

# **MODULE – 1**

## **INTRODUCTION TO DATABASES**

### **OVERVIEW OF DATABASE LANGUAGES AND ARCHITECTURES**

### **CONCEPTUAL DATA MODELING USING ENTITIES AND RELATIONSHIPS**

# Types of Databases and Database Applications

2

- Numeric and Textual Databases
- Multimedia Databases
- Geographic Information Systems (GIS)
- Data Warehouses and Online Analytical Processing (OLAP)
- Real-time and Active Databases

# Basic Definitions

3

- **Database:** A collection of related data.
- **Data:** Known facts that can be recorded and have an implicit meaning.
- **Mini-world or Universe of discourse(UoD):** Some part of the real world about which data is stored in a database. For example, student grades and transcripts at a university.
- **Database Management System (DBMS):** A collection of programs that enables users to create and maintain database. DBMS is a general purpose software system that facilitates the processes of defining, constructing, manipulating and sharing databases among various users and applications.
- **Database System:** The DBMS software together with the database. Sometimes, the applications are also included.

# Typical DBMS Functionality

4

- Define a database : In terms of data types, structures and constraints.
- Construct or Load the Database on a secondary storage medium.
- Manipulating the database : Querying, generating reports, insertions, deletions and modifications to its content.
- Concurrent Processing and Sharing by a set of users and programs – yet, keeping all data valid and consistent.

# Typical DBMS Functionality

## Other features:

- ▣ Metadata : The information present in the database in the form of data catalog or data dictionary.
- ▣ Protection or Security measures to prevent unauthorized access
- ▣ “Active” processing to take internal actions on data
- ▣ Presentation and Visualization of data

# Example of a Database (with a Conceptual Data Model)

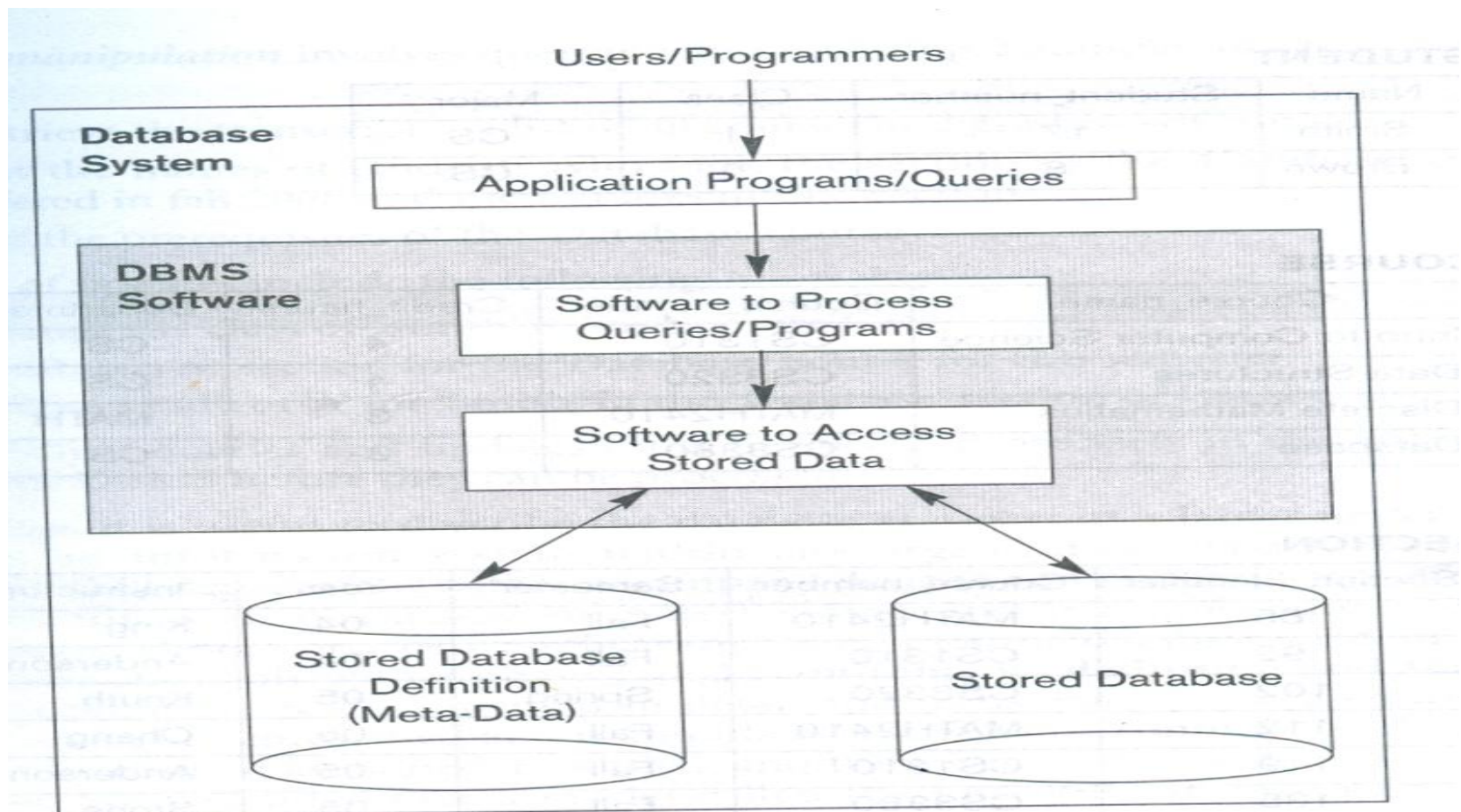
6

- **Mini-world for the example:** Part of a UNIVERSITY environment.
  
- **Some mini-world *entities*:**
  - STUDENTs
  - COURSEs
  - SECTIONs (of COURSEs)
  - (academic) DEPARTMENTs
  - INSTRUCTORs

*Note:* The above could be expressed in the ENTITY-RELATIONSHIP data model.

# A Simplified Database System Environment

7



# Example of a Database (with a Conceptual Data Model)

8

- **Some mini-world *relationships*:**
  - SECTIONs *are of* specific COURSEs
  - STUDENTs *take* SECTIONs
  - COURSEs *have* prerequisite COURSEs
  - INSTRUCTORs *teach* SECTIONs
  - COURSEs *are offered by* DEPARTMENTs
  - STUDENTs *major in* DEPARTMENTs

**Note:** The above could be expressed in the *ENTITY-RELATIONSHIP* data model.



# Example of a Database System

**Figure 1.2** An example of a database that stores student records and their grades.

STUDENT	Name	StudentNumber	Class	Major
	Smith	17	1	CS
	Brown	8	2	CS

COURSE	CourseName	CourseNumber	CreditHours	Department
	Intro to Computer Science	CS1310	4	CS
	Data Structures	CS3320	4	CS
	Discrete Mathematics	MATH2410	3	MATH
	Database	CS3380	3	CS

SECTION	SectionIdentifier	CourseNumber	Semester	Year	Instructor
	85	MATH2410	Fall	98	King
	92	CS1310	Fall	98	Anderson
	102	CS3320	Spring	99	Knuth
	112	MATH2410	Fall	99	Chang
	119	CS1310	Fall	99	Anderson
	135	CS3380	Fall	99	Stone

GRADE_REPORT	StudentNumber	SectionIdentifier	Grade
	17	112	B
	17	119	C
	8	85	A
	8	92	A
	8	102	B
	8	135	A

PREREQUISITE	CourseNumber	PrerequisiteNumber
	CS3380	CS3320
	CS3380	MATH2410
	CS3320	CS1310

# Main Characteristics of the Database Approach

10

- Self-describing nature of a database system:
  - A DBMS **catalog** stores the *description* of the database. The description is called **meta-data**. This allows the DBMS software to work with different databases.
  
- Insulation between programs and data:
  - **Program – data independence**
    - Allows changing data storage structures and operations without having to change the DBMS access programs.
  - **Program – operation independence**
    - User application programs can operate on the data by invoking operations through their names and parameters regardless how operations are implemented.

# Main Characteristics of the Database Approach

11

## □ Data Abstraction:

- The characteristics that allows program – data independence and program – operation independence is called data abstraction.
- A **data model** is used to hide storage details and present the users with a *conceptual view* of the database.

## □ Support of multiple views of the data:

- Each user may see a different view of the database, which describes *only* the data of interest to that user.

# Main Characteristics of the Database Approach

12

- Sharing of data and multiuser transaction processing :
  - Allowing a set of concurrent users to retrieve and to update the database. Concurrency control within the DBMS guarantees that each **transaction** is correctly executed or completely aborted.
  
  - OLTP (Online Transaction Processing) is a major part of database applications.

# Database Users

Users may be divided into those who actually use and control the content (called “Actors on the Scene”) and those who enable the database to be developed and the DBMS software to be designed and implemented (called “Workers Behind the Scene”).

# Database Users

## Actors on the scene

- ❑ **Database administrators:** responsible for authorizing access to the database, for co – ordinating and monitoring its use, acquiring software, and hardware resources, controlling its use and monitoring efficiency of operations.
- ❑ **Database Designers:** responsible to define the content, the structure, the constraints, and functions or transactions against the database. They must communicate with the end-users and understand their needs.
- ❑ **End-users:** they use the data for queries, reports and some of them actually update the database content.
- ❑ **System Analysts and Application Programmers(Software Engineers):**
  - System Analyst determine the requirements of end users, especially Naïve and parametric end users and develop specifications for canned transactions that meet these requirements.
  - Application Programmers implement these specifications as programs, then they test, debug, document and maintain these canned transactions.

# Categories of End-users

15

- **Casual** : access database occasionally when needed
  
- **Naive or Parametric** : they make up a large section of the end-user population. They use previously well-defined functions in the form of “canned transactions” against the database. Examples are bank-tellers or reservation clerks who do this activity for an entire shift of operations.

# Categories of End-users

16

- **Sophisticated** : these include business analysts, scientists, engineers, others thoroughly familiar with the system capabilities. Many use tools in the form of software packages that work closely with the stored database.
- **Stand-alone** : mostly maintain personal databases using ready-to-use packaged applications. An example is a tax program user that creates his or her own internal database.



# Database Users

17

- Workers behind the scene
  - DBMS System Designers and Implementers
  - Tool Developers
  - Operators and Maintenance Personnel

# Advantages of Using the Database Approach

18

- Controlling redundancy in data storage and in development and maintenance efforts.
- Restricting unauthorized access to data.
- Providing persistent storage for program Objects
- Providing Storage Structures for efficient Query Processing

# Advantages of Using the Database Approach

19

- Providing backup and recovery services.
- Providing multiple interfaces to different classes of users.
- Representing complex relationships among data.
- Enforcing integrity constraints on the database.
- Permitting Inferencing and Actions using rules

# Additional Implications of Using the Database Approach

20

- **Potential for enforcing standards:**
  - ▣ This is very crucial for the success of database applications in large organizations Standards refer to data item names, display formats, screens, report structures, meta-data (description of data) etc.
  
- **Reduced application development time:**
  - ▣ Incremental time to add each new application is reduced.

# Additional Implications of Using the Database Approach

21

- **Flexibility to change data structures:**
  - ▣ Database structure may evolve as new requirements are defined.
  
- **Availability of up-to-date information:**
  - ▣ Very important for on-line transaction systems such as airline, hotel, car reservations.
  
- **Economies of scale:**
  - ▣ By consolidating data and applications across departments wasteful overlap of resources and personnel can be avoided.

# Brief History of Database Applications

22

- **Early Database Applications:** The Hierarchical and Network Models were introduced in mid 1960's and dominated during the seventies. A bulk of the worldwide database processing still occurs using these models.
- **Relational Model based Systems:** The model that was originally introduced in 1970 was heavily researched and experimented with in IBM and the universities. Relational DBMS Products emerged in the 1980's.

# Historical Development of Database Technology

23

- **Object-oriented applications:** OODBMSs were introduced in late 1980's and early 1990's to cater to the need of complex data processing in CAD and other applications. Their use has not taken off much.
- **Data on the Web and E-commerce Applications:** Web contains data in HTML (Hypertext markup language) with links among pages. This has given rise to a new set of applications and E-commerce is using new standards like XML (eXtended Markup Language).

