Chapter 3

MORE SQL

Outline

- More Complex SQL Retrieval Queries
- Specifying Constraints as Assertions and Actions as Triggers
- Views in SQL
- Schema Change Statements in SQL
- Database Programming

NULLS IN SQL QUERIES

- SQL allows queries that check if a value is NULL (missing or undefined or not applicable)
- SQL uses **IS** or **IS NOT** to compare NULLs because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate .
- Query 14: Retrieve the names of all employees who do not have supervisors.

```
Q14:    SELECT   FNAME, LNAME
        FROM     EMPLOYEE
        WHERE    SUPERSSN  IS  NULL
```

Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result

NESTING OF QUERIES

- A complete SELECT query, called a *nested query* , can be specified within the WHERE-clause of another query, called the *outer query*
- Many of the previous queries can be specified in an alternative form using nesting
- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:  SELECT        FNAME, LNAME, ADDRESS
     FROM          EMPLOYEE
     WHERE         DNO IN  (SELECT  DNUMBER
     FROM          DEPARTMENT
     WHERE         DNAME='Research' )
```

CORRELATED NESTED QUERIES

- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query* , the two queries are said to be *correlated*
- The result of a correlated nested query is *different for each tuple (or combination of tuples) of the relation(s) the outer query*
- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q12: SELECT     E.FNAME, E.LNAME
     FROM       EMPLOYEE AS E
     WHERE      E.SSN IN (SELECT    ESSN
                FROM  DEPENDENT
                WHERE         ESSN=E.SSN AND
                      E.FNAME=DEPENDENT_NAME)
```

CORRELATED NESTED QUERIES (cont.)

- In Q12, the nested query has a different result *for each tuple* in the outer query
- A query written with nested SELECT... FROM... WHERE... blocks and using the = or IN comparison operators can ***always*** be expressed as a single block query. For example, Q12 may be written as in Q12A

```
Q12A:    SELECT    E.FNAME, E.LNAME
         FROM      EMPLOYEE E, DEPENDENT D
         WHERE     E.SSN=D.ESSN AND
                   E.FNAME=D.DEPENDENT_NAME
```

THE EXISTS FUNCTION

- EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not
- We can formulate Query 12 in an alternative form that uses EXISTS as Q12B below

THE EXISTS FUNCTION (cont.)

- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q12B:   SELECT    FNAME, LNAME
        FROM      EMPLOYEE
        WHERE     EXISTS    (SELECT   *
                  FROM      DEPENDENT
                  WHERE     SSN=ESSN AND
                  FNAME=DEPENDENT_NAME)
```

THE EXISTS FUNCTION (cont.)

- Query 6: Retrieve the names of employees who have no dependents.

```
Q6:     SELECT   FNAME, LNAME
        FROM     EMPLOYEE
        WHERE    NOT EXISTS  (SELECT   *
                 FROM  DEPENDENT
                 WHERE SSN=ESSN)
```

  – In Q6, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If *none exist* , the EMPLOYEE tuple is selected
  – EXISTS is necessary for the expressive power of SQL

More Comparison Operators

- … {=, >, >=, <, <=, <>} {ANY, SOME, ALL} …
- Query 17: Retrieve all employees whose salary is greater than the salary of all employees in dept. 5

```
Q17: SELECT     *
     FROM       Employee
     WHERE      Salary > ALL (SELECT Salary
                              FROM    Employee
                              WHERE   DNO=5)
```

EXPLICIT SETS

- It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query
- Query 13: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

```
Q13:    SELECT    DISTINCT ESSN
        FROM      WORKS_ON
        WHERE     PNO IN  (1, 2, 3)
```

Renaming of Attributes in SQL

- Rename any attribute that appears in the result of a query by adding the qualifier AS followed by the desired new name

Q8A: SELECT E.Lname AS Employee_name, S.Lname AS Supervisor_name

FROM EMPLOYEE AS E, EMPLOYEE AS S

WHERE E.Super_ssn=S.Ssn

SET OPERATIONS

- SQL has directly incorporated some set operations
- There is a union operation (**UNION)**, and in *some versions* of SQL there are set difference (**MINUS or EXCEPT)** and intersection (**INTERSECT)** operations
- The resulting relations of these set operations are sets of tuples; *duplicate tuples are eliminated from the result*
- The set operations apply only to *union compatible relations* ; the two relations must have the same attributes and the attributes must appear in the same order

SET OPERATIONS (cont.)

- Query 4: Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

