DATABASE SYSTEM CONCEPTS AND CONCEPTUAL MODELING, RELATIONAL DATA MODEL

Contents

- Basic Definitions
- Data Models
- Data Models and Their Categories
- Schemas, Instances, and States
- High level Conceptual model
- ER Model Concepts
  - Entities and Attributes
  - Entity Types, Value Sets, and Key Attributes
  - Relationships and Relationship Types
  - Weak Entity Types
  - Roles and Attributes in Relationship Types
  - ER Diagrams - Notation

Basic Definitions

Database:

- A collection of related data.

Data:

- Known facts that can be recorded and have an implicit meaning.

Mini-world:

- 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 software package/ system to facilitate the creation and maintenance of a computerized database.

Database System:

- The DBMS software together with the data itself. Sometimes, the applications are also included.

Impact of Databases and Database Technology

- Businesses: Banking, Insurance, Retail, Transportation, Healthcare, Manufacturing
- Service Industries: Financial, Real-estate, Legal, Electronic Commerce, Small businesses
- Education : Resources for content and Delivery
- More recently: Social Networks, Environmental and Scientific Applications, Medicine and Genetics
- Personalized Applications: based on smart mobile devices

Data Models

- **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 Model Structure and Constraints:**
  - Constructs are used to define the database structure
  - Constructs typically include ***elements*** (and their ***data types***) as well as groups of elements (e.g. ***entity, record, table***), and ***relationships*** among such groups
  - Constraints specify some restrictions on valid data; these constraints must be enforced at all times

Data Models (continued)

- **Data Model Operations:**
  - These operations are used for specifying database *retrievals* and *updates* by referring to the constructs of the data model.
  - Operations on the data model may include ***basic model operations*** (e.g. generic insert, delete, update) and ***user-defined operations*** (e.g. compute_student_gpa, update_inventory)

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. These are usually specified in an ad-hoc manner through DBMS design and administration manuals
- **Implementation (representational) data models:**
  - Provide concepts that fall between the above two, used by many commercial DBMS implementations (e.g. relational data models used in many commercial systems).
- **Self-Describing Data Models:**
  - Combine the description of data with the data values. Examples include XML, key-value stores and some NOSQL systems.

Schemas versus 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.

Schemas versus 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*

Database Schema vs. Database State

- Database State:
  - Refers to the ***content*** of a database at a moment in time.
- Initial Database State:
  - Refers to the database state when it is initially loaded into the system.
- Valid State:
  - A state that satisfies the structure and constraints of the database.

Database Schema vs. Database State (continued)

- Distinction
  - The ***database schema*** changes very infrequently.
  - The ***database state*** changes every time the database is updated.

- **Schema** is also called **intension**.
- **State** is also called **extension**.

Example of a Database Schema

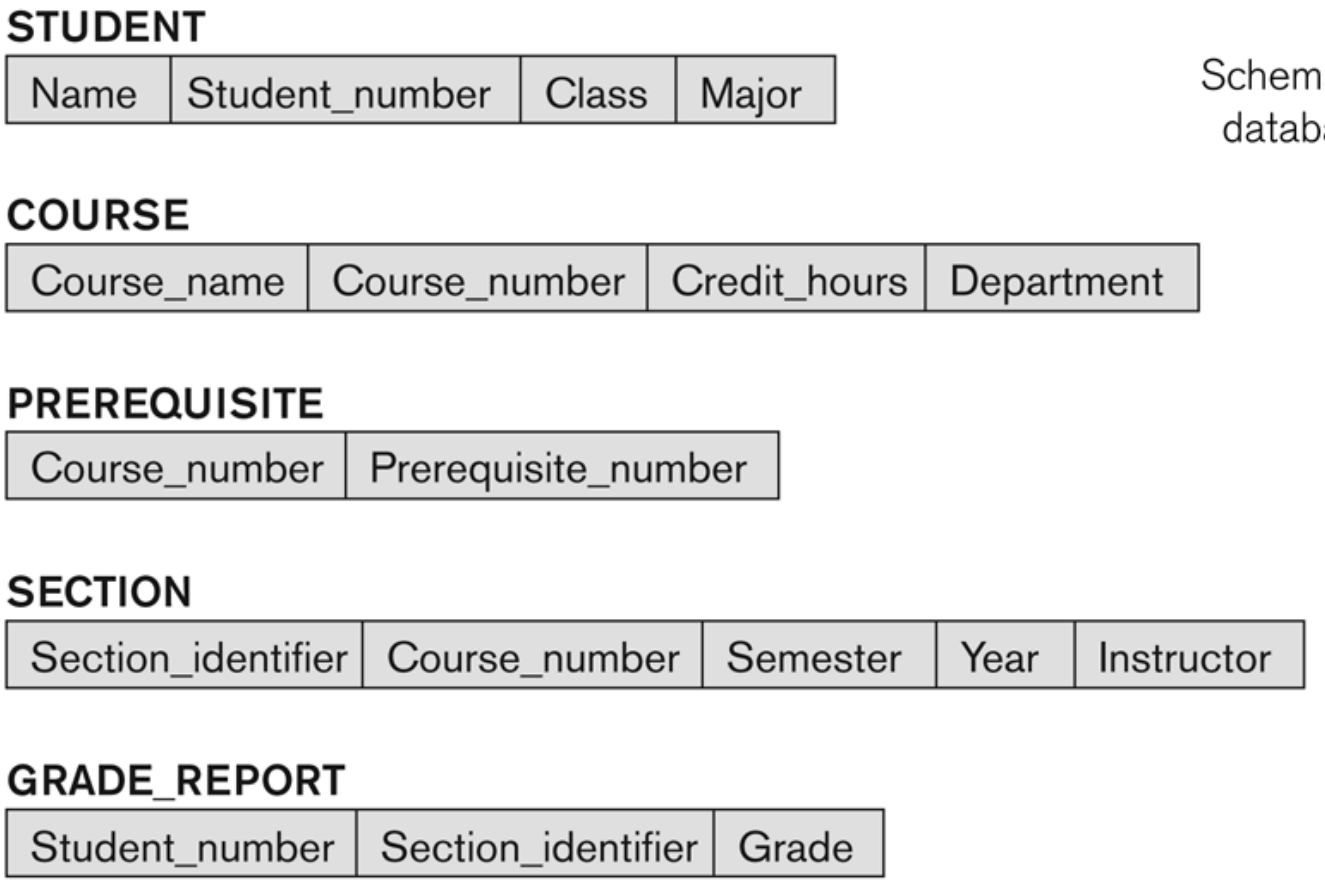


Figure 2.1
Schema diagram for the database in Figure 1.2.

Example of a database state

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

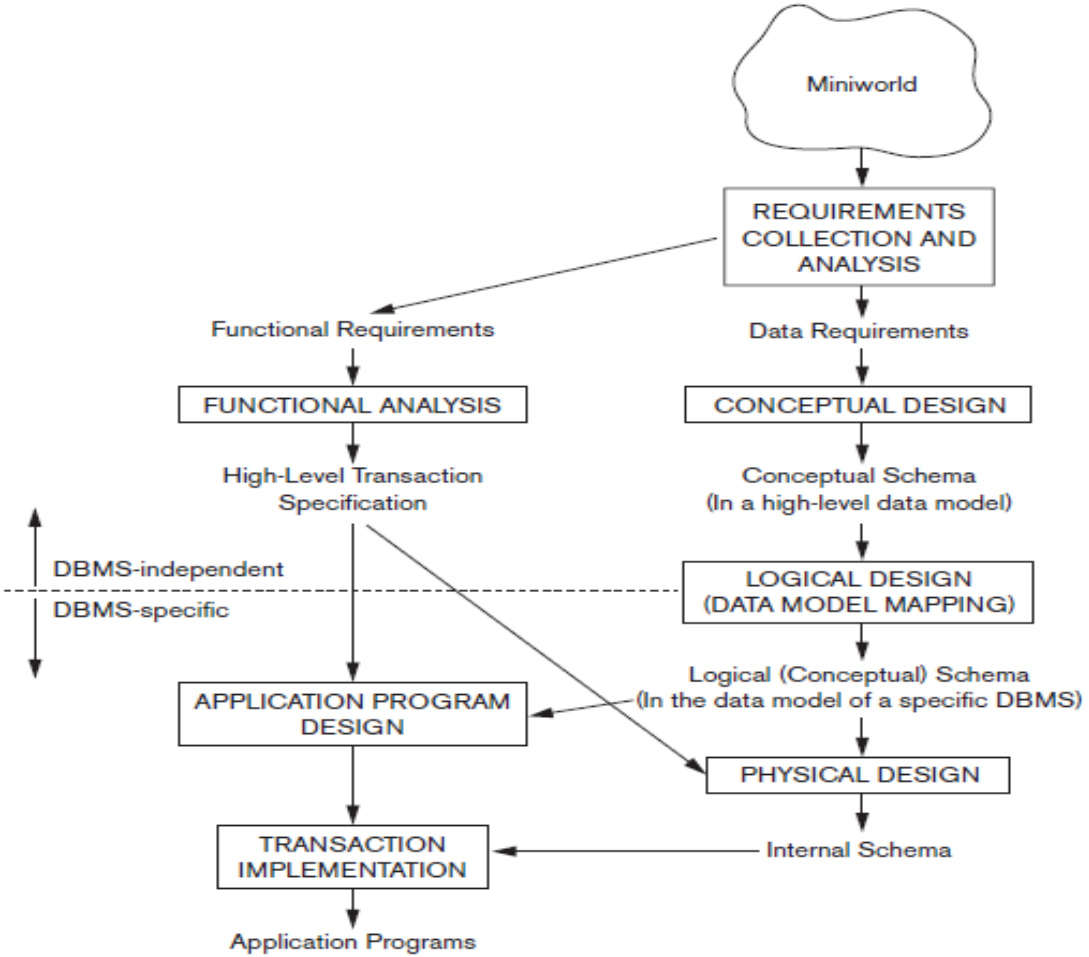
Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

High-Level Conceptual Data Models for Database Design

- A very important phase in designing a successful database application.
- The modeling concepts of the entity–relationship (ER) model, which is a popular high-level conceptual data model.
- Conceptual design
- Logical design
- Physical design

Database design process



Methodologies for Conceptual Design

- Entity Relationship (ER) Diagrams
- Enhanced Entity Relationship (EER) Diagrams
- Use of Design Tools in industry for designing and documenting large scale designs
- The UML (Unified Modeling Language) Class Diagrams are popular in industry to document conceptual database designs

Example COMPANY Database

- 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.

Example COMPANY Database (Continued)

- The database will store each EMPLOYEE's social security number, address, salary, sex, and birthdate.
  - Each employee *works for* one department but may *work on* several projects.
  - The DB will keep track of the number of hours per week that an employee currently works on each project.
  - It is required to keep track of the *direct supervisor* of each employee.
- Each employee may *have* a number of DEPENDENTs.
  - For each dependent, the DB keeps a record of name, sex, birthdate, and relationship to the employee.

ER Model Concepts

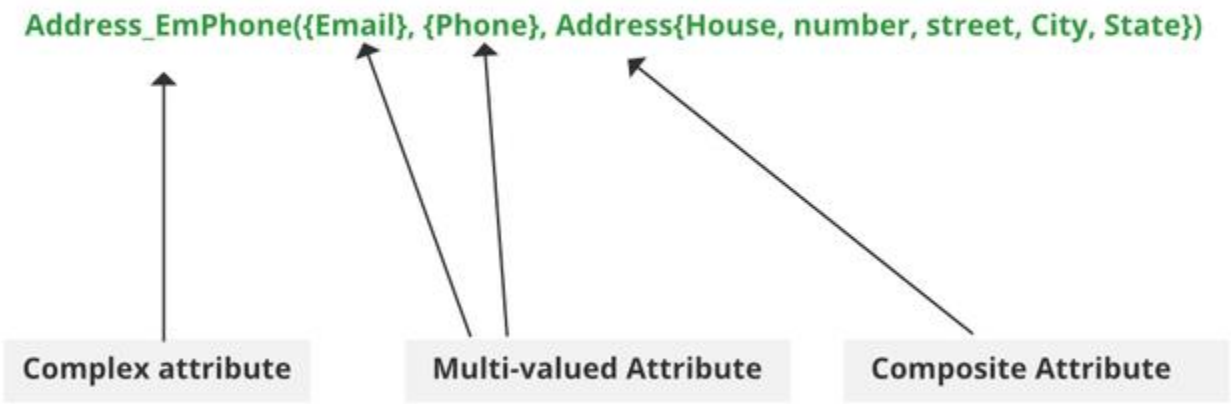
- Entities and Attributes
  - Entity is a basic concept for the ER model. Entities are specific things or objects in the mini-world that are represented in the database.
    - For example the EMPLOYEE John Smith, the Research DEPARTMENT, the ProductX PROJECT
  - Attributes are properties used to describe an entity.
    - For example an EMPLOYEE entity may have the attributes Name, SSN, Address, Sex, BirthDate
  - A specific entity will have a value for each of its attributes.
    - For example a specific employee entity may have Name='John Smith', SSN='123456789', Address ='731, Fondren, Houston, TX', Sex='M', BirthDate='09-JAN-55‘
  - Each attribute has a *value set* (or data type) associated with it – e.g. integer, string, date, enumerated type, …

Types of Attributes (1)

- Simple
  - Each entity has a single atomic value for the attribute. For example, SSN or Zipcode.
- Composite
  - The attribute may be composed of several components. For example:
    - Address(Apt#, House#, Street, City, State, ZipCode, Country), or
    - Name(FirstName, MiddleName, LastName).
    - Composition may form a hierarchy where some components are themselves composite.
- Multi-valued
  - An entity may have multiple values for that attribute. For example, Color of a CAR or PreviousDegrees of a STUDENT.
    - Denoted as {Color} or {PreviousDegrees}.

- Several types of attributes occur in the ER model: simple versus composite, single valued versus multivalued, and stored versus derived.

Complex Attribute



Types of Attributes (2)

- In general, composite and multi-valued attributes may be nested arbitrarily to any number of levels, although this is rare.
  - For example, PreviousDegrees of a STUDENT is a composite multi-valued attribute denoted by {PreviousDegrees (College, Year, Degree, Field)}
  - Multiple PreviousDegrees values can exist
  - Each has four subcomponent attributes:
    - College, Year, Degree, Field

Example of a composite attribute

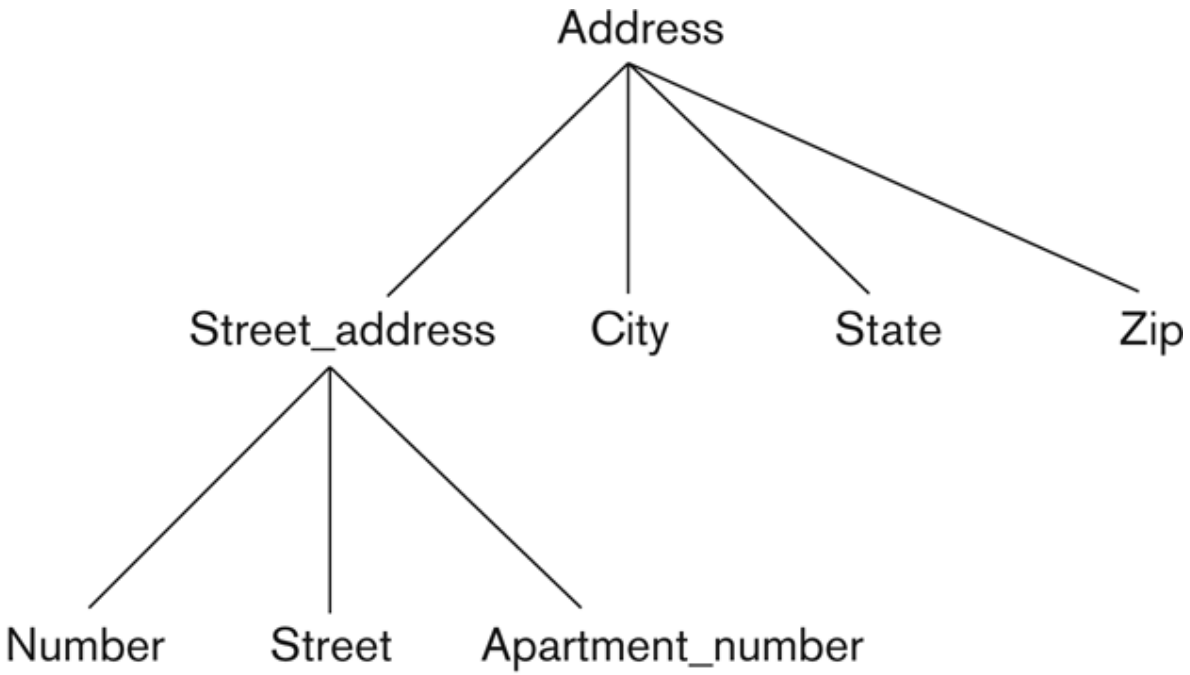


Figure 3.4
A hierarchy of composite attributes.

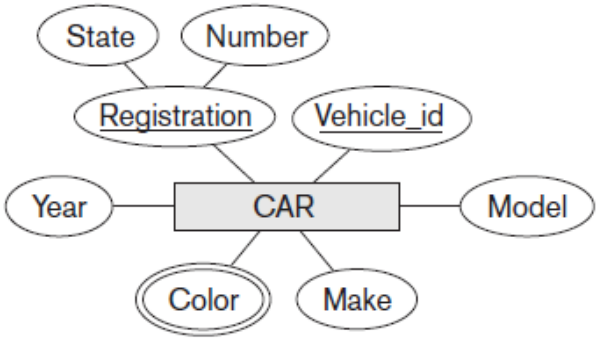
Entity Types and Key Attributes (1)

- Entities with the same basic attributes are grouped or typed into an entity type.
  - For example, the entity type EMPLOYEE and PROJECT.
- An attribute of an entity type for which each entity must have a unique value is called a key attribute of the entity type.
  - For example, SSN of EMPLOYEE.

Entity Types and Key Attributes (2)

- A key attribute may be composite.
  - VehicleTagNumber is a key of the CAR entity type with components (Number, State).
- An entity type may have more than one key.
  - The CAR entity type may have two keys:
    - VehicleIdentificationNumber (popularly called VIN)
    - VehicleTagNumber (Number, State), aka license plate number.
- Each key is underlined (Note: this is different from the relational schema where only one “primary key is underlined).

(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.

Entity Set

- Each entity type will have a collection of entities stored in the database
  - Called the **entity set** or sometimes **entity collection**
- Previous slide shows three CAR entity instances in the entity set for CAR
- Same name (CAR) used to refer to both the entity type and the entity set
- However, entity type and entity set may be given different names
- Entity set is the current *state* of the entities of that type that are stored in the database

Entity Sets -- instructor and student

76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

instructor

98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

student

Value Sets (Domains) of Attributes

- Each simple attribute is associated with a value set
  - E.g., Lastname has a value which is a character string of upto 15 characters, say
  - Date has a value consisting of MM-DD-YYYY where each letter is an integer
- A **value set** specifies the set of values associated with an attribute

Attributes and Value Sets

- Value sets are similar to data types in most programming languages – e.g., integer, character (n), real, bit
- Mathematically, an attribute A for an entity type E whose value set is V is defined as a function

  A : E -> P(V)

Where P(V) indicates a power set (which means all possible subsets) of V. The above definition covers simple and multivalued attributes.

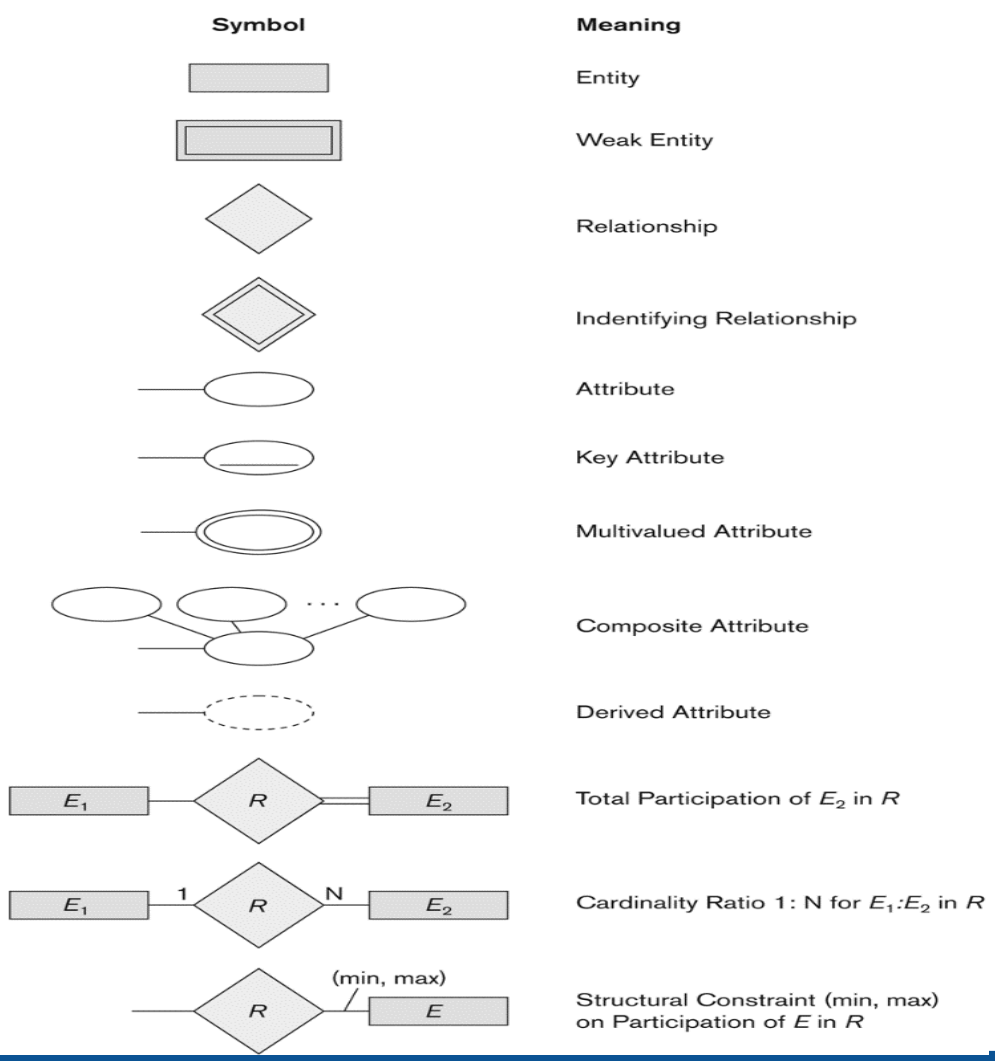
- We refer to the value of attribute A for entity e as A(e).