# Extending Database Capabilities

24

- **New functionality is being added to DBMSs in the following areas:**
  - Scientific Applications
  - Image Storage and Management
  - Audio and Video data management
  - Data Mining
  - Spatial data management
  - Time Series and Historical Data Management

*The above gives rise to new research and development in incorporating new data types, complex data structures, new operations and storage and indexing schemes in database systems.*



# When not to use a DBMS

25

- **Main inhibitors (costs) of using a DBMS:**
  - ▣ High initial investment and possible need for additional hardware.
  - ▣ Overhead for providing generality, security, concurrency control, recovery, and integrity functions.
  
- **When a DBMS may be unnecessary:**
  - ▣ If the database and applications are simple, well defined, and not expected to change.
  - ▣ If there are stringent real-time requirements that may not be met because of DBMS overhead.
  - ▣ If access to data by multiple users is not required.

# When not to use a DBMS

26

- **When no DBMS may suffice:**
  - ▣ If the database system is not able to handle the complexity of data because of modeling limitations
  - ▣ If the database users need special operations not supported by the DBMS.

# Data Models

- **Data Model:** A set of concepts to describe the *structure* of a database, and certain *constraints* that the database should obey.
- **Data Model Operations:** Operations for specifying database retrievals and updates by referring to the concepts of the data model. Operations on the data model may include *basic operations* and *user-defined operations*.

# Categories of data models

28

- **Conceptual (high-level, semantic) data models**
  - ▣ Provide concepts that are close to the way many users *perceive* data. (Also called **entity-based** or **object-based** data models.)
  
- **Physical (low-level, internal) data models**
  - ▣ Provide concepts that describe details of how data is stored in the computer.
  
- **Implementation (representational) data models**
  - ▣ Provide concepts that fall between the above two, balancing user views with some computer storage details.

# Schemas, Instances and Database State

29

## □ Database Schema

- The *description* of a database. Includes descriptions of the database structure and the constraints that should hold on the database.

## □ Schema Diagram

- A diagrammatic display of (some aspects of) a database schema.

## □ Schema Construct

- A component of the schema or an object within the schema, e.g., STUDENT, COURSE.

## □ Database State

- The actual data stored in a database at a *particular moment in time*. Also called **snapshot or instance or occurrence**.

# Database Schema Vs. Database State

30

## □ Initial Database State

- Refers to the database when it is loaded

## □ Valid State

- A state that satisfies the structure and constraints of the database.

## □ Distinction

- The **database schema** changes *very infrequently*. The **database state** changes *every time the database is updated*.
- **Schema is also called intension, whereas state is called extension.**

# Three-Schema Architecture

31

- Proposed to support DBMS characteristics of:
  - ▣ **Program-data independence.**
  - ▣ Support of **multiple views** of the data.

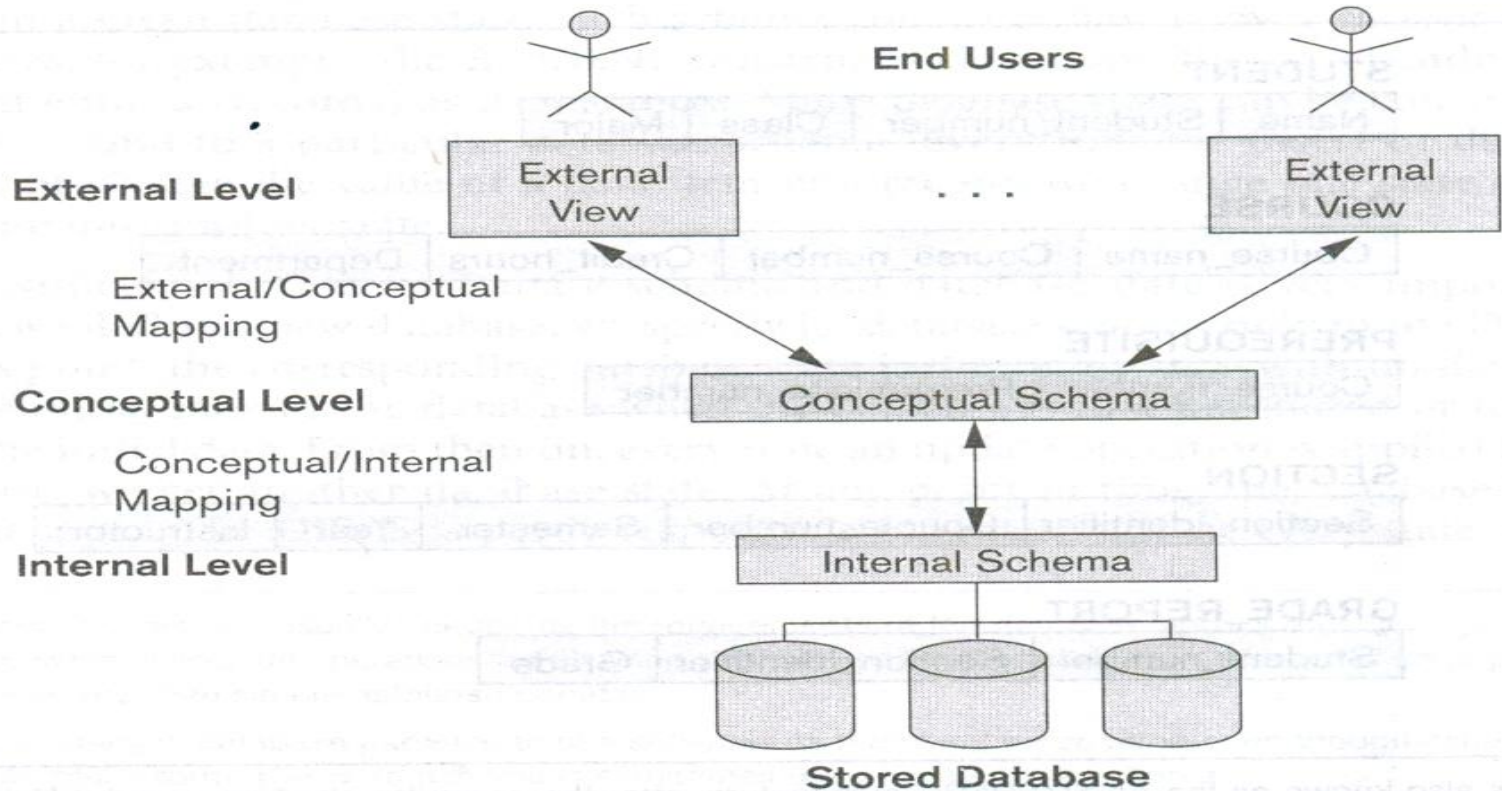
# Three-Schema Architecture

- Defines DBMS schemas at *three levels*:
  - ▣ **Internal schema** at the internal level to describe physical storage structures and access paths. Typically uses a *physical* data model.
  - ▣ **Conceptual schema** at the conceptual level to describe the structure and constraints for the *whole* database for a community of users. Uses a *conceptual* or an *implementation* data model.
  - ▣ **External schemas** at the external level to describe the various user views. Usually uses the same data model as the conceptual level.



# Three-Schema Architecture

33



# Three-Schema Architecture

**Mappings** among schema levels are needed to transform requests and data. Programs refer to an external schema, and are mapped by the DBMS to the internal schema for execution.

# Data Independence

35

- The capacity to change the schema at one level of a database system without having to change the system at the next higher level.
  
- Two types of data independences:
  - **Logical Data Independence:** The capacity to change the conceptual schema without having to change the external schemas and their application programs.
  
  - **Physical Data Independence:** The capacity to change the internal schema without having to change the conceptual schema.

# Data Independence

- 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.
- The higher-level schemas themselves are *unchanged*.
- Hence, the application programs need not be changed since they refer to the external schemas.

# DBMS Languages

- **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.

# DBMS Languages

38

- **Data Manipulation Language (DML):** Used to specify database retrievals and updates.
  - ▣ DML commands (**data sublanguage**) can be *embedded* in a general-purpose programming language (**host language**), such as COBOL, C or an Assembly Language.
  - ▣ Alternatively, *stand-alone* DML commands can be applied directly (**query language**).

# DML Types

39

- **High Level or Non-procedural DML:** e.g., SQL, are *set-oriented* and specify what data to retrieve than how to retrieve. Also called *declarative* languages.
- **Low Level or Procedural DML:** record-at-a-time; they specify *how* to retrieve data and include constructs such as looping.

# DBMS Interfaces

40

- Stand-alone query language interfaces.
- Programmer interfaces for embedding DML in programming languages:
  - ▣ Pre-compiler Approach
  - ▣ Procedure (Subroutine) Call Approach



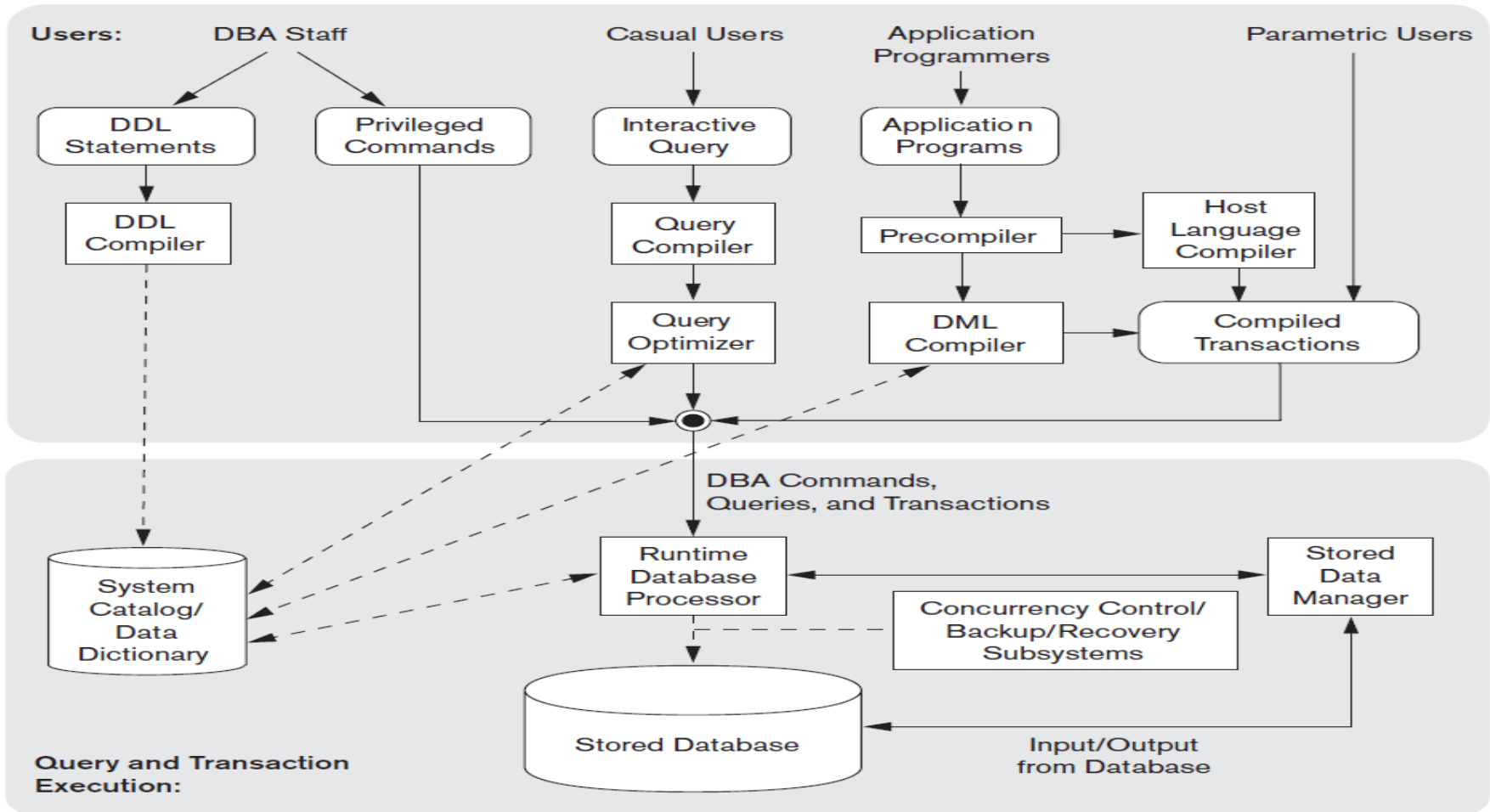
# DBMS Interfaces

41

- User-friendly interfaces:
  - ▣ Menu-based, popular for browsing on the web
  - ▣ Forms-based, designed for naïve users
  - ▣ Graphics-based (Point and Click, Drag and Drop etc.)
  - ▣ Natural language: requests in written English
  - ▣ Speech as Input and Output
  - ▣ Web Browser as an interface
  - ▣ Parametric interfaces (e.g., bank tellers) using function keys.
  - ▣ Interfaces for the DBA:
    - Creating accounts, granting authorizations
    - Setting system parameters
    - Changing schemas or access path