```
Q4:  (SELECT  PNAME
      FROM          PROJECT, DEPARTMENT, EMPLOYEE
      WHERE         DNUM=DNUMBER AND MGRSSN=SSN
      AND           LNAME='Smith')
      UNION         (SELECT  PNAME
      FROM          PROJECT, WORKS_ON, EMPLOYEE
      WHERE         PNUMBER=PNO AND ESSN=SSN AND
                         LNAME='Smith')
```

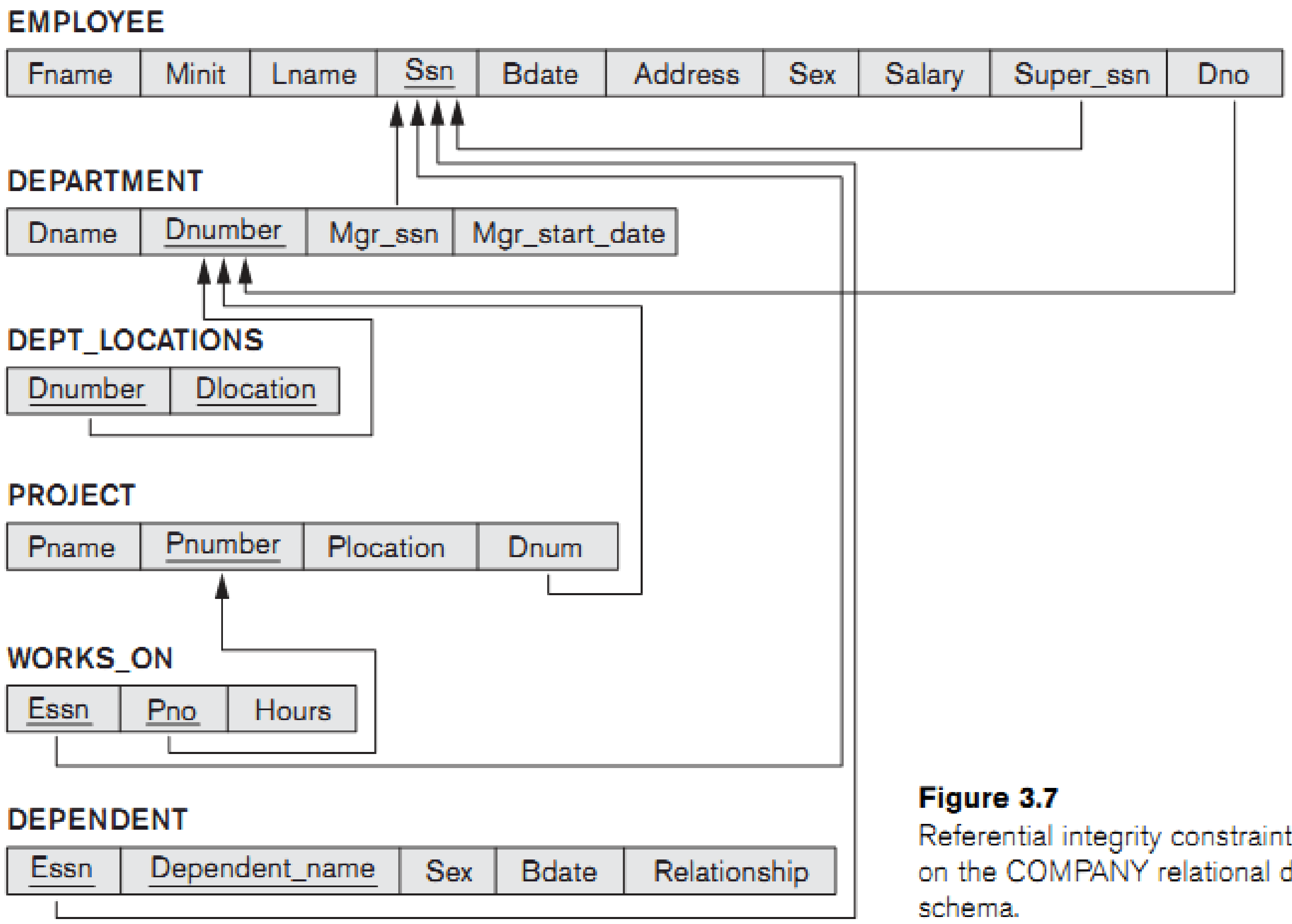


Figure 3.7
Referential integrity constraints displayed on the COMPANY relational database schema.

SET OPERATIONS (cont.)

- Q3: Retrieve the name of each employee who works on all the projects controlled by department number 5

- **Q3A: SELECT** Fname, Lname

  FROM EMPLOYEE

  WHERE NOT EXISTS

  ((**SELECT** Pnumber **FROM** PROJECT **WHERE** Dnum=5)

  EXCEPT (**SELECT** Pno **FROM** WORKS_ON **WHERE** Ssn=Essn));

Joined Relations Feature in SQL2

- Can specify a "joined relation" in the FROM-clause
- Looks like any other relation but is the result of a join
- Includes inner join and outer join
- Allows the user to specify different types of joins (regular "theta" JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN, CROSS JOIN, etc)

Joined Relations Feature in SQL2 (cont.)

- **Q1: SELECT FNAME, LNAME, ADDRESS**
  FROM EMPLOYEE, DEPARTMENT
  WHERE DNAME='Research' AND DNUMBER=DNO
- could be written as:

```
Q1:  SELECT      FNAME, LNAME, ADDRESS
     FROM       (EMPLOYEE JOIN DEPARTMENT
                 ON DNUMBER=DNO)
     WHERE       DNAME='Research'
```

or as:

```
Q1:  SELECT      FNAME, LNAME, ADDRESS
     FROM        (EMPLOYEE NATURAL JOIN (DEPARTMENT
                 AS DEPT(DNAME, DNO, MSSN, MSDATE)))
     WHERE       DNAME='Research'
```

Joined Relations Feature in SQL2 (cont.)

```
Q8:  SELECT        E.FNAME, E.LNAME, S.FNAME, S.LNAME
     FROM          EMPLOYEE E S
     WHERE         E.SUPERSSN=S.SSN
```

can be written as:

```
Q8:  SELECT        E.FNAME, E.LNAME, S.FNAME, S.LNAME
     FROM  (EMPLOYEE E LEFT OUTER JOIN EMPLOYEES
           ON  E.SUPERSSN=S.SSN)
```

Joined Relations Feature in SQL2 (cont.)

- Another Example;

```
Q2: SELECT Pnumber, Dnum, Lname, Address, Bdate
      FROM   PROJECT, DEPARTMENT, EMPLOYEE
     WHERE Dnum=Dnumber AND Mgr_ssn=Ssn AND
                Plocation='Stafford';
```

```
Q2: SELECT  PNUMBER, DNUM, LNAME,
            BDATE, ADDRESS
    FROM    (PROJECT JOIN
            DEPARTMENT ON DNUM=DNUMBER)
           JOIN EMPLOYEE ON MGRSSN=SSN) )
    WHERE   PLOCATION='Stafford'
```

AGGREGATE FUNCTIONS

- Include **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**
- Query 15: Find the maximum salary, the minimum salary, and the average salary among all employees.

```
Q15:    SELECT   MAX(SALARY),
                 MIN(SALARY), AVG(SALARY)
        FROM     EMPLOYEE
```

  – Some SQL implementations *may not allow more than one function* in the SELECT-clause

AGGREGATE FUNCTIONS (cont.)

- Query 16: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

```
Q16: SELECT    MAX(SALARY), MIN(SALARY),
               AVG(SALARY)
     FROM      EMPLOYEE, DEPARTMENT
     WHERE     DNO=DNUMBER AND
               DNAME='Research'
```

AGGREGATE FUNCTIONS (cont.)

- Queries 17 and 18: Retrieve the total number of employees in the company (Q17), and the number of employees in the 'Research' department (Q18).