Displaying an Entity type

- In ER diagrams, an entity type is displayed in a rectangular box
- Attributes are displayed in ovals
  - Each attribute is connected to its entity type
  - Components of a composite attribute are connected to the oval representing the composite attribute
  - Each key attribute is underlined
  - Multivalued attributes displayed in double ovals
- See the full ER notation in advance on the next slide

NOTATION for ER diagrams

Figure 3.14
Summary of the notation for ER diagrams.



Entity Type CAR with two keys and a corresponding Entity Set

(a)

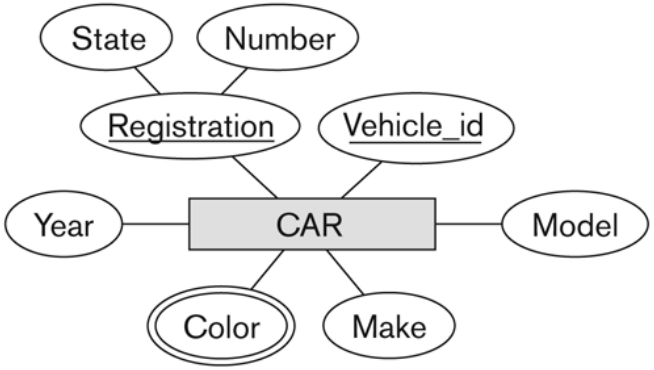


Figure 3.7
The CAR entity type with two key attributes, Registration and Vehicle_id. (a) ER diagram notation. (b) Entity set with three entities.

(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})

Initial Conceptual Design of Entity Types for the COMPANY Database Schema

- Based on the requirements, we can identify four initial entity types in the COMPANY database:
  - DEPARTMENT
  - PROJECT
  - EMPLOYEE
  - DEPENDENT

Initial Design of Entity Types:
EMPLOYEE, DEPARTMENT, PROJECT, DEPENDENT

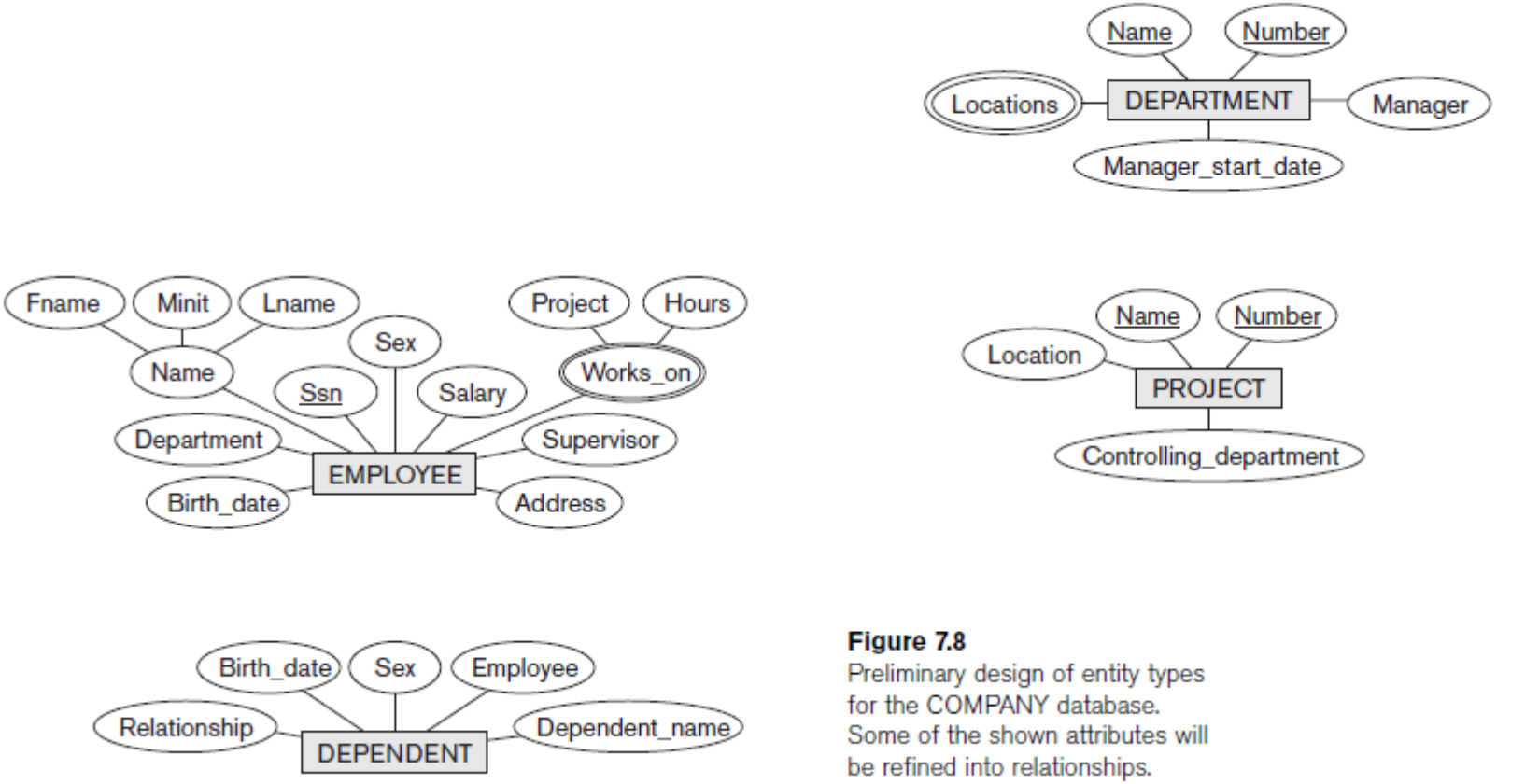


Figure 7.8
Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

Relationship Types, Relationship Sets, Roles, and Structural Constraints

Refining the initial design by introducing relationships

- The initial design is typically not complete
- Some aspects of the requirements will be represented as **relationships**
- The ER model has three main concepts:
  - Entities (and their entity types and entity sets)
  - Attributes (simple, composite, multivalued)
  - Relationships (and their relationship types and relationship sets)

Relationships and Relationship Types (1)

- A **relationship** relates two or more distinct entities with a specific meaning.
  - For example, EMPLOYEE John Smith *works on* the ProductX PROJECT, or EMPLOYEE Franklin Wong *manages* the Research DEPARTMENT.
- Relationships of the same type are grouped or typed into a **relationship type**.
  - For example, the WORKS_ON relationship type in which EMPLOYEEs and PROJECTs participate, or the MANAGES relationship type in which EMPLOYEEs and DEPARTMENTs participate.
- The degree of a relationship type is the number of participating entity types.
  - Both MANAGES and WORKS_ON are *binary* relationships.

Relationship instances of the WORKS_FOR N:1 relationship between EMPLOYEE and DEPARTMENT

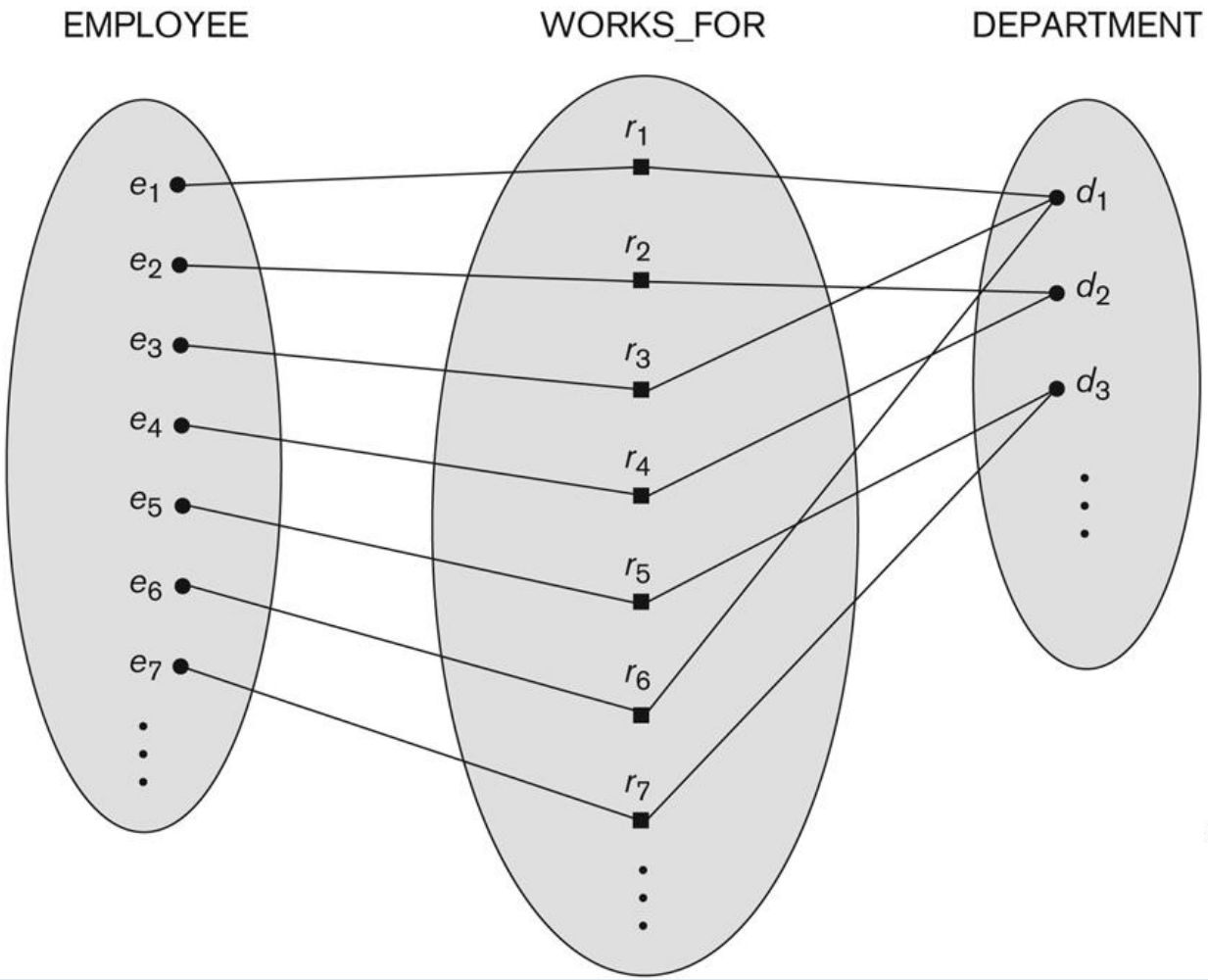


Figure 3.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Relationship instances of the M:N WORKS_ON relationship between EMPLOYEE and PROJECT

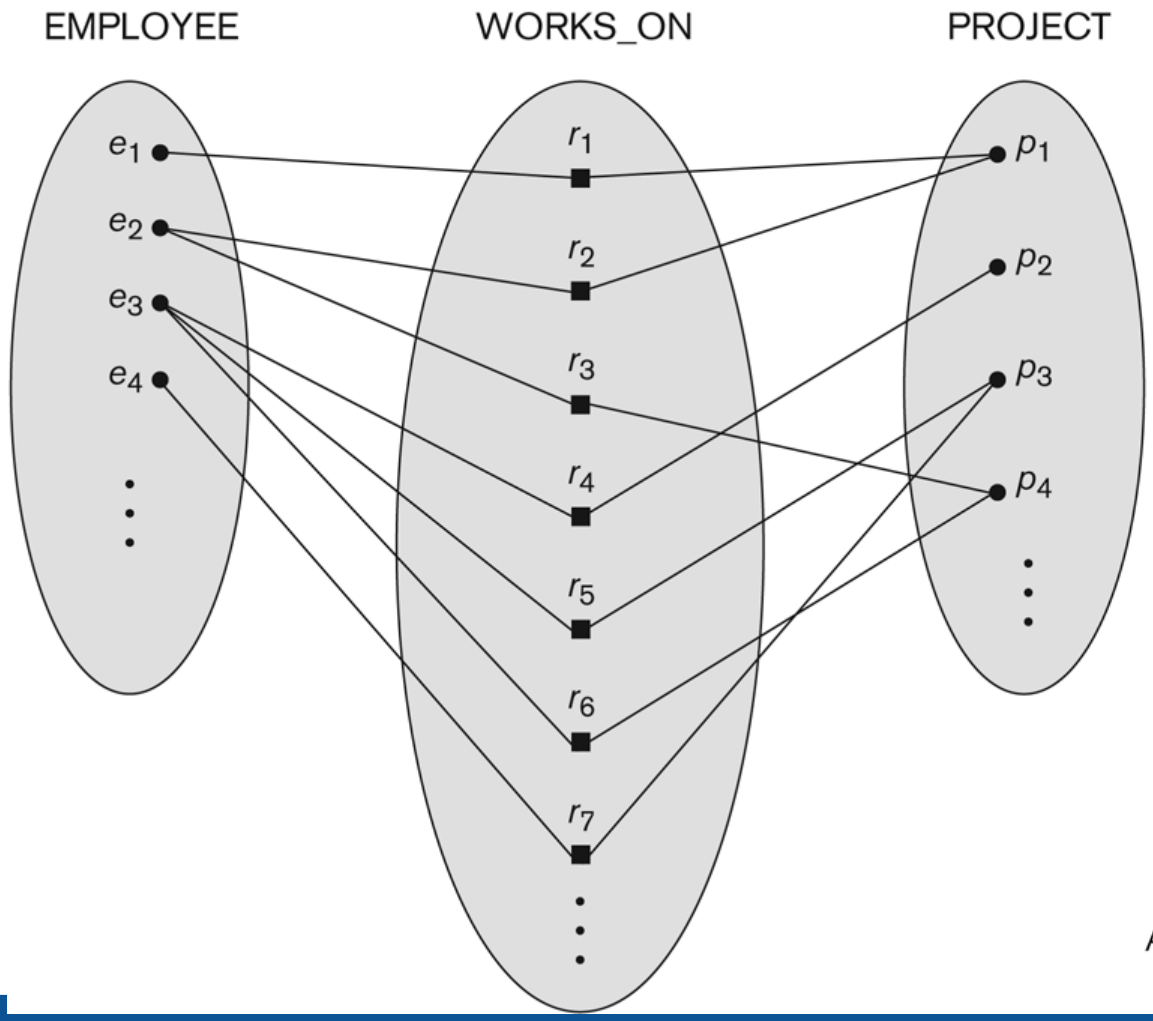


Figure 3.13
An M:N relationship, WORKS_ON.

Relationship type vs. relationship set (1)

- Relationship Type:
  - Is the schema description of a relationship
  - Identifies the relationship name and the participating entity types
  - Also identifies certain relationship constraints
- Relationship Set:
  - The current set of relationship instances represented in the database
  - The current *state* of a relationship type

Relationship type vs. relationship set (2)

- Each instance in the set relates individual participating entities – one from each participating entity type

- In ER diagrams, we represent the *relationship type* as follows:
  - Diamond-shaped box is used to display a relationship type
  - Connected to the participating entity types via straight lines
  - Note that the relationship type is not shown with an arrow. The name should be typically be readable from left to right and top to bottom.

Refining the COMPANY database schema by introducing relationships

- By examining the requirements, six relationship types are identified
- All are *binary* relationships(degree 2)
- Listed below with their participating entity types:
  - WORKS_FOR (between EMPLOYEE, DEPARTMENT)
  - MANAGES (also between EMPLOYEE, DEPARTMENT)
  - CONTROLS (between DEPARTMENT, PROJECT)
  - WORKS_ON (between EMPLOYEE, PROJECT)
  - SUPERVISION (between EMPLOYEE (as subordinate), EMPLOYEE (as supervisor))
  - DEPENDENTS_OF (between EMPLOYEE, DEPENDENT)

ER DIAGRAM – Relationship Types are:

WORKS_FOR, MANAGES, WORKS_ON, CONTROLS, SUPERVISION, DEPENDENTS_OF

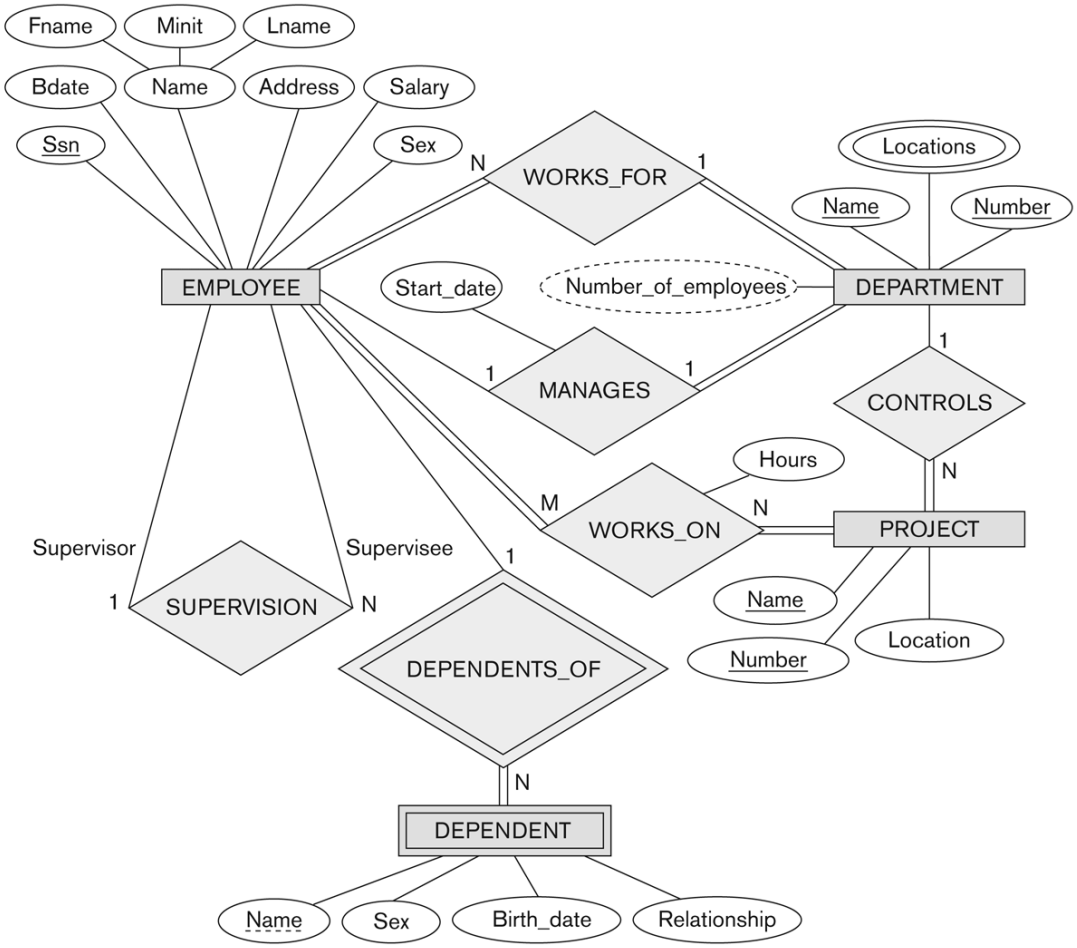


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Discussion on Relationship Types

- In the refined design, some attributes from the initial entity types are refined into relationships:
  - Manager of DEPARTMENT -> MANAGES
  - Works_on of EMPLOYEE -> WORKS_ON
  - Department of EMPLOYEE -> WORKS_FOR
  - etc
- In general, more than one relationship type can exist between the same participating entity types
  - MANAGES and WORKS_FOR are distinct relationship types between EMPLOYEE and DEPARTMENT
  - Different meanings and different relationship instances.