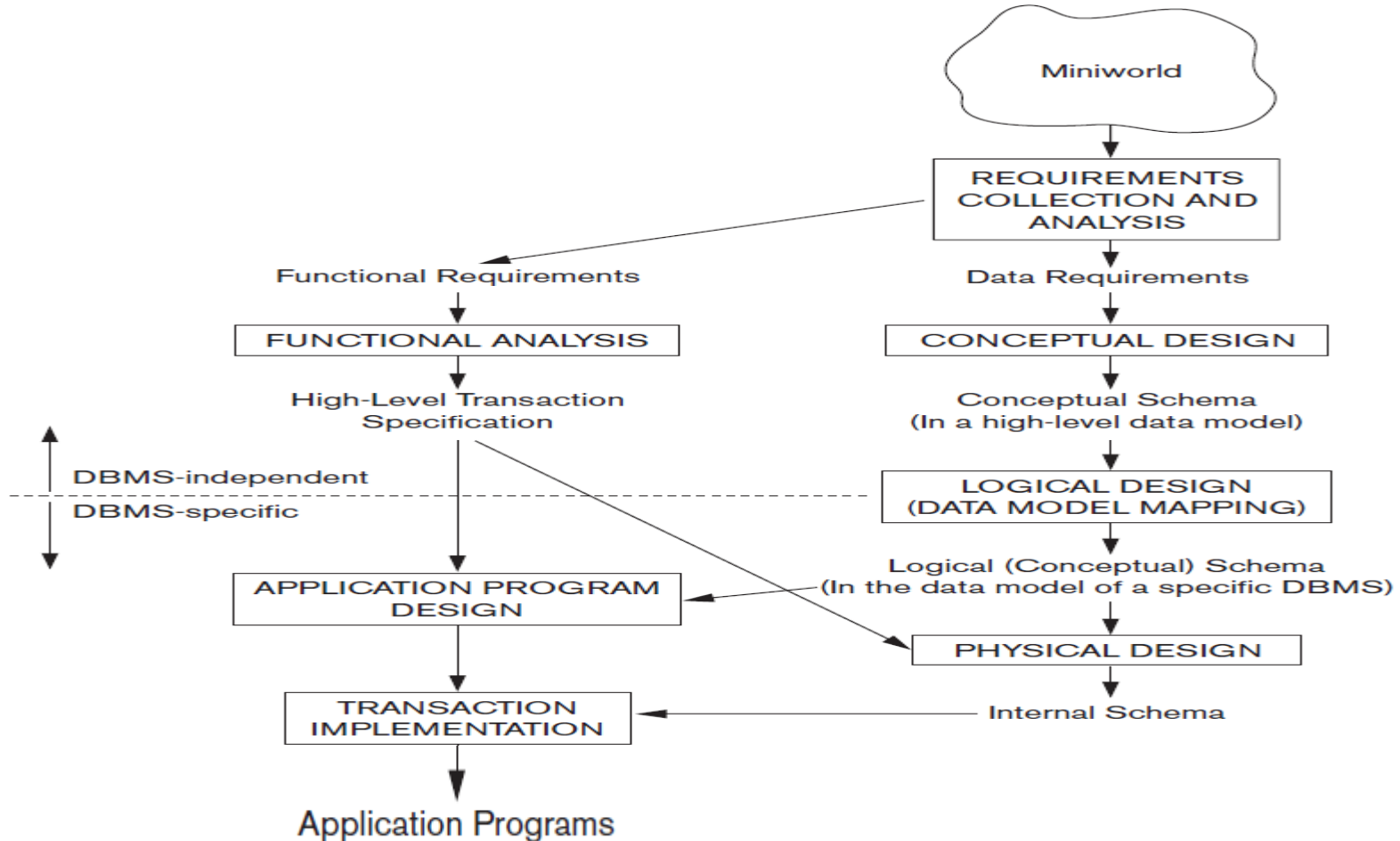
# DBMS Component Modules

42



# Using High – Level Conceptual Data Models for Database Design

43



# An Example Database Application

44

- Requirements of the Company (oversimplified for illustrative purposes)
  - 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 name, number and is located at a single location.

# Contd...

45

- We store each EMPLOYEE's social security number, address, salary, sex, and birth date. 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, birth date, and relationship to employee.

# Entity Types, Entity Sets, Attributes and Keys

46

- Entity
  - ▣ A thing in the real world with an independent existence.
  - ▣ Eg: A particular person, Car, House or an Employee
  
- Attribute
  - ▣ The particular property that describes the entity.
  - ▣ Eg: An Employee's name, age, address, salary and job

# Types of Attributes

47

- Composite versus simple(Atomic) Attributes
- Single Valued versus Multivalued Attributes
- Stored versus Derived Attributes
- Complex Attributes
  - ▣ For example, {Address\_phone({Phone(Area\_Code, Phone\_Number)}, Address(Street\_Address(Number, Street, Apartment\_Number), City, Zip\_Code))}
- Null Values

# Entity Types

48

- A collection of entities that have the same attributes.
- Each entity type in the database is described by its name and attributes.
- An entity type describes the **schema** or **intension** for a set of entities that share the same structure.
- Eg: EMPLOYEE and COMPANY and a list of attributes for each.
- An entity type is represented in ER diagrams as a rectangular box enclosing the entity type name.



# Entity Set

49

- The collection of entities of a particular entity type is grouped into an entity set.
- This is also called as **Extension** of the entity type.

# Key Attribute of an Entity Type

- An entity type usually has an attribute whose values are distinct for each individual entity in the entity set.
- Such an attribute is called as key attribute and its values can be used to identify each entity uniquely.
- Eg: No two companies will have the same name, so name attribute is the key attribute in COMPANY.
- In ER diagrammatic notation, each key attribute has its name **underlined** inside the oval.

# Value Sets (Domains) of Attributes

- Each simple attribute of an entity type is associated with a **value set** (or **domain** of values), which specifies the set of values that may be assigned to that attribute for each individual entity.
- Value sets are typically specified using the basic data types.

# ENTITY SET corresponding to the ENTITY TYPE CAR

52

Entity Type

CAR

Registration(RegistrationNumber, State), VehicleID, Make, Model, Year, (Color)

car<sub>1</sub>  
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 1999, (red, black))  
car<sub>2</sub>  
((ABC 123, NEW YORK), WP9872, Nissan 300ZX, 2-door, 2002, (blue))  
car<sub>3</sub>  
((VSY 720, TEXAS), TD729, Buick LeSabre, 4-door, 2003, (white, blue))  
.  
.  
.

Entity Set

# Relationship Types, Relationship Sets, and Instances

53

- A **relationship type**  $R$  among  $n$  entity types  $E_1, E_2, \dots, E_n$  defines a set of associations – or a **relationship set** – among entities from these entity types.
- Mathematically, the relationship set  $R$  is a set of **relationship instances**  $r_i$ , where each  $r_i$  associates  $n$  individual entities.

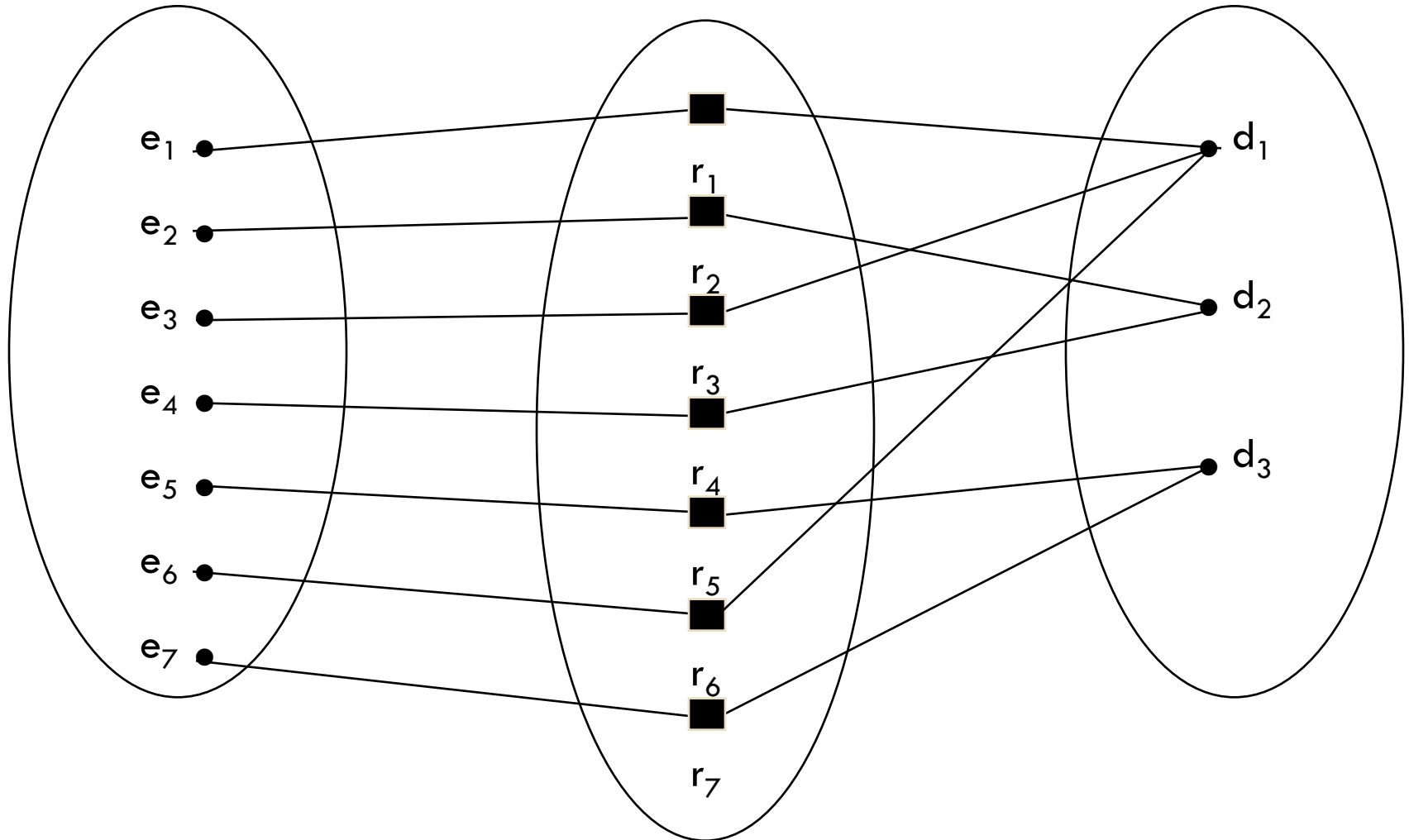
# Example 1...