```
Q17:    SELECT    COUNT (*)
        FROM      EMPLOYEE

Q18:    SELECT    COUNT (*)
        FROM      EMPLOYEE, DEPARTMENT
        WHERE     DNO=DNUMBER AND
                  DNAME='Research’
```

GROUPING

- In many cases, we want to apply the aggregate functions *to subgroups of tuples in a relation*
- Each subgroup of tuples consists of the set of tuples that have *the same value* for the *grouping attribute(s)*
- The function is applied to each subgroup independently
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*

GROUPING (cont.)

- Query 20: For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q20:SELECT       DNO, COUNT (*), AVG (SALARY)
    FROM         EMPLOYEE
    GROUP BY     DNO
```

  – In Q20, the EMPLOYEE tuples are divided into groups--each group having the same value for the grouping attribute DNO
  – The COUNT and AVG functions are applied to each such group of tuples separately
  – The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples
  – A join condition can be used in conjunction with grouping

GROUPING (cont.)

- Query 21: For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
Q21:    SELECT     PNUMBER, PNAME, COUNT (*)
        FROM       PROJECT, WORKS_ON
        WHERE      PNUMBER=PNO
        GROUP BY   PNUMBER, PNAME
```

  – In this case, the grouping and functions are applied *after* the joining of the two relations

THE HAVING-CLAUSE

- Sometimes we want to retrieve the values of these functions for only those *groups that satisfy certain conditions*
- The HAVING-clause is used for specifying a selection condition on groups (rather than on individual tuples)

THE HAVING-CLAUSE (cont.)

- Query 22: For each project *on which more than two employees work* , retrieve the project number, project name, and the number of employees who work on that project.

```
Q22:    SELECT     PNUMBER, PNAME, COUNT
                   (*)
        FROM       PROJECT, WORKS_ON
        WHERE      PNUMBER=PNO
        GROUP BY   PNUMBER, PNAME
        HAVING     COUNT (*) > 2
```

ORDER BY

- The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s) in ascending or descending order
- Query 28: Retrieve a list of employees and the projects each works in, ordered by the employee's department, and within each department ordered alphabetically by employee last name.

```
Q28:        SELECT      DNAME, LNAME, FNAME, PNAME
            FROM        DEPARTMENT, EMPLOYEE,
                        WORKS_ON, PROJECT
            WHERE       DNUMBER=DNO AND SSN=ESSN
            AND         PNO=PNUMBER
            ORDER BY    DNAME, LNAME [DESC|ASC]
```

Outline

- More Complex SQL Retrieval Queries
- **Specifying Constraints as Assertions and Actions as Triggers**
- Views in SQL
- Schema Change Statements in SQL
- Database Programming

Constraints as Assertions

- General constraints: constraints that do not fit in the basic SQL categories
- Mechanism: `CREAT ASSERTION`
  - components include: a constraint name, followed by `CHECK`, followed by a condition

Assertions: An Example

- “The salary of an employee must not be greater than the salary of the manager of the department that the employee works for’’

```
CREAT ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS (SELECT *
 FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
 WHERE E.SALARY > M.SALARY AND
       E.DNO=D.NUMBER AND D.MGRSSN=M.SSN))
```

SQL Triggers

- Objective: to monitor a database and take action when a condition occurs
- Triggers are expressed in a syntax similar to assertions and include the following:
  - event (e.g., an update operation)
  - condition
  - action (to be taken when the condition is satisfied)

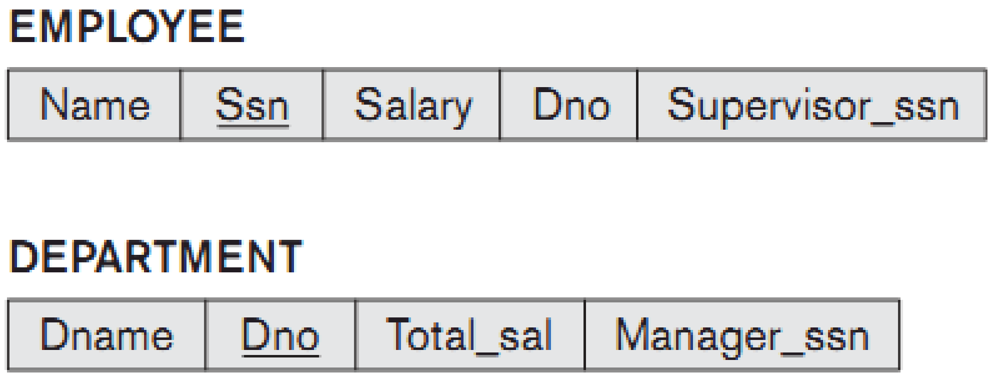
SQL Triggers: An Example

- A trigger to compare an employee's salary to his/her supervisor during insert or update operations:

```
CREATE TRIGGER INFORM_SUPERVISOR
BEFORE INSERT OR UPDATE OF
   SALARY, SUPERVISOR_SSN ON EMPLOYEE
   FOR EACH ROW
WHEN
   (NEW.SALARY> (SELECT SALARY FROM EMPLOYEE
                     WHERE SSN=NEW.SUPERVISOR_SSN))
   INFORM_SUPERVISOR (NEW.SUPERVISOR_SSN,NEW.SSN;
```

SQL Triggers

- More example



EMPLOYEE

Name	Ssn	Salary	Dno	Supervisor_ssn

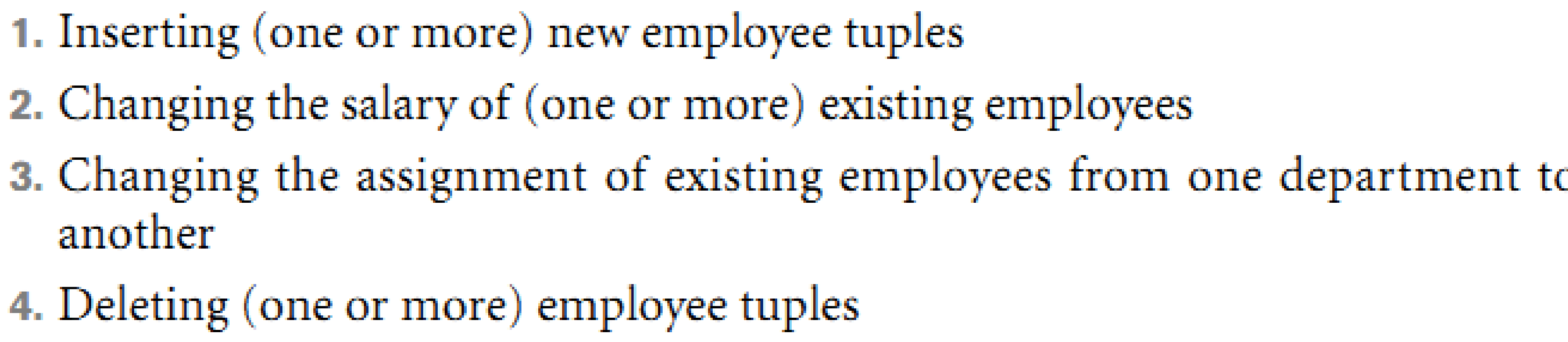
DEPARTMENT

Dname	Dno	Total_sal	Manager_ssn

- How to update Total_sal automatically??

SQL Triggers

- Firstly, determine the events which can impact on what we want



1. Inserting (one or more) new employee tuples
2. Changing the salary of (one or more) existing employees
3. Changing the assignment of existing employees from one department t another
4. Deleting (one or more) employee tuples

- Secondly, for each event, do we need a condition?
  - Exp: event 1: if dno is not null
- Finally, write the action to solve
  - Exp: recompute the total salary of department

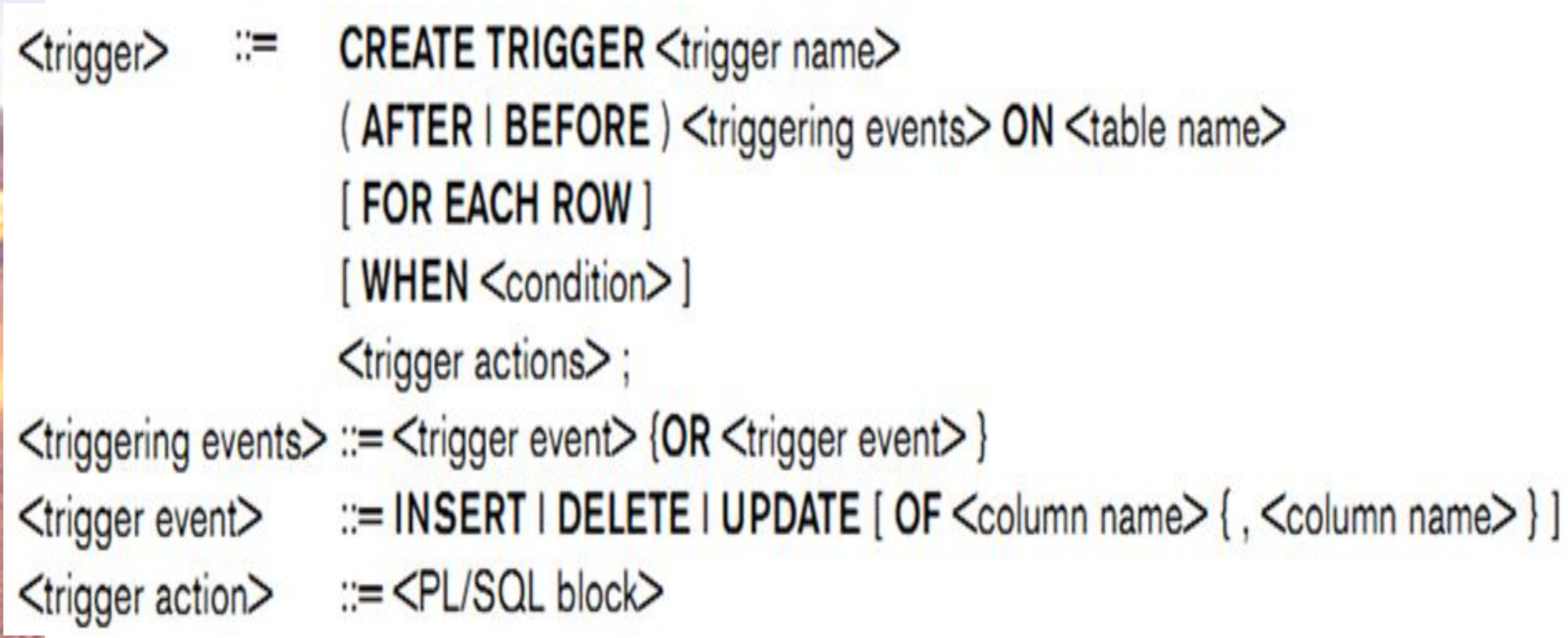
```
R1:  CREATE TRIGGER Total_sal1
     AFTER INSERT ON EMPLOYEE
     FOR EACH ROW
     WHEN ( NEW.Dno IS NOT NULL )
          UPDATE DEPARTMENT
          SET Total_sal = Total_sal + NEW.Salary
          WHERE Dno = NEW.Dno;
```