Constraints on Relationships

- Constraints on Relationship Types
  - (Also known as ratio constraints)
  - Cardinality Ratio (specifies *maximum* participation)
    - One-to-one (1:1)
    - One-to-many (1:N) or Many-to-one (N:1)
    - Many-to-many (M:N)
  - Existence Dependency Constraint (specifies *minimum* participation) (also called participation constraint)
    - zero (optional participation, not existence-dependent)
    - one or more (mandatory participation, existence-dependent)

Many-to-one (N:1) Relationship

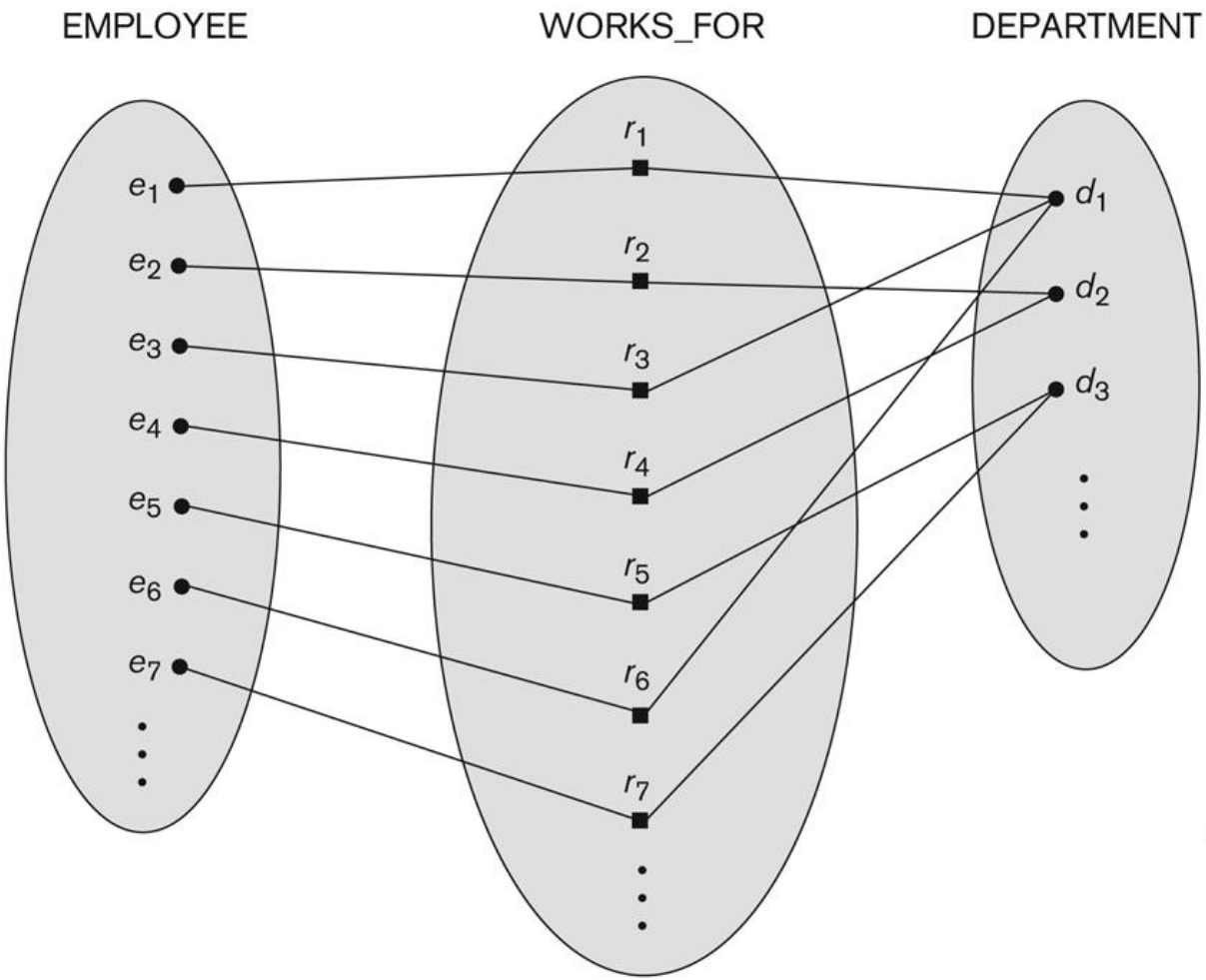


Figure 3.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Many-to-many (M:N) Relationship

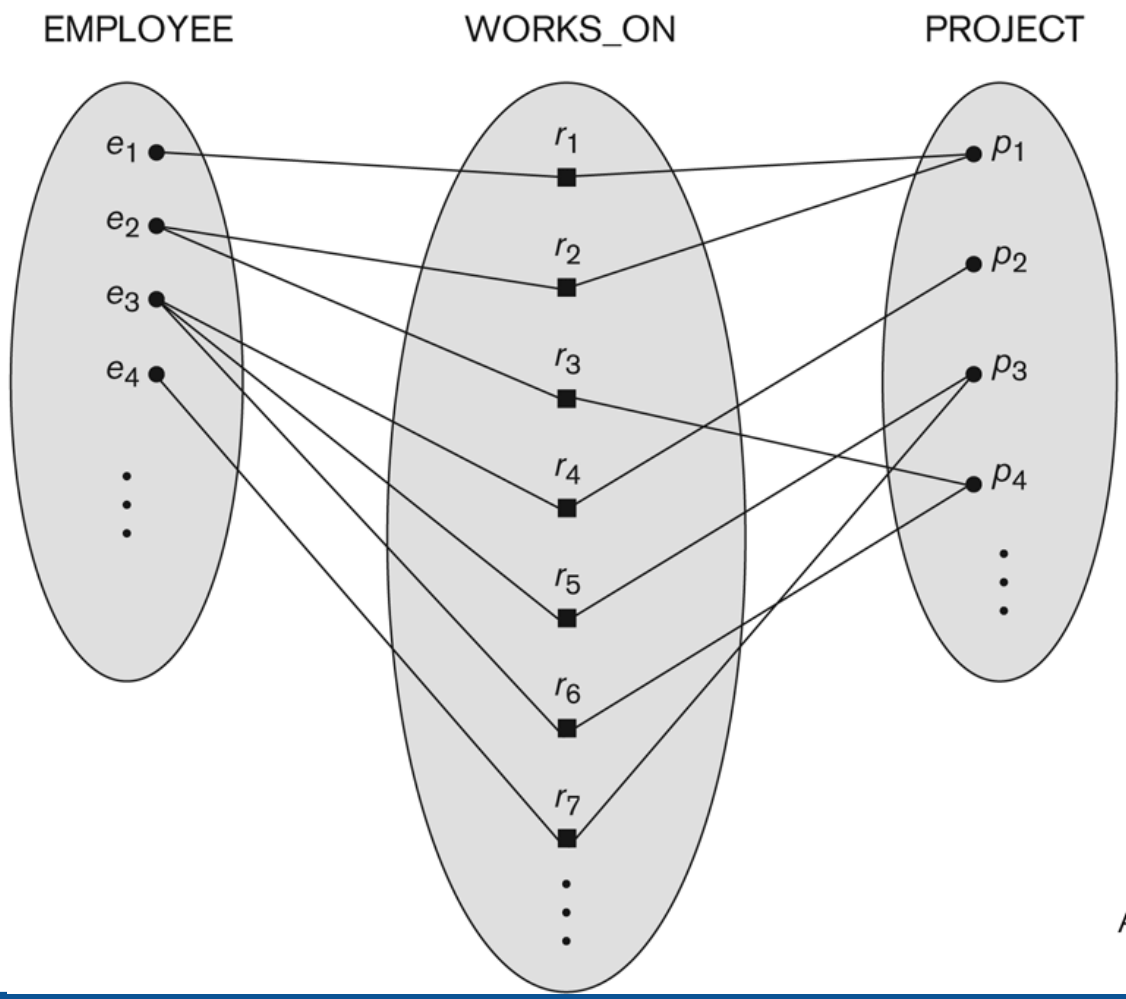


Figure 3.13
An M:N relationship, WORKS_ON.

Recursive Relationship Type

- A relationship type between the same participating entity type in **distinct roles**
- Also called a **self-referencing** relationship type.
- Example: the SUPERVISION relationship
- EMPLOYEE participates twice in two distinct roles:
  - supervisor (or boss) role
  - supervisee (or subordinate) role
- Each relationship instance relates two distinct EMPLOYEE entities:
  - One employee in *supervisor* role
  - One employee in *supervisee* role

Displaying a recursive relationship

- In a recursive relationship type.
  - Both participations are same entity type in different roles.
  - For example, SUPERVISION relationships between EMPLOYEE (in role of supervisor or boss) and (another) EMPLOYEE (in role of subordinate or worker).
- In following figure, first role participation labeled with 1 and second role participation labeled with 2.
- In ER diagram, need to display role names to distinguish participations.

A Recursive Relationship Supervision`

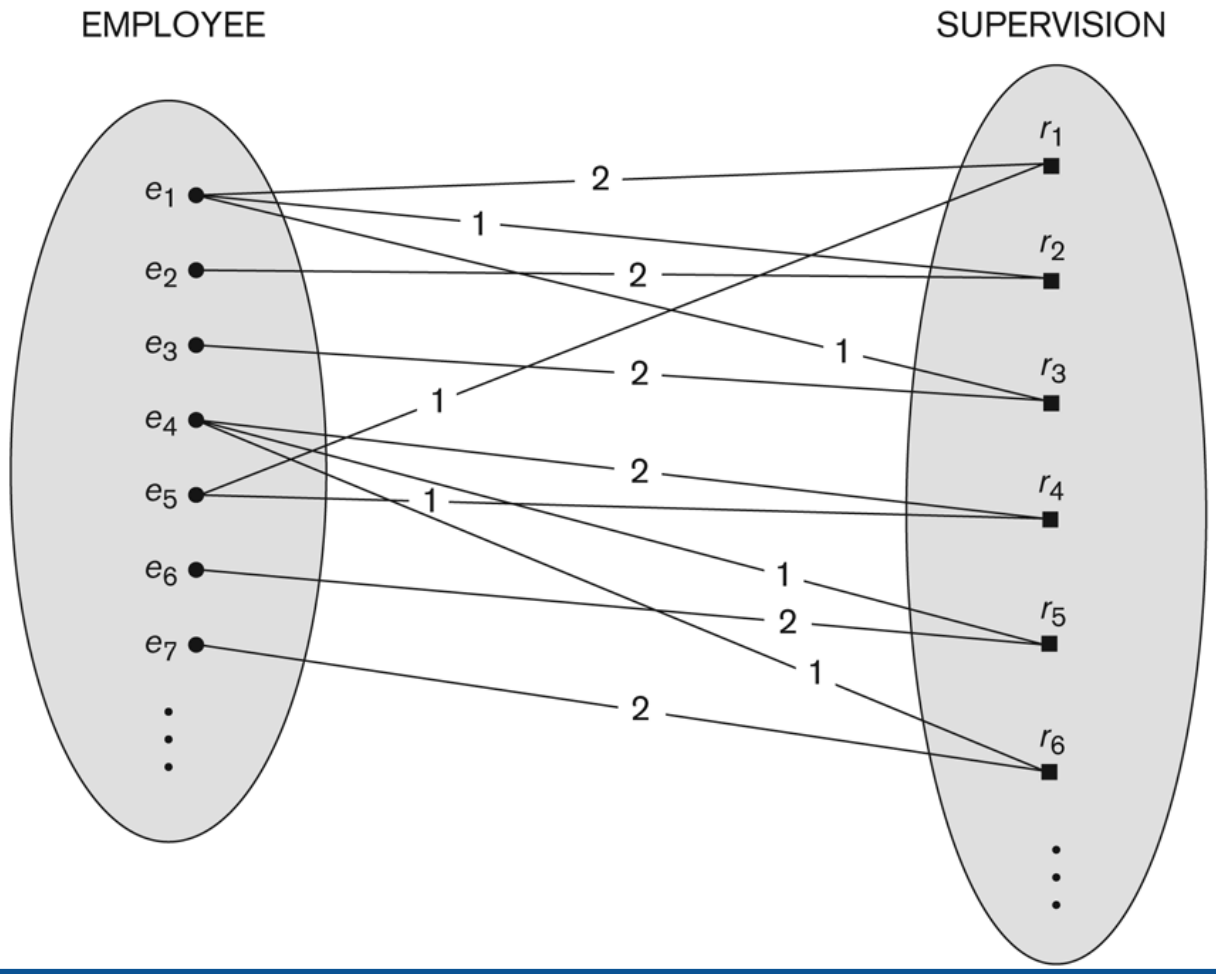


Figure 3.11
A recursive relationship SUPERVISION between EMPLOYEE in the *supervisor* role (1) and EMPLOYEE in the *subordinate* role (2).

Recursive Relationship Type is: SUPERVISION (participation role names are shown)

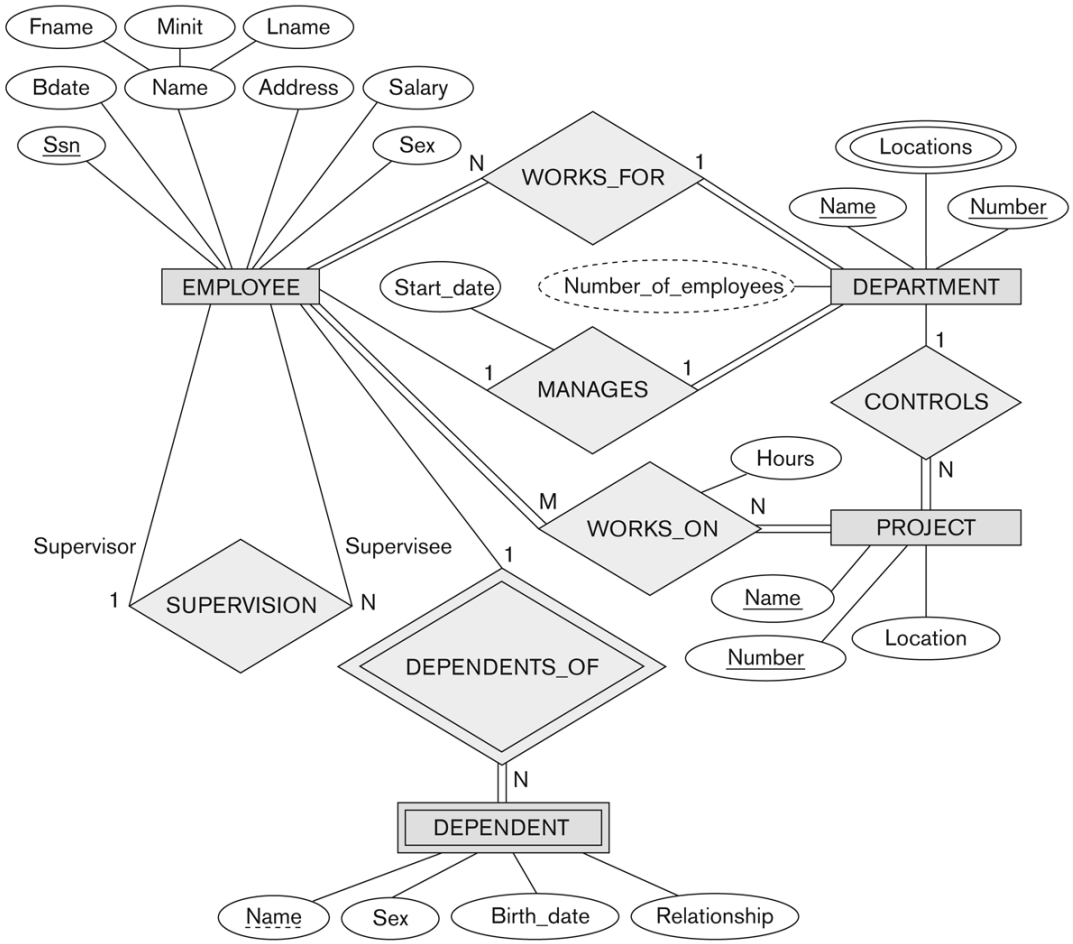


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Weak Entity Types

- An entity that does not have a key attribute and that is identification-dependent on another entity type.
- A weak entity must participate in an identifying relationship type with an owner or identifying entity type
- Entities are identified by the combination of:
  - A partial key of the weak entity type
  - The particular entity they are related to in the identifying relationship type
- **Example:**
  - A DEPENDENT entity is identified by the dependent's first name, *and* the specific EMPLOYEE with whom the dependent is related
  - Name of DEPENDENT is the *partial key*
  - DEPENDENT is a *weak entity type*
  - EMPLOYEE is its identifying entity type via the identifying relationship type DEPENDENT_OF

Attributes of 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.
    - A value of HoursPerWeek depends on a particular (employee, project) combination
  - Most relationship attributes are used with M:N relationships
    - In 1:N relationships, they can be transferred to the entity type on the N-side of the relationship

Example Attribute of a Relationship Type: Hours of WORKS_ON

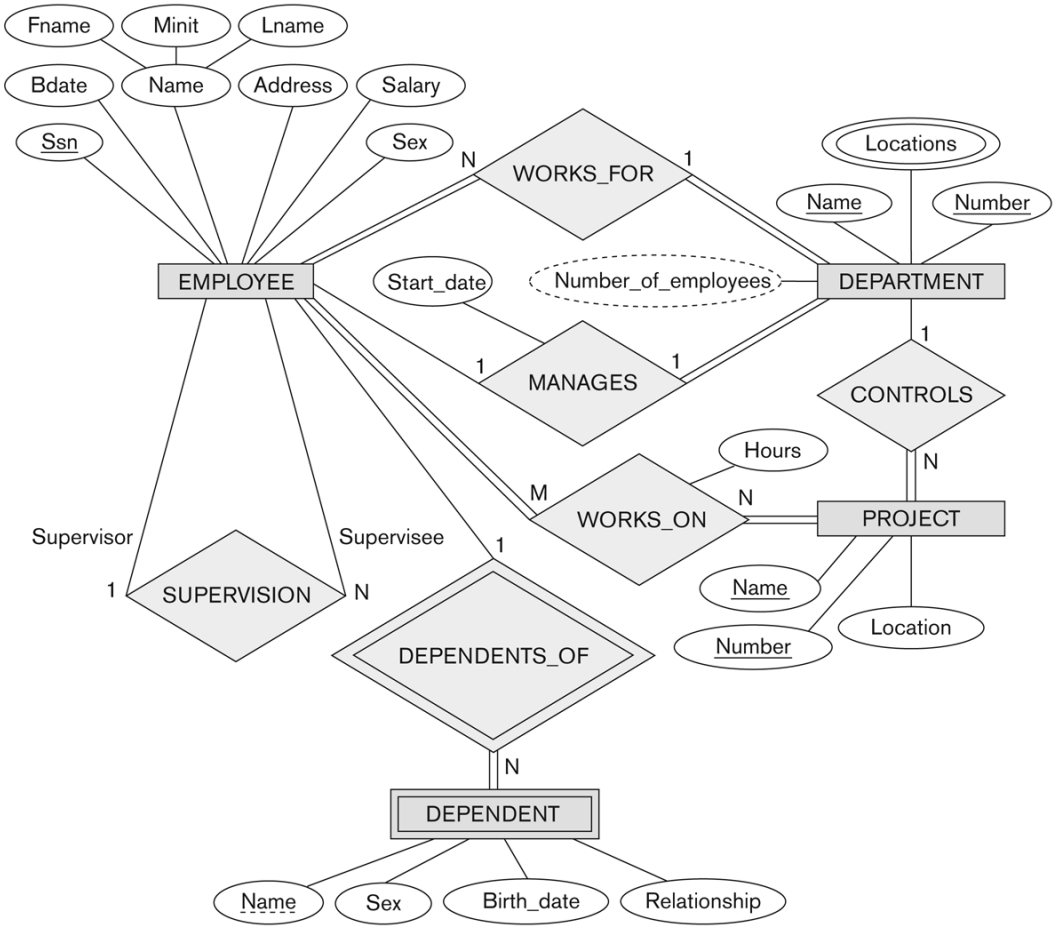


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Alternative diagrammatic notation

- ER diagrams is one popular example for displaying database schemas
- Many other notations exist in the literature and in various database design and modeling tools
- UML class diagrams is representative of another way of displaying ER concepts that is used in several commercial design tools

Total Participation

- Total participation, also known as **mandatory participation**, occurs when every instance of an entity must be involved in a relationship with another entity. This means that for every instance of the participating entity, there must be a corresponding instance in the related entity.

Partial Participation

- Partial participation, also known as **optional participation**, occurs when not every instance of an entity must be involved in a relationship with another entity. In this case, some instances of the participating entity may not have a corresponding instance in the related entity.

Total and partial

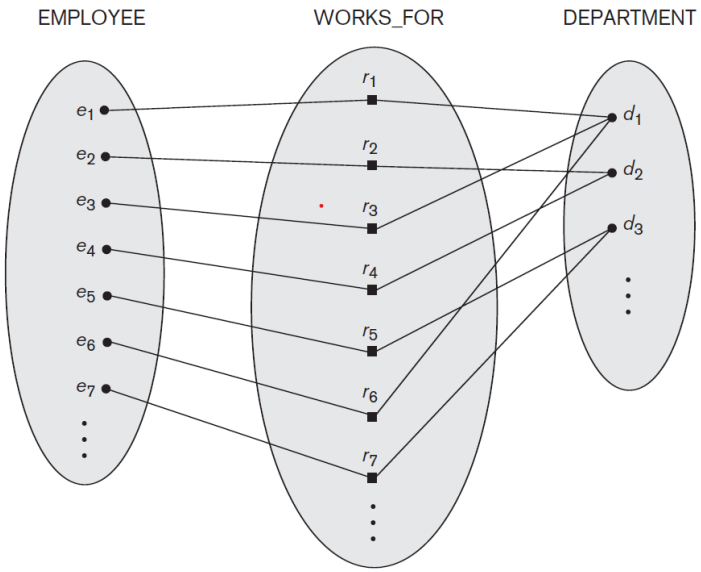
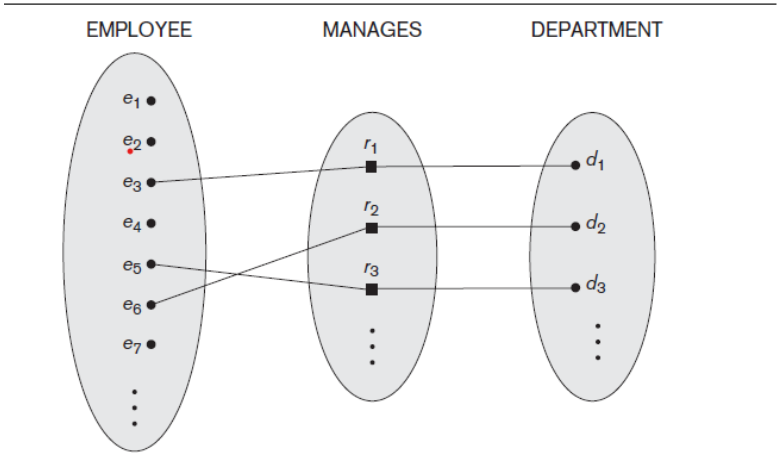


Figure 7.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.