54

EMPLOYEE

WORKS\_FOR

DEPARTMENT



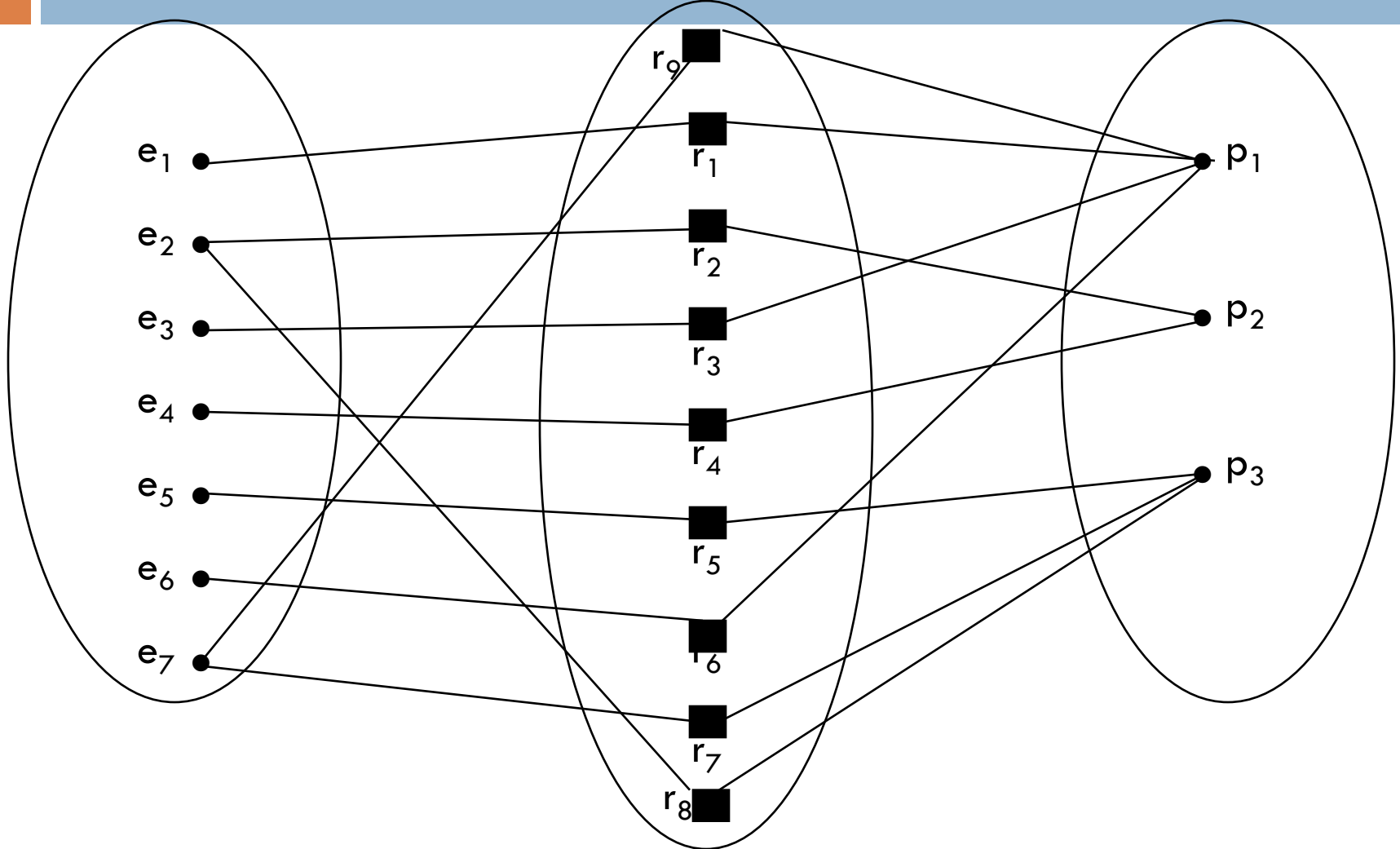
# Example 2...

EMPLOYEE

WORKS\_ON

PROJECT

55



# Relationship Degree

56

- The degree of a relationship type is the number of participating entity types.
- A relationship type of degree two is called **binary**, and one of degree three is called **ternary**.



# Role Names and Recursive Relationships

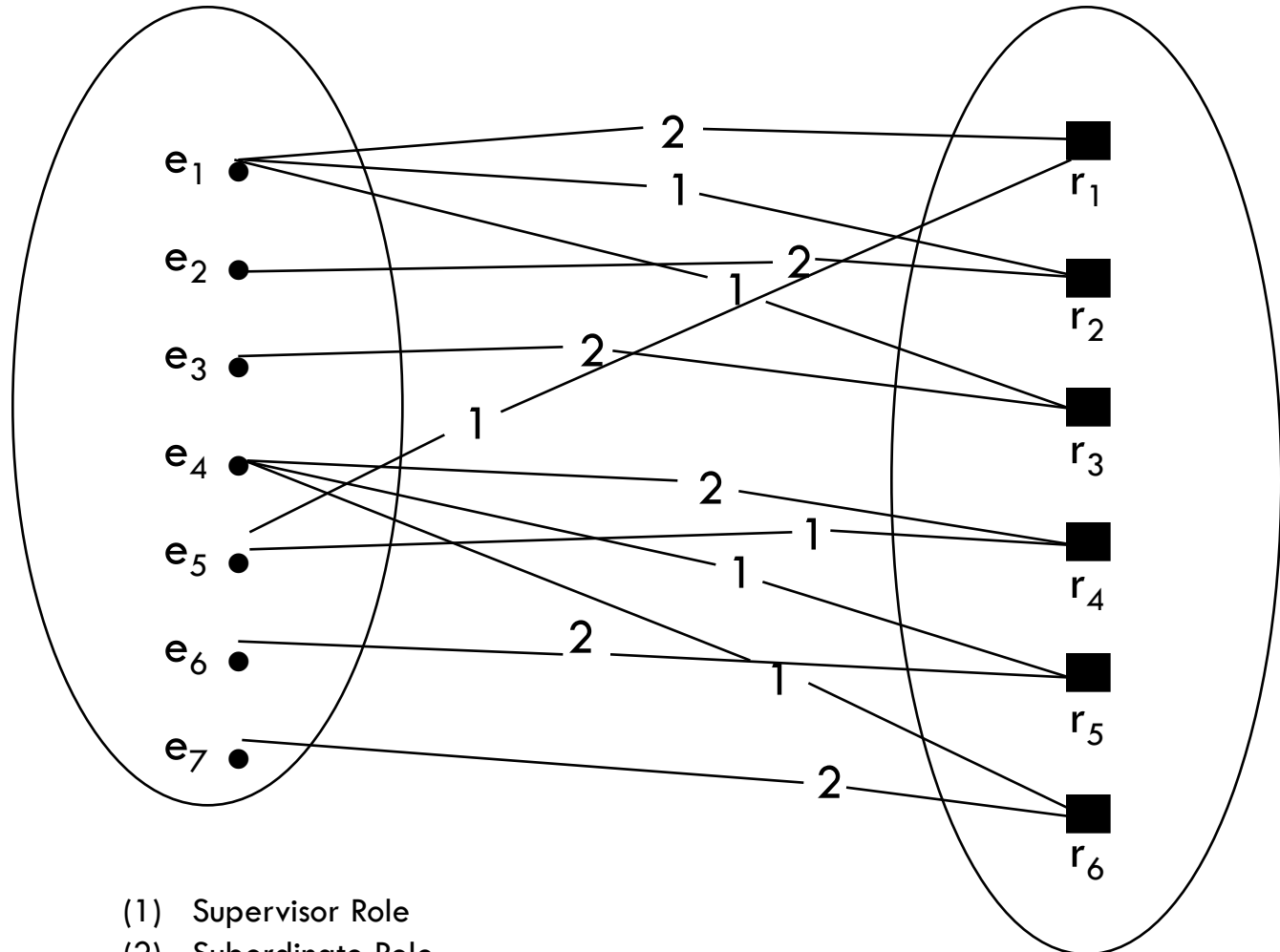
- Each entity type that participates in a relationship type plays a particular **role** in the relationship.
- The role name signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means.
- Some entity types participates more than once in a relationship type in different roles. Such relationship types are called **recursive relationships**.

# Example...

58

EMPLOYEE

SUPERVISION



- (1) Supervisor Role
- (2) Subordinate Role

# Constraints on Relationship Types

59

- There are two types of relationship constraints:
  - Cardinality ratio
    - The cardinality ratio for binary relationship specifies the maximum number of relationship instances that an entity can participate in.
    - The possible cardinality ratio for binary relationship types are 1:1, 1:N, N:1 and M:N
  - Participation
    - The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type.
    - This constraint specifies the minimum number of relationship instances that each entity can participate in.
- The cardinality ratio and participation constraint together called as **structural constraint**.

