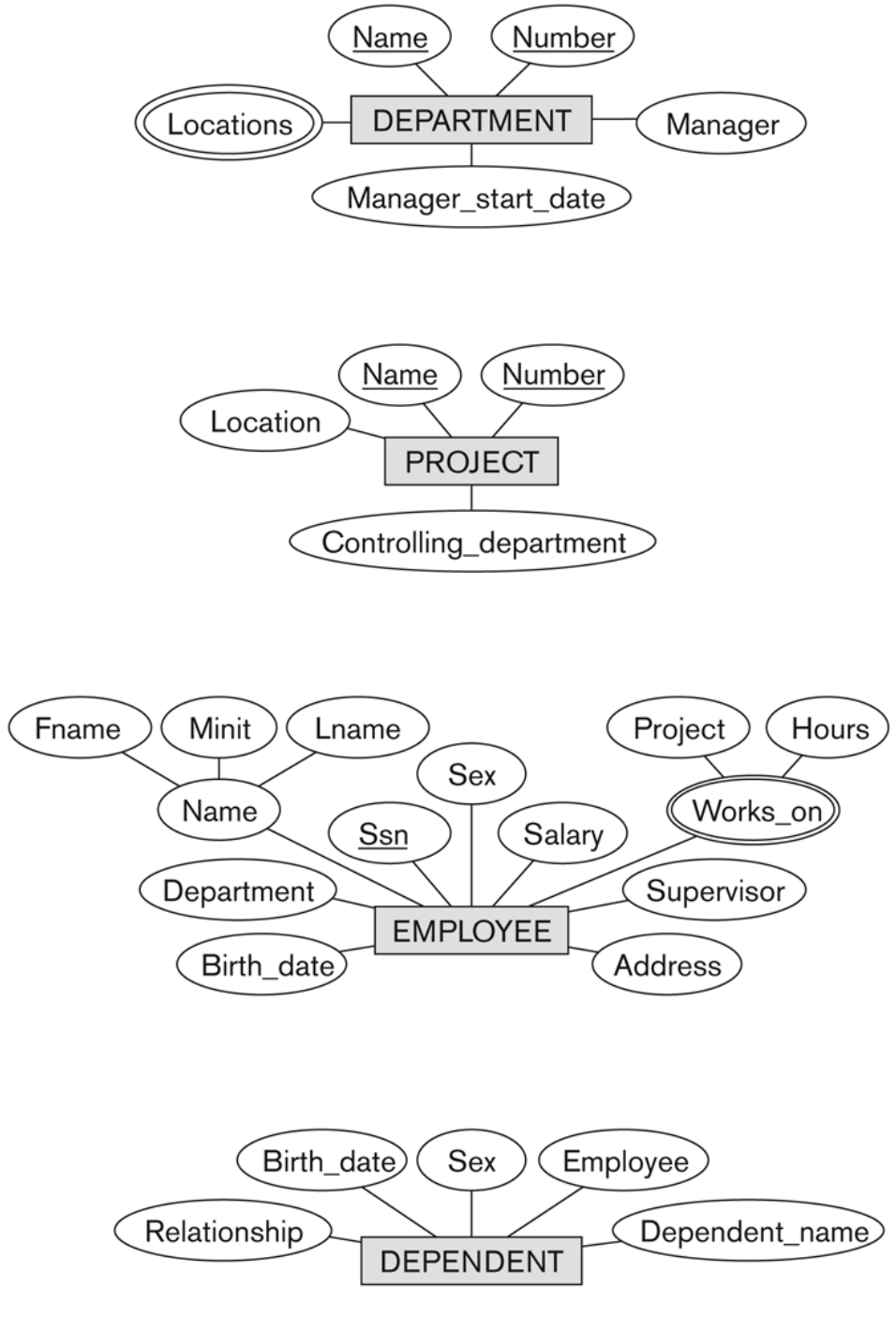


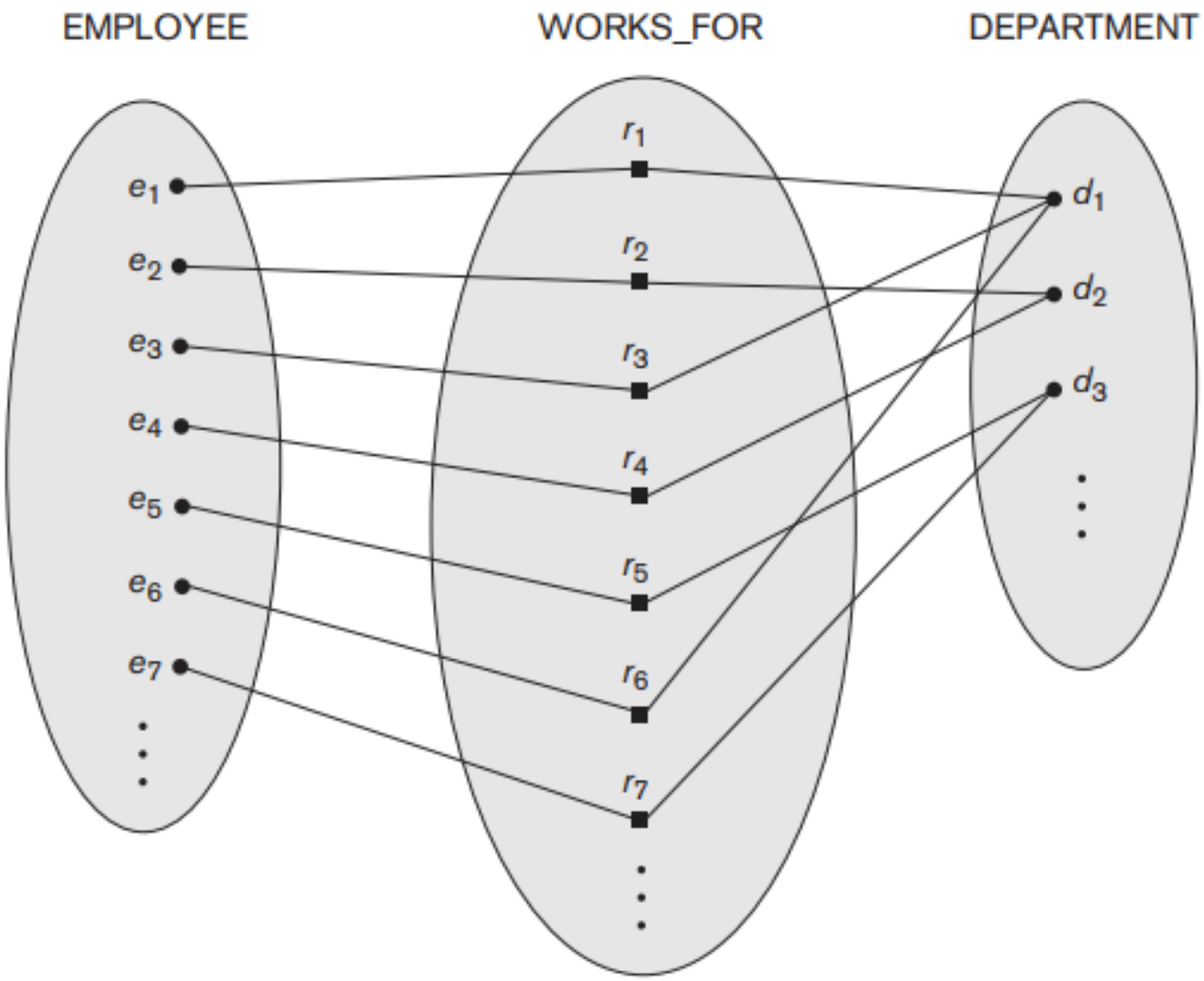
Figure 3.8
Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