Attributes of Relationship Types

- Attributes of 1:1 or 1:N relationship types can be migrated to one entity type

For a 1:N relationship type

- Relationship attribute can be migrated only to entity type on N-side of relationship

For M:N relationship types

- Some attributes may be determined by combination of participating entities
- Must be specified as relationship attributes

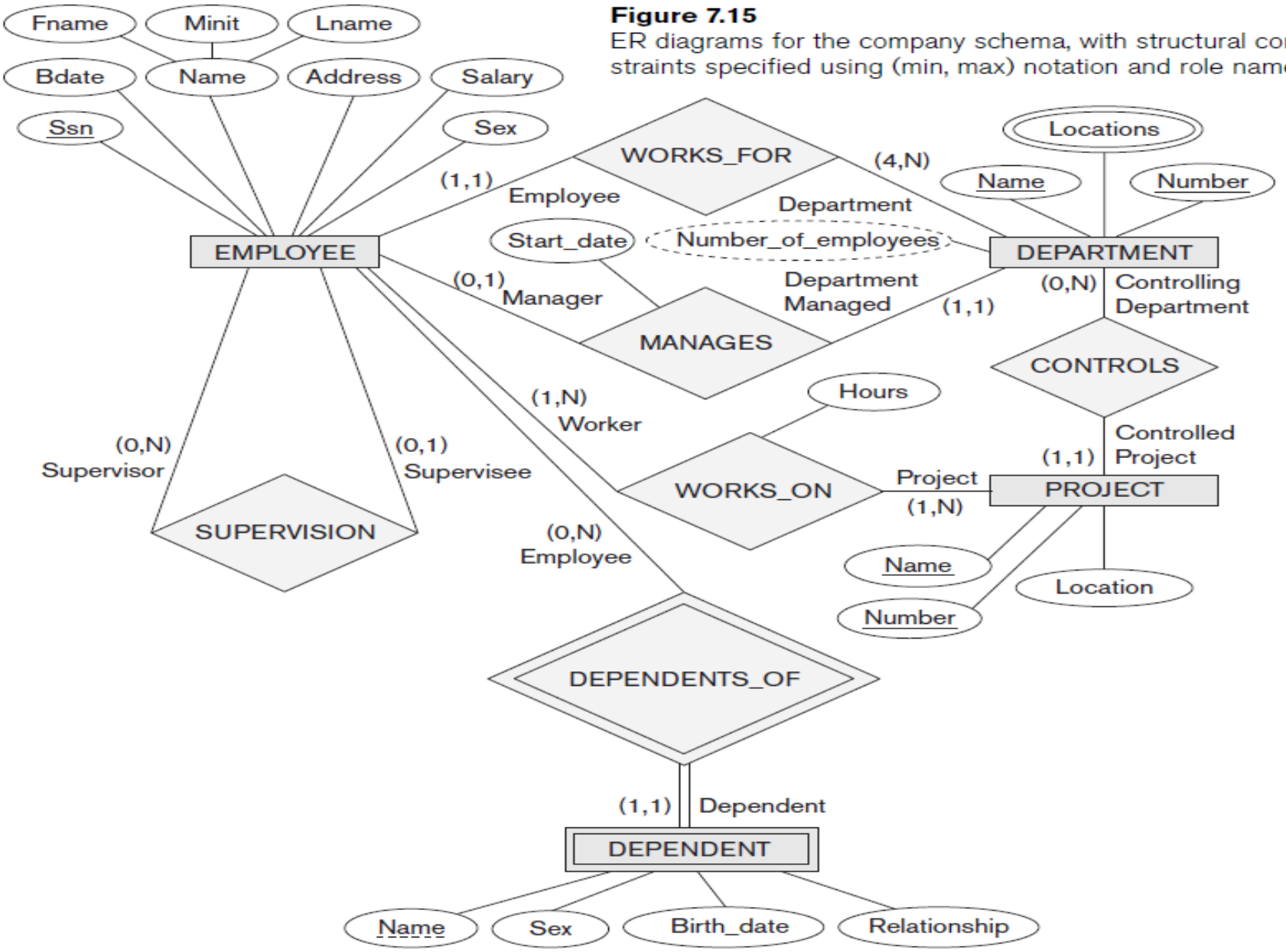
Notation for Constraints on Relationships

- Cardinality ratio (of a binary relationship): 1:1, 1:N, N:1, or M:N
  - Shown by placing appropriate numbers on the relationship edges.
- Participation constraint (on each participating entity type): total (called existence dependency) or partial.
  - Total shown by double line, partial by single line.
- NOTE: These are easy to specify for Binary Relationship Types.

Refining the ER Design for the COMPANY Database

- Change attributes that represent relationships into relationship types
- Determine cardinality ratio and participation constraint of each relationship type

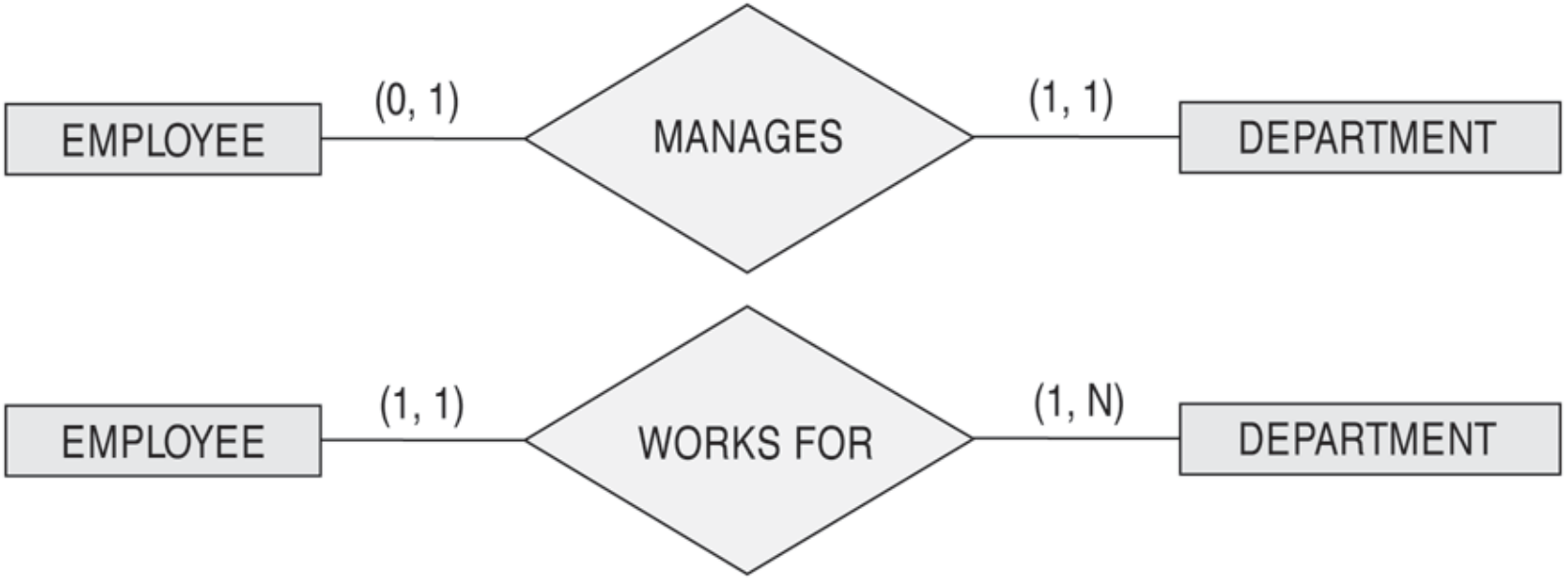
Figure 7.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.



Alternative (min, max) notation for relationship structural constraints:

- Specified on each participation of an entity type E in a relationship type R
- Specifies that each entity e in E participates in at least *min* and at most *max* relationship instances in R
- Default(no constraint): min=0, max=n (signifying no limit)
- Must have min≤max, min≥0, max ≥1
- Derived from the knowledge of mini-world constraints
- Examples:
  - A department has exactly one manager and an employee can manage at most one department.
    - Specify (0,1) for participation of EMPLOYEE in MANAGES
    - Specify (1,1) for participation of DEPARTMENT in MANAGES
  - An employee can work for exactly one department but a department can have any number of employees.
    - Specify (1,1) for participation of EMPLOYEE in WORKS_FOR
    - Specify (0,n) for participation of DEPARTMENT in WORKS_FOR

The (min,max) notation for relationship constraints



Read the min,max numbers next to the entity type and looking **away from** the entity type

COMPANY ER Schema Diagram using (min, max) notation

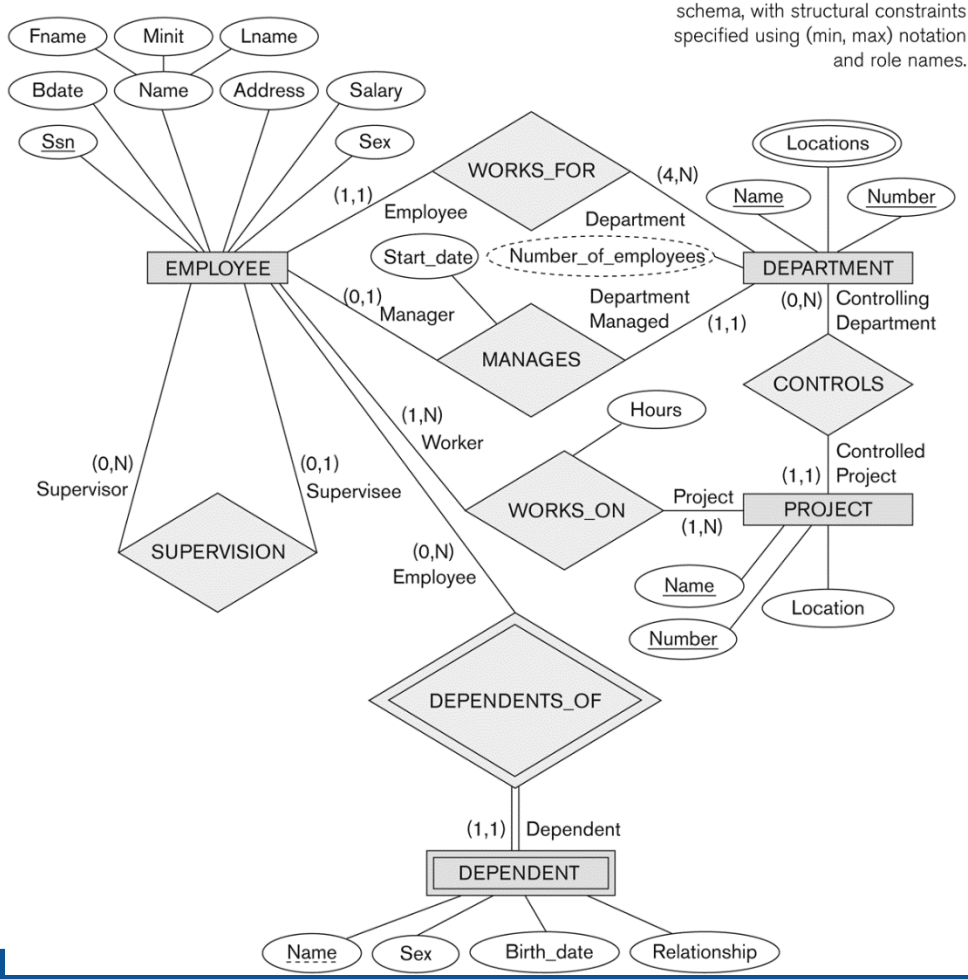


Figure 3.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.

Find the relationship constraints??

Teacher Table

Teacher_ID	Teacher_Name
1	Mr. Johnson
2	Mr. Thompson

Student Table

Student_ID	Student_Name	Teacher_ID
101	Alice	1
102	Bob	1
103	Charlie	2
104	David	2

Find the cardinality ratio??

Suppose entity set A = {a,b,c,d,e} and entity set B = {w,x,y,z} and they participate in a relationship R and the instances in R are: { (a,w), (b,w), (c,x), (d,x), (e,y) }. Which one of the following is correct?

- Cardinality ratio A:B is many-to-one; A participates partially; B participates completely.
- Cardinality ratio A:B is one-to-many; A participates completely; B participates partially.
- Cardinality ratio A:B is many-to-one; A participates completely; B participates partially.
- Cardinality ratio A:B is many-to-many; A participates completely; B participates partially.

9) Suppose ***Author*** is a relationship type with two participating entity types ***Person*** and ***Book***. What is the appropriate cardinality ratio for Person:Book?

- 1:N
- N:1
- M:N
- 1:1

16) Consider the following sets:

C = { p: weak entity type; q: multi-valued attribute; r: derived attribute: s: relationship type}
D = { w: dashed-line ellipse; x: diamond box; y: double-line rectangle; z: double-line ellipse}

The correct match between elements of C and D is:

- p -- z; q -- y; r -- w; s -- x
- p -- y; q -- z; r -- x; s -- w
- p -- y; q -- z; r -- w; s -- x
- p -- x; q -- w; r -- z; s -- y

- Suppose X is a composite attribute of an entity type and has three components – C1, C2 and C3, where only C2 is multi-valued, if domain sets of C1, C2 and C3 have 5, 3, and 4 elements respectively, what is the size of the domain of X?

Enhanced Entity-Relationship (EER) Modeling

Chapter Outline

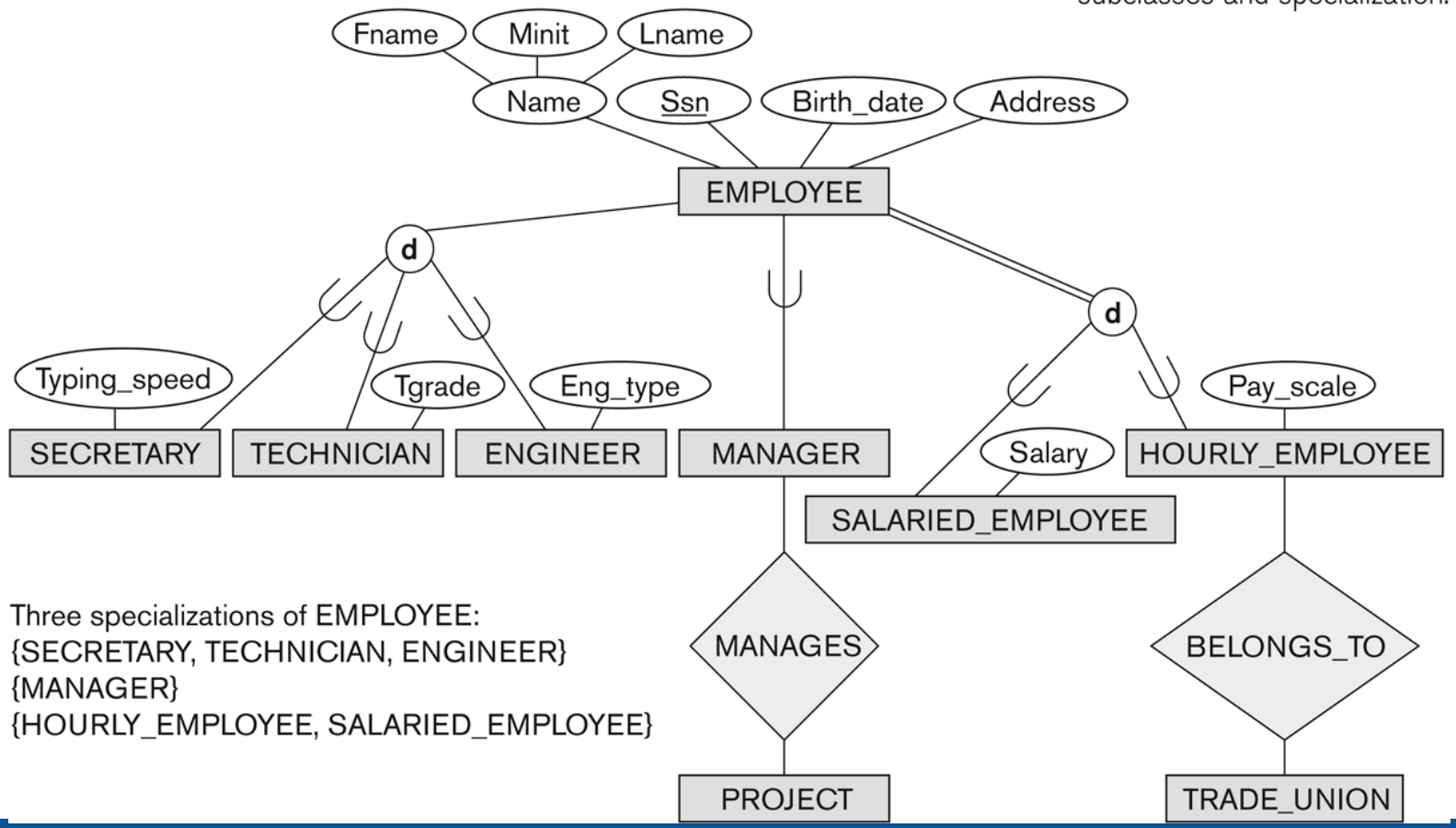
- EER stands for Enhanced ER or Extended ER
- EER Model Concepts
  - Includes all modeling concepts of basic ER
  - Additional concepts:
    - subclasses/superclasses
    - specialization/generalization
    - categories (UNION types)
    - attribute and relationship inheritance
  - Constraints on Specialization/Generalization
- The additional EER concepts are used to model applications more completely and more accurately
  - EER includes some object-oriented concepts, such as inheritance
- Knowledge Representation and Ontology Concepts

Subclasses and Superclasses (1)

- An entity type may have additional meaningful subgroupings of its entities
  - Example: EMPLOYEE may be further grouped into:
    - SECRETARY, ENGINEER, TECHNICIAN, …
      - Based on the EMPLOYEE's Job
    - MANAGER
      - EMPLOYEEs who are managers (the role they play)
    - SALARIED_EMPLOYEE, HOURLY_EMPLOYEE
      - Based on the EMPLOYEE's method of pay
- EER diagrams extend ER diagrams to represent these additional subgroupings, called *subclasses* or *subtypes*

Subclasses and Superclasses

Figure 4.1
EER diagram notation to represent subclasses and specialization.



Subclasses and Superclasses (2)

- Each of these subgroupings is a subset of EMPLOYEE entities
- Each is called a subclass of EMPLOYEE
- EMPLOYEE is the superclass for each of these subclasses
- These are called superclass/subclass relationships:
  - EMPLOYEE/SECRETARY
  - EMPLOYEE/TECHNICIAN
  - EMPLOYEE/MANAGER
  - ...

Subclasses and Superclasses (3)

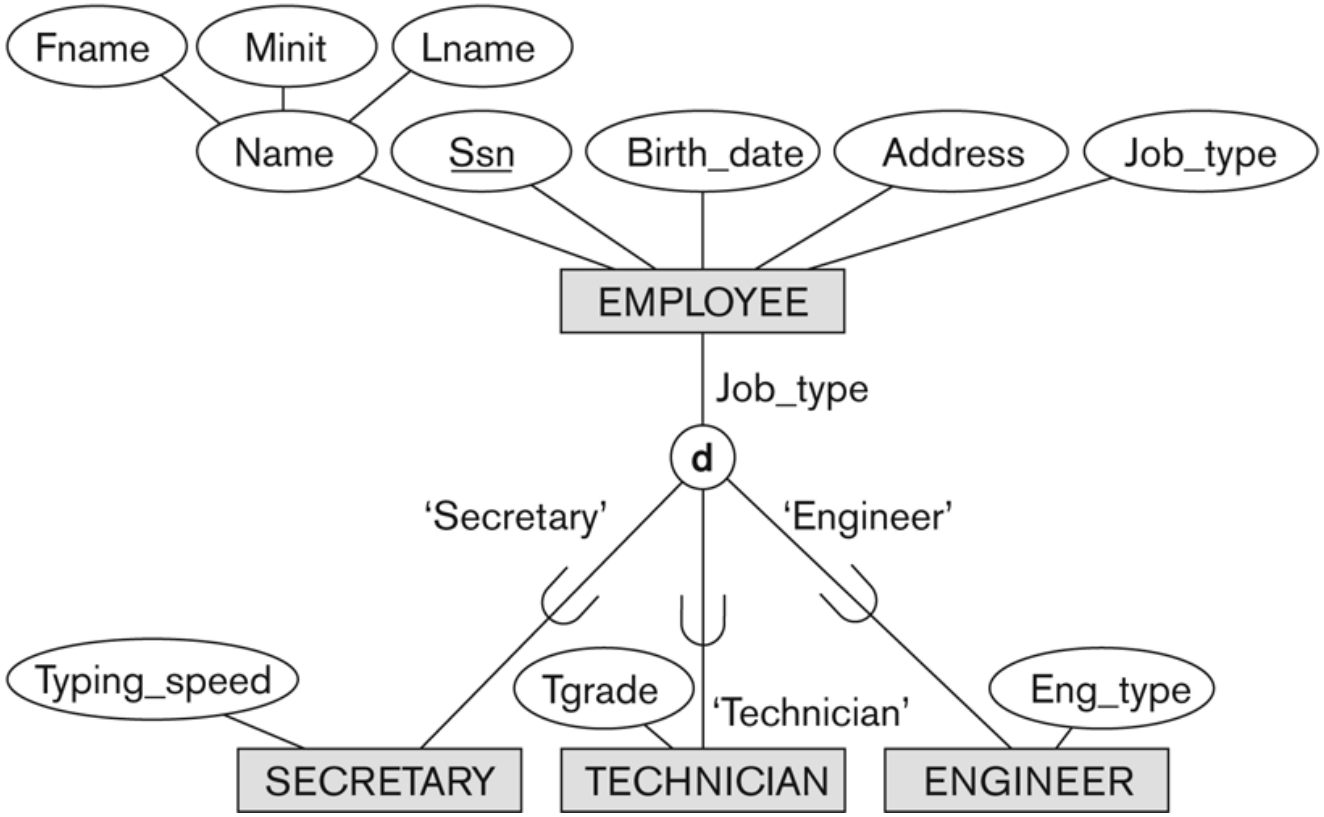
- These are also called IS-A relationships
  - SECRETARY IS-A EMPLOYEE, TECHNICIAN IS-A EMPLOYEE, ….
- Note: An entity that is member of a subclass represents the same real-world entity as some member of the superclass:
  - The subclass member is the same entity in a *distinct specific role*
  - An entity cannot exist in the database merely by being a member of a subclass; it must also be a member of the superclass
  - A member of the superclass can be optionally included as a member of any number of its subclasses

Subclasses and Superclasses (4)

- Examples:
  - A salaried employee who is also an engineer belongs to the two subclasses:
    - ENGINEER, and
    - SALARIED_EMPLOYEE
  - A salaried employee who is also an engineering manager belongs to the three subclasses:
    - MANAGER,
    - ENGINEER, and
    - SALARIED_EMPLOYEE
- It is not necessary that every entity in a superclass be a member of some subclass

Representing Specialization in EER Diagrams

Figure 4.4
EER diagram notation for an attribute-defined specialization on Job_type.