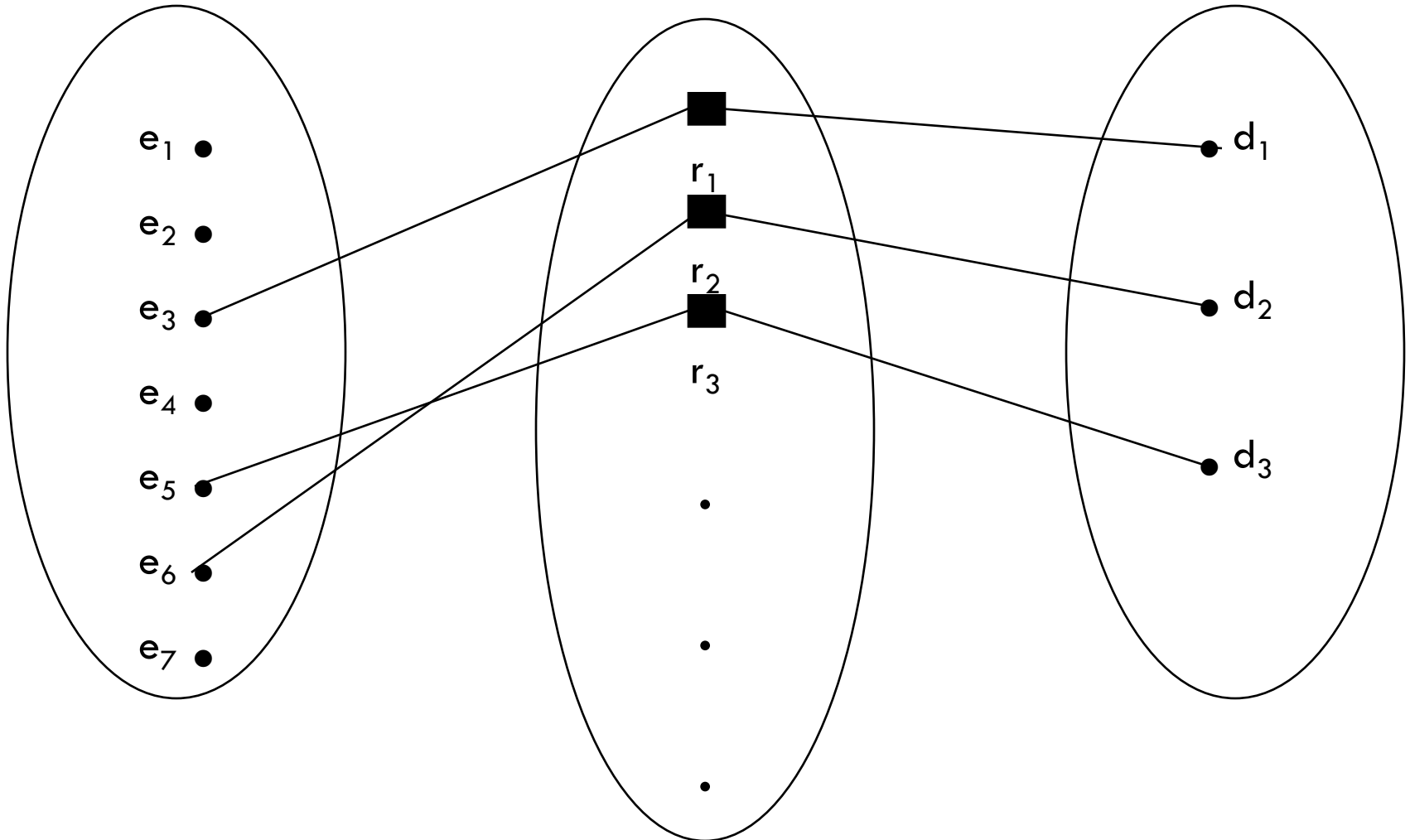
# 1:1 Relationship

60

EMPLOYEE

MANAGES

DEPARTMENT



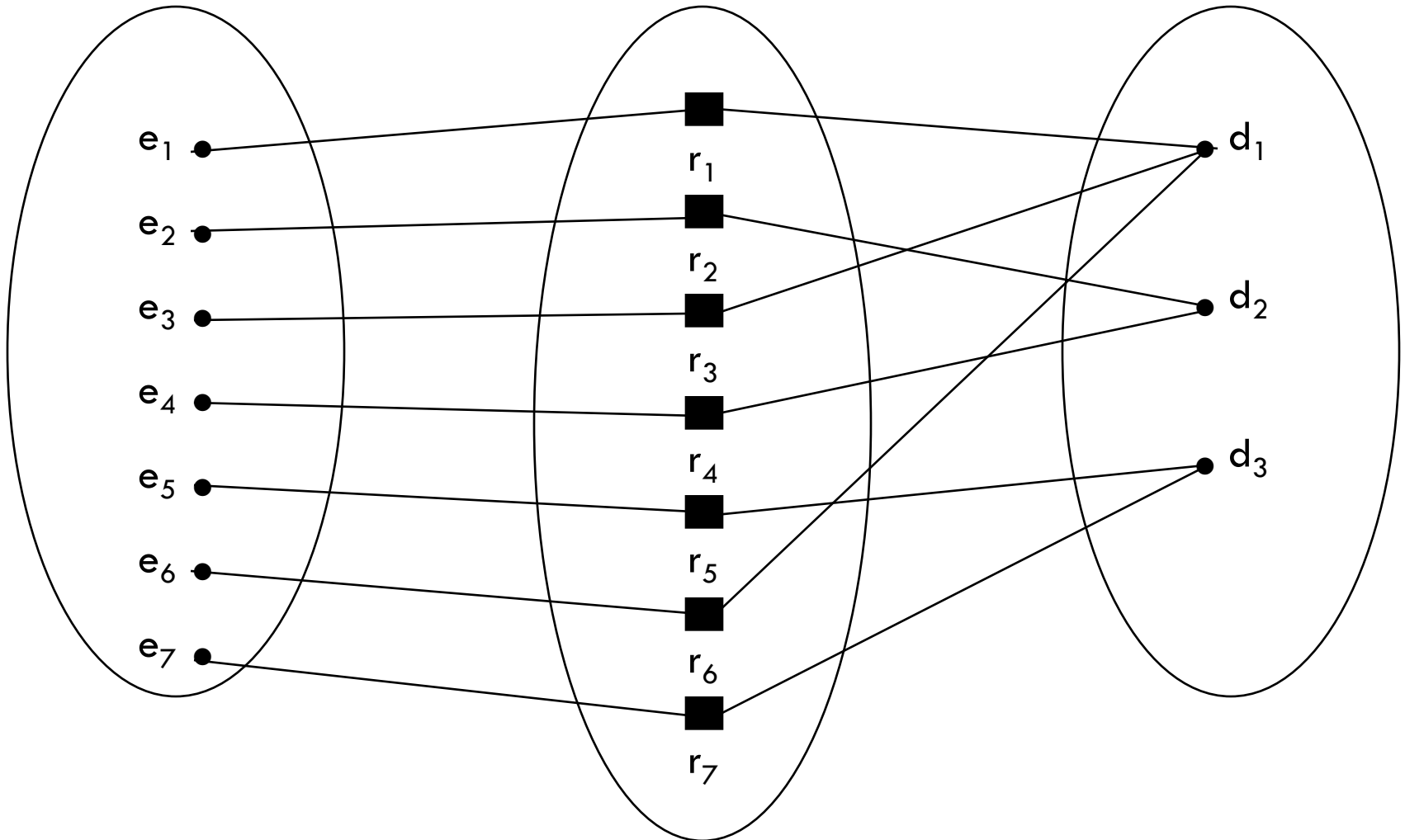
# 1:N Relationship

61

EMPLOYEE

WORKS\_FOR

DEPARTMENT

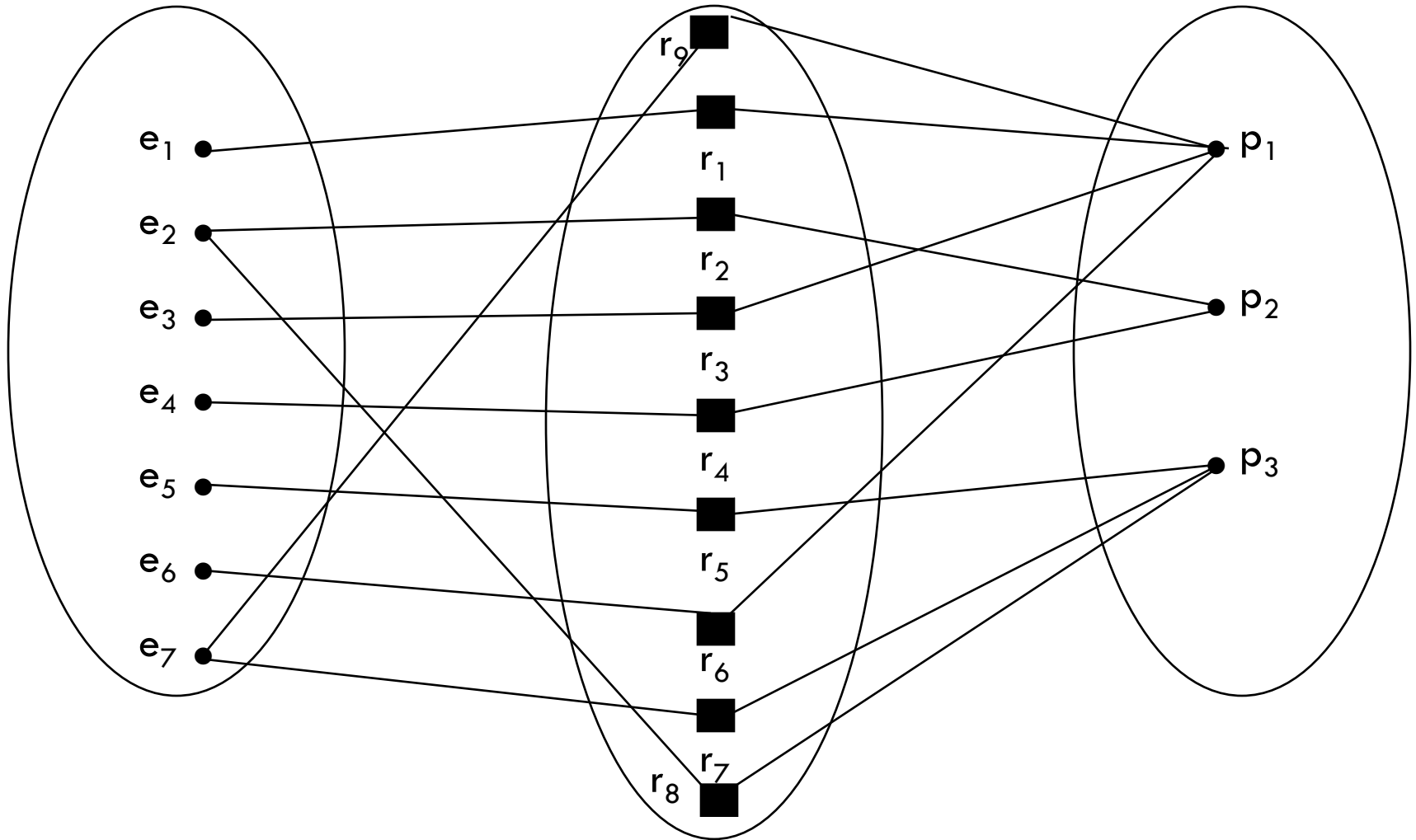


# M:N Relationship

62 EMPLOYEE

WORKS\_ON

PROJECT



# Participation Constraint and Existence Dependence

63

- The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type.
- This is also called as minimum cardinality constraint.
- Two types of participation constraint:
  - ▣ Total Participation, also called as Existence Dependency
  - ▣ Partial Participation

# Attributes as Relationship Types

- A relationship type can have attributes; for example, HoursPerWeek of WORKS\_ON; its value for each relationship instance describes the number of hours per week that an EMPLOYEE works on a PROJECT.



# Weak Entity Types

65

- Entity types that do not have key attributes of their own are called **weak entity types**.
- The entity types which contain key attributes of their own are called **regular entity types** or **strong entity types**.
- Entities belonging to weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. This other entity type is called as **identifying entity type** or **owner entity type**.
- The relationship type that relates a weak entity type to its owner is called as **identifying relationship** of the weak entity type.
- The weak entity type normally has a partial key, which is the set of attributes that can uniquely identify weak entities that are related to the same owner entity.

# Example...

66

## Example:

Suppose that a `DEPENDENT` entity is identified by the dependent's first name and birth date, *and* the specific `EMPLOYEE` that the dependent is related to.

`DEPENDENT` is a weak entity type with `EMPLOYEE` as its identifying entity type via the identifying relationship type `DEPENDENT_OF`.

# Notations Used in ER Diagrams

67

Symbol	Meaning
	ENTITY TYPE
	WEAK ENTITY TYPE
	RELATIONSHIP TYPE
	IDENTIFYING RELATIONSHIP TYPE
	ATTRIBUTE
	KEY ATTRIBUTE
	MULTIVALUED ATTRIBUTE
	COMPOSITE ATTRIBUTE
	DERIVED ATTRIBUTE
	TOTAL PARTICIPATION OF E <sub>2</sub> IN R
	CARDINALITY RATIO 1:N FOR E <sub>1</sub> :E <sub>2</sub> IN R
	STRUCTURAL CONSTRAINT (min, max) ON PARTICIPATION OF E IN R

