MODULE – 1

INTRODUCTION TO DATABASES

OVERVIEW OF DATABASE LANGUAGES AND ARCHITECTURES

CONCEPTUAL DATA MODELING USING ENTITIES AND RELATIONSHIPS

Types of Databases and Database Applications

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

Basic Definitions

- **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

- 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

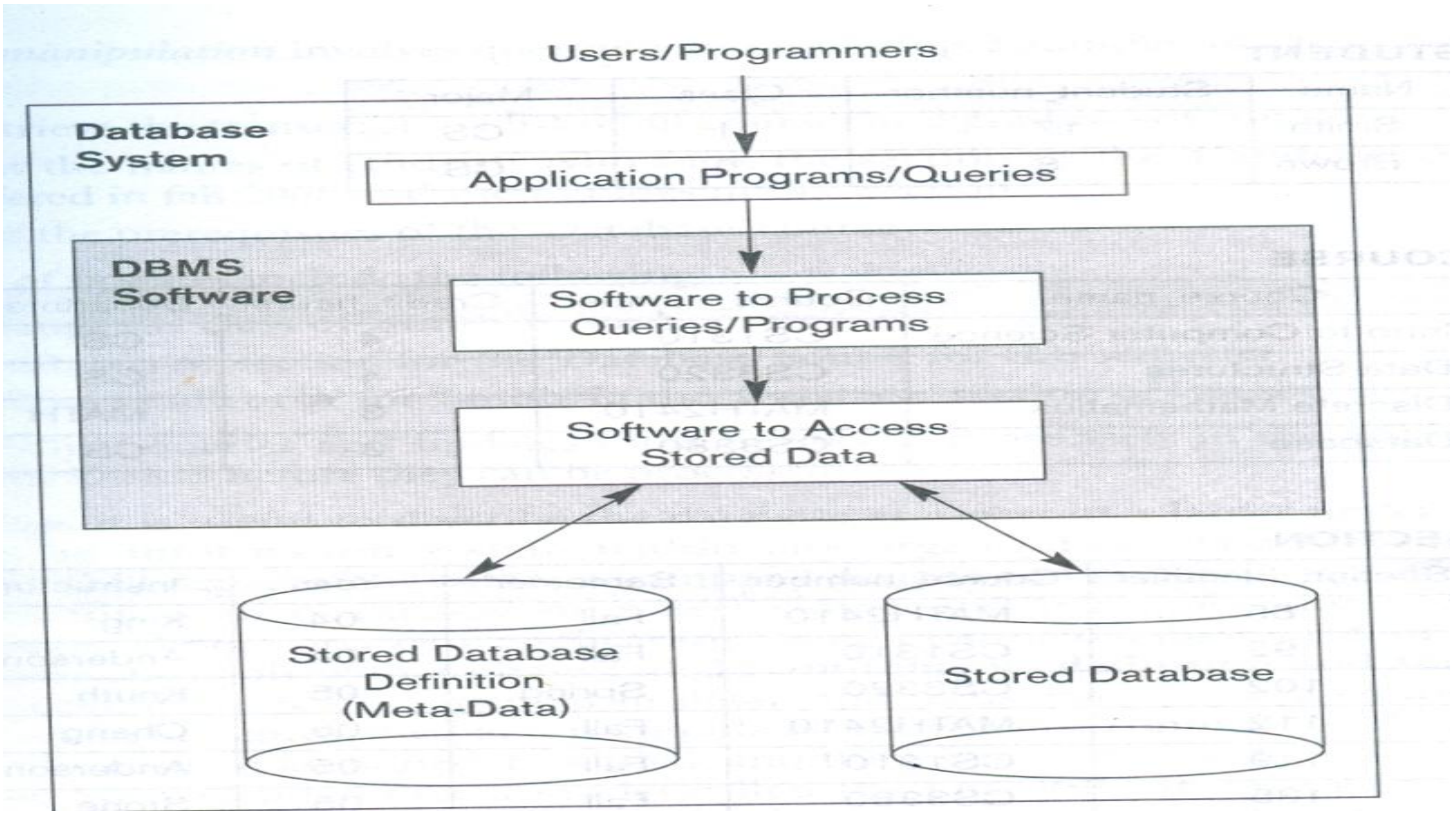
Example of a Database (with a Conceptual Data Model)

- **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


Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

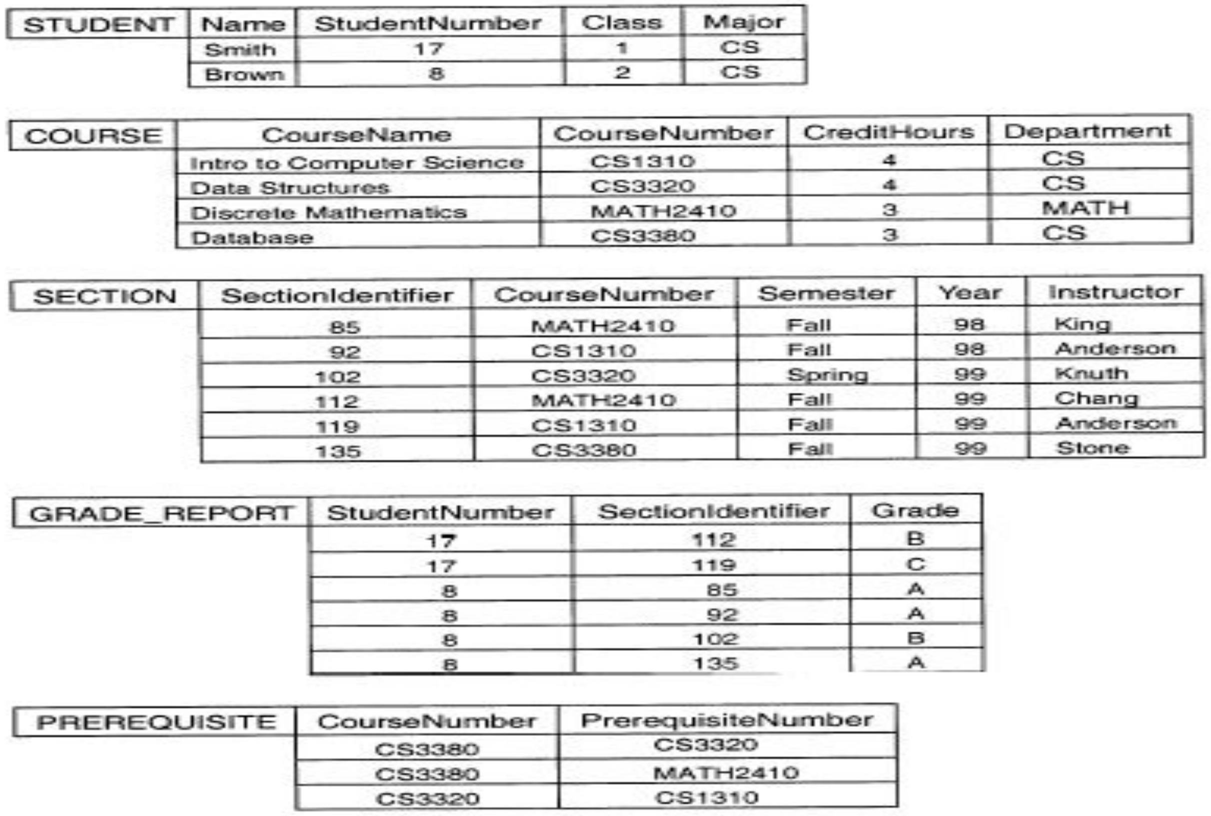
Example of a Database (with a Conceptual Data Model)

- **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

- 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

- 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

- <u>Sharing of data and multiuser transaction processing :</u>
  - 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