Attribute Inheritance in Superclass / Subclass Relationships

- An entity that is member of a subclass *inherits*
  - All attributes of the entity as a member of the superclass
  - All relationships of the entity as a member of the superclass
- Example:
  - In the previous slide, SECRETARY (as well as TECHNICIAN and ENGINEER) inherit the attributes Name, SSN, …, from EMPLOYEE
  - Every SECRETARY entity will have values for the inherited attributes

Specialization (1)

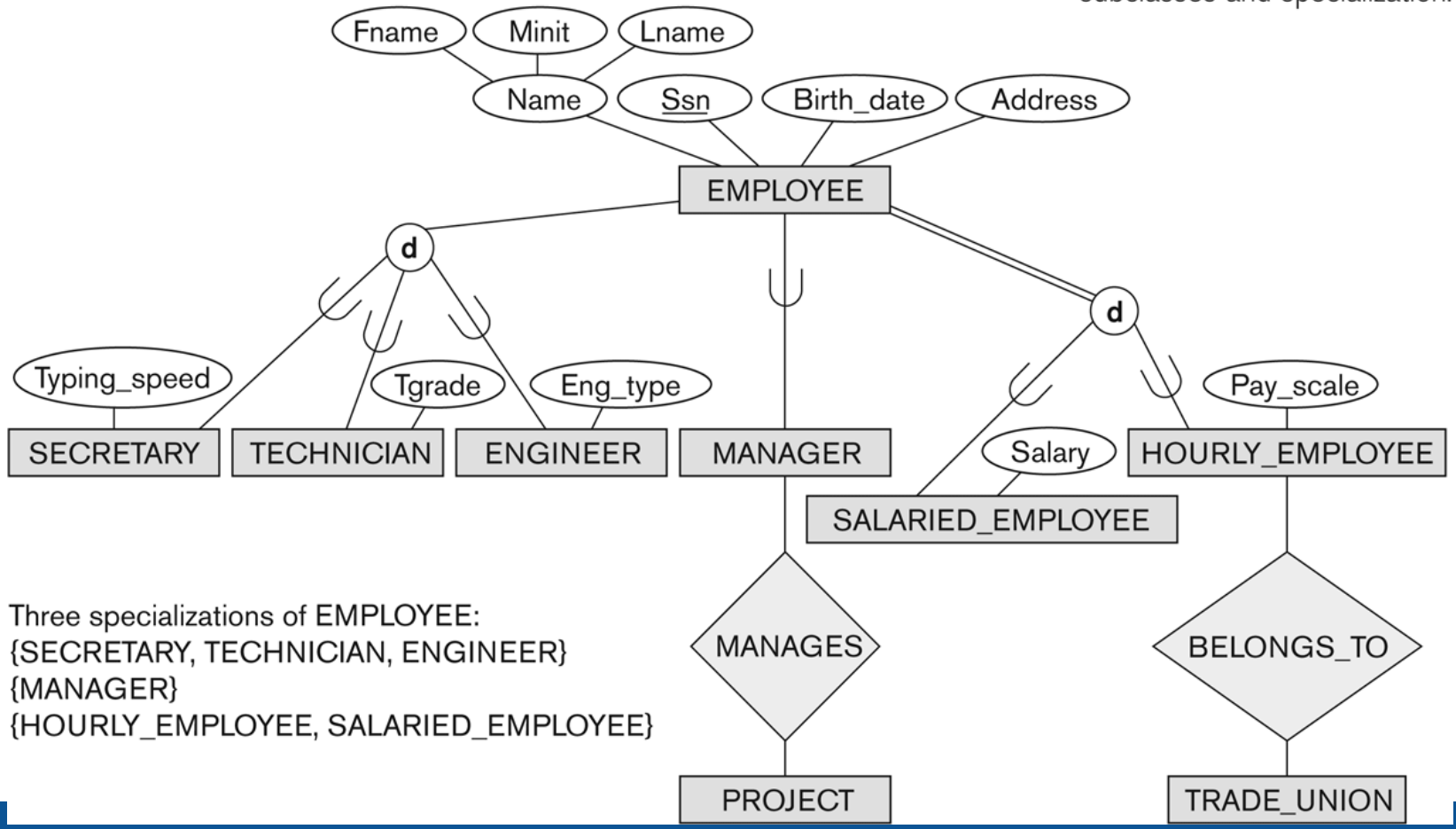
- Specialization is the process of defining a set of subclasses of a superclass
- The set of subclasses is based upon some distinguishing characteristics of the entities in the superclass
  - Example: {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of EMPLOYEE based upon *job type.*
  - Example: MANAGER *is a specialization of EMPLOYEE based on the role the employee plays*
    - May have several specializations of the same superclass

Specialization (2)

- Example: Another specialization of EMPLOYEE based on *method of pay* is {SALARIED_EMPLOYEE, HOURLY_EMPLOYEE}.
  - Superclass/subclass relationships and specialization can be diagrammatically represented in EER diagrams
  - Attributes of a subclass are called *specific* or *local* attributes.
    - For example, the attribute TypingSpeed of SECRETARY
  - The subclass can also participate in specific relationship types.
    - For example, a relationship BELONGS_TO of HOURLY_EMPLOYEE

Specialization (3)

Figure 4.1
EER diagram notation to represent subclasses and specialization.



Three specializations of EMPLOYEE:
{SECRETARY, TECHNICIAN, ENGINEER}
{MANAGER}
{HOURLY_EMPLOYEE, SALARIED_EMPLOYEE}

Generalization

- Generalization is the reverse of the specialization process
- Several classes with common features are generalized into a superclass;
  - original classes become its subclasses
- Example: CAR, TRUCK generalized into VEHICLE;
  - both CAR, TRUCK become subclasses of the superclass VEHICLE.
  - We can view {CAR, TRUCK} as a specialization of VEHICLE
  - Alternatively, we can view VEHICLE as a generalization of CAR and TRUCK

Generalization (2)

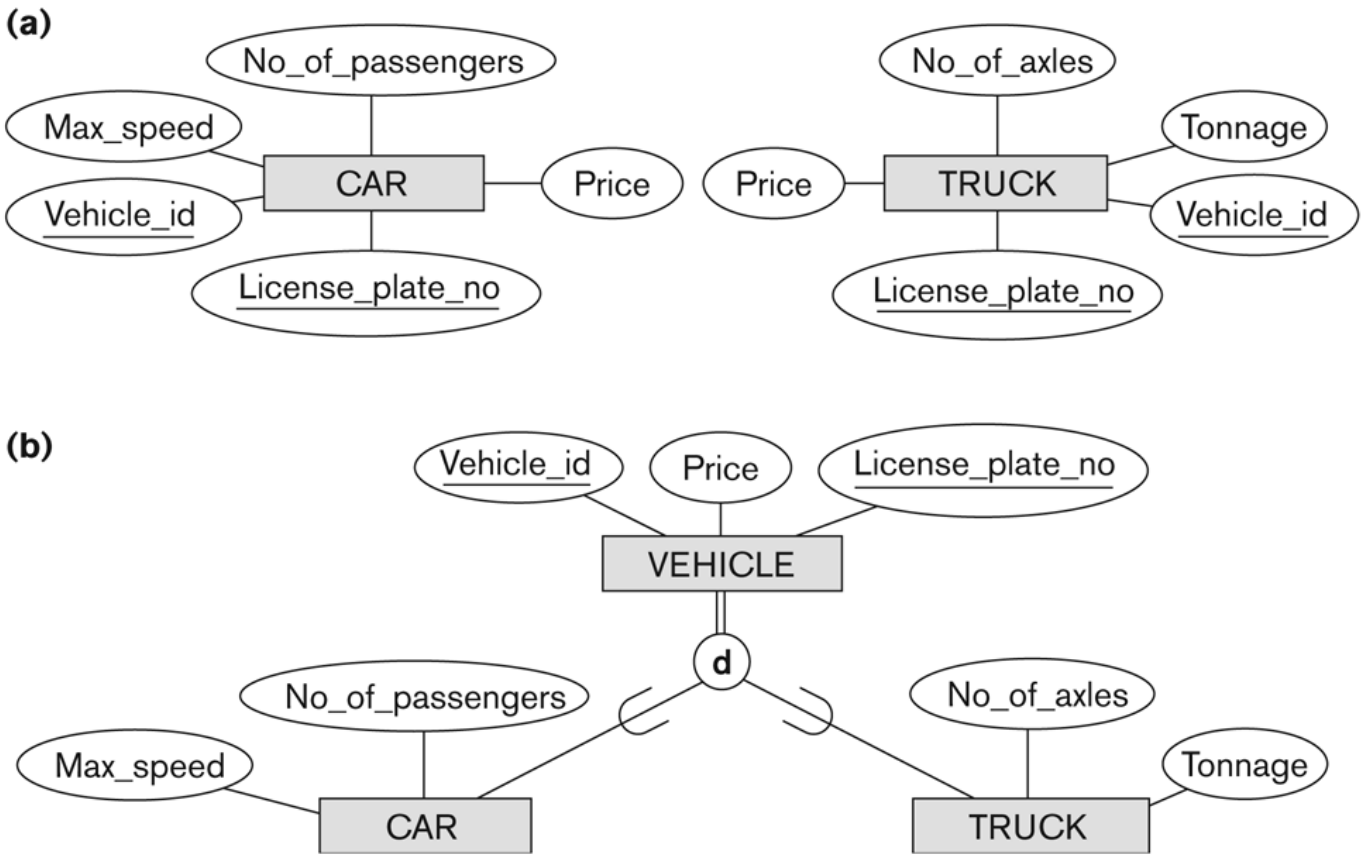


Figure 4.3
Generalization. (a) Two entity types, CAR and TRUCK. (b) Generalizing CAR and TRUCK into the superclass VEHICLE.

Generalization and Specialization (1)

- Diagrammatic notations are sometimes used to distinguish between generalization and specialization
  - Arrow pointing to the generalized superclass represents a generalization
  - Arrows pointing to the specialized subclasses represent a specialization
  - We *do not use* this notation because it is often subjective as to which process is more appropriate for a particular situation
  - We advocate not drawing any arrows

Generalization and Specialization (2)

- Data Modeling with Specialization and Generalization
  - A superclass or subclass represents a collection (or set or grouping) of entities
  - It also represents a particular *type of entity*
  - Shown in rectangles in EER diagrams (as are entity types)
  - We can call all entity types (and their corresponding collections) ***classes***, whether they are entity types, superclasses, or subclasses

Types of Specialization

- Predicate-defined (or condition-defined) : based on some predicate. E.g., based on value of an attribute, say, Job-type, or Age.
- Attribute-defined: shows the name of the attribute next to the line drawn from the superclass toward the subclasses (see Fig. 4.1)
- User-defined: membership is defined by the user on an entity by entity basis

Constraints on Specialization and Generalization (1)

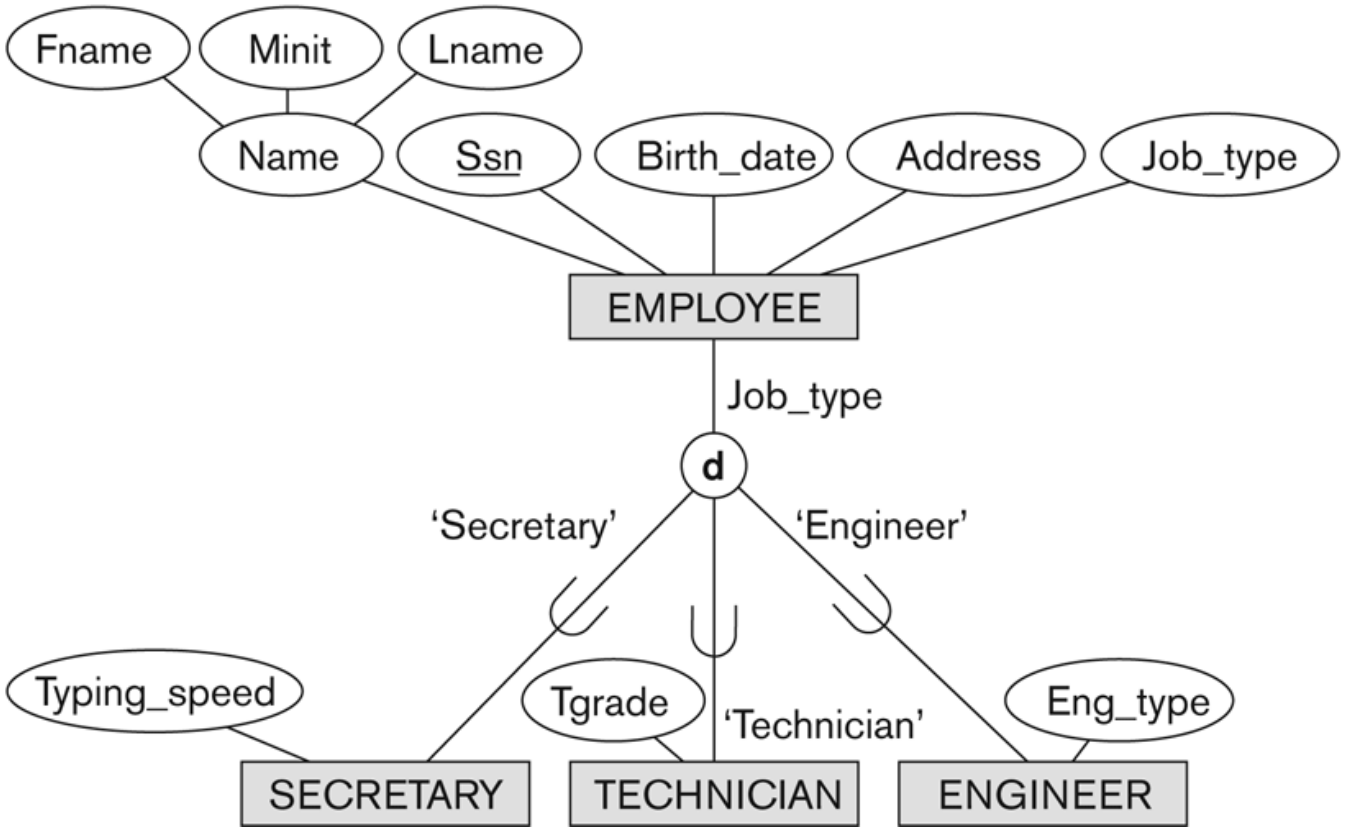
- If we can determine exactly those entities that will become members of each subclass by a condition, the subclasses are called predicate-defined (or condition-defined) subclasses
  - Condition is a constraint that determines subclass members
  - Display a predicate-defined subclass by writing the predicate condition next to the line attaching the subclass to its superclass

Constraints on Specialization and Generalization (2)

- If all subclasses in a specialization have membership condition on same attribute of the superclass, specialization is called an attribute-defined specialization
  - Attribute is called the defining attribute of the specialization
  - Example: JobType is the defining attribute of the specialization {SECRETARY, TECHNICIAN, ENGINEER} of EMPLOYEE
- If no condition determines membership, the subclass is called user-defined
  - Membership in a subclass is determined by the database users by applying an operation to add an entity to the subclass
  - Membership in the subclass is specified individually for each entity in the superclass by the user

Displaying an attribute-defined specialization in EER diagrams

Figure 4.4
EER diagram notation for an attribute-defined specialization on Job_type.



Constraints on Specialization and Generalization (3)

- Two basic constraints can apply to a specialization/generalization:
  - Disjointness Constraint:
  - Completeness Constraint:

Constraints on Specialization and Generalization (4)

- Disjointness Constraint:
  - Specifies that the subclasses of the specialization must be *disjoint*:
    - an entity can be a member of at most one of the subclasses of the specialization
  - Specified by ***d*** in EER diagram
  - If not disjoint, specialization is *overlapping*:
    - that is the same entity may be a member of more than one subclass of the specialization
  - Specified by ***o*** in EER diagram

Constraints on Specialization and Generalization (5)

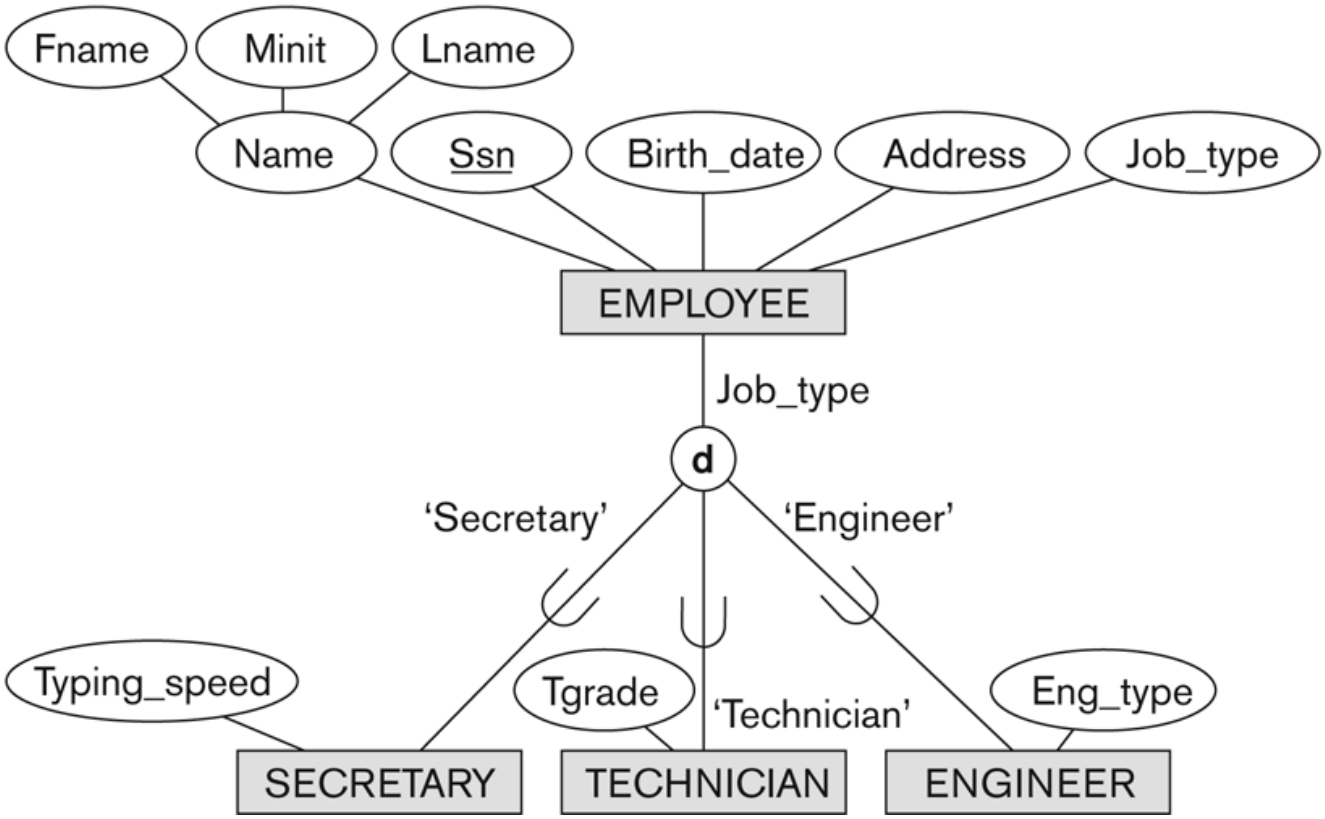
- Completeness (Exhaustiveness) Constraint:
  - *Total* specifies that every entity in the superclass must be a member of some subclass in the specialization/generalization
  - Shown in EER diagrams by a ***double line***
  - *Partial* allows an entity not to belong to any of the subclasses
  - Shown in EER diagrams by a single line

Constraints on Specialization and Generalization (6)

- Hence, we have four types of specialization/generalization:
  - Disjoint, total
  - Disjoint, partial
  - Overlapping, total
  - Overlapping, partial
- Note: Generalization usually is total because the superclass is derived from the subclasses.