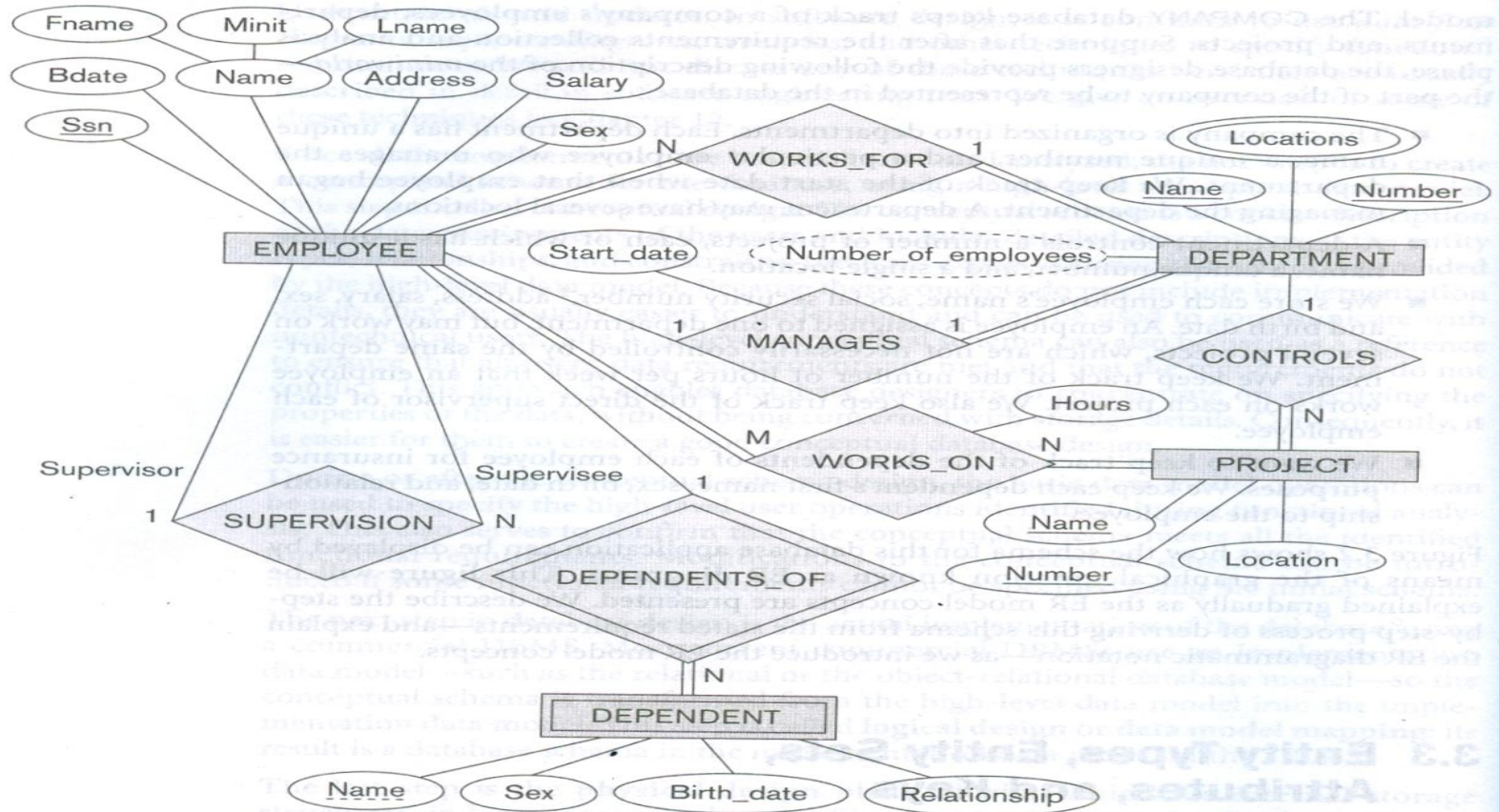
# Proper Naming of Schema Constructs

68

- Use singular names for entity types, rather than plural one.
- Verbs tend to indicate the relationship types.
- Another naming consideration involves choosing the binary relationship names to make the ER diagram of the schema readable from left to right and top to bottom.

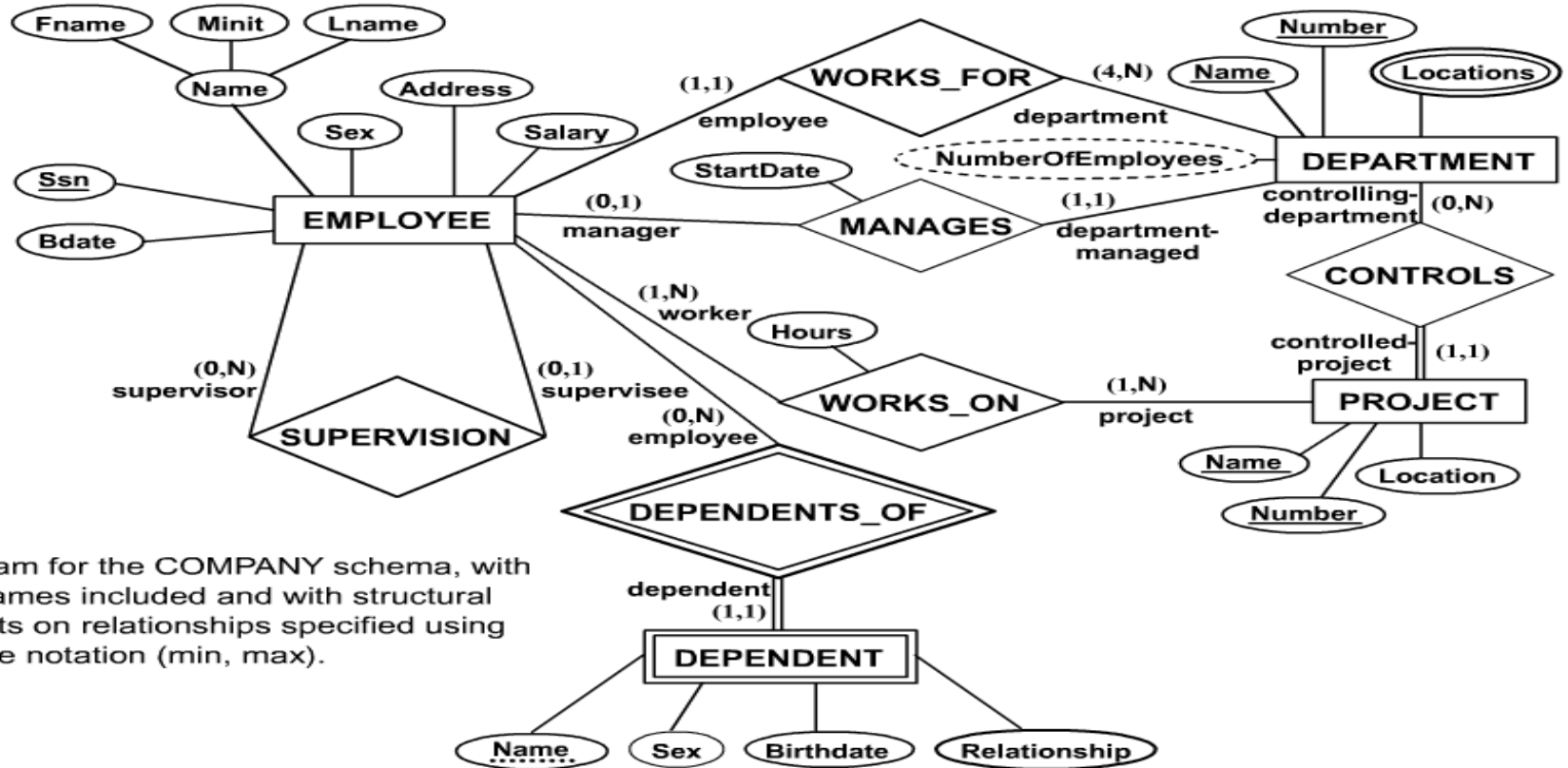
# E – R Diagram for COMPANY

69



# Alternative Notations for ER Diagrams

## Alternative ER Notations



ER diagram for the COMPANY schema, with all role names included and with structural constraints on relationships specified using alternative notation (min, max).

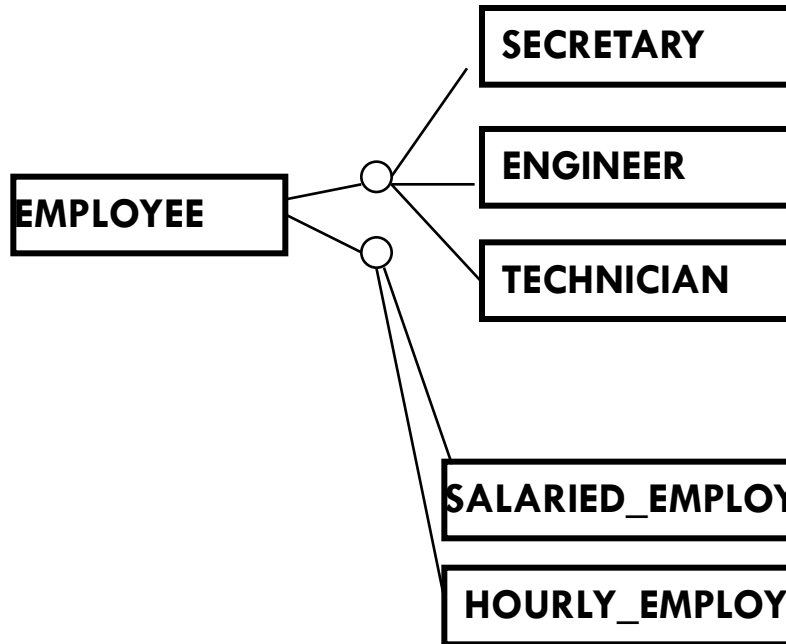
# Relationships of Higher Degree

71

- Relationship types of degree 2 are called **binary**
- Relationship types of degree 3 are called **ternary** and of degree  $n$  are called  **$n$ -ary**
- In general, an  $n$ -ary relationship *is not* equivalent to  $n$  binary relationships

# Subclasses and Superclasses

❑ An entity type may have additional meaningful subgroupings of its entities



**Superclasses:** EMPLOYEE

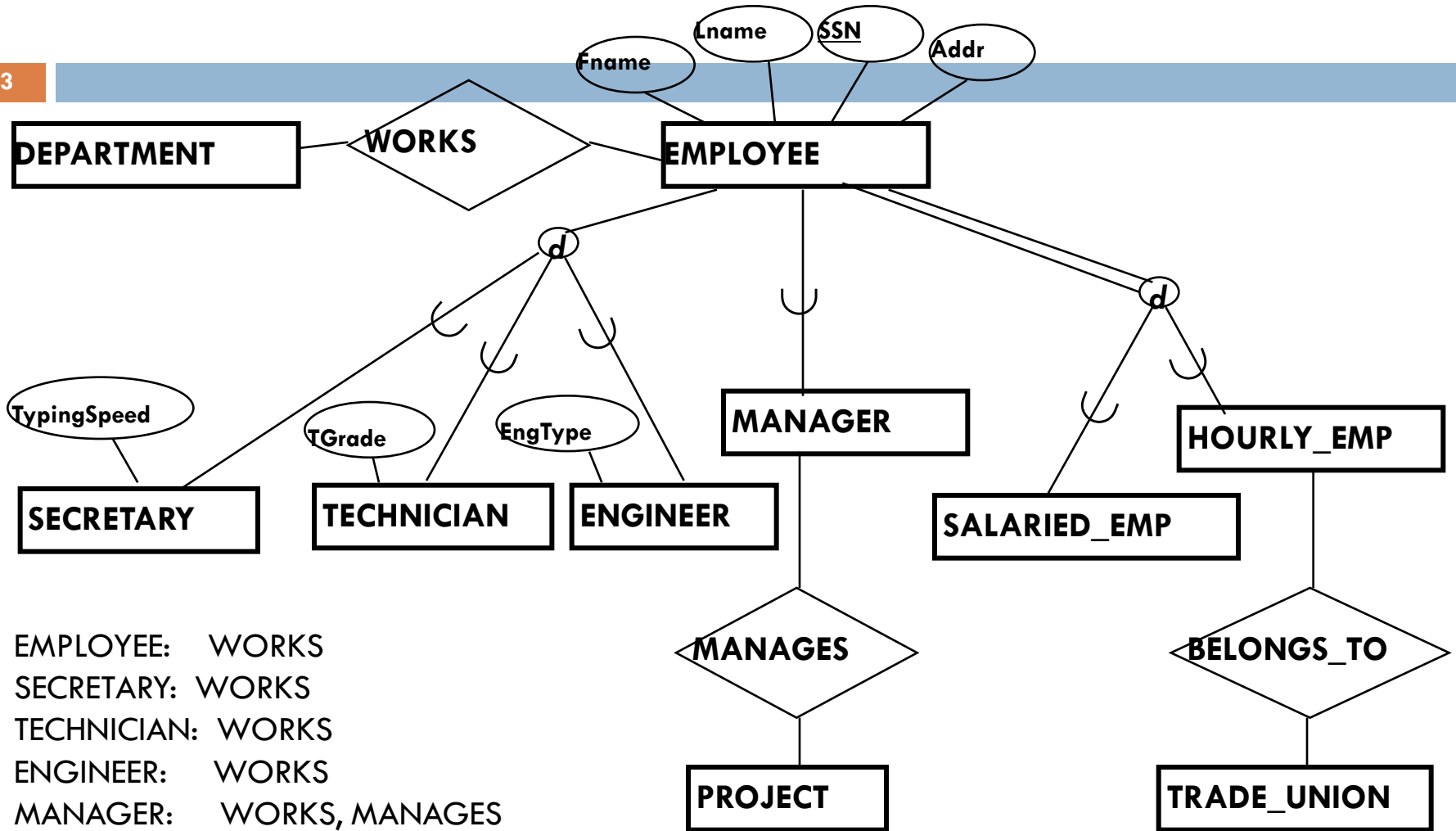
**Subclasses:** SECRETARY, ENGINEER,  
TECHNICIAN,  
SALARIED\_EMPLOYEE,  
HOURLY\_EMPLOYEE

**Every entity that is a member of one of these subgroupings is also an employee**



# Example

73



EMPLOYEE: WORKS  
 SECRETARY: WORKS  
 TECHNICIAN: WORKS  
 ENGINEER: WORKS  
 MANAGER: WORKS, MANAGES  
 SALARIED\_EMP: WORKS  
 HOURLY\_EMP: WORKS, BELONGS\_TO

# Why class/subclass relationships and specializations

74

- Certain attributes may apply to some but not all entities of the superclass.
  - ▣ 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.