- **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

- **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

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

Advantages of Using the Database Approach

- 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

- 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

- **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

- **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

- **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

- **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

- **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

- **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

- **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

- **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

- **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

- **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.**

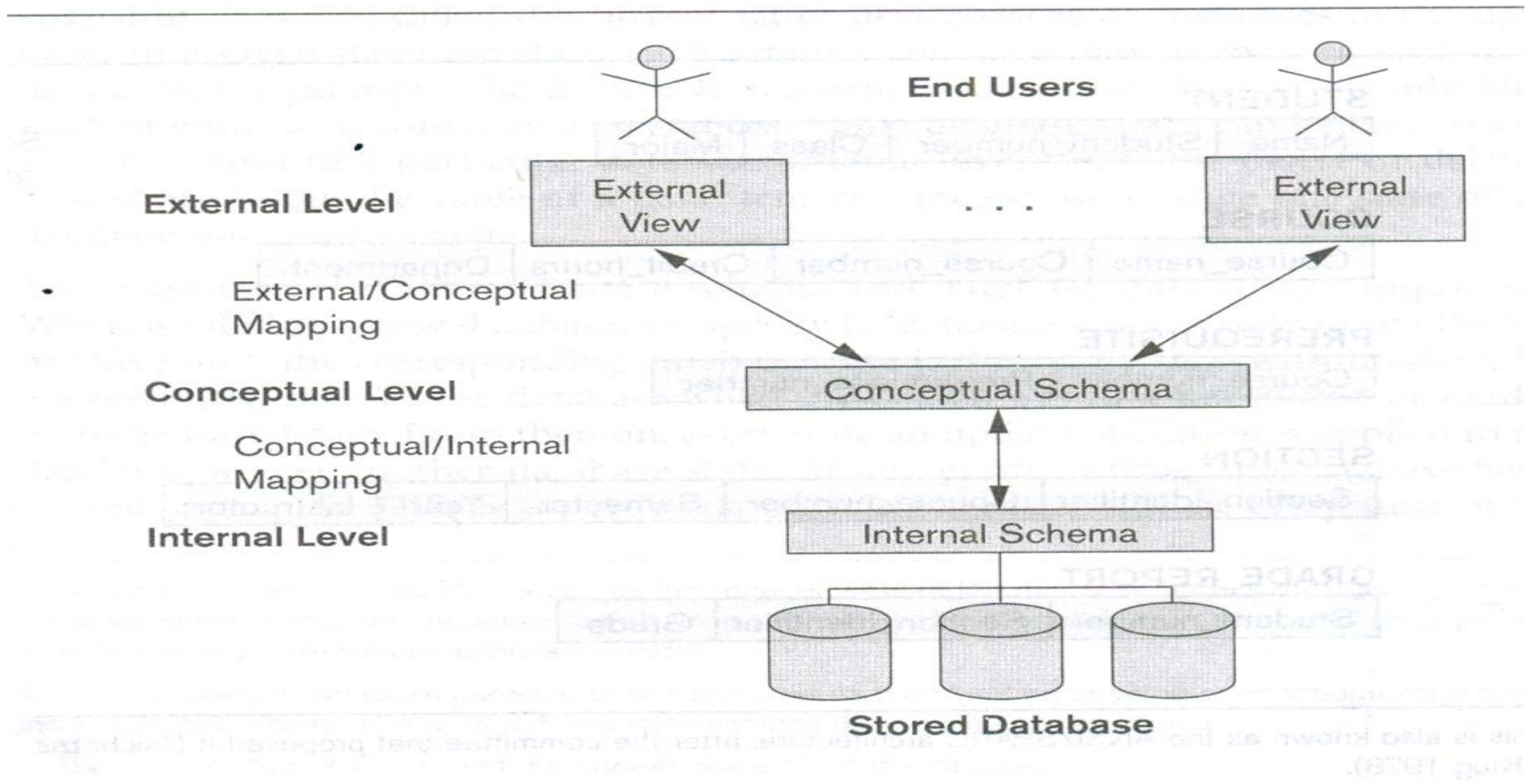
Three-Schema Architecture

- 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



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

- 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.

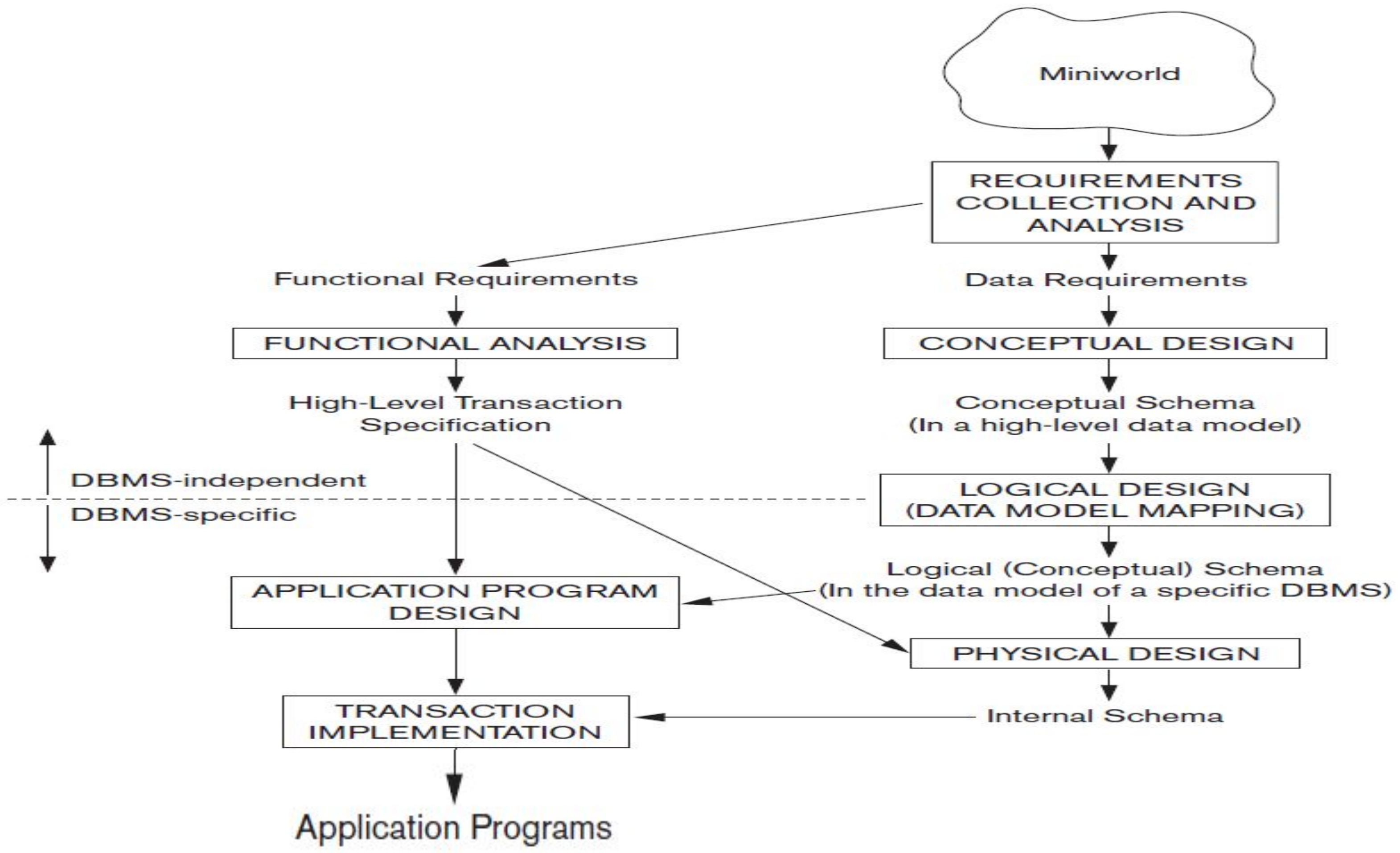
Data Independence

- 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.

Using High – Level Conceptual Data Models for Database Design


Miniworld
REQUIREMENTS COLLECTION AND ANALYSIS
Functional Requirements
Data Requirements
FUNCTIONAL ANALYSIS
CONCEPTUAL DESIGN
High-Level Transaction Specification
Conceptual Schema (In a high-level data model)
DBMS-independent
DBMS-specific
LOGICAL DESIGN (DATA MODEL MAPPING)
Logical (Conceptual) Schema (In the data model of a specific DBMS)
APPLICATION PROGRAM DESIGN
PHYSICAL DESIGN
TRANSACTION IMPLEMENTATION
Internal Schema
Application Programs