```
R4:  CREATE TRIGGER Total_sal4
     AFTER DELETE ON EMPLOYEE
     FOR EACH ROW
     WHEN ( OLD.Dno IS NOT NULL  )
          UPDATE DEPARTMENT
          SET Total_sal = Total_sal – OLD.Salary
          WHERE Dno = OLD.Dno;
```

```
R2:  CREATE TRIGGER Total_sal2
     AFTER UPDATE OF Salary ON EMPLOYEE
     FOR EACH ROW
     WHEN ( NEW.Dno IS NOT NULL )
          UPDATE DEPARTMENT
          SET Total_sal = Total_sal + NEW.Salary – OLD.Salary
          WHERE Dno = NEW.Dno;

R3:  CREATE TRIGGER Total_sal3
     AFTER UPDATE OF Dno ON EMPLOYEE
     FOR EACH ROW
          BEGIN
          UPDATE DEPARTMENT
          SET Total_sal = Total_sal + NEW.Salary
          WHERE Dno = NEW.Dno;
          UPDATE DEPARTMENT
          SET Total_sal = Total_sal – OLD.Salary
          WHERE Dno = OLD.Dno;
          END;
```

SQL Triggers - Syntax



```
<trigger>          ::=  CREATE TRIGGER <trigger name>
                        ( AFTER I BEFORE ) <triggering events> ON <table name>
                        [ FOR EACH ROW ]
                        [ WHEN <condition> ]
                        <trigger actions> ;
<triggering events> ::= <trigger event> {OR <trigger event> }
<trigger event>     ::= INSERT I DELETE I UPDATE [ OF <column name> { , <column name> } ]
<trigger action>    ::= <PL/SQL block>
```

Outline

- More Complex SQL Retrieval Queries
- Specifying Constraints as Assertions and Actions as Triggers
- **Views in SQL**
- Schema Change Statements in SQL

Views in SQL

- A view is a “virtual” table that is derived from other tables
- Allows for limited update operations (since the table may not physically be stored)
- Allows full query operations
- A convenience for expressing certain operations

Specification of Views

- SQL command: `CREATE VIEW`
  - a table (view) name
  - a possible list of attribute names (for example, when arithmetic operations are specified or when we want the names to be different from the attributes in the base relations)
  - a query to specify the table contents

SQL Views: An Example

- Specify a different WORKS_ON table