Relationship

- **Relationship**
  - Association between several entities
  - Ex: Employee – worksfor – Department
- **Relationship type**
  - A relationship type R among n entity types E1, E2,…..En defines a set of association among entities from these entity types.
- **Relationship set**
  - Is a set of relationships of the same type.

Relationship

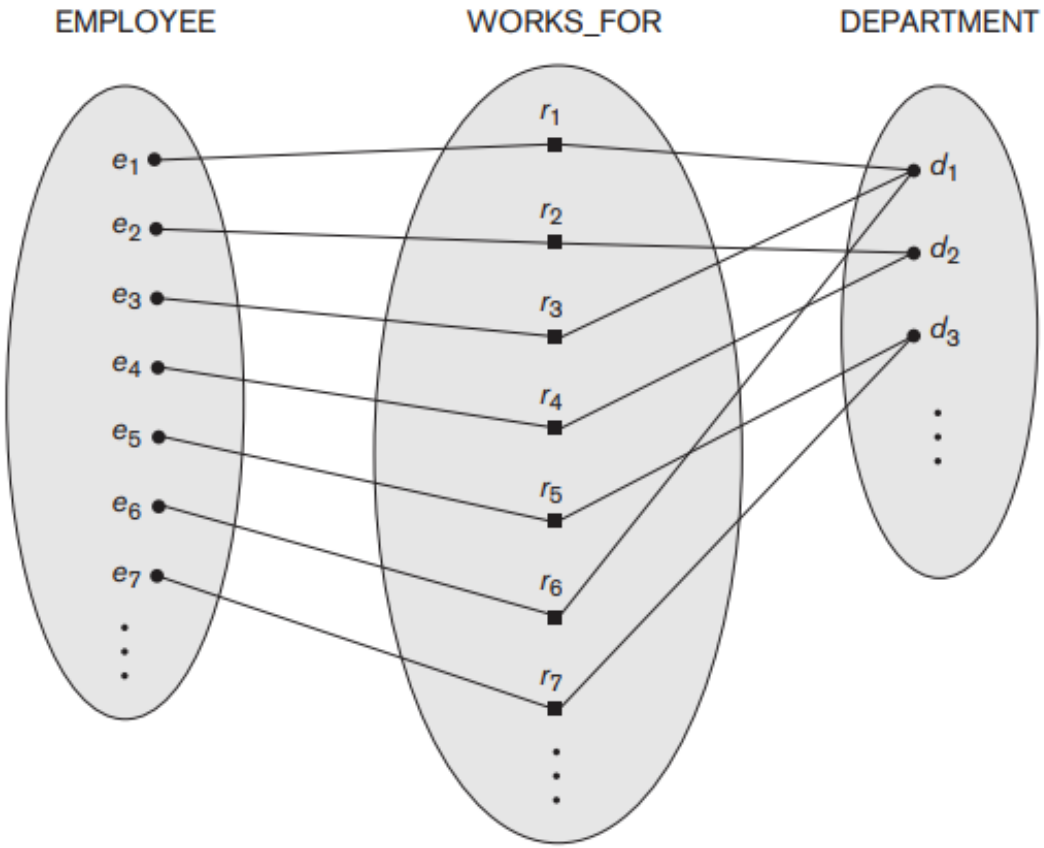
EMPLOYEE **works_for** DEPARTMENT



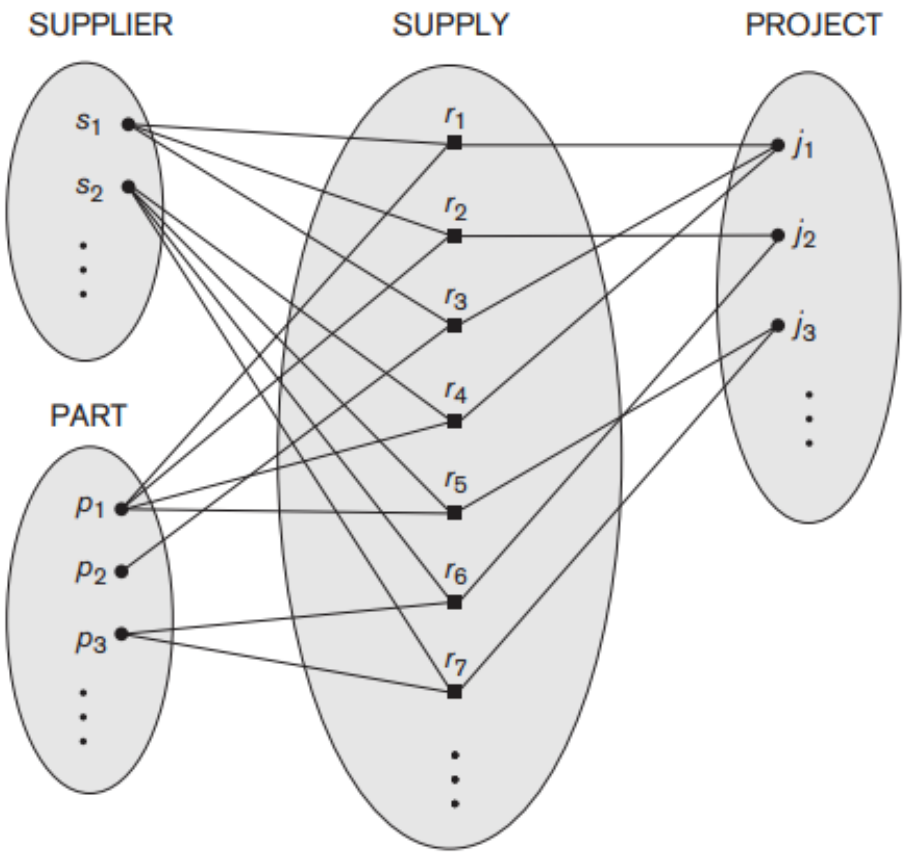
Relationship

- **Relationship instance**
  - Associates n individual entities and each entity in relationship is member of entity type.
  - Ex: If works_for is the relationship between the Employee entity and the Department entity, then ‘Ram works for CS Department’, ‘Shyam works for electrical Department’ etc., are relationship instances of the relationship works_for.
- **Degree of a Relationship**
  - Is the number of entity types involved in a relationship.
- Types
  - One Entity - Unary
  - Two Entities - Binary
  - Three Entities - Ternary

Relationship



Binary Relationship



Ternary Relationship

Relationship

Role Names and Recursive Relationships

Roles Names

- A role name specifies what role an entity type plays in a specific relationship
- Role names are sometimes used in ER- diagrams to clarify the roles of the participating entity types.