Example of disjoint partial Specialization

Figure 4.4
EER diagram notation for an attribute-defined specialization on Job_type.



Example of overlapping total Specialization

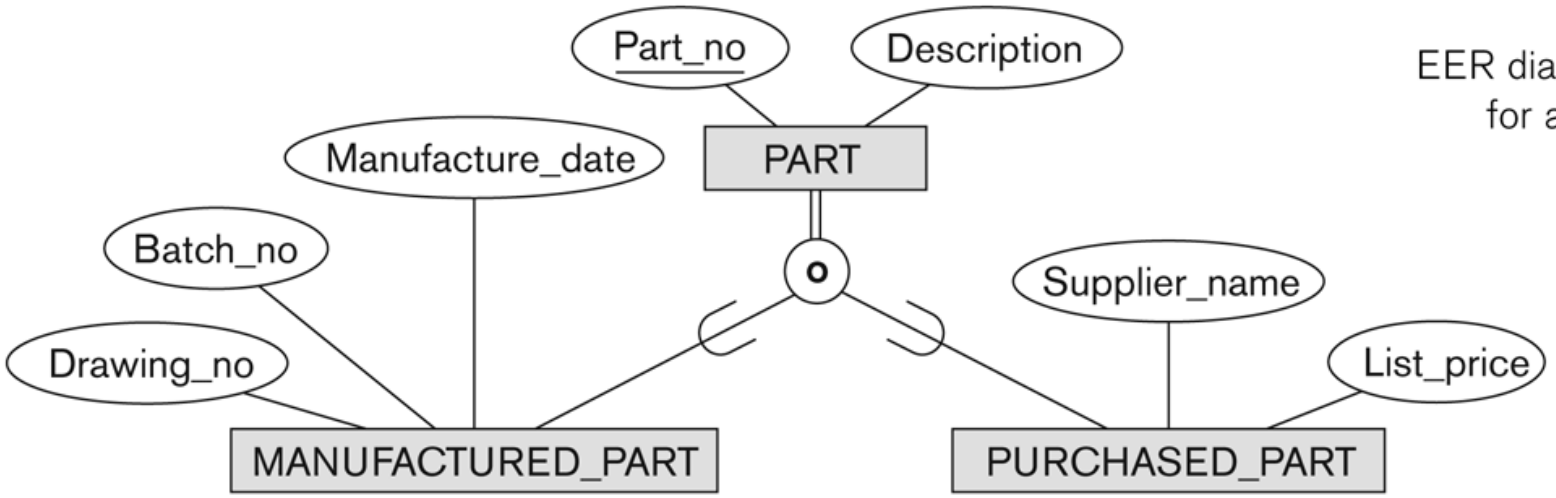


Figure 4.5
EER diagram notation for an overlapping (nondisjoint) specialization.

Specialization/Generalization Hierarchies, Lattices & Shared Subclasses (1)

- A subclass may itself have further subclasses specified on it
  - forms a hierarchy or a lattice
- ***Hierarchy*** has a constraint that every subclass has only one superclass (called ***single inheritance***); this is basically a ***tree structure***
- In a ***lattice***, a subclass can be subclass of more than one superclass (called ***multiple inheritance***)

Shared Subclass “Engineering_Manager”

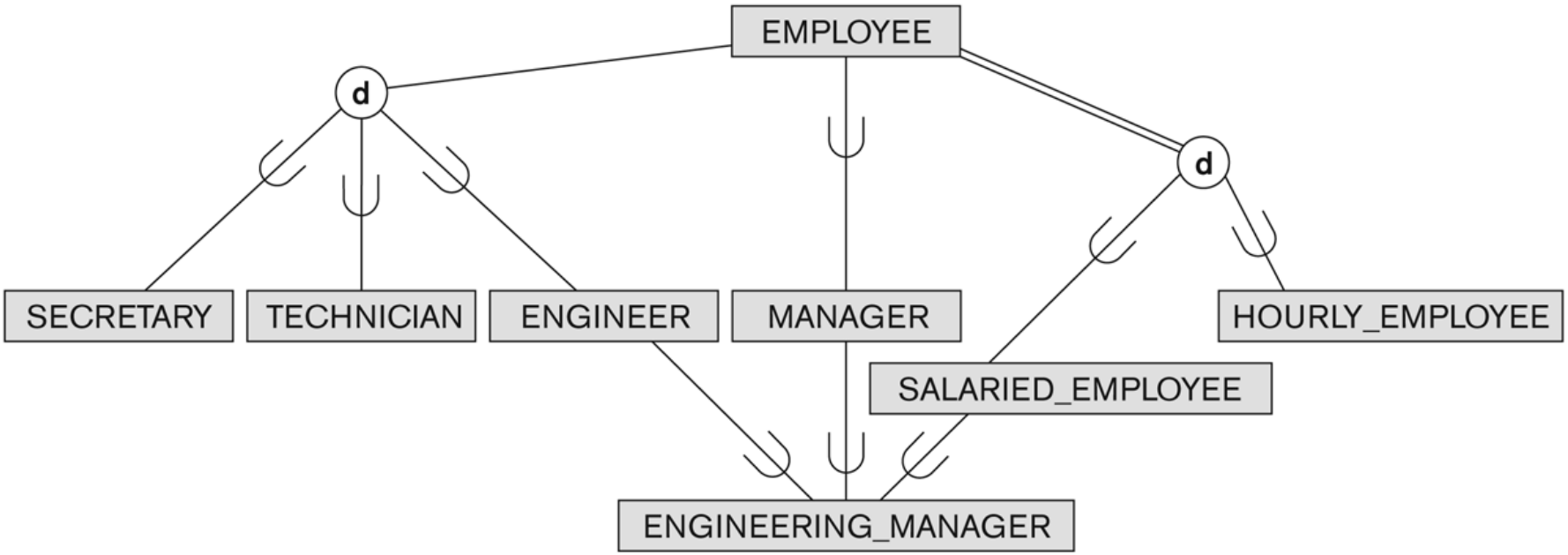


Figure 4.6
A specialization lattice with shared subclass ENGINEERING_MANAGER.

Specialization/Generalization Hierarchies, Lattices & Shared Subclasses (2)

- In a lattice or hierarchy, a subclass inherits attributes not only of its direct superclass, but also of all its predecessor superclasses
- A subclass with more than one superclass is called a shared subclass (multiple inheritance)
- Can have:
  - *specialization* hierarchies or lattices, or
  - *generalization* hierarchies or lattices,
  - depending on how they were *derived*
- We just use *specialization* (to stand for the end result of either specialization or generalization)

Specialization/Generalization Hierarchies, Lattices & Shared Subclasses (3)

- In *specialization*, start with an entity type and then define subclasses of the entity type by successive specialization
  - called a *top down* conceptual refinement process
- In *generalization*, start with many entity types and generalize those that have common properties
  - Called a *bottom up* conceptual synthesis process
- In practice, a *combination of both processes* is usually employed

Specialization / Generalization Lattice Example (UNIVERSITY)

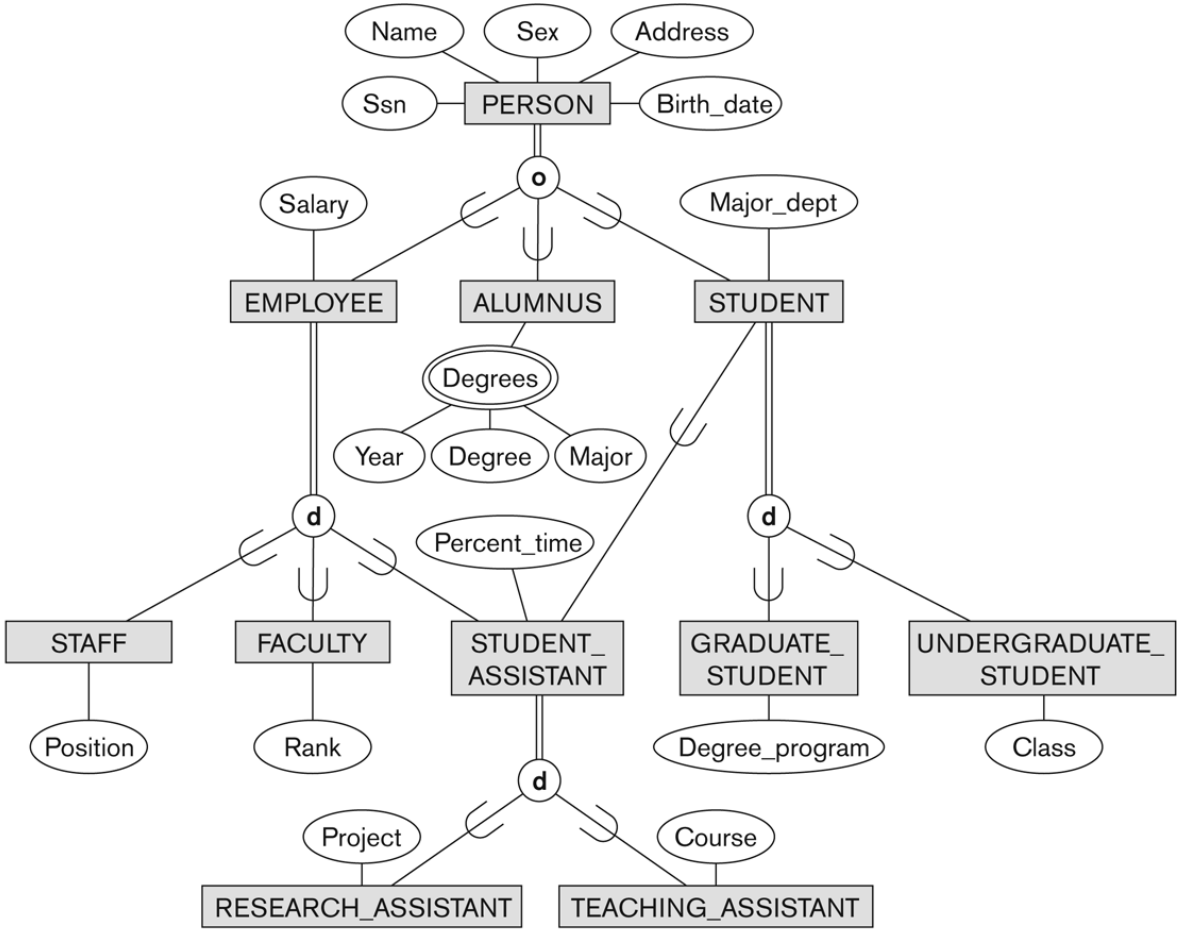


Figure 4.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.

Categories (UNION TYPES) (1)

- All of the *superclass/subclass relationships* we have seen thus far have a single superclass
- A shared subclass is a subclass in:
  - *more than one* distinct superclass/subclass relationships
  - each relationships has a *single* superclass
  - shared subclass leads to multiple inheritance
- In some cases, we need to model a *single superclass/subclass relationship* with *more than one* superclass
- Superclasses can represent different entity types
- Such a subclass is called a category or UNION TYPE

Categories (UNION TYPES) (2)

- Example: In a database for vehicle registration, a vehicle owner can be a PERSON, a BANK (holding a lien on a vehicle) or a COMPANY.
  - A *category* (UNION type) called OWNER is created to represent a subset of the *union* of the three superclasses COMPANY, BANK, and PERSON
  - A category member must exist in ***at least one (typically just one)*** of its superclasses
- Difference from *shared subclass*, which is a:
  - subset of the *intersection* of its superclasses
  - shared subclass member must exist in ***all*** of its superclasses

Two categories (UNION types): OWNER, REGISTERED_VEHICLE

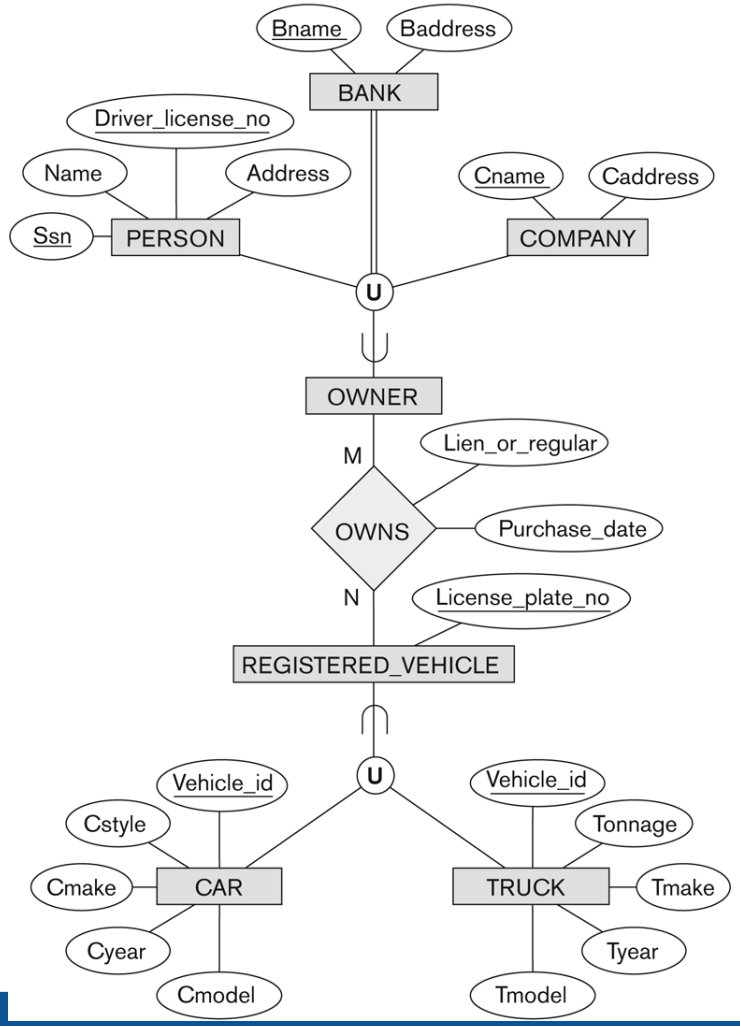


Figure 4.8
Two categories (union types): OWNER and REGISTERED_VEHICLE.

Formal Definitions of EER Model (1)

- Class C:
  - A type of entity with a corresponding set of entities:
    - could be entity type, subclass, superclass, or category
- Note: The definition of *relationship type* in ER/EER should have 'entity type' replaced with 'class‘ to allow relationships among classes in general
- Subclass S is a class whose:
  - Type inherits all the attributes and relationship of a class C
  - Set of entities must always be a subset of the set of entities of the other class C
    - $S \subseteq C$
  - C is called the superclass of S
  - A superclass/subclass relationship exists between S and C

Formal Definitions of EER Model (2)

- Specialization Z: Z = {S1, S2,…, Sn} is a set of subclasses with same superclass G; hence, G/Si is a superclass relationship for i = 1, …., n.
  - G is called a generalization of the subclasses {S1, S2,…, Sn}
  - Z is total if we always have:
    - S1 ∪ S2 ∪ … ∪ Sn = G;
    - Otherwise, Z is partial.
  - Z is disjoint if we always have:
    - Si ∩ S2 empty-set for i ≠ j;
  - Otherwise, Z is overlapping.

Formal Definitions of EER Model (3)

- Subclass S of C is predicate defined if predicate (condition) p on attributes of C is used to specify membership in S;
  - that is, S = C[p], where C[p] is the set of entities in C that satisfy condition p
- A subclass not defined by a predicate is called user-defined
- Attribute-defined specialization: if a predicate A = ci (where A is an attribute of G and ci is a constant value from the domain of A) is used to specify membership in each subclass Si in Z
  - Note: If ci ≠ cj for i ≠ j, and A is single-valued, then the attribute-defined specialization will be disjoint.

Formal Definitions of EER Model (4)

- Category or UNION type T
  - A class that is a subset of the *union* of n defining superclasses D1, D2,...Dn, n>1:
    - $T \subseteq (D1 \cup D2 \cup \ldots \cup Dn)$
  - Can have a predicate pi on the attributes of Di to specify entities of Di that are members of T.
  - If a predicate is specified on every Di: $T = (D1[p1] \cup D2[p2] \cup \ldots \cup Dn[pn])$

Alternative diagrammatic notations

- ER/EER diagrams are a specific notation for displaying the concepts of the model diagrammatically
- DB design tools use many alternative notations for the same or similar concepts
- One popular alternative notation uses *UML class diagrams*
- see next slides for UML class diagrams and other alternative notations

UML Example for Displaying Specialization / Generalization

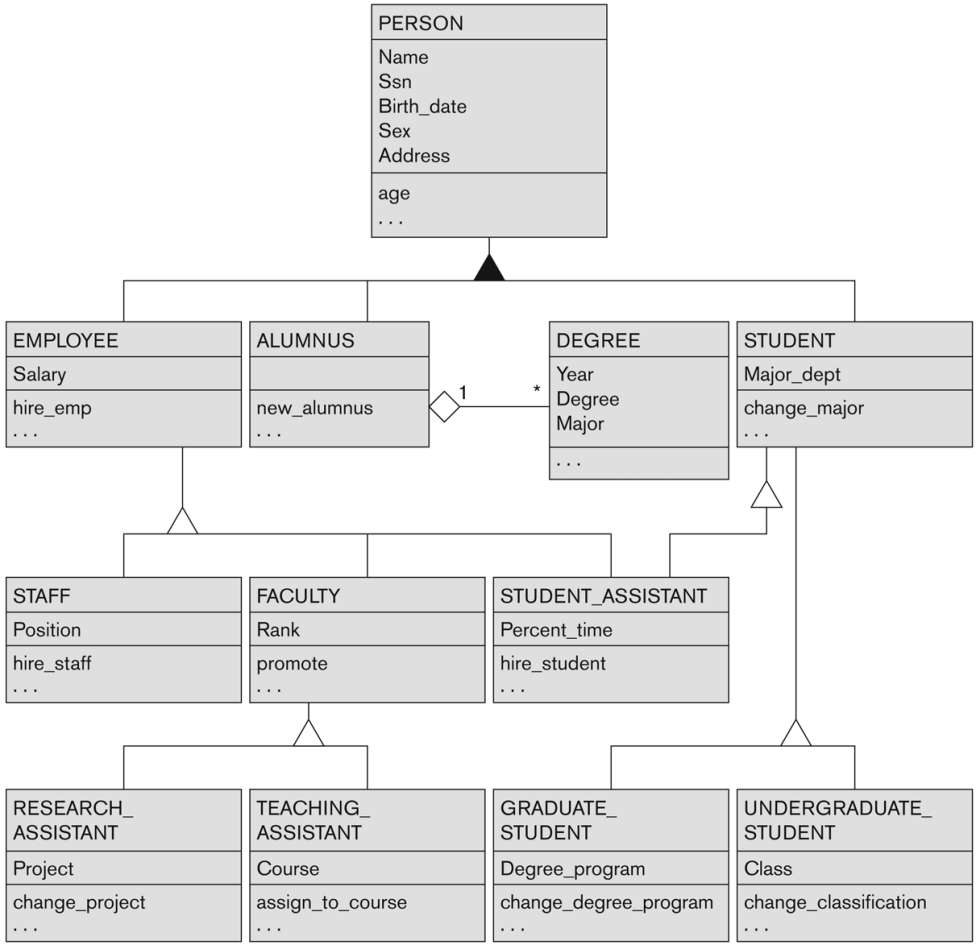


Figure 4.10
A UML class diagram corresponding to the EER diagram in Figure 4.7, illustrating UML notation for specialization/generalization.

Alternative Diagrammatic Notations

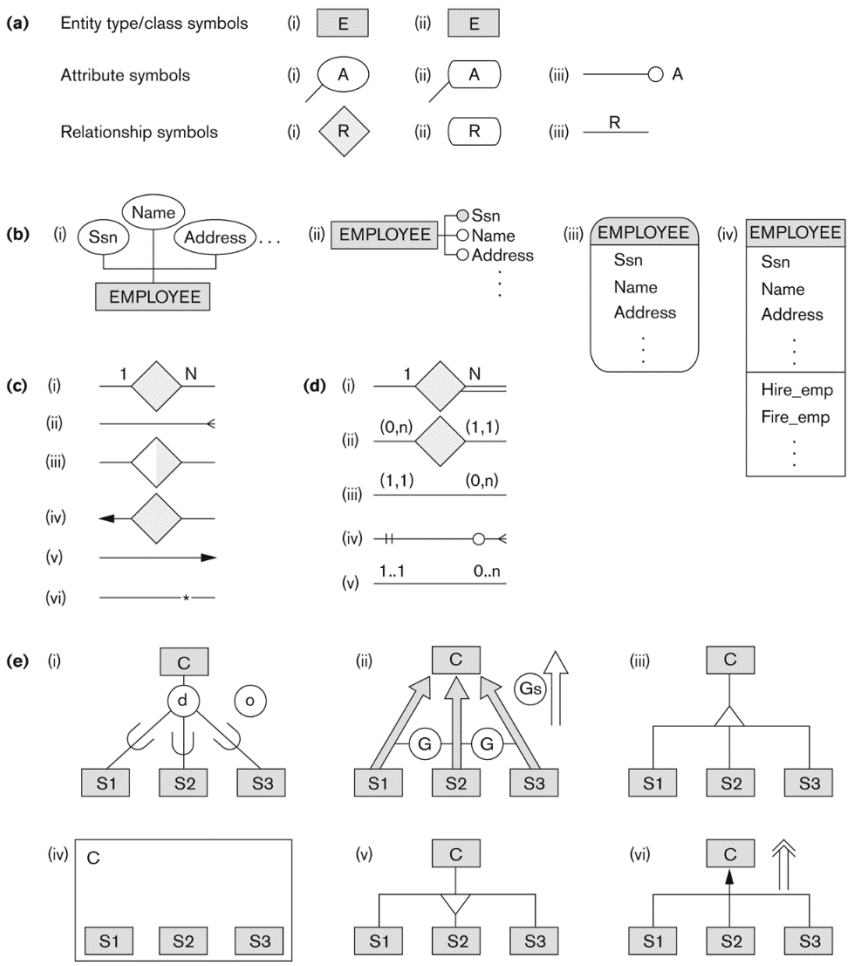


Figure A.1
Alternative notations. (a) Symbols for entity type/class, attribute, and relationship. (b) Displaying attributes. (c) Displaying cardinality ratios. (d) Various (min, max) notations. (e) Notations for displaying specialization/generalization.