```
CREATE VIEW WORKS_ON_NEW AS
SELECT FNAME, LNAME, PNAME, HOURS
   FROM EMPLOYEE, PROJECT, WORKS_ON
   WHERE SSN=ESSN AND PNO=PNUMBER
   GROUP BY PNAME;
```

Using a Virtual Table

- We can specify SQL queries on a newly table (view):

```
SELECT FNAME, LNAME FROM WORKS_ON_NEW
WHERE PNAME='Seena';
```

- When no longer needed, a view can be dropped:

```
DROP VIEW WORKS_ON_NEW;
```

Efficient View Implementation

- Query modification: present the view query in terms of a query on the underlying base tables
  - disadvantage: inefficient for views defined via complex queries (especially if additional queries are to be applied to the view within a short time period)
  - Exp:

```
SELECT    Fname, Lname
FROM      EMPLOYEE, PROJECT, WORKS_ON
WHERE     Ssn=Essn AND Pno=Pnumber
          AND Pname='ProductX';
```

Efficient View Implementation

- View materialization: involves physically creating and keeping a temporary table
  - assumption: other queries (such as insert, update, delete, etc.) on the view will follow
  - concerns: maintaining correspondence between the base table and the view when the base table is updated
  - strategy: incremental update
  - kept as a materialized (physically stored) table as long as it is being queried

View Update

- Update on a single view without aggregate operations: update may map to an update on the underlying base table
- Views involving joins: an update *may* map to an update on the underlying base relations
  – not always possible

```
CREATE VIEW WORKS_ON1
AS SELECT   Fname, Lname, Pname, Hours
FROM        EMPLOYEE, PROJECT, WORKS_ON
WHERE       Ssn=Essn AND Pno=Pnumber;
```

```
UPDATE WORKS_ON1
SET     Pname = 'ProductY'
WHERE   Lname='Smith' AND Fname='John'
        AND Pname='ProductX';
```

```
UPDATE WORKS_ON
SET       Pno =    ( SELECT   Pnumber
                     FROM     PROJECT
                     WHERE    Pname='ProductY' )
WHERE     Essn IN  ( SELECT   Ssn
                     FROM     EMPLOYEE
                     WHERE    Lname='Smith' AND Fname='John' )
          AND
          Pno =    ( SELECT   Pnumber
                     FROM     PROJECT
                     WHERE    Pname='ProductX' );
```

```
UPDATE PROJECT    SET    Pname = 'ProductY'
WHERE      Pname = 'ProductX';
```

Un-updatable Views

- Views defined using groups and aggregate functions are not updateable
- Views defined on multiple tables using joins are generally not updateable
- CREATE VIEW syntax has a **WITH CHECK OPTION** will prevent data being added or modified within the view that cannot subsequently be retrieved from the view

Un-updatable Views

- WITH CHECK OPTION

Exp:

```
CREATE VIEW emp_dep5
AS
SELECT SSN, Lname, Fname, Dno FROM
EMPLOYEE WHERE DNO = 5 WITH CHECK OPTION
```

INSERT INTO emp_dep5 VALUES (‘111112222’, ‘Bob’, ‘Smith’, 3)

Cannot insert successfully

Views in SQL

```
CREATE VIEW
[<database_name>.][<owner>.]view_na
me [ (column[,...n])]
AS
select_statement
[ WITH CHECK OPTION ]
```

Outline

- More Complex SQL Retrieval Queries
- Specifying Constraints as Assertions and Actions as Triggers
- Views in SQL
- **Schema Change Statements in SQL**

DROP Command

- Two drop behavior options: CASCADE & RESTRICT

Examples:

DROP SCHEMA COMPANY CASCADE;

DROP TABLE DEPENDENT RESTRICT;