Recursive Relationships

- In a relationship where same entity type participate more than once in a relationship type in different roles
- Ex: the SUPERVISION relationship
- EMPLOYEE participates twice in two distinct roles:
  - supervisor (or boss) role
  - supervisee (or subordinate) role

Relationship

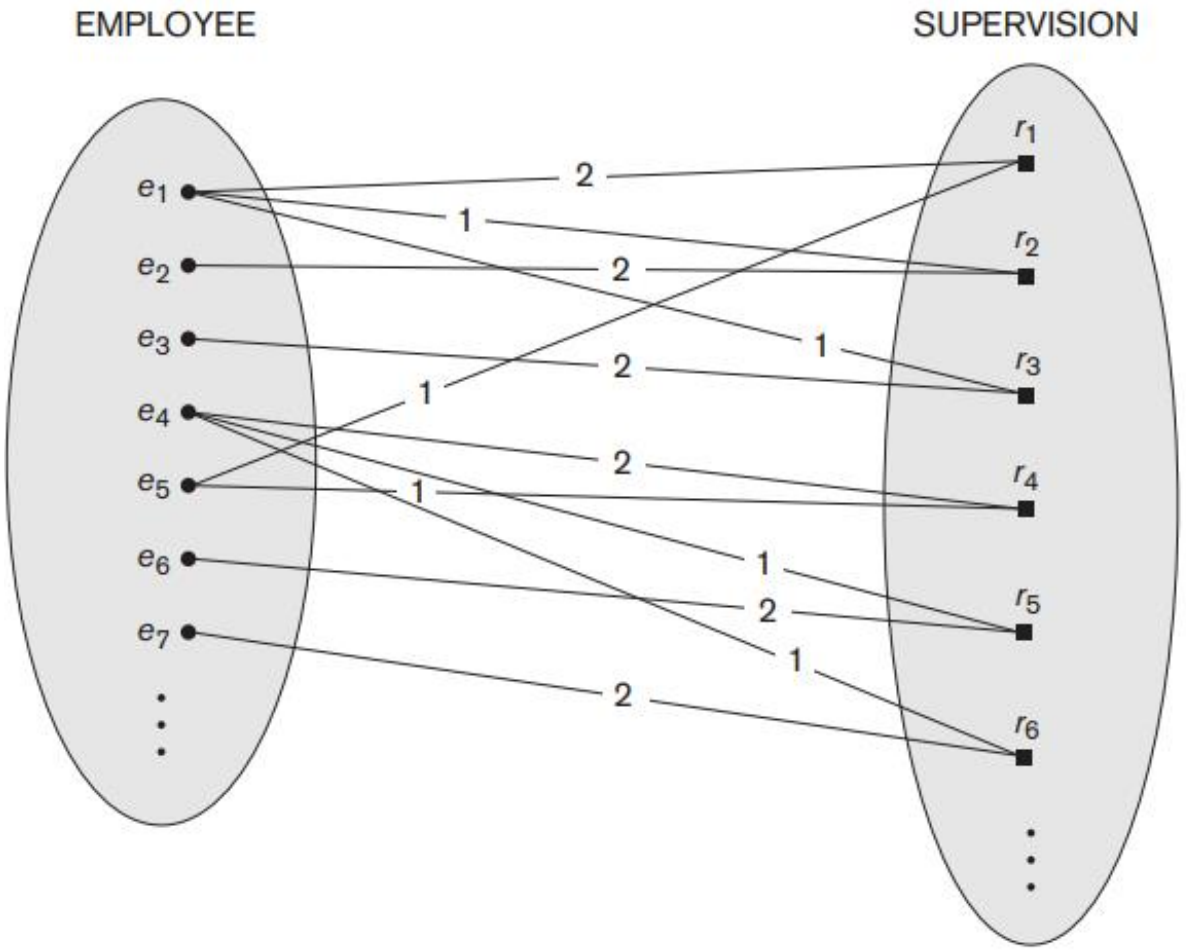


Figure 3.11
A recursive relationship SUPERVISION between EMPLOYEE in the *supervisor* role (1) and EMPLOYEE in the *subordinate* role (2).

Relationship

Constraints on Binary Relationship Types

- Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set.
- Two main types:
  - Cardinality ratio
  - Participation constraints

Relationship

Cardinality Ratios for Binary Relationships

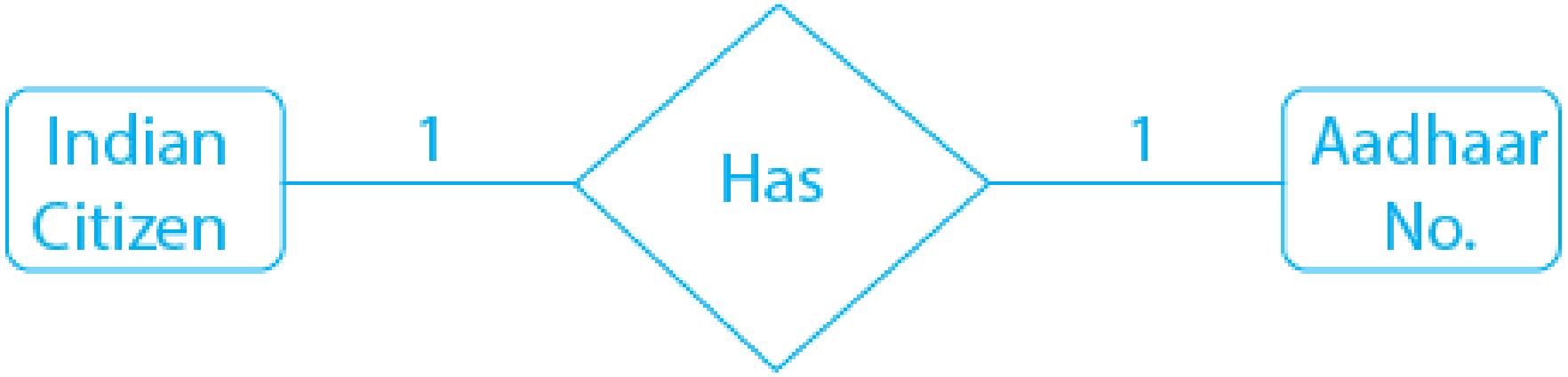
- Specifies the maximum number of relationship instances that an entity can participate in.
- Possible Cardinality ratios for binary relationship types are :
  - one-to-one (1:1)
  - one-to-many (1:N)
  - many-to-many (M:N)