An Example Database Application

- 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…

- 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

- 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

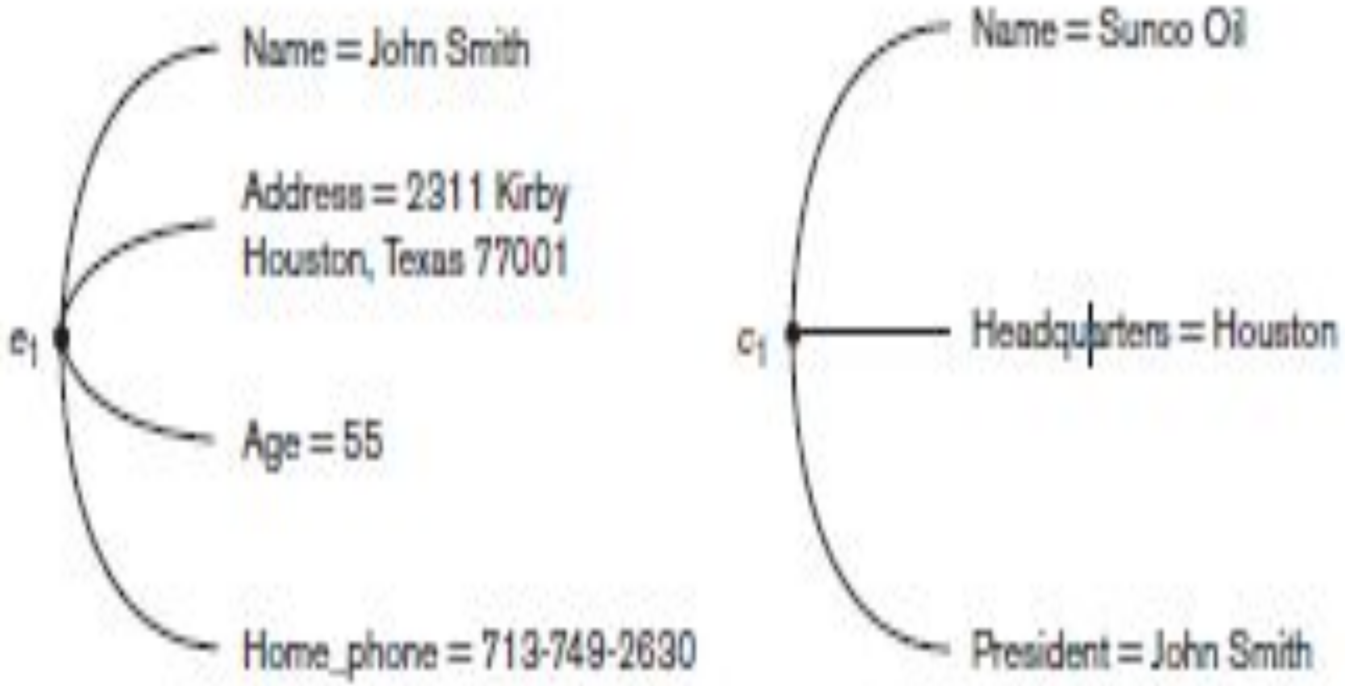


Figure 3.3
Two entities, EMPLOYEE e_1, and COMPANY c_1, and their attributes.

Types of Attributes

- 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

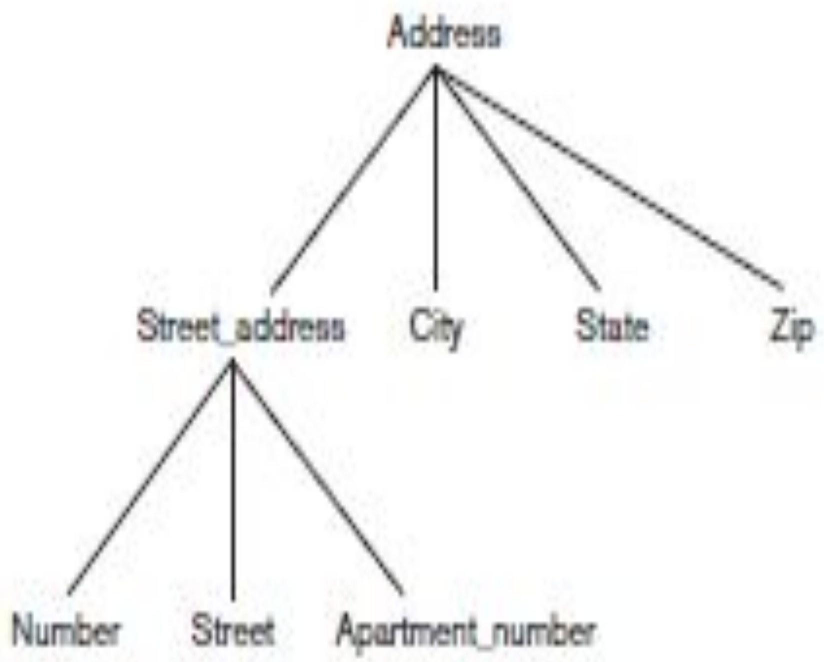


Figure 3.4
A hierarchy of composite attributes.

Entity Types

- 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.

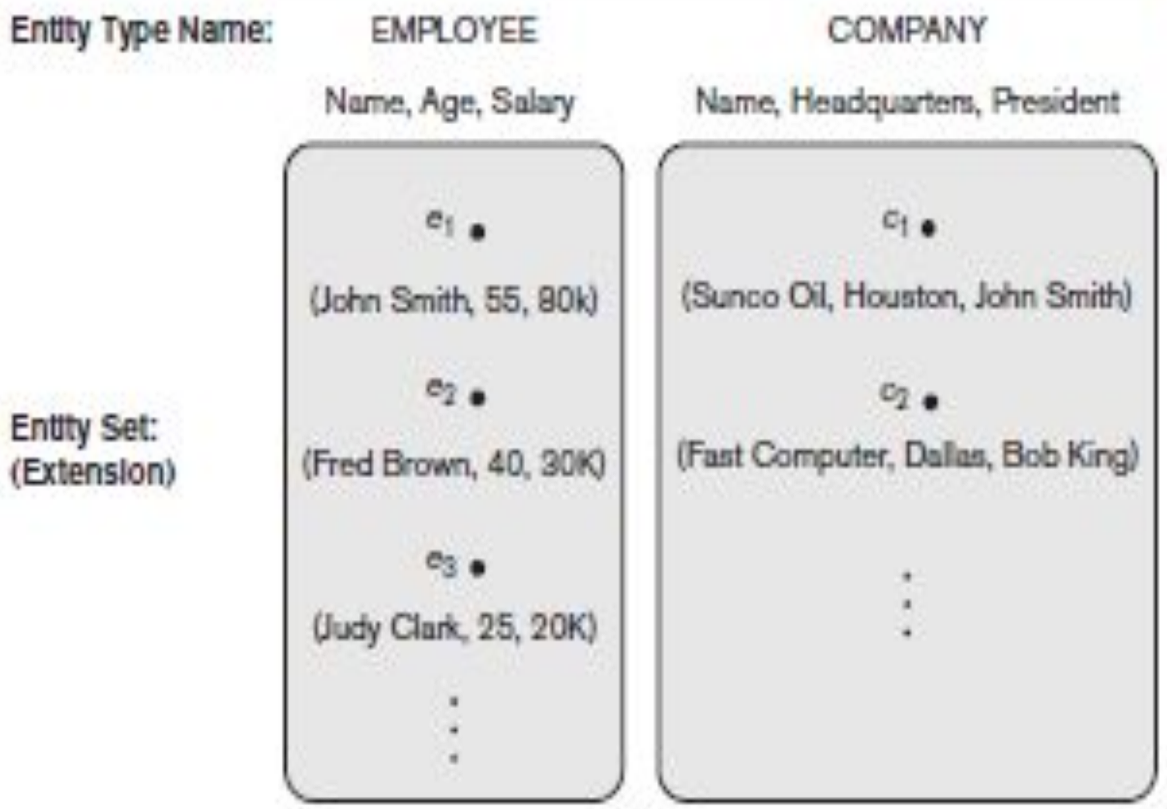


Figure 3.6
Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

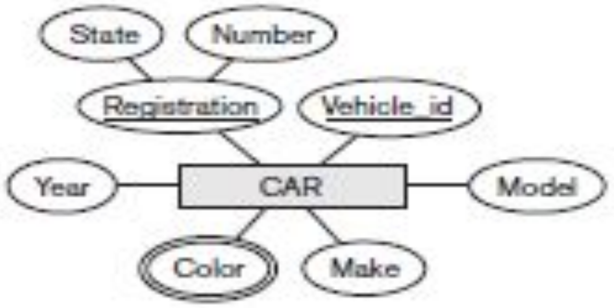
Entity Set

- 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.

(a)



(b)

CAR
Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}

CAR$_1$
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR$_2$
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR$_3$
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

⋮

Figure 3.7
The CAR entity type with two key attributes, Registration and Vehicle_id. (a) ER diagram notation. (b) Entity set with three entities.