EMPLOYEE (Name, SSN, BirthDate, Address)

SECRETARY (Name, SSN, BirthDate, Address, **TypingSpeed**)

ENGINEER (Name, SSN, BirthDate, Address, **EngineerType**)

TECHNICIAN (Name, SSN, BirthDate, Address, **TGrade**)

# Why need class/subclass relationships and specializations

75

- Some relationship types may be participated in only by entities that are members of the subclass.

# Subclasses vs. Superclasses

76

- The set of entities in each subclasses is a **subset** of the entities that belong to EMPLOYEE
- Each is called a subclass of EMPLOYEE
- EMPLOYEE is the superclass for each of these subclasses
- The relationship between a superclass and any one of its subclass is called a **superclass/subclass** or **class/subclass relationship**.

e.g., EMPLOYEE/SECRETARY and EMPLOYEE/TECHNICIAN are two class/subclass relationships.

# Properties of Superclasses and Subclasses

77

- A member entity of the subclass represents ***the same real-world entity*** as some member of the superclass.
- The subclass member is the same as the entity in the superclass, but in a ***distinct specific role***.
- When implementing a superclass/subclass relationship, a member of the subclass may be represented as ***a distinct database object*** – a distinct record that is related via the key attribute to its superclass entity.

# Properties of Superclasses and Subclasses (cont.)

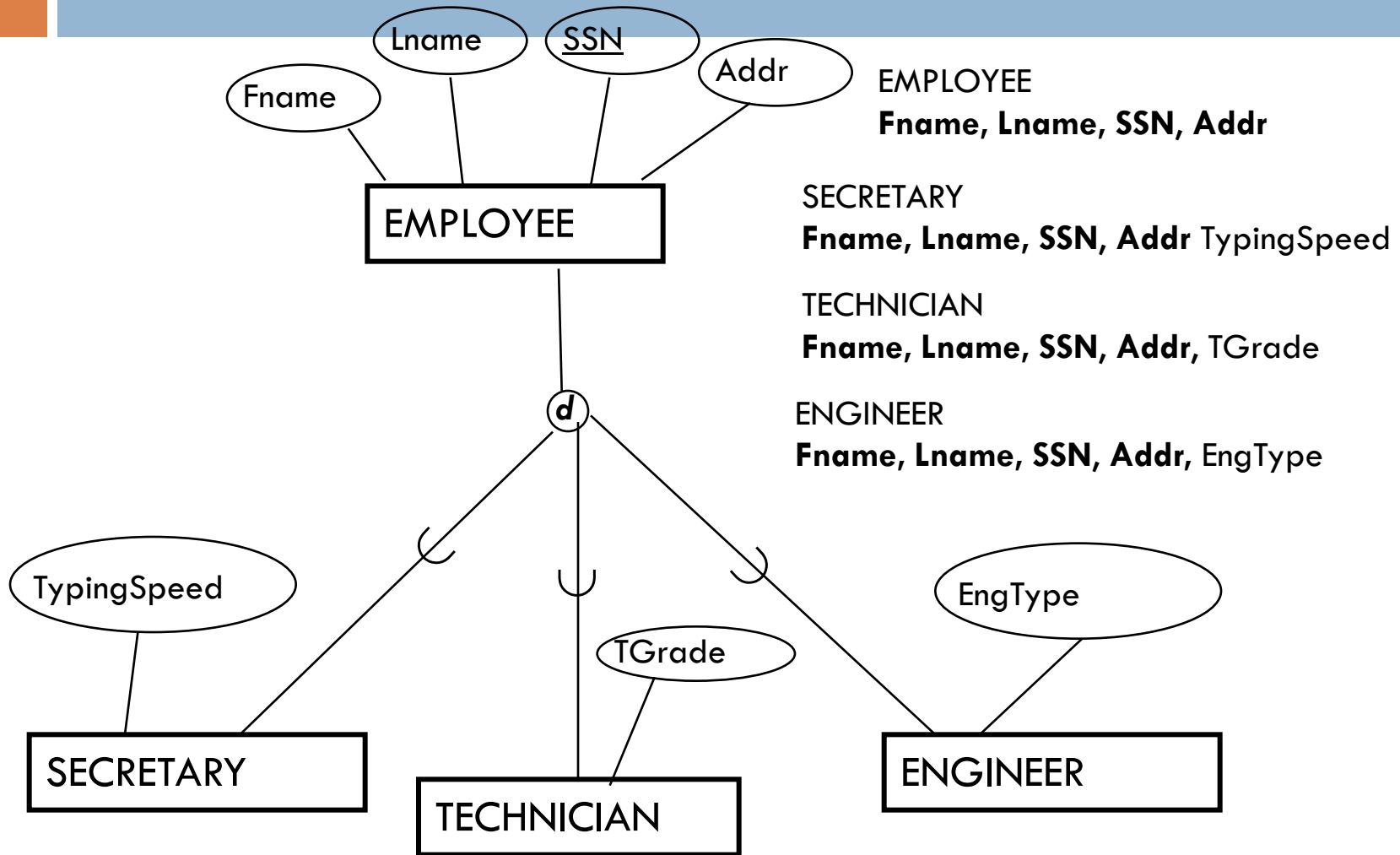
78

- An entity **CANNOT** exist in the DB merely by being a member of a subclass. **It must also be a member of the superclass.**
- An entity can be a member of more than one subclass.
  - Example: A salaried employee who is also an engineer belongs to the two subclasses ENGINEER and SALARIED\_EMPLOYEE
- It is not necessary that every entity in a superclass be a member of some subclass
  - Example: A technical writer is an employee but does not belong to any subclasses.

# Type inheritance

- The **type** of an entity is defined by the **attributes** it possesses and the **relationship types** which it participates.
- An entity that is a member of a subclass **inherits** all the attributes of the entity as a member of the superclass, as well as all the **relationships** in which the superclass participates.

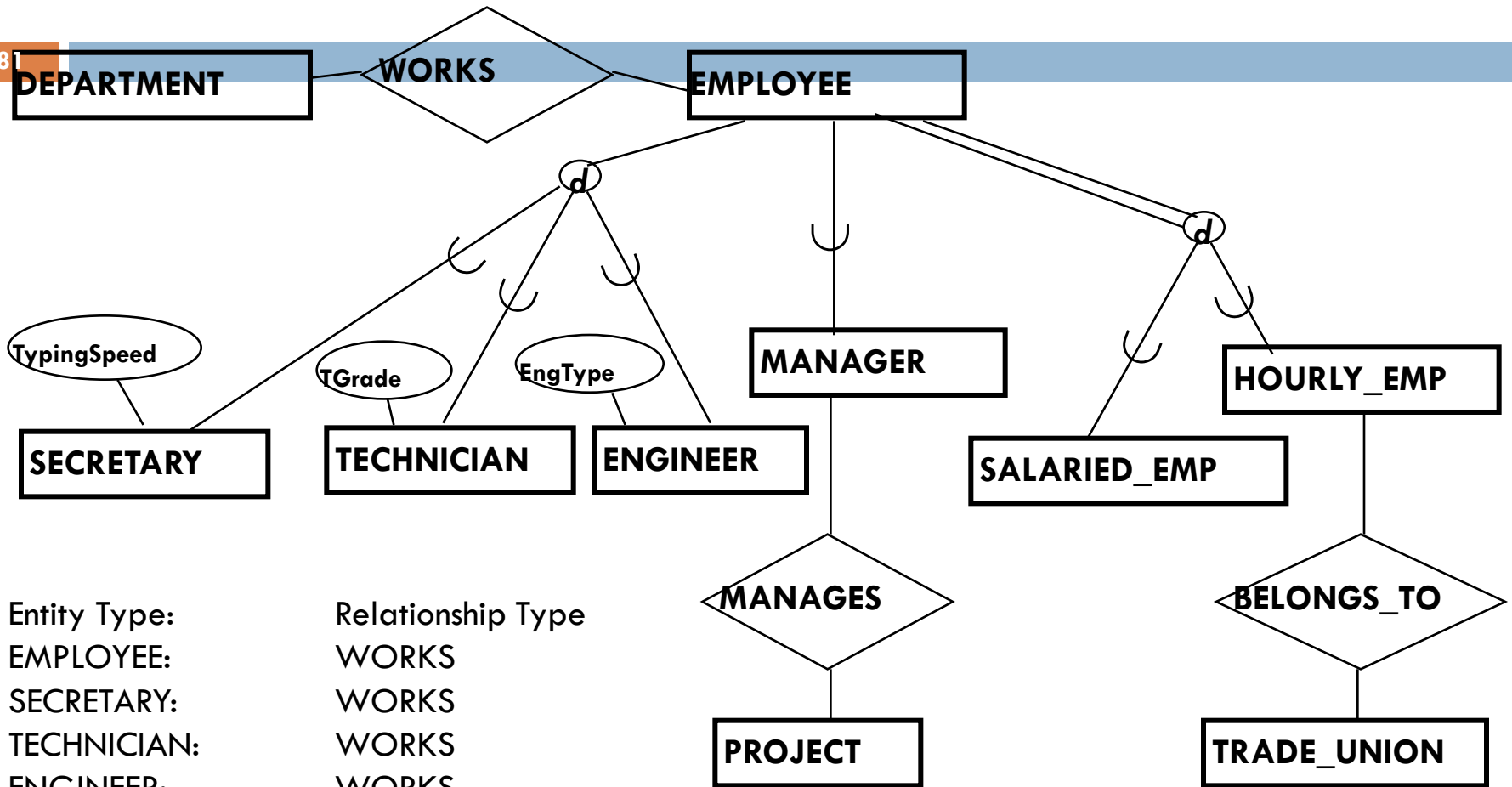
# Example





# Example

81



Entity Type:	Relationship Type
EMPLOYEE:	WORKS
SECRETARY:	WORKS
TECHNICIAN:	WORKS
ENGINEER:	WORKS
MANAGER:	WORKS, MANAGES
SALARIED_EMP:	WORKS
HOURLY_EMP:	WORKS, BELONGS_TO