Ex: employee head-of department (1:1), student enrolls course (m:n), lecturer offers course (1:n) assuming a course is taught by a single lecturer

Relationship

One-to-One Relationship (1:1)

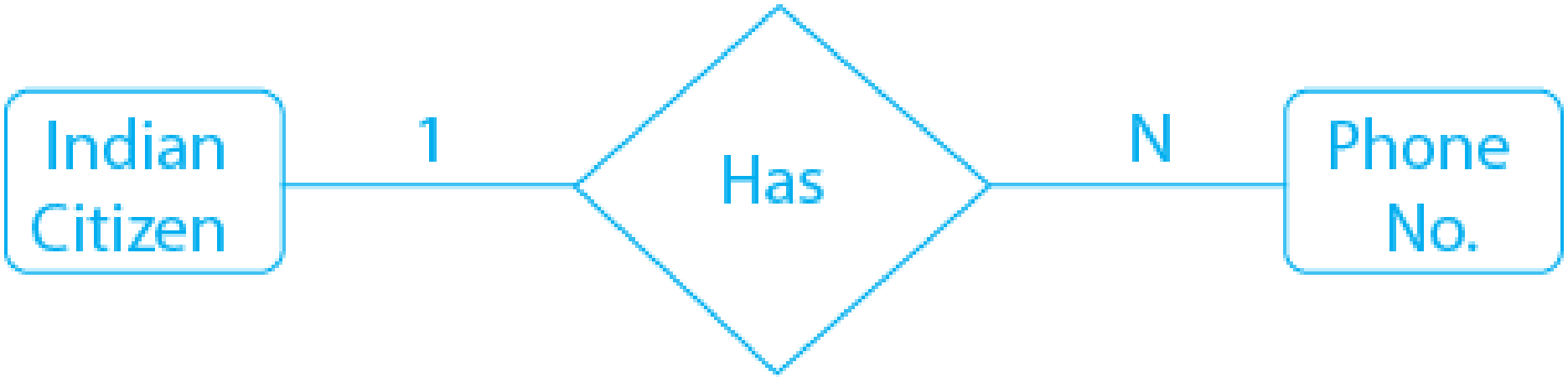
- This relationship shows that every Indian citizen has exactly 1 Aadhaar Number and, an Aadhaar Number is unique and can belong to exactly one citizen.

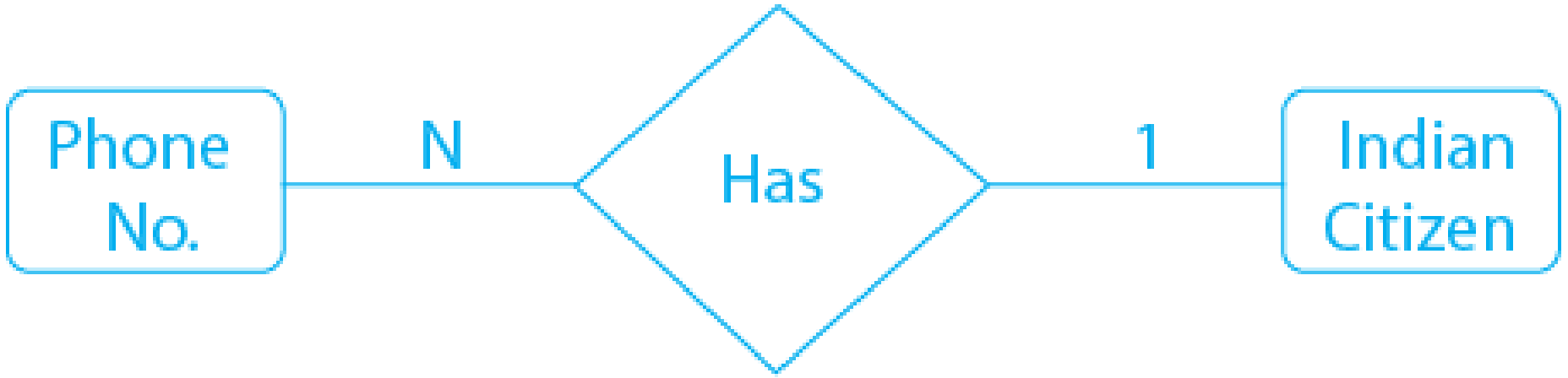


Relationship

One-to-Many Relationship (1:N)

- This means that an entity of the first entity set can map to at most N entities of the second entity set.
- However, an entity of the second entity set can map to at most 1 entity of the first entity set.

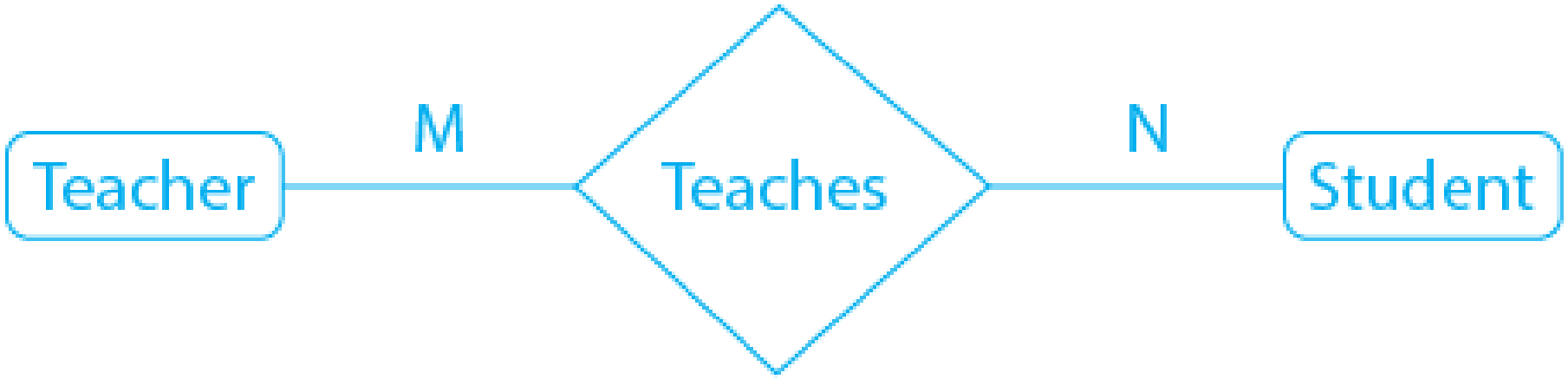




Relationship

Many-to-One Relationship (N:1)

- This is the exact opposite of a one-to-many relationship. So, we can take the previous example and reverse it. It will become a many-to-one relationship.