Knowledge Representation (KR)-1

- Deals with modeling and representing a certain domain of knowledge.
- Typically done by using some formal model of representation and by creating an Ontology
- An ontology for a specific domain of interest describes a set of concepts and interrelationships among those concepts
- An Ontology serves as a “schema” which enables interpretation of the knowledge in a “knowledge-base”

Knowledge Representation (KR)-2

COMMON FEATURES between KR and Data Models:

- Both use similar set of abstractions – classification, aggregation, generalization, and identification.
- Both provide concepts, relationships, constraints, operations and languages to represent knowledge and model data

DIFFERENCES:

- KR has broader scope: tries to deal with missing and incomplete knowledge, default and common-sense knowledge etc.

Knowledge Representation (KR)-3

DIFFERENCES (continued):

- KR schemes typically include rules and reasoning mechanisms for inferencing
- Most KR techniques involve data and metadata. In data modeling, these are treated separately
- KR is used in conjunction with artificial intelligence systems to do decision support applications

For more details on spatial, temporal and multimedia data modeling, see Chapter 26. For details on use of Ontologies see Sections 27.4.3 and 27.7.4.

General Basis for Conceptual Modeling

- TYPES OF DATA ABSTRACTIONS
  - CLASSIFICATION and INSTANTIATION
  - AGGREGATION and ASSOCIATION (relationships)
  - GENERALIZATION and SPECIALIZATION
  - IDENTIFICATION
- CONSTRAINTS
  - CARDINALITY (Min and Max)
  - COVERAGE (Total vs. Partial, and Exclusive (Disjoint) vs. Overlapping)

Ontologies

- Use conceptual modeling and other tools to develop “a specification of a conceptualization”
  - **Specification** refers to the language and vocabulary (data model concepts) used
  - **Conceptualization** refers to the description (schema) of the concepts of a particular field of knowledge and the relationships among these concepts
- Many medical, scientific, and engineering ontologies are being developed as a means of standardizing concepts and terminology

Summary

- Introduced the EER model concepts
  - Class/subclass relationships
  - Specialization and generalization
  - Inheritance
- Constraints on EER schemas
- These augment the basic ER model concepts introduced in Chapter 3
- EER diagrams and alternative notations were presented
- Knowledge Representation and Ontologies were introduced and compared with Data Modeling

Scenario: Healthcare System

General Context

- In a healthcare system, different types of personnel play various roles, including doctors, nurses, and administrative staff. These personnel share common attributes but also have unique attributes and roles specific to their functions.

Entities and Relationships

- Person: A general entity type representing all personnel in the healthcare system.
- Doctor: A subclass of Person with attributes specific to doctors.
- Nurse: A subclass of Person with attributes specific to nurses.
- AdminStaff: A subclass of Person with attributes specific to administrative staff.
- Department: Entity representing various departments in the healthcare system.
- Patient: Entity representing patients in the healthcare system.
- Treatment: A relationship between Doctor, Nurse, and Patient.

Attributes

- Person: PersonID, Name, Address, Phone
- Doctor: DoctorID, Specialization, LicenseNumber
- Nurse: NurseID, CertificationAdminStaff: AdminID, Role
- Department: DeptID, DeptName
- Patient: PatientID, PatientName, DOB, Address

Relationships

- WorksIn: Relationship between Person and Department.
- Treats: Relationship between Doctor and Patient.
- Assists: Relationship between Nurse and Doctor.
- Administers: Relationship between AdminStaff and Department.

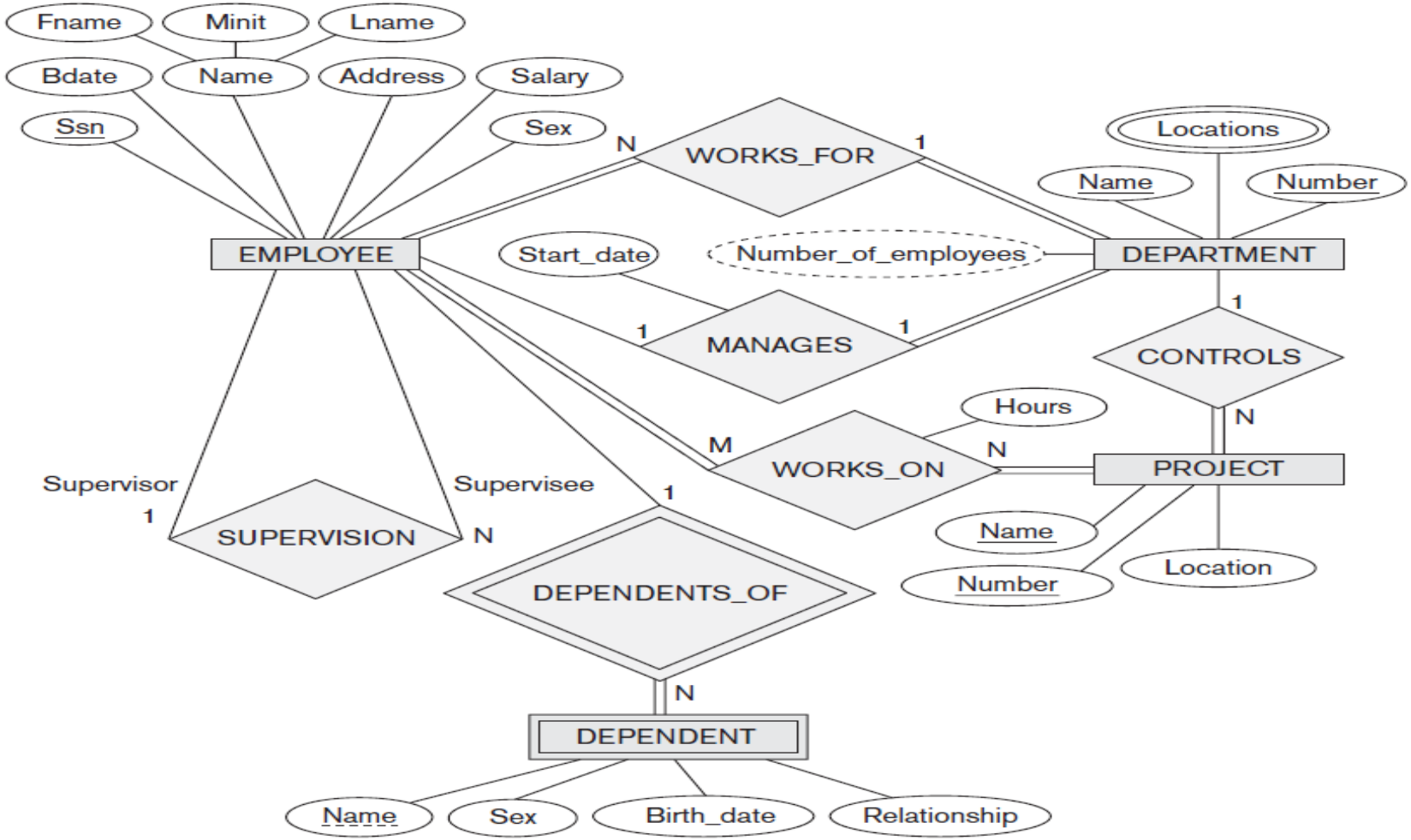
Relational Database Design by ER- and EER-to-Relational Mapping

Relational Database Design Using ER-to-Relational Mapping

- ER-to-Relational Mapping Algorithm

Step 1: Mapping of Regular Entity Types
Step 2: Mapping of Weak Entity Types
Step 3: Mapping of Binary 1:1 Relationship Types
Step 4: Mapping of Binary 1:N Relationship Types
Step 5: Mapping of Binary M:N Relationship Types
Step 6: Mapping of Multivalued Attributes
Step 7: Mapping of N-ary Relationship Types

Figure 9.1
The ER conceptual schema diagram for the COMPANY database.



Step 1

- Figure out all the regular/strong entity from the diagram and then create a corresponding relation(table) that includes all the simple attributes.
- Choose one of the attributes as a primary key. If composite, the simple attributes together form the primary key.
- For the given ER-Diagram we have Employee, Department and Project as strong/regular entity, as they are enclosed in single rectangle.
- So, we create respective relations.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary

DEPARTMENT

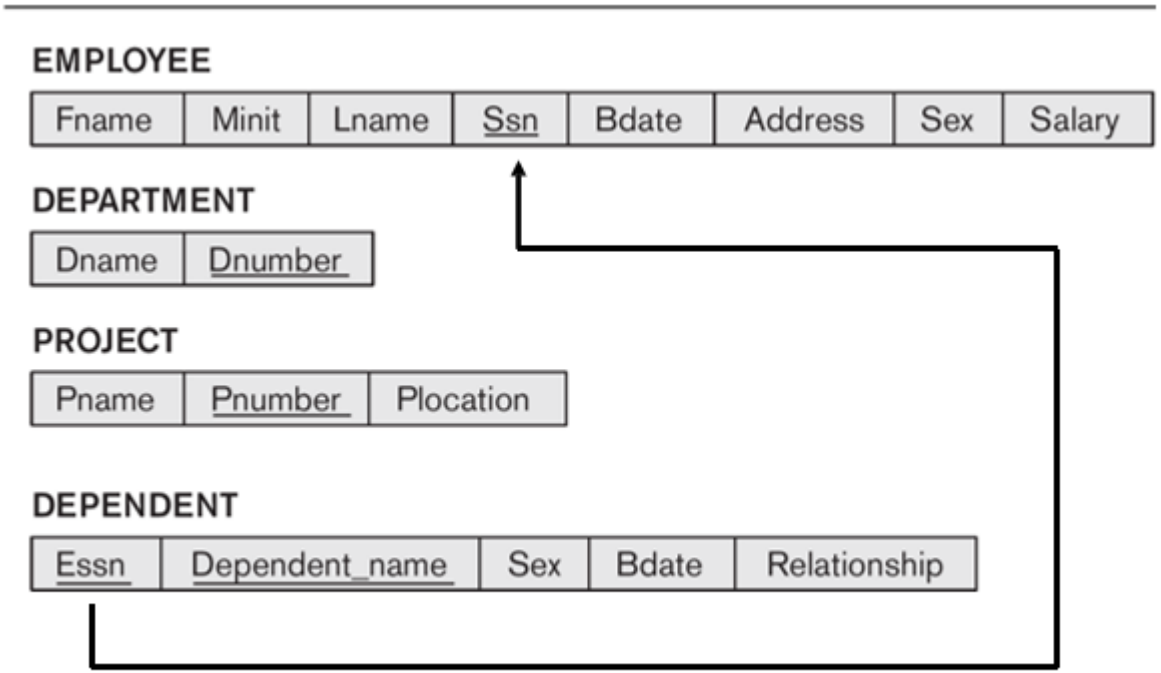
Dname	Dnumber

PROJECT

Pname	Pnumber	Plocation

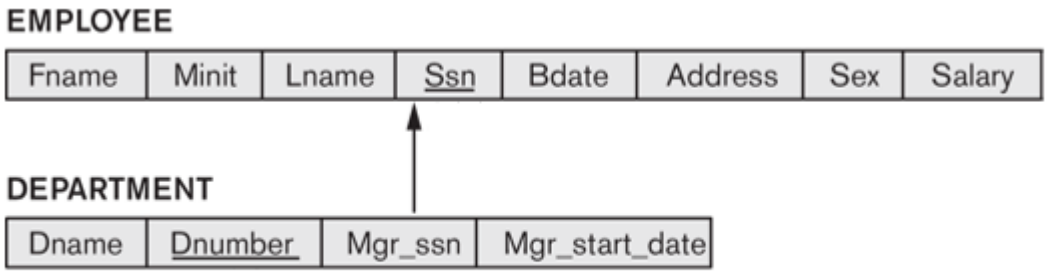
Step 2

- Figure out the weak entity types from the diagram and create a corresponding relation(table) that includes all its simple attributes.
- Add as foreign key all of the primary key attributes in the entity corresponding to the owner entity.
- The primary key is a combination of all the primary key attributes from the owner and the primary key of the weak entity.
- For the given ER-Diagram we have Dependent as a weak entity, as it is enclosed in a double rectangle that is indicative of an entity being weak.
- The Dependent relation(table) is created that is shown in the figure below.


EMPLOYEE
Fname
Minit
Lname
Ssn
Bdate
Address
Sex
Salary
DEPARTMENT
Dname
Dnumber
PROJECT
Pname
Pnumber
Plocation
DEPENDENT
Essn
Dependent_name
Sex
Bdate
Relationship

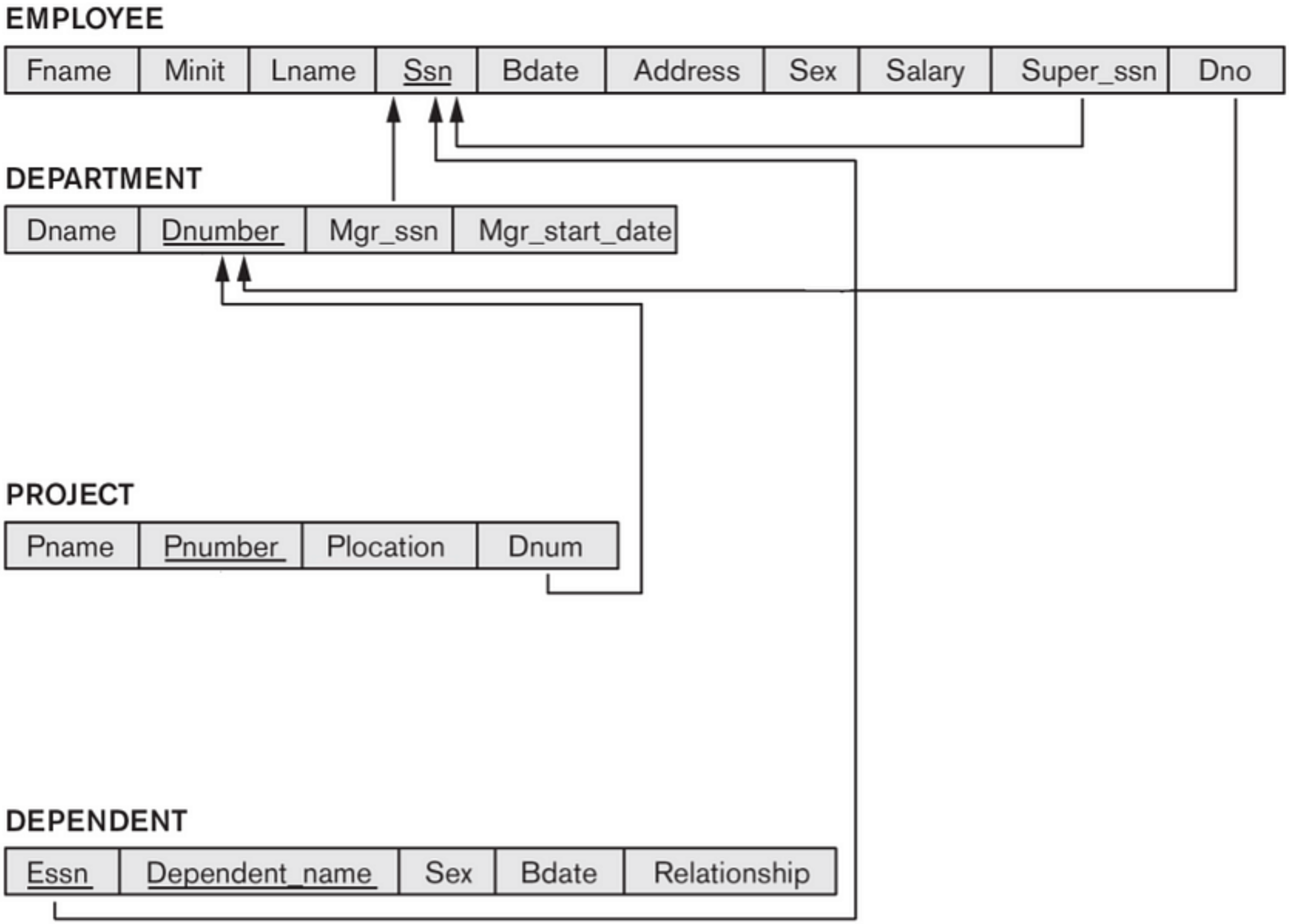
Step 3

- Now we need to figure out the entities from ER diagram for which there exists a 1-to-1 relationship.
- The entities for which there exists a 1-to-1 relationship, choose one relation(table) as S, the other as T.
- Better if S has total participation (reduces the number of NULL values).
- Then we need to add to S all the simple attributes of the relationship if there exists any.
- After that, we add as a foreign key in S the primary key attributes of T.
- For the given ER-Diagram there exists a 1-to-1 relationship between Employee and Department entity.
- Here Department has total participation therefore consider it as relation S and Employee as relation T.
- The 1-to-1 mapping between Employee and Department is depicted in the figure below.


EMPLOYEE
Fname
Minit
Lname
Ssn
Bdate
Address
Sex
Salary
DEPARTMENT
Dname
Dnumber
Mgr_ssn
Mgr_start_date

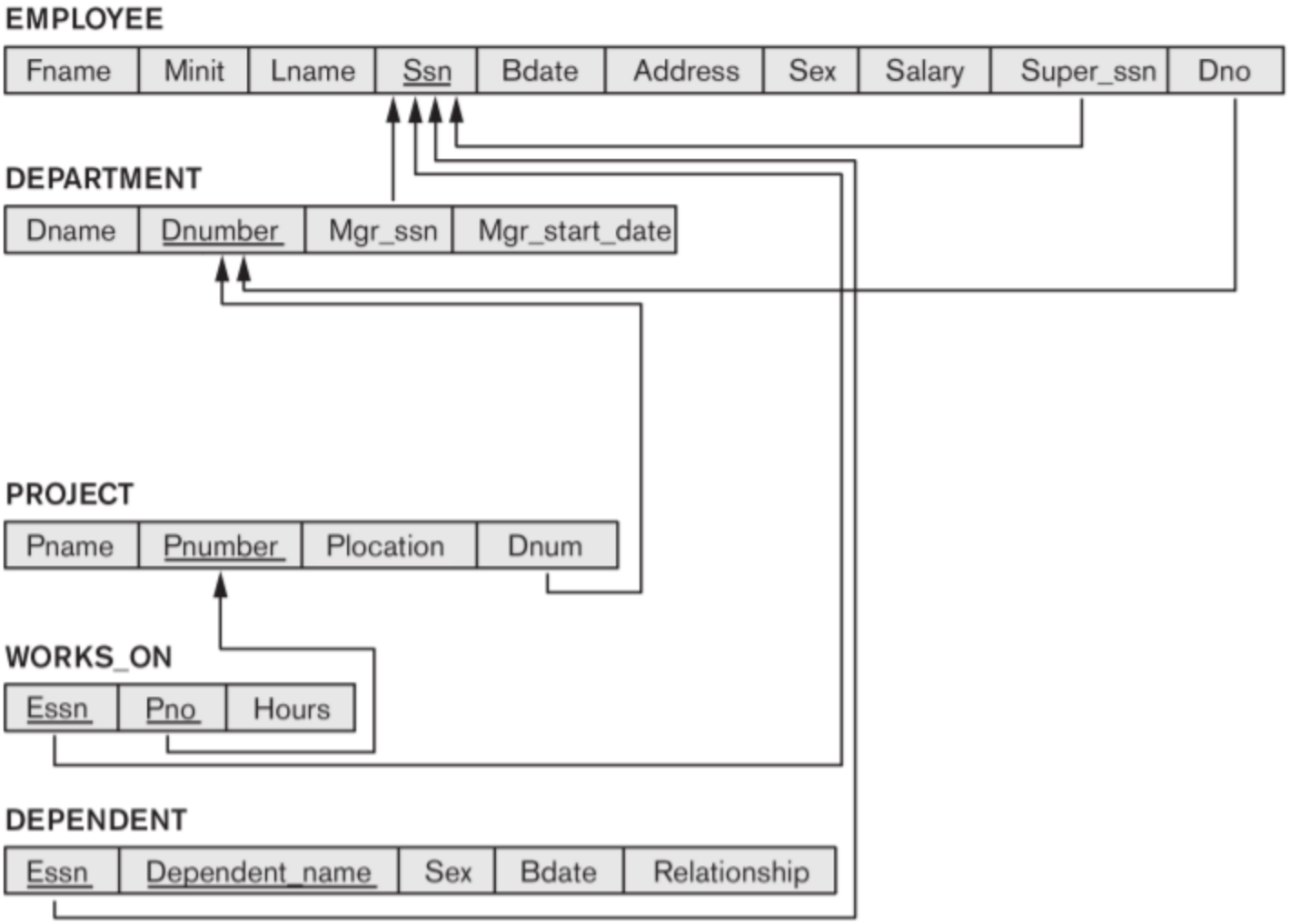
Step 4

- Now we need to figure out the entities from ER diagram for which there exists a 1-to-N relationship.
- The entities for which there exists a 1-to-N relationship, choose a relation as S as the type at N-side of relationship and other as T.
- Then we add as a foreign key to S all of the primary key attributes of T.
- In the given ER diagram there are two 1-to-N relationships that exists between Employee-Department and Employee-Dependent entity.
- The 1-to-N mapping between Employee-Department and Employee-Dependent is depicted in the figure below.