# Specialization

82

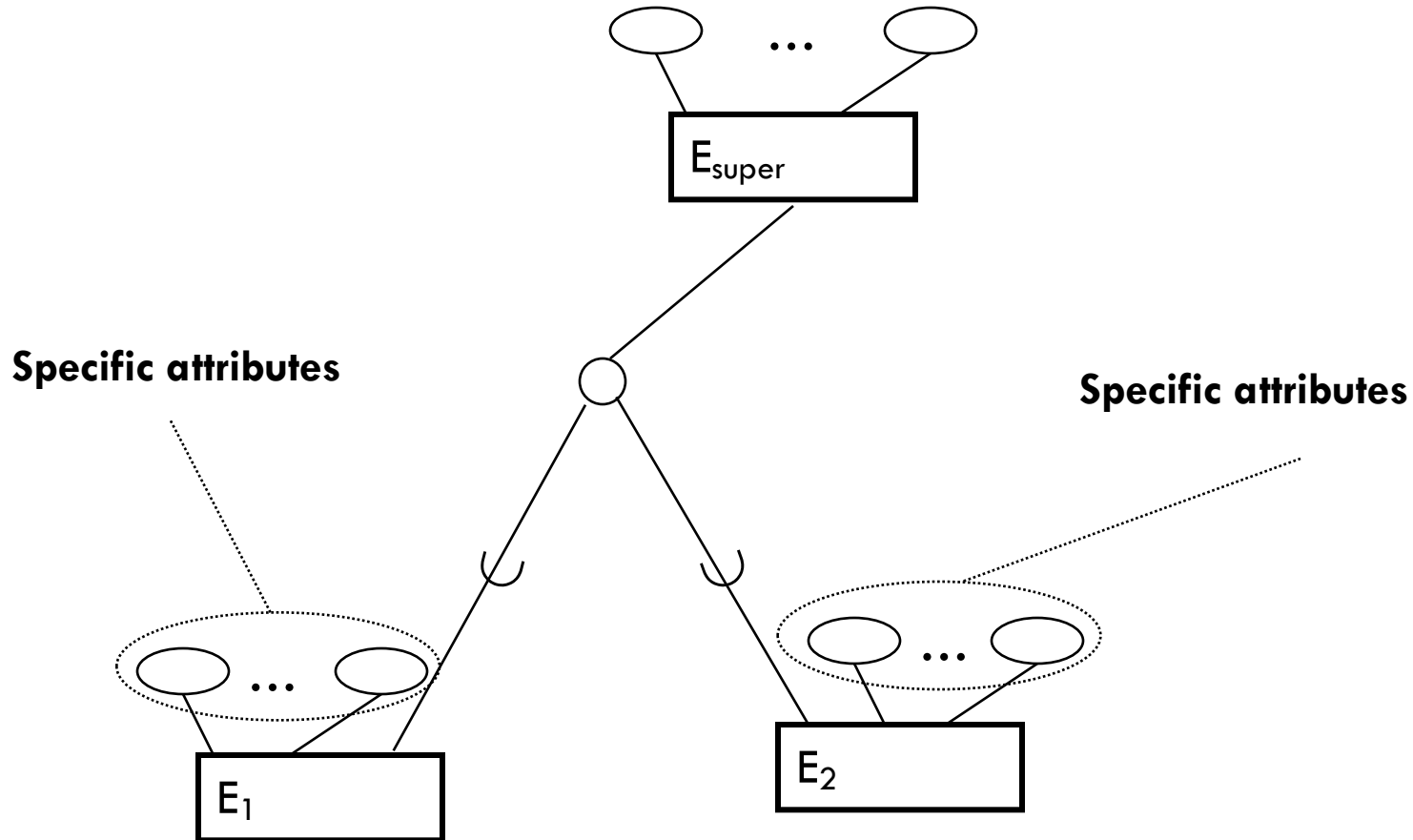
- The process of defining a set of subclass of an entity type (the **superclass** of the specialization).
- The set of subclasses that form a specialization is defined on the basis of some ***distinguishing characteristics*** of the entities in the superclass.  
{SECRETARY, ENGINEER, TECHNICIAN} is a specialization of EMPLOYEE based on the *job type* of each entity.

# Specialization (cont.)

83

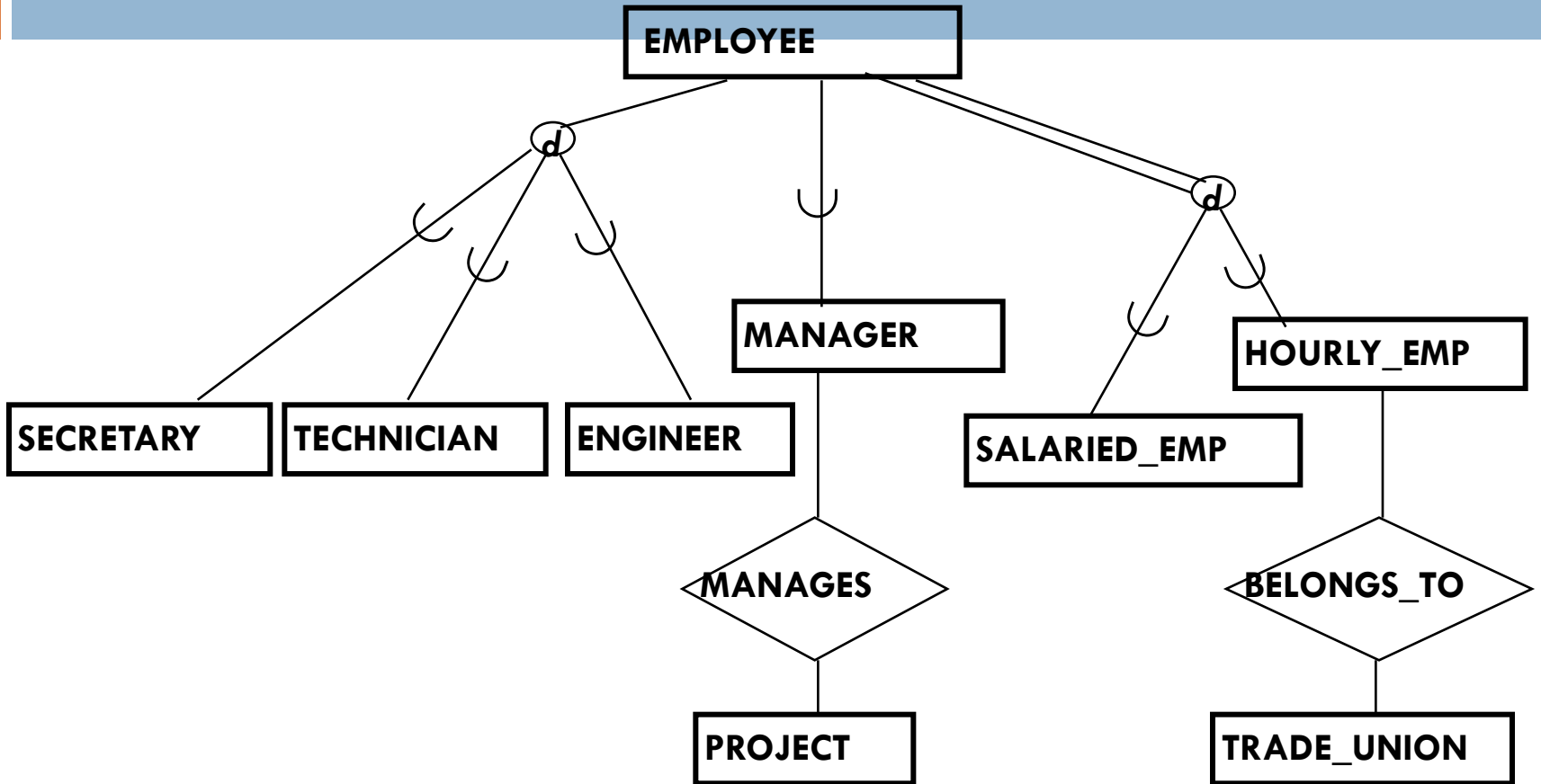
- The same entity type may have several specializations based on different distinguishing characteristics.
- The EMPLOYEE entity type may have two specializations:
  - ▣ Based on the *methods of pay*:  
{SALARIED\_EMPLOYEE, HOURLY\_EMPLOYEE}
  - ▣ Based on the type of job:  
{SECRETARY, ENGINEER, TECHNICIAN}
- A subclass can participate in **specific relationship type**

# Diagrammatic representation of specialization in an EER diagram



# Example

85



# Specialization

The specialization process allows us to do the following:

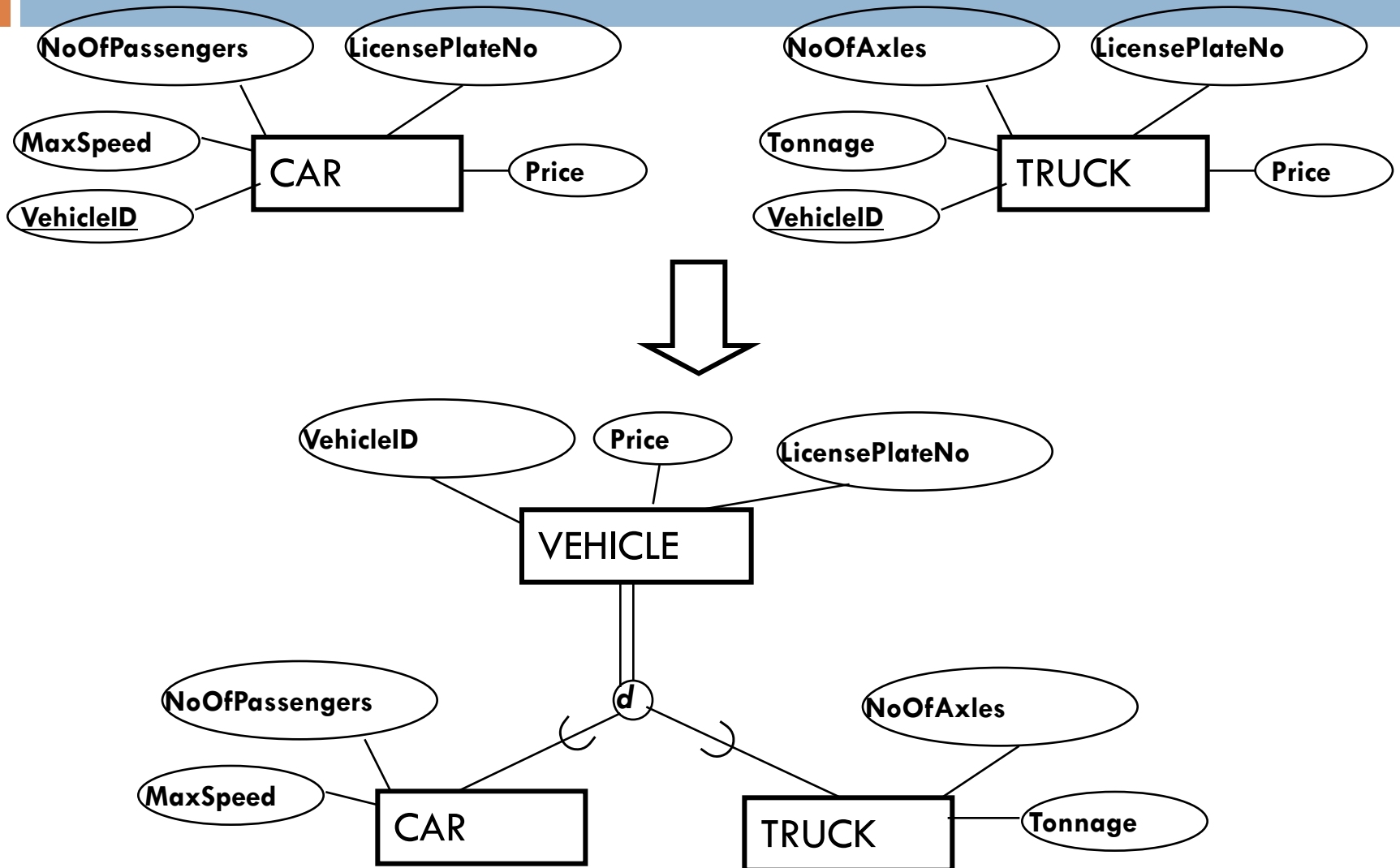
- Define a set of subclass of an entity type
- Establish additional specific attributes with each subclass
- Establish additional specific relationship types between each subclass and other entity types or other subclasses

# Generalization

87

- **Generalization** is the reverse of specialization process. It defines a generalized entity type from the given entity types.

# Generalization (cont.)





# Generalization (cont.)

89

- We can view {CAR, TRUCK} as a specialization of VEHICLE
- Alternatively, we can view VEHICLE as a generalization of CAR and TRUCK

# Generalization (cont.)

- Generalization suppresses the difference among several entity types, identifying their common features, and **generalize** them into **a single superclass** of which the original types are special **subclasses**.
- The decision as to which process, *generalization* or *specialization*, is more appropriate in a particular situation is often subjective.

# Generalization (cont.)

- Generalization suppresses the difference among several entity types, identifying their common features, and **generalize** them into **a single superclass** of which the original types are special **subclasses**.
- The decision as to which process, *generalization* or *specialization*, is more appropriate in a particular situation is often subjective.

# Questions

1. What are the advantages of DBMS. Explain.
2. Explain the three-schema architecture with a neat diagram.
3. With a neat diagram, explain the various components of DBMS.
4. Explain the different ways of interacting with the databases.
5. Explain the different types of attributes with examples for each.
6. Write an E-R diagram for hospital database by considering at least 5 entities.
7. Differentiate between specialization and generalization.