Relationship

Many-to-Many Relationship (M:N)

- This means that the entity of the first entity set can map to at most M entities of the second entity set and the entity of the second entity set can map to at most N entities of the first entity set.

Relationship

Participation Constraints and Existence Dependence

- Specifies whether the existence of an entity depends on its being related to another entity via the relationship type.
- Specifies the minimum number of relationship instances that each entity can participate in and is sometimes called the minimum cardinality constraints.
- Two types:
  - Total participation
  - Partial participation

Relationship

Total Participation

- Every entity instance must be connected through the relationship to another instance of the other participating entity types.
- Here the participation of Employee in works_for is called total participation.
- Means, every entity in the total set of employee entities must be related to a department entity via Works_for.
- Total participation is also called as existence dependency.

Relationship

Partial participation

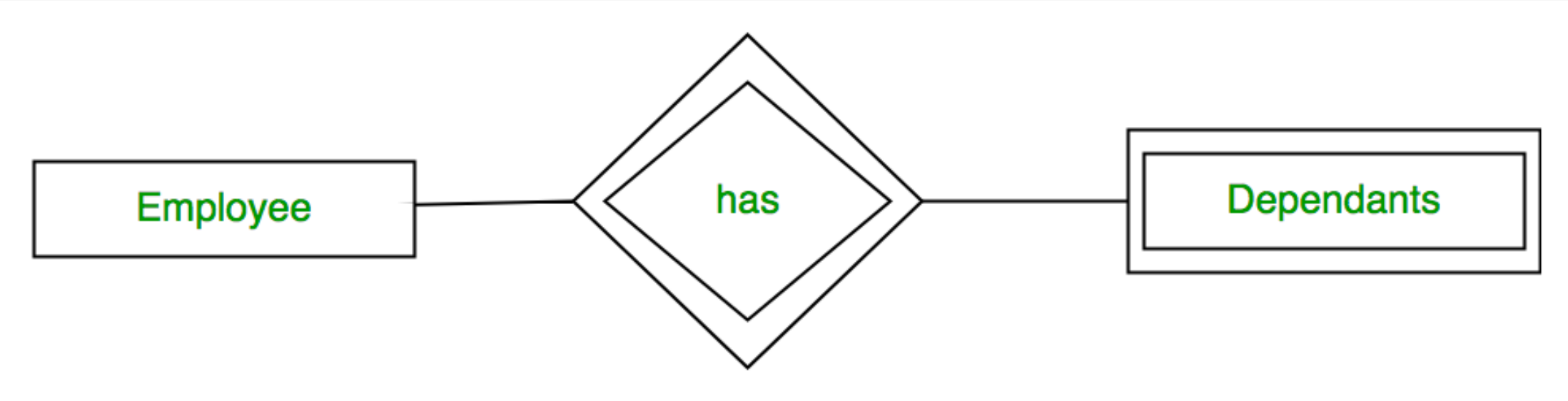
- Is where the entity can exist without participating in a relationship with another entity.
- For example, the entity course may exist within an organization, even though it has no current students.
- The cardinality ratio and participation constraints, taken together, as the **structural constraints** of a relationship type.

Relationship

- Double line between the entity set “Student” and relationship set “Enrolled in” signifies total participation.
- It specifies that each student must be enrolled in at least one course.
- Single line between the entity set “Course” and relationship set “Enrolled in” signifies partial participation.
- It specifies that there might exist some courses for which no enrollments are made.

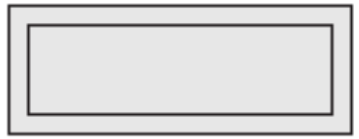
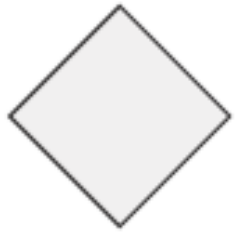
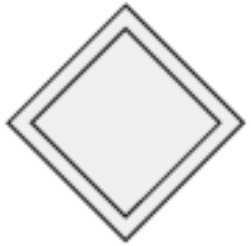
Weak entity types

- Entity that depends on other entity for its existence and doesn’t have key attribute of its own.
- Ex: DEPENDENTS of EMPLOYEE
- The relationship type that relates a weak entity type to its owner is called identifying relationship of the weak entity type.
- Has a partial key, which is the set of attributes that can uniquely identify weak entities that are related to the same owner entity.