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

Entity Type

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

car$_1$
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 1999, (red, black))
car$_2$
((ABC 123, NEW YORK), WP9872, Nissan 300ZX, 2-door, 2002, (blue))
car$_3$
((VSY 720, TEXAS), TD729, Buick LeSabre, 4-door, 2003, (white, blue))

.

.

.

Entity Set

Initial Conceptual Design of the COMPANY Database

1. **An entity type DEPARTMENT with attributes Name, Number, Locations,** Manager, and Manager_start_date. Locations is the only multivalued attribute.We can specify that both Name and Number are (separate) key attributes because each was specified to be unique.

2. An entity type PROJECT with attributes Name, Number, Location, and Controlling_department. Both Name and Number are (separate) key attributes.

3. An entity type EMPLOYEE with attributes Name, Ssn, Sex, Address, Salary, Birth_date, Department, and Supervisor. Both Name and Address may be composite attributes; however, this was not specified in the requirements. We must go back to the users to see if any of them will refer to the individual components of Name—First_name, Middle_initial, Last_name—or of Address. In our example, Name is modeled as a composite attribute, whereas Address is not, presumably after consultation with the users.

4. An entity type DEPENDENT with attributes Employee, Dependent_name, Sex, Birth_date, and Relationship (to the employee).

Initial Conceptual Design of the COMPANY Database

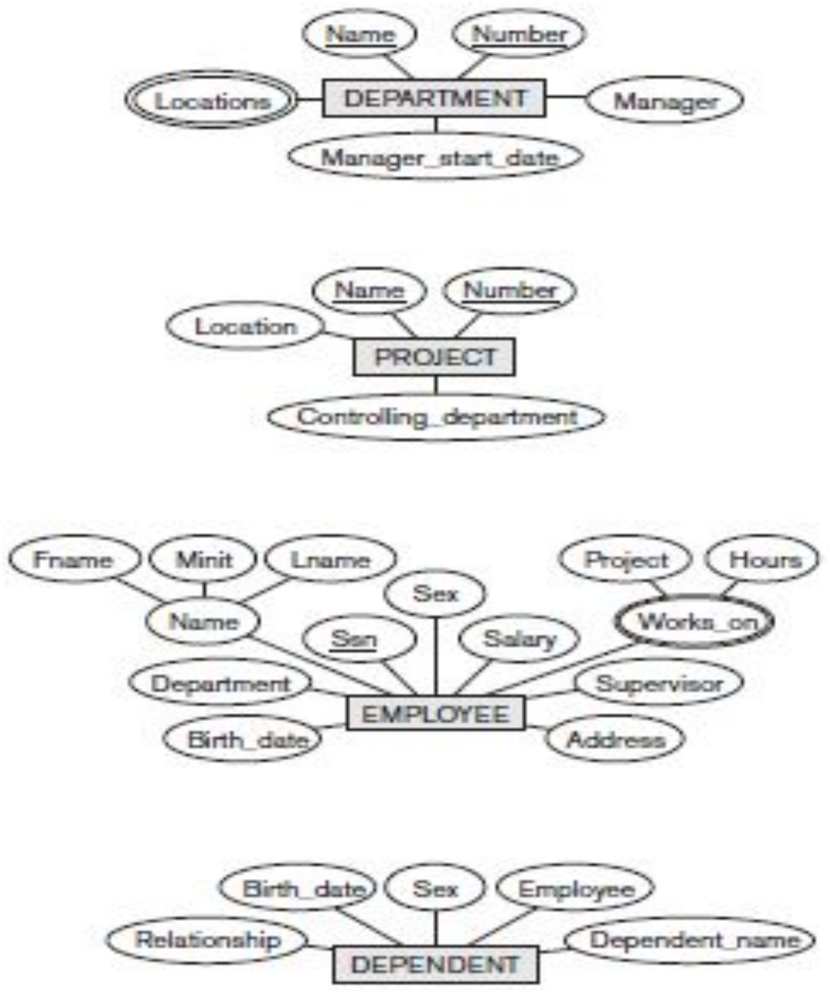


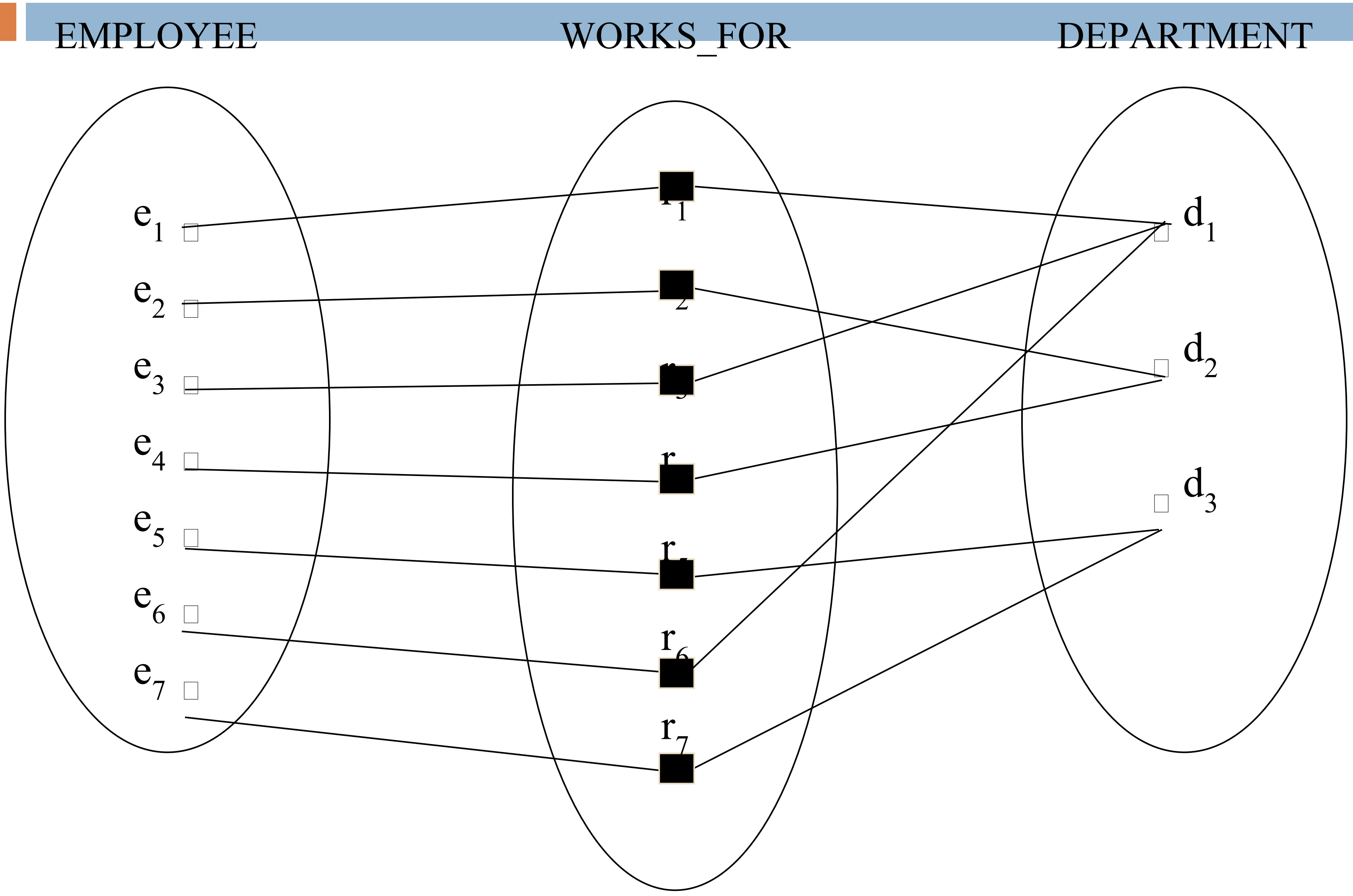
Figure 3.8
Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