EMPLOYEE
Fname
Minit
Lname
Ssn
Bdate
Address
Sex
Salary
Super_ssn
Dno
DEPARTMENT
Dname
Dnumber
Mgr_ssn
Mgr_start_date
PROJECT
Pname
Pnumber
Plocation
Dnum
DEPENDENT
Essn
Dependent_name
Sex
Bdate
Relationship

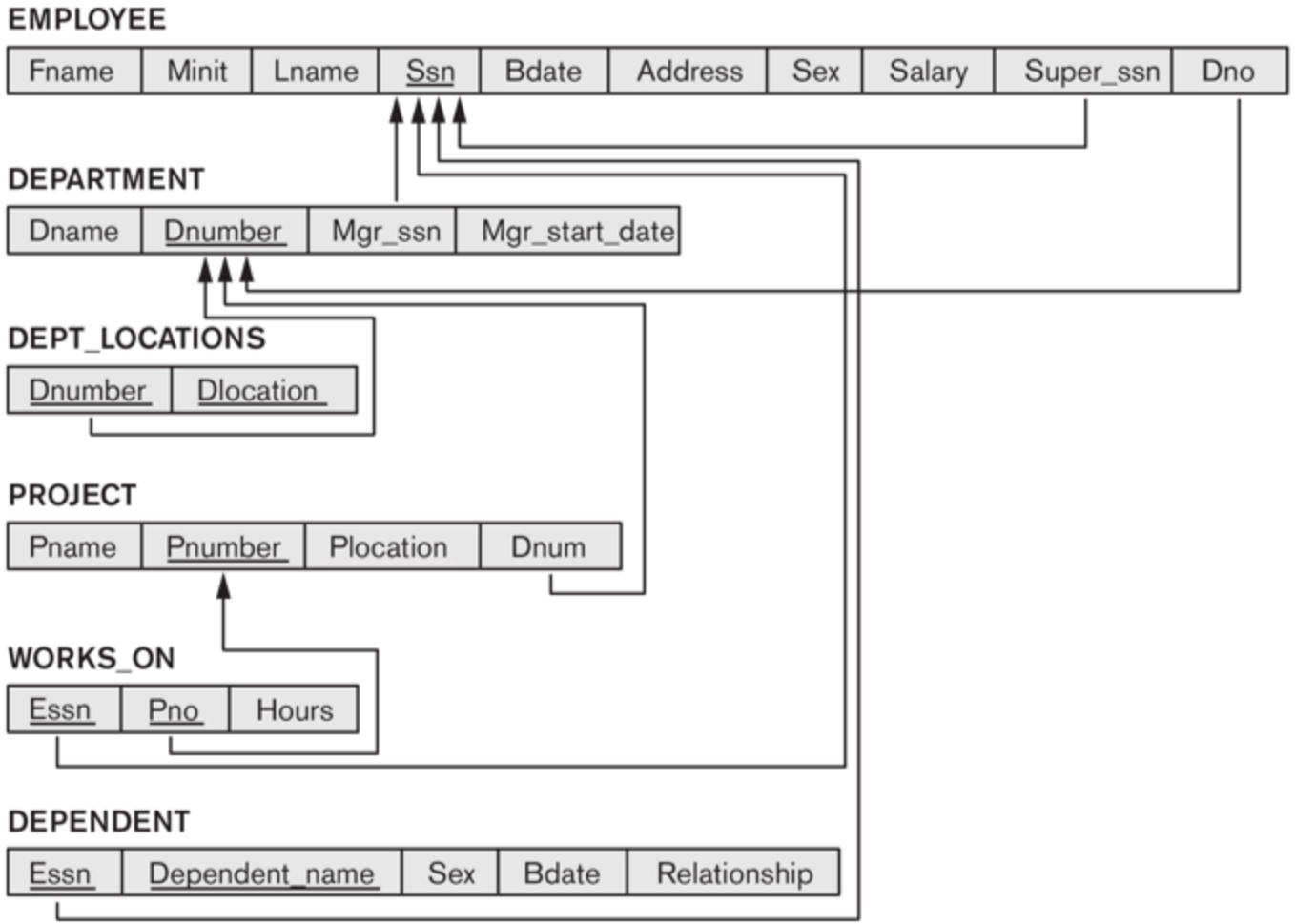
Step 5

- Now we need to figure out the entities from ER diagram for which there exists an M-to-N relationship.
- Create a new relation(table) S.
- The primary keys of relations(tables) between which M-to-N relationship exists, are added to the new relation S created, that acts as a foreign key.
- Then we,add any simple attributes of the M-to-N relationship to S.
- For the given ER-Diagram there exists M-to-N relationship between Employee and Project entity.
- The new table Works_On is created for mapping the relationship between Employee and Project relation(table).


EMPLOYEE
Fname
Minit
Lname
Ssn
Bdate
Address
Sex
Salary
Super_ssn
Dno
DEPARTMENT
Dname
Dnumber
Mgr_ssn
Mgr_start_date
PROJECT
Pname
Pnumber
Plocation
Dnum
WORKS_ON
Essn
Pno
Hours
DEPENDENT
Essn
Dependent_name
Sex
Bdate
Relationship

Step 6

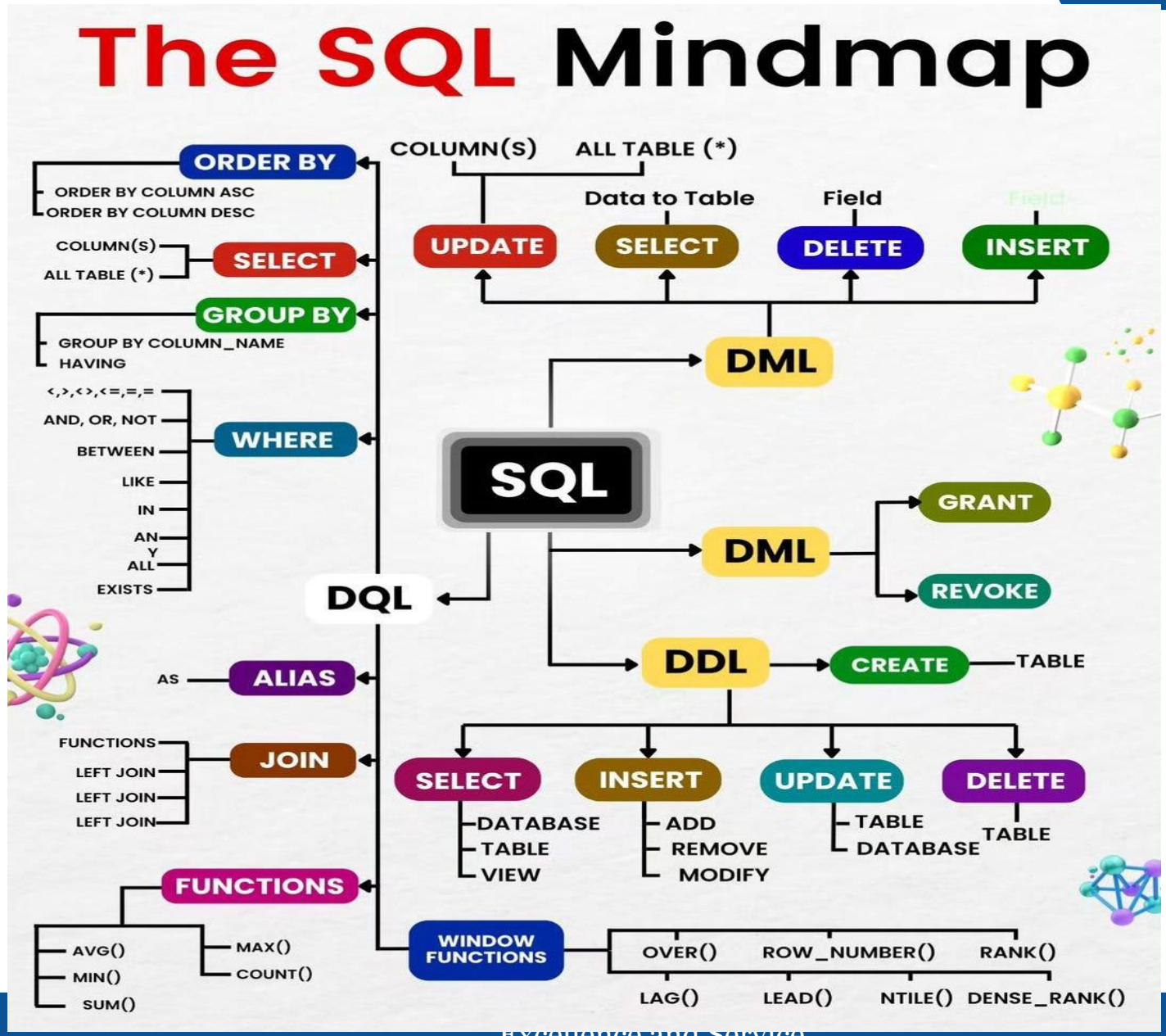
- Now identify the relations(tables) that contain multi-valued attributes.
- Then we need to create a new relation S
- In the new relation S we add as foreign keys the primary keys of the corresponding relation.
- Then we add the multi-valued attribute to S; the combination of all attributes in S forms the primary key.
- For the given ER-Diagram there exists a multi-valued attribute (*Locations*) in Department relation(table).
- So, we create a new relation called *Dept_Locations*. To this new relation we add the primary key of *Department* Table that is *D_Number* and the multi-valued attribute *Locations.*
-


EMPLOYEE
Fname
Minit
Lname
Ssn
Bdate
Address
Sex
Salary
Super_ssn
Dno
DEPARTMENT
Dname
Dnumber
Mgr_ssn
Mgr_start_date
DEPT_LOCATIONS
Dnumber
Dlocation
PROJECT
Pname
Pnumber
Plocation
Dnum
WORKS_ON
Essn
Pno
Hours
DEPENDENT
Essn
Dependent_name
Sex
Bdate
Relationship

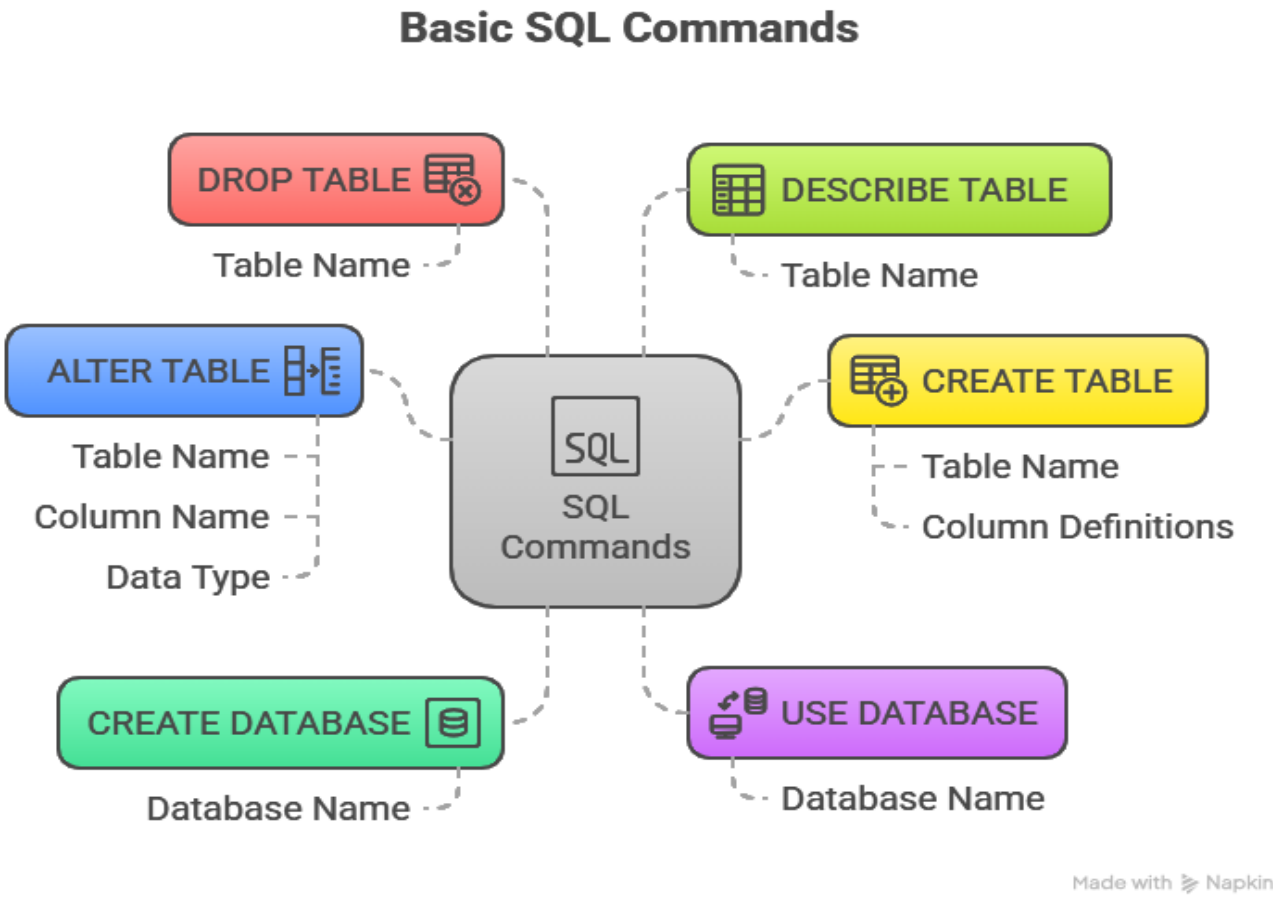
Correspondence between ER and Relational Models

ER MODEL	RELATIONAL MODEL
Entity type	*Entity* relation
1:1 or 1:N relationship type	Foreign key (or *relationship* relation)
M:N relationship type	*Relationship* relation and *two* foreign keys
n-ary relationship type	*Relationship* relation and *n* foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

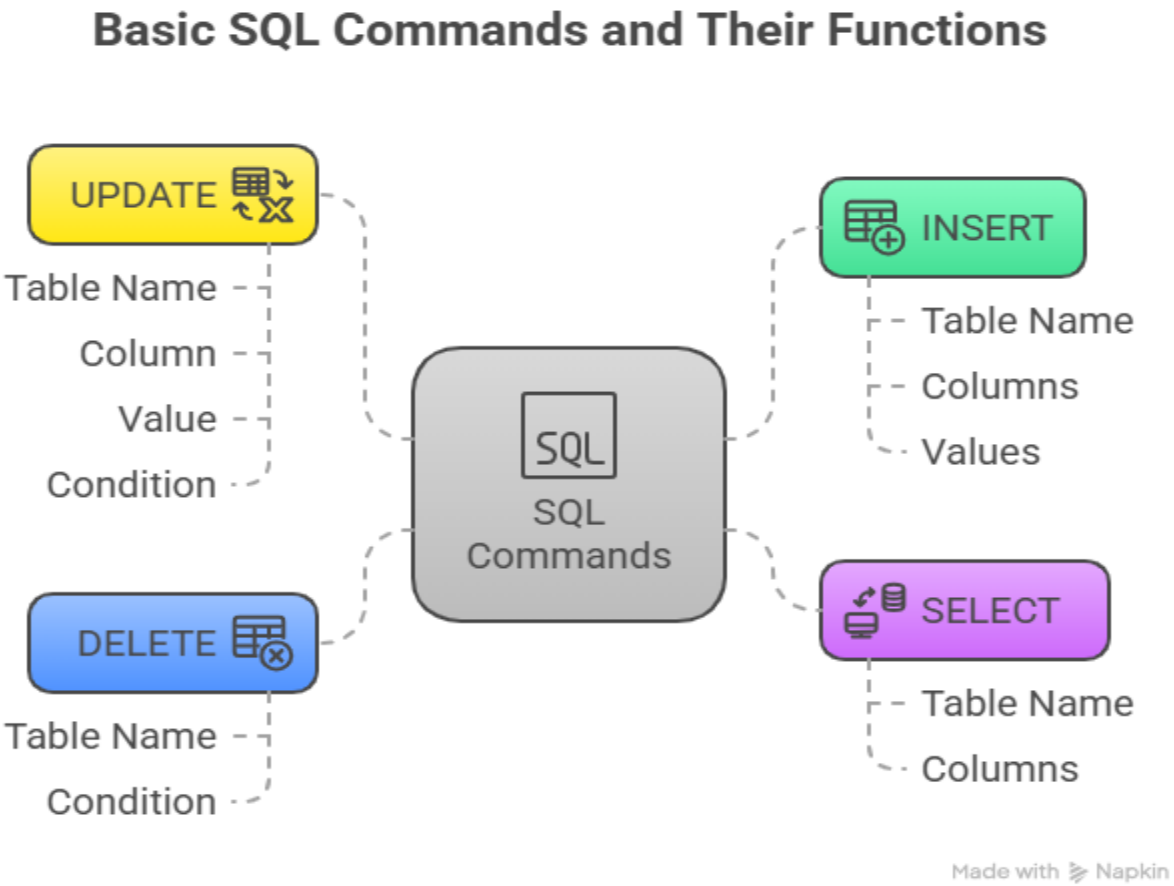
SQL (Basic, Intermediate and Advanced)


The SQL Mindmap
SQL
DML
DQL
DML
DDL
ORDER BY
ORDER BY COLUMN ASC
ORDER BY COLUMN DESC
SELECT
COLUMN(S)
ALL TABLE (*)
GROUP BY
GROUP BY COLUMN_NAME
HAVING
WHERE
<,>,<>,<=,=,=
AND, OR, NOT
BETWEEN
LIKE
IN
ANY
ALL
EXISTS
ALIAS
AS
JOIN
FUNCTIONS
LEFT JOIN
LEFT JOIN
LEFT JOIN
FUNCTIONS
AVG()
MIN()
SUM()
MAX()
COUNT()
WINDOW FUNCTIONS
OVER()
ROW_NUMBER()
RANK()
LAG()
LEAD()
NTILE()
DENSE_RANK()
COLUMN(S)
ALL TABLE (*)
Data to Table
Field
UPDATE
SELECT
DELETE
INSERT
GRANT
REVOKE
CREATE
TABLE
SELECT
INSERT
UPDATE
DELETE
DATABASE
TABLE
VIEW
ADD
REMOVE
MODIFY
TABLE
DATABASE
TABLE

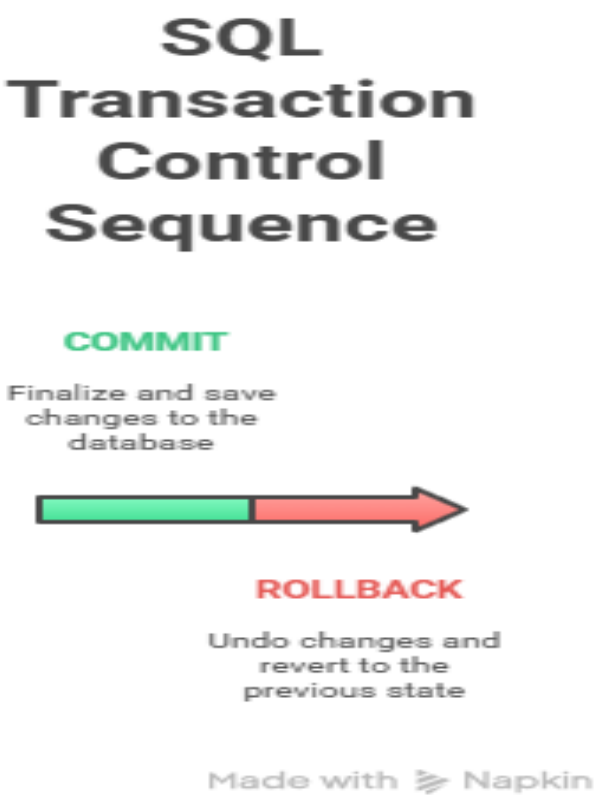
DDL


Basic SQL Commands
DROP TABLE
Table Name
DESCRIBE TABLE
Table Name
ALTER TABLE
Table Name
Column Name
Data Type
SQL
SQL Commands
CREATE TABLE
Table Name
Column Definitions
CREATE DATABASE
Database Name
USE DATABASE
Database Name
Made with Napkin

DML


Basic SQL Commands and Their Functions
UPDATE
Table Name
Column
Value
Condition
DELETE
Table Name
Condition
SQL
SQL Commands
INSERT
Table Name
Columns
Values
SELECT
Table Name
Columns
Made with Napkin

TCL


SQL
Transaction
Control
Sequence
COMMIT
Finalize and save changes to the database
ROLLBACK
Undo changes and revert to the previous state
Made with Napkin

Intermediate SQL Commands

- DDLALTER TABLE table_name MODIFY column datatype;
- ALTER TABLE table_name DROP COLUMN column_name;
- RENAME TABLE old_name TO new_name;
- CREATE INDEX idx_name ON table_name (column_name);

DML + Query Features
SELECT DISTINCT column FROM table_name;
SELECT column FROM table_name WHERE column BETWEEN val1 AND val2;
SELECT column FROM table_name WHERE column LIKE 'A%';GROUP BY, HAVING, ORDER BY clauses
JOIN operations (INNER, LEFT, RIGHT JOIN)

DCL (Data Control Language)
GRANT SELECT, INSERT ON table_name TO user_name;
REVOKE INSERT ON table_name FROM user_name;

Advanced SQL Commands

- Nested/Correlated Subqueries
- Common Table Expressions (CTE)
- Set Operators
- Views and Procedures
- Triggers and Transactions