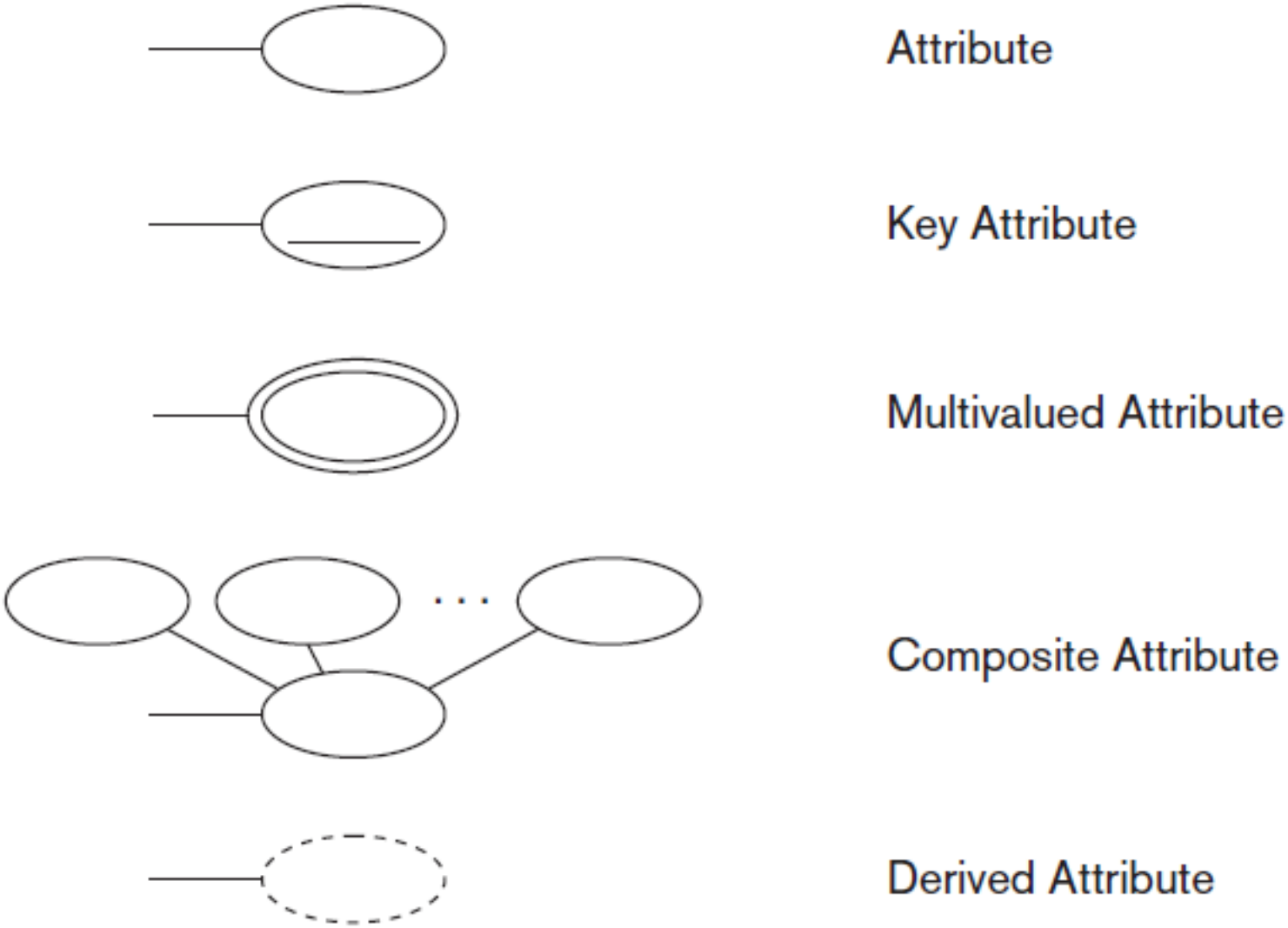


Weak entity types

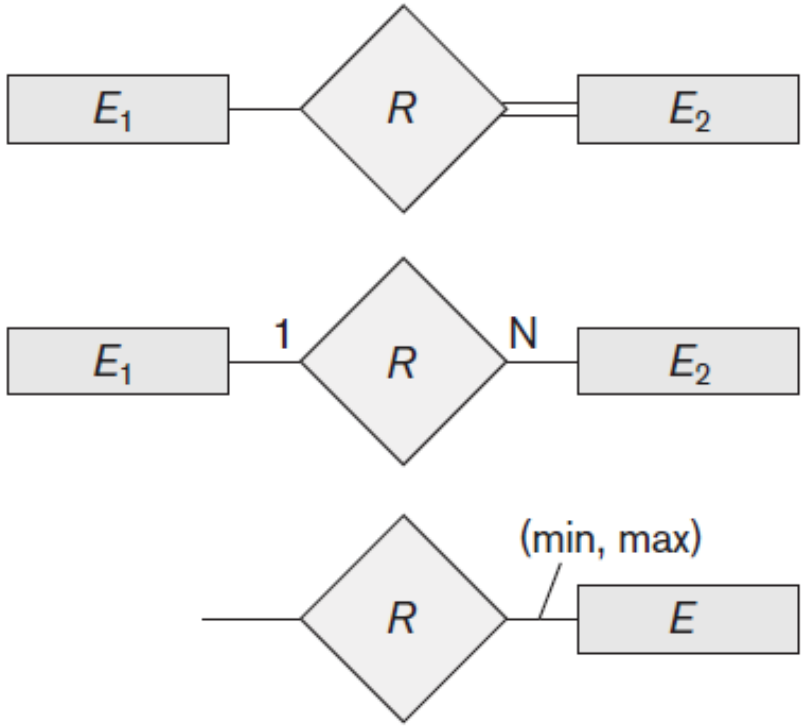
ER Notations

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Indentifying Relationship

ER Notations



ER Notations



Entity-Relationship Diagram

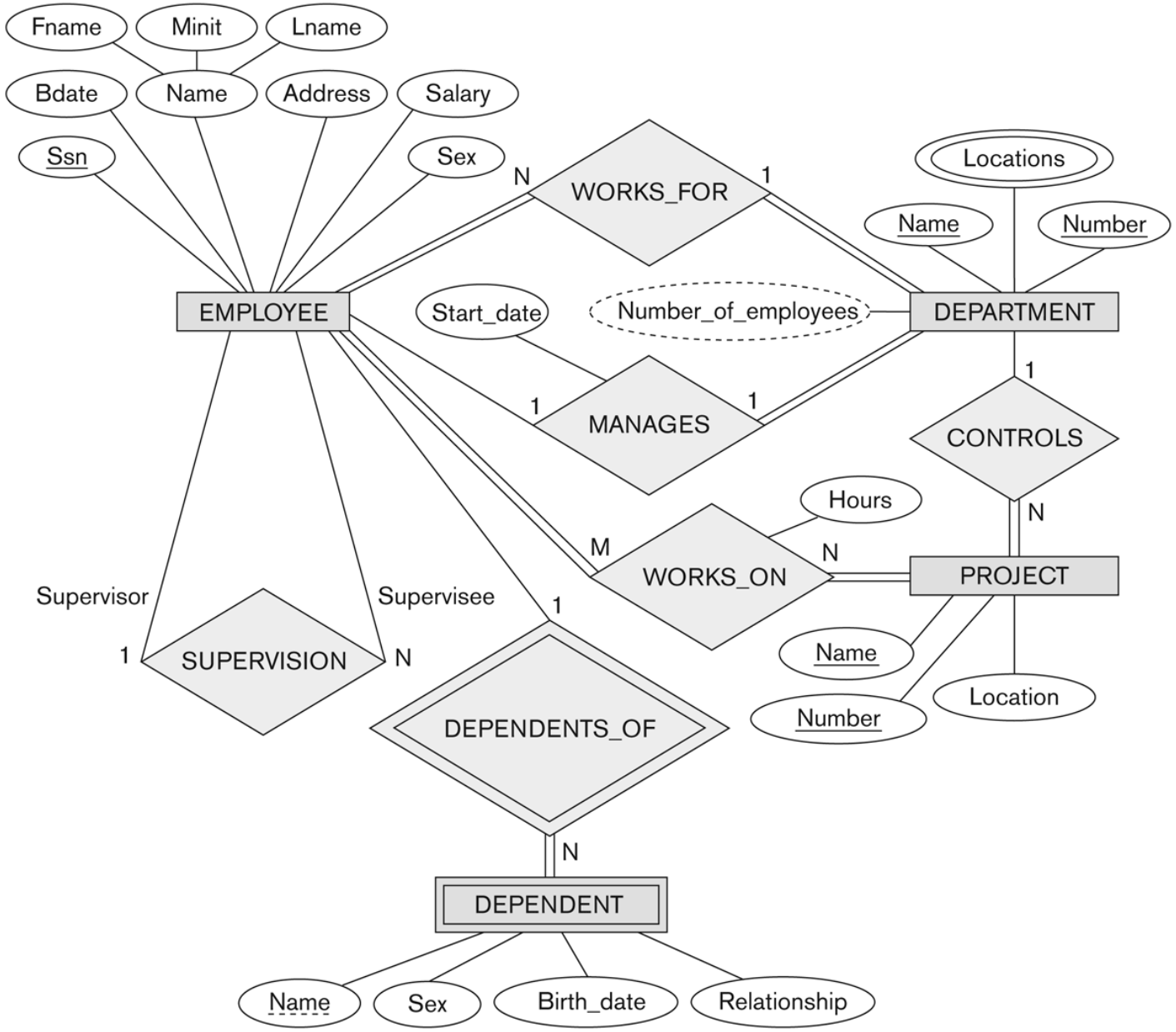


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Specialization and Generalization