Relationship Types, Relationship Sets, and Instances

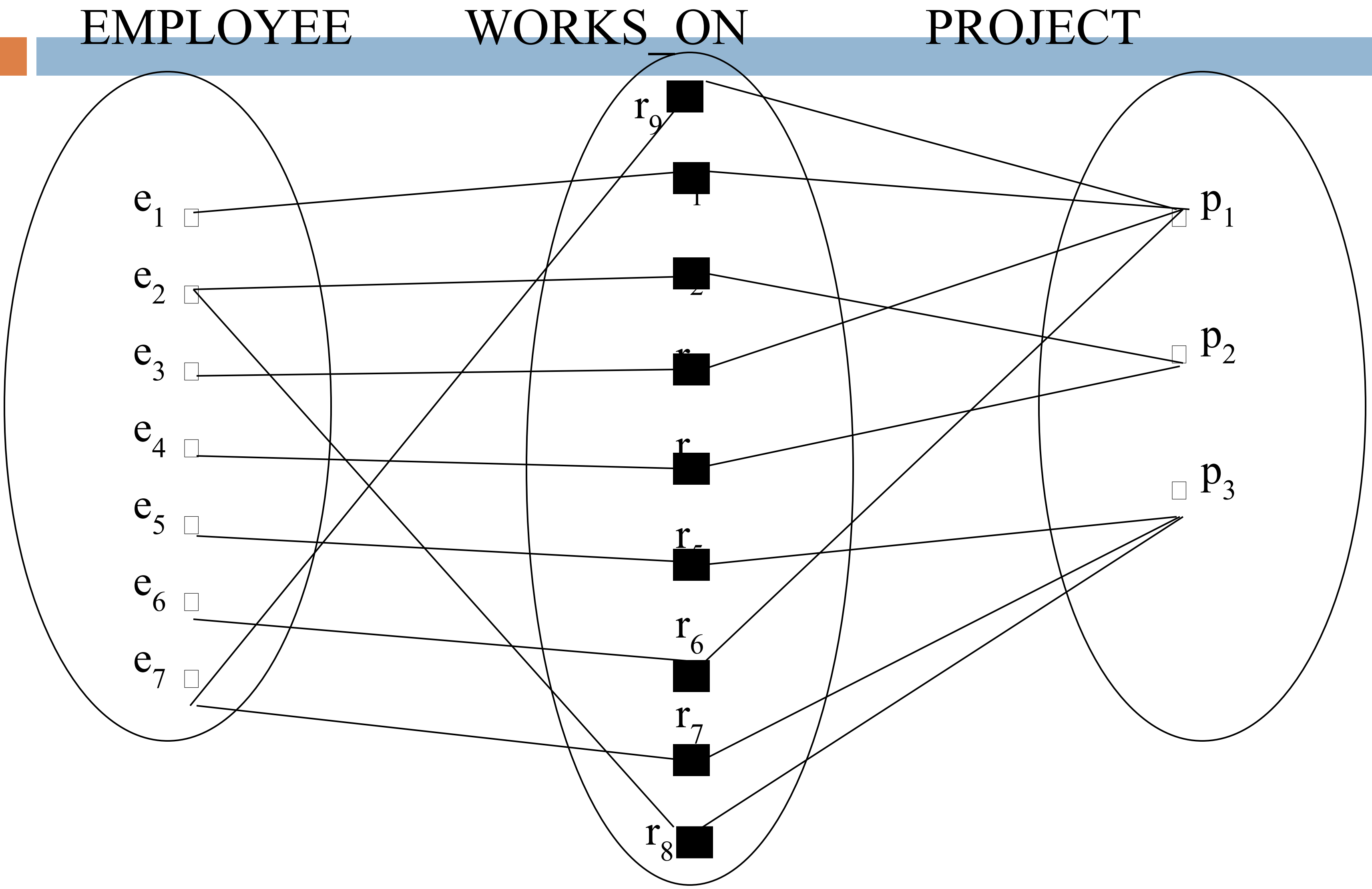
- A **relationship type** R among n entity types E_1, E_2, … E_n defines a set of associations – or a **relationship set** – among entities from these entity types.

- Mathematically, the relationship set R is a set of **relationship instances** r_i, where each r_i associates n individual entities.

Example 1…


EMPLOYEE
WORKS_FOR
DEPARTMENT
e1
e2
e3
e4
e5
e6
e7
r1
r2
r3
r4
r5
r6
r7
d1
d2
d3

Example 2…


EMPLOYEE
WORKS_ON
PROJECT
e_1
e_2
e_3
e_4
e_5
e_6
e_7
r_9
r_6
r_7
r_8
p_1
p_2
p_3

Relationship Degree

- 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**.

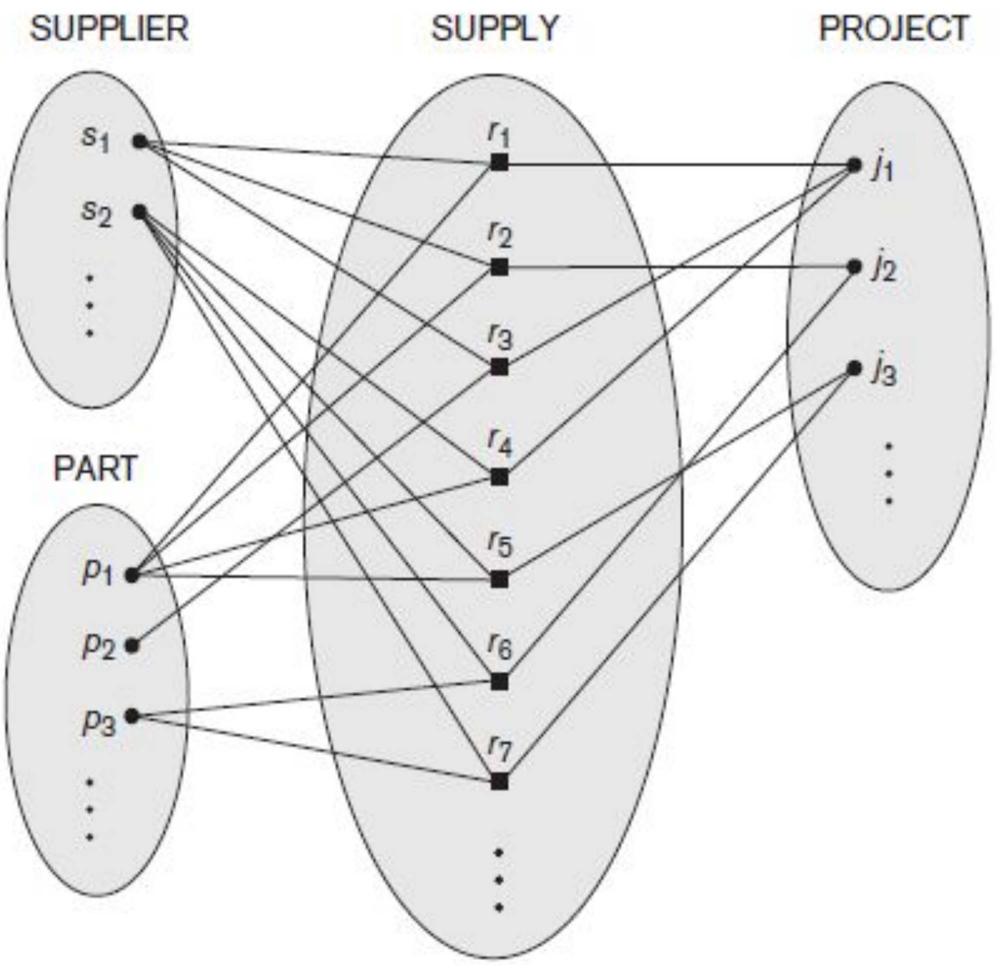


Figure 3.10
Some relationship instances in the SUPPLY ternary relationship set.