- **Specialization** is the process of defining a set of subclasses of an entity type; this entity type is called the superclass of the specialization.
- The set of subclasses that forms a specialization is defined on the basis of some distinguishing characteristic of the entities in the superclass.
- For example, the set of subclasses {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of the superclass EMPLOYEE that distinguishes among employee entities based on the job type of each employee.

Specialization and Generalization

- We may have several specializations of the same entity type based on different distinguishing characteristics.
- For example, another specialization of the EMPLOYEE entity type may yield the set of subclasses {SALARIED_EMPLOYEE, HOURLY_EMPLOYEE}; this specialization distinguishes among employees based on the method of pay.

Specialization and Generalization

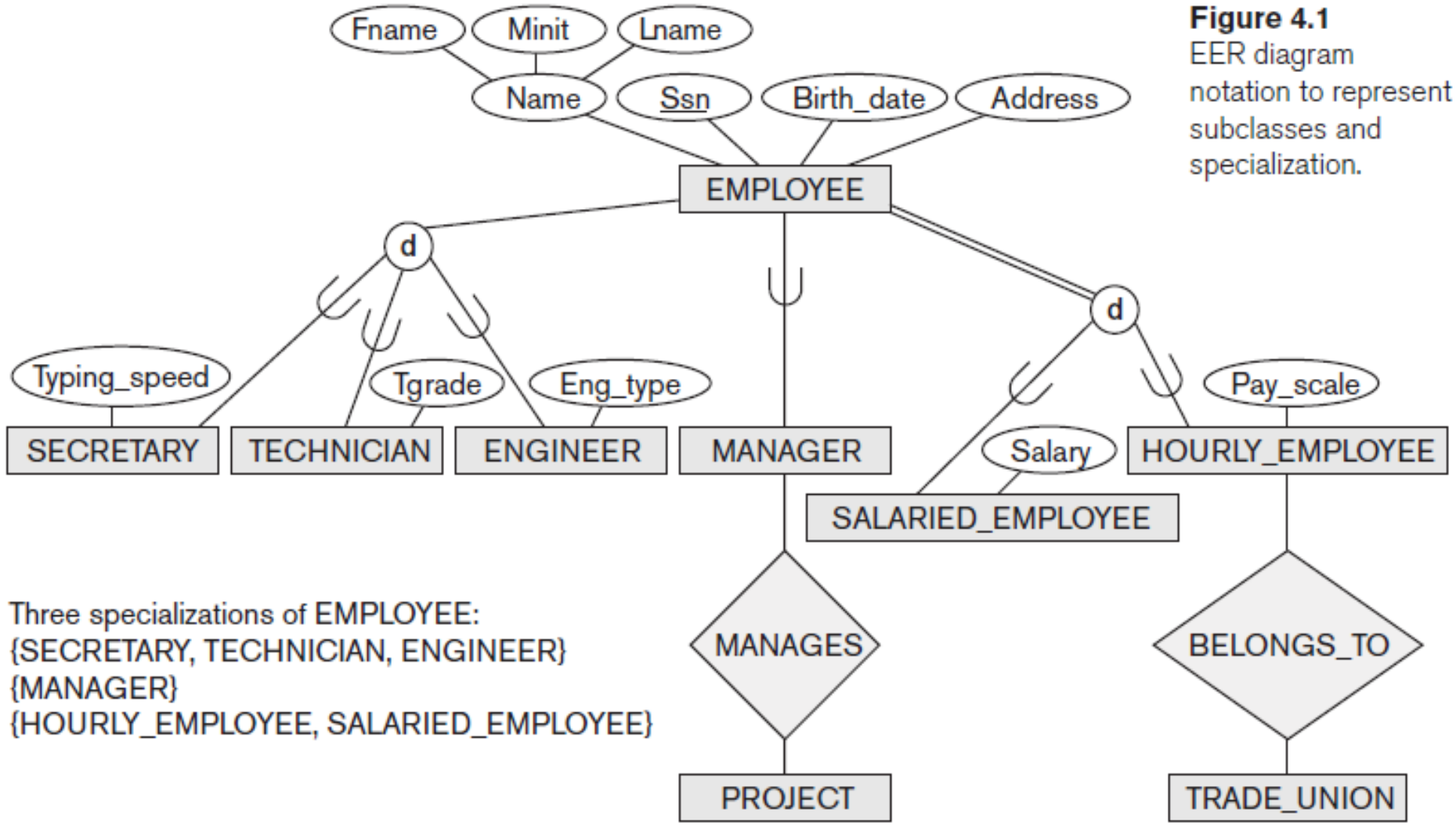
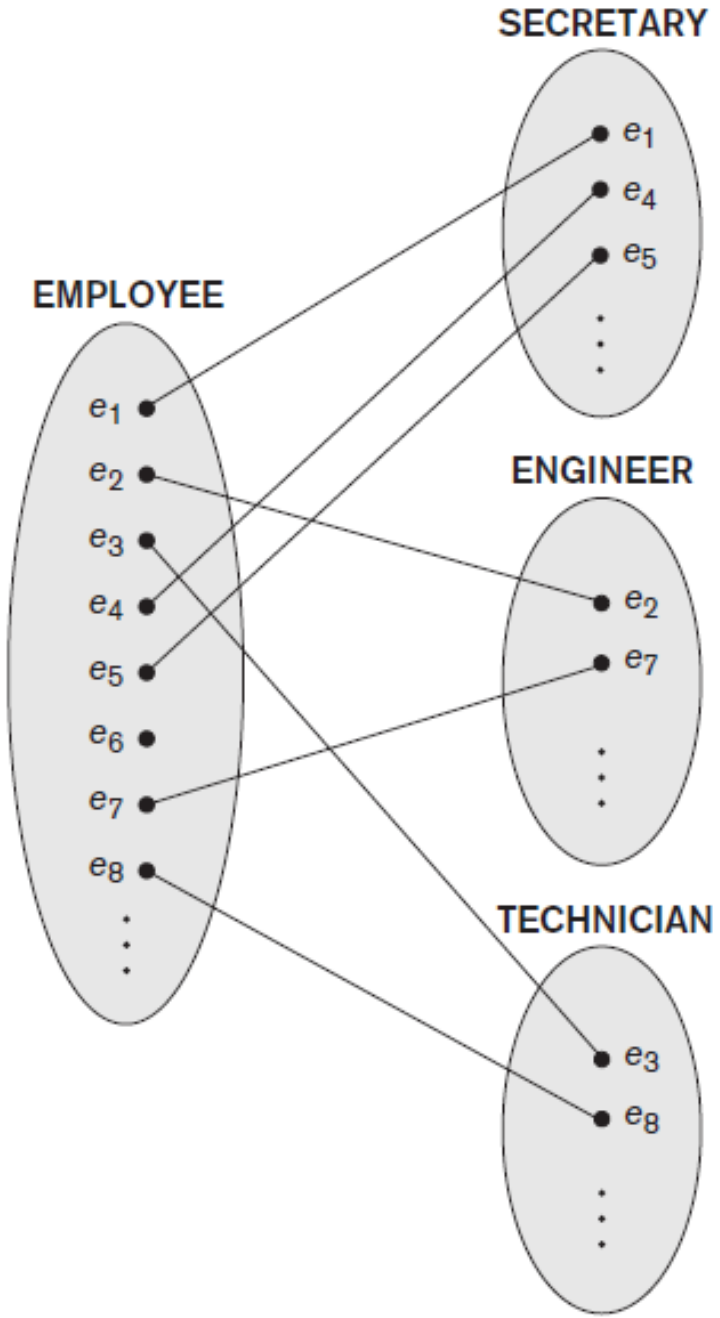


Figure 4.1
EER diagram notation to represent subclasses and specialization.

Specialization and Generalization



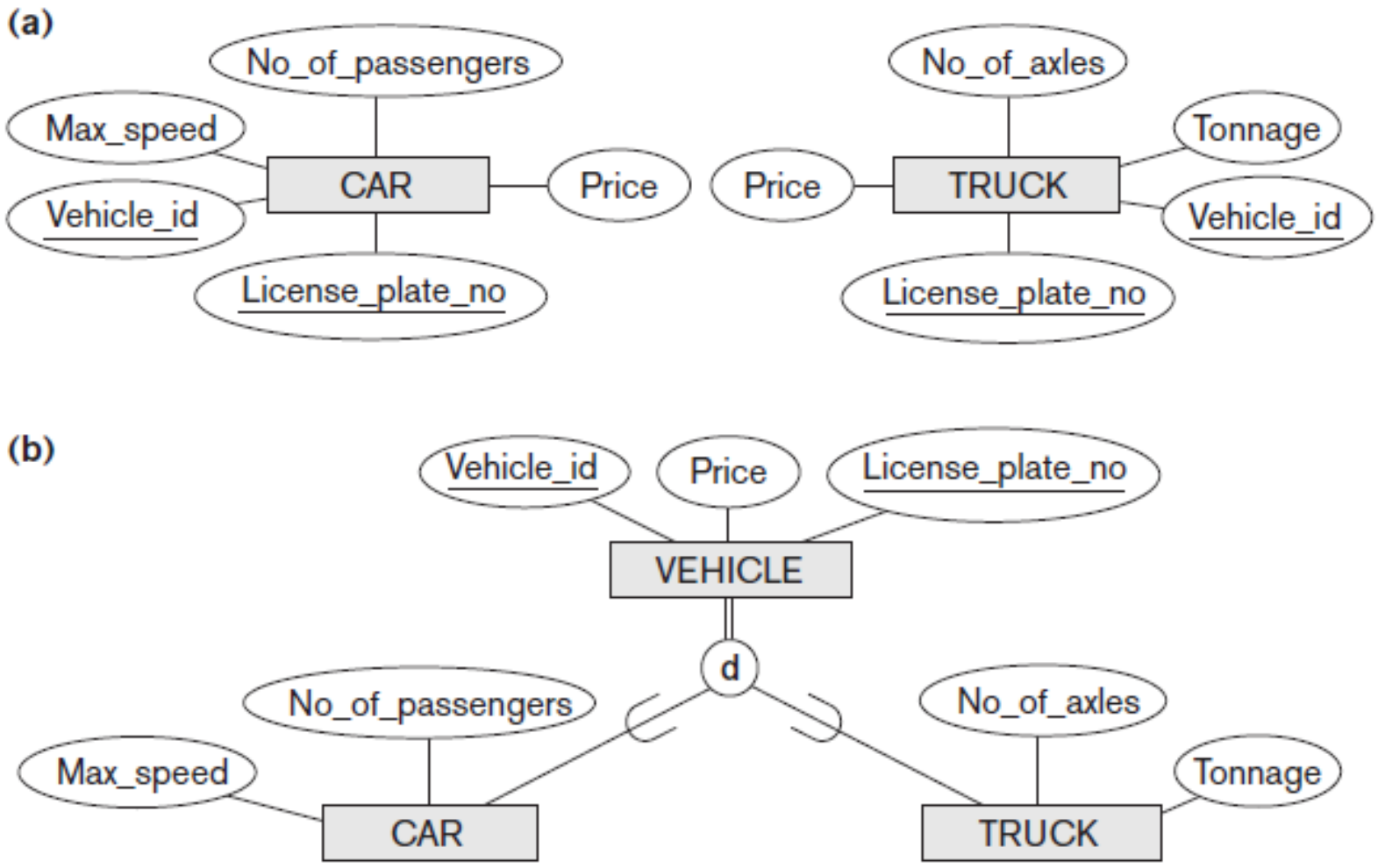
Specialization and Generalization

- There are two main reasons for including class/subclass relationships and specializations.
- The first is that certain attributes may apply to some but not all entities of the superclass entity type.
- A subclass is defined in order to group the entities to which these attributes apply.
- The members of the subclass may still share the majority of their attributes with the other members of the superclass.
- The second reason for using subclasses is that some relationship types may be participated in only by entities that are members of the subclass.

Specialization and Generalization

- **Generalization:** Think of a reverse process of abstraction in which we suppress the differences among several entity types, identify their common features, and generalize them into a single superclass of which the original entity types are special subclasses.
- For example, consider the entity types CAR and TRUCK shown in Figure.
- Because they have several common attributes, they can be generalized into the entity type VEHICLE.
- Both CAR and TRUCK are now subclasses of the generalized superclass VEHICLE. We use the term generalization to refer to the process of defining a generalized entity type from the given entity types.

Specialization and Generalization



End of Module 1