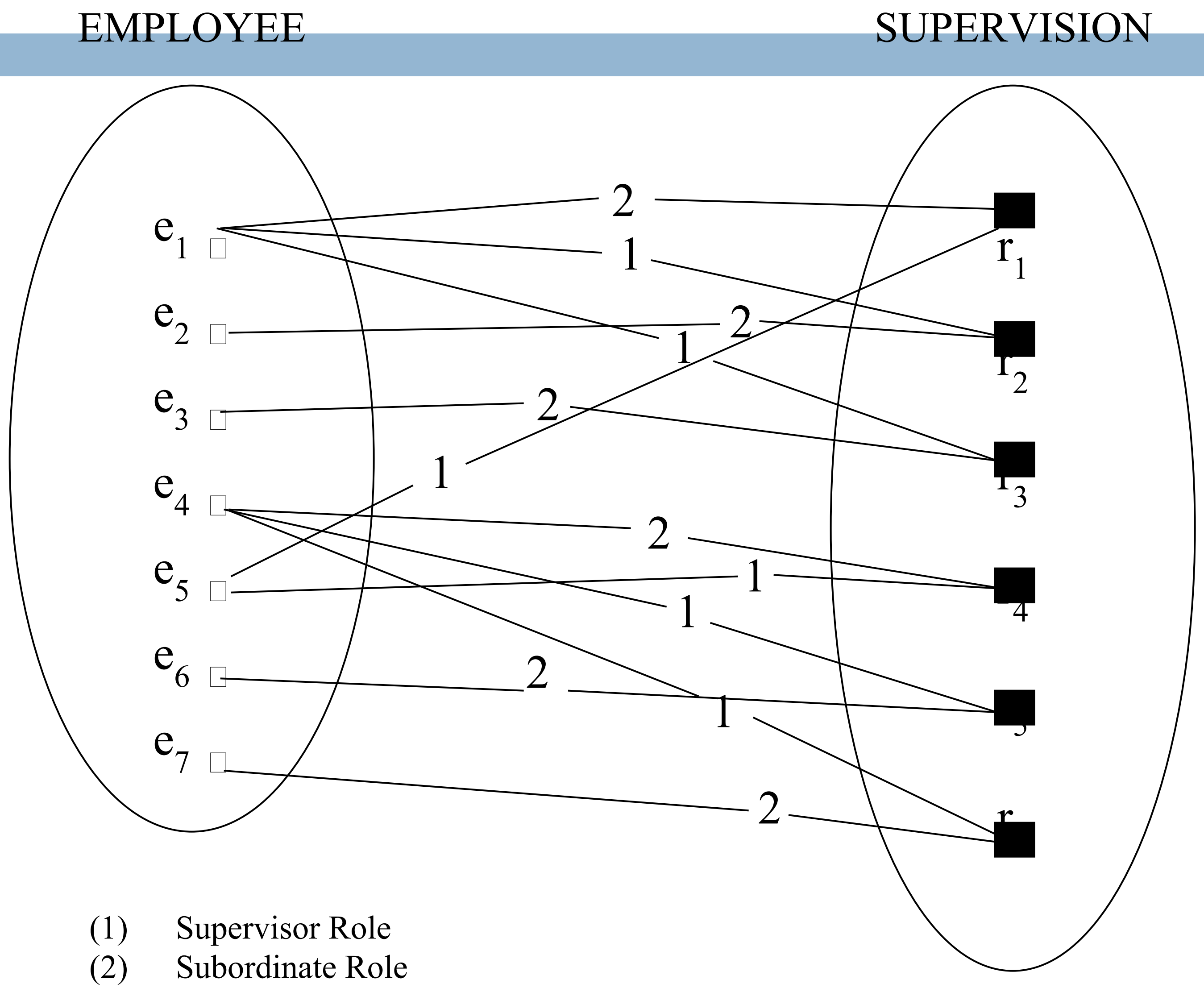
Role Names and Recursive Relationships

- 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…



(1) Supervisor Role
(2) Subordinate Role

Constraints on Relationship Types

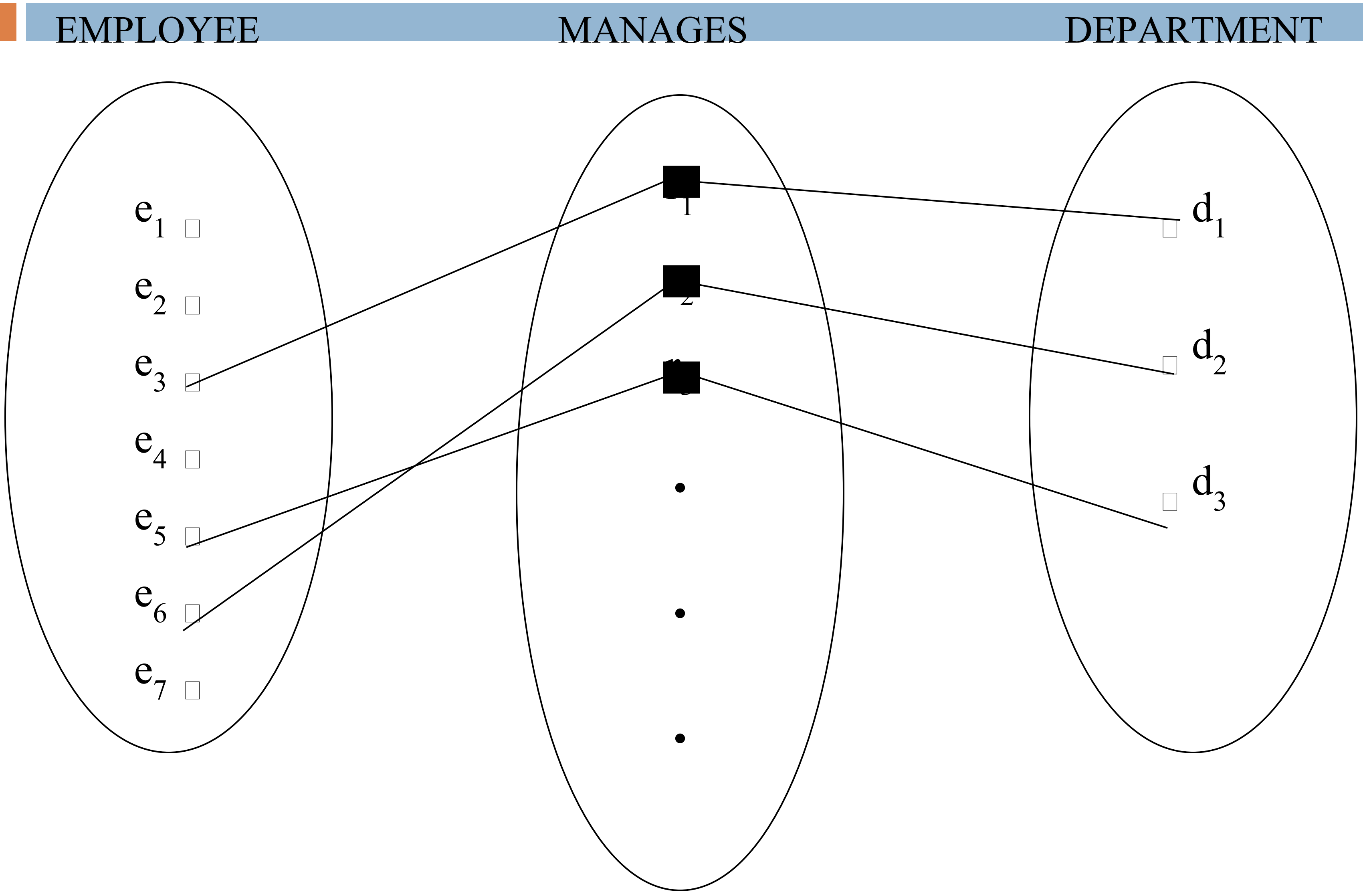
- There are two types of relationship constraints:

  - 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.(Minimum Cardinality ratio)

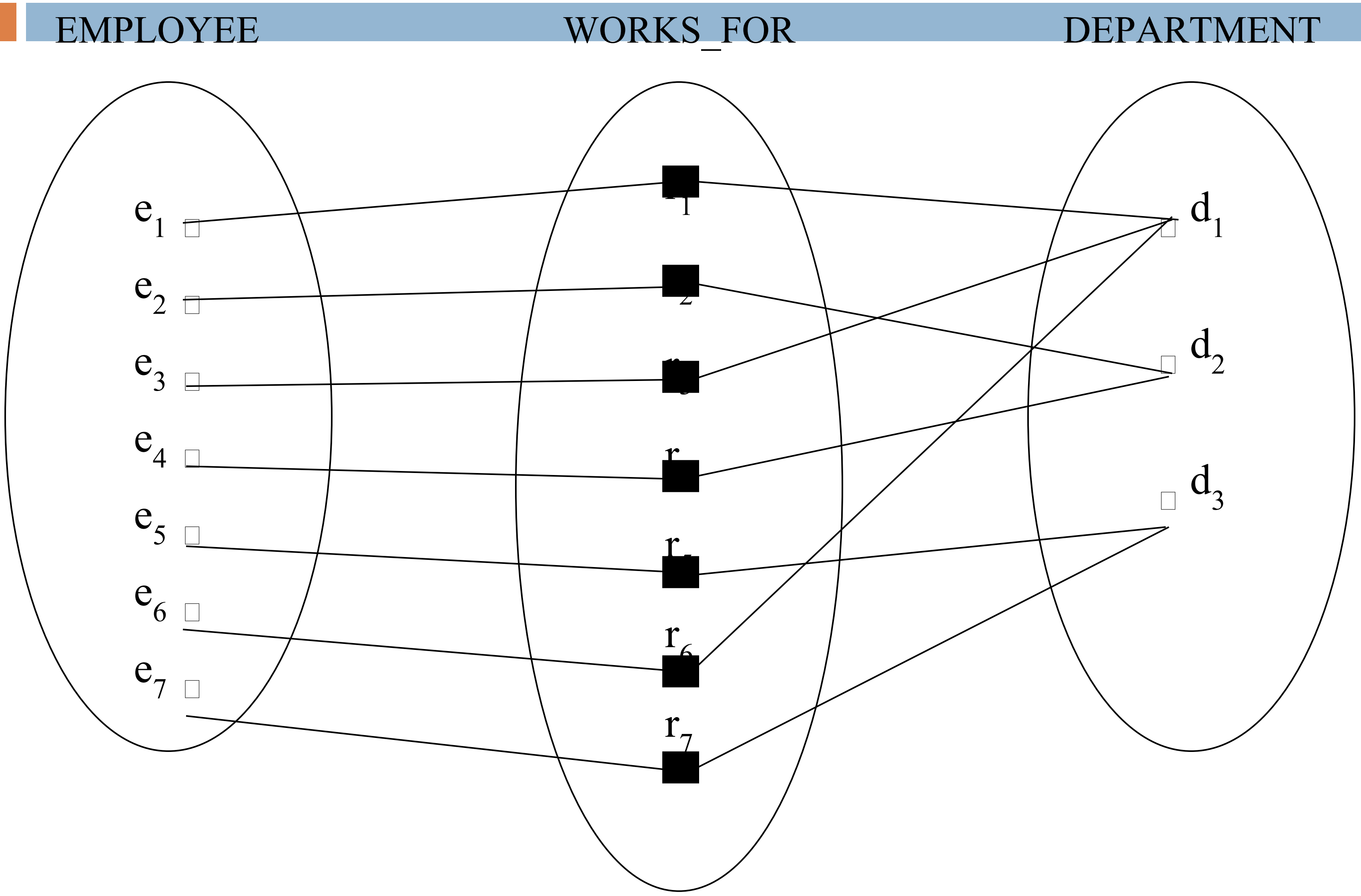
      1. total (Existence dependency) 2. Partial

- The cardinality ratio and participation constraint together called as **structural constraint**.

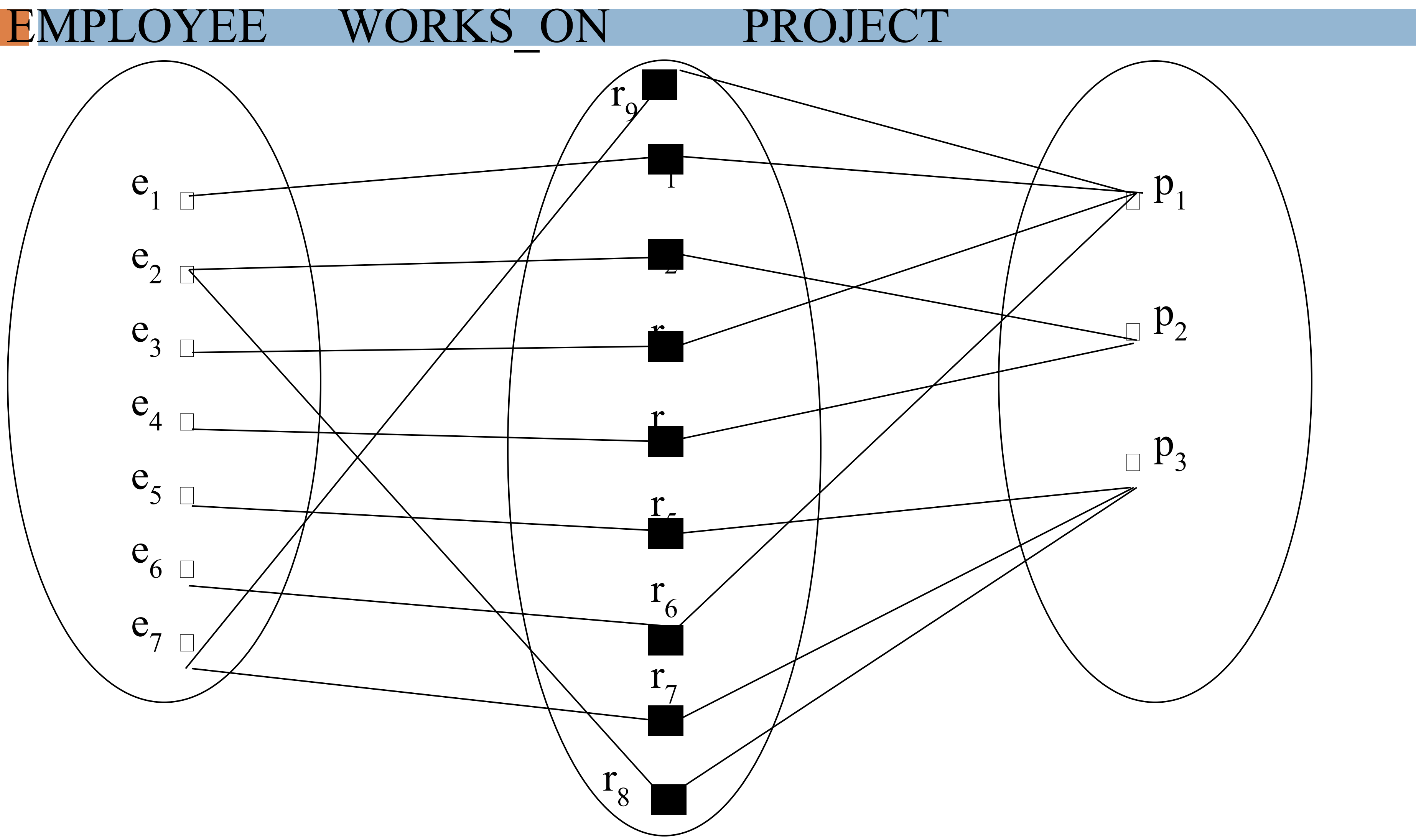
1:1 Relationship



1:N Relationship



M:N Relationship



Participation Constraint and Existence Dependence

- 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

- 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…

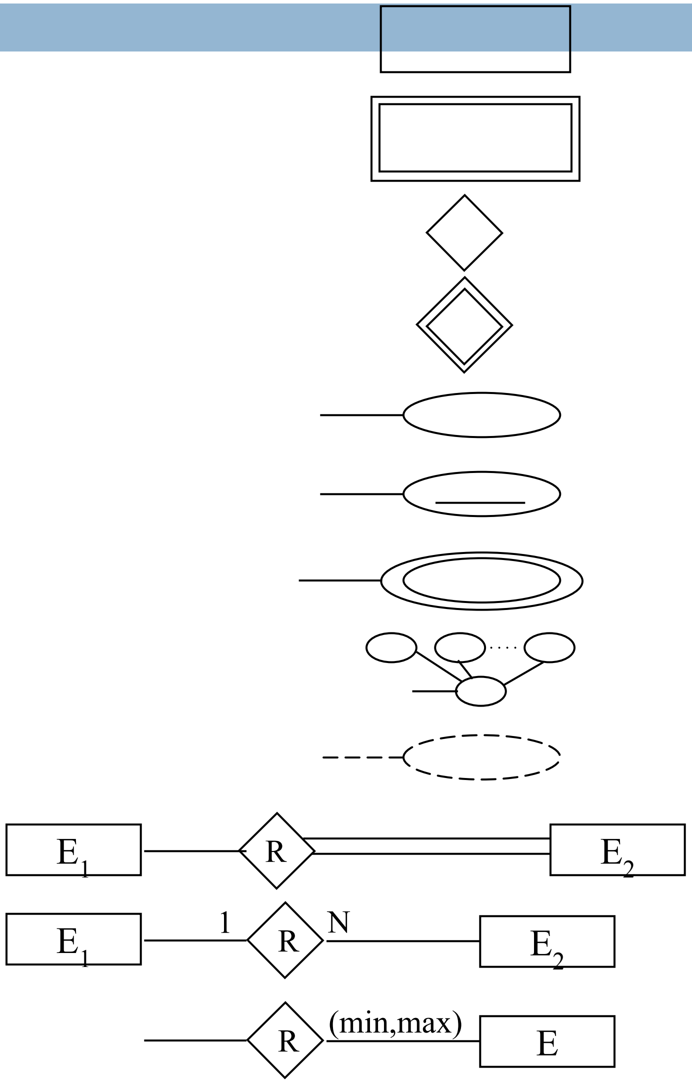
Example:

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

Symbol



Meaning

ENTITY TYPE

WEAK ENTITY TYPE

RELATIONSHIP TYPE

IDENTIFYING RELATIONSHIP TYPE

ATTRIBUTE

KEY ATTRIBUTE

MULTIVALUED ATTRIBUTE

COMPOSITE ATTRIBUTE

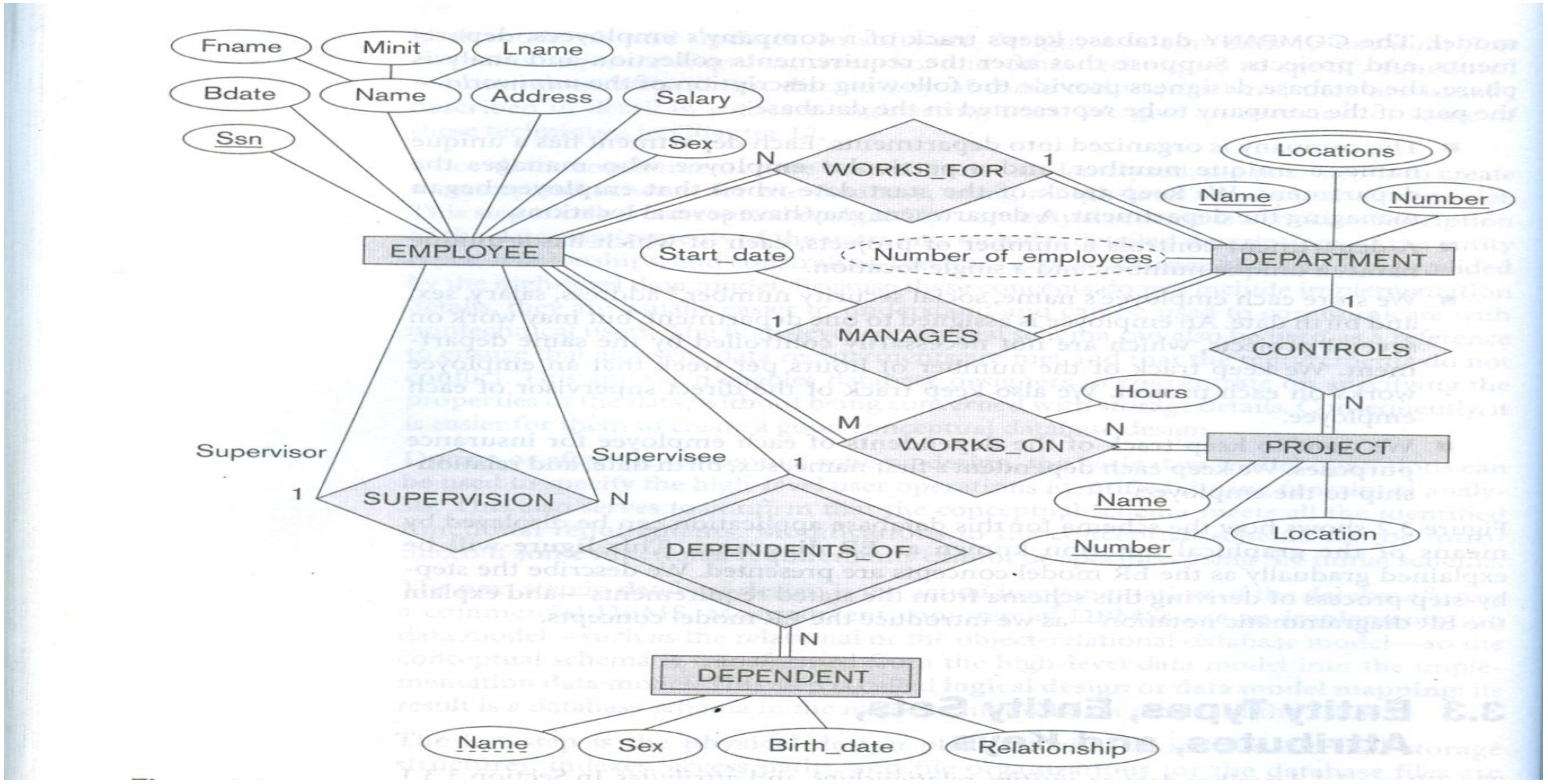
DERIVED ATTRIBUTE

TOTAL PARTICIPATION OF E_2 IN R

CARDINALITY RATIO 1:N FOR E_1:E_2 IN R

STRUCTURAL CONSTRAINT (min, max) ON PARTICIPATION OF E IN R

E – R Diagram for COMPANY

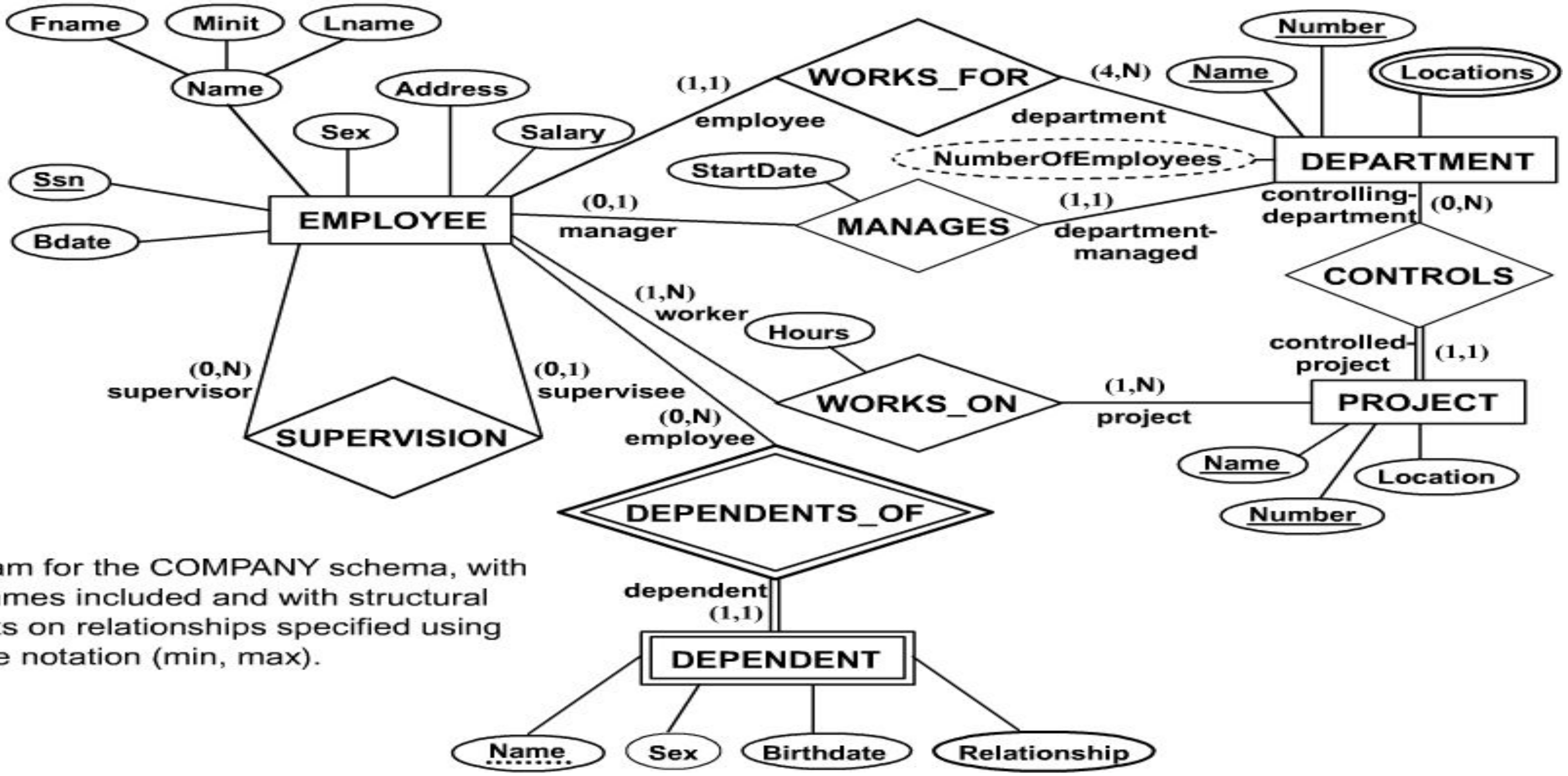


Proper Naming of Schema Constructs

- 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.

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).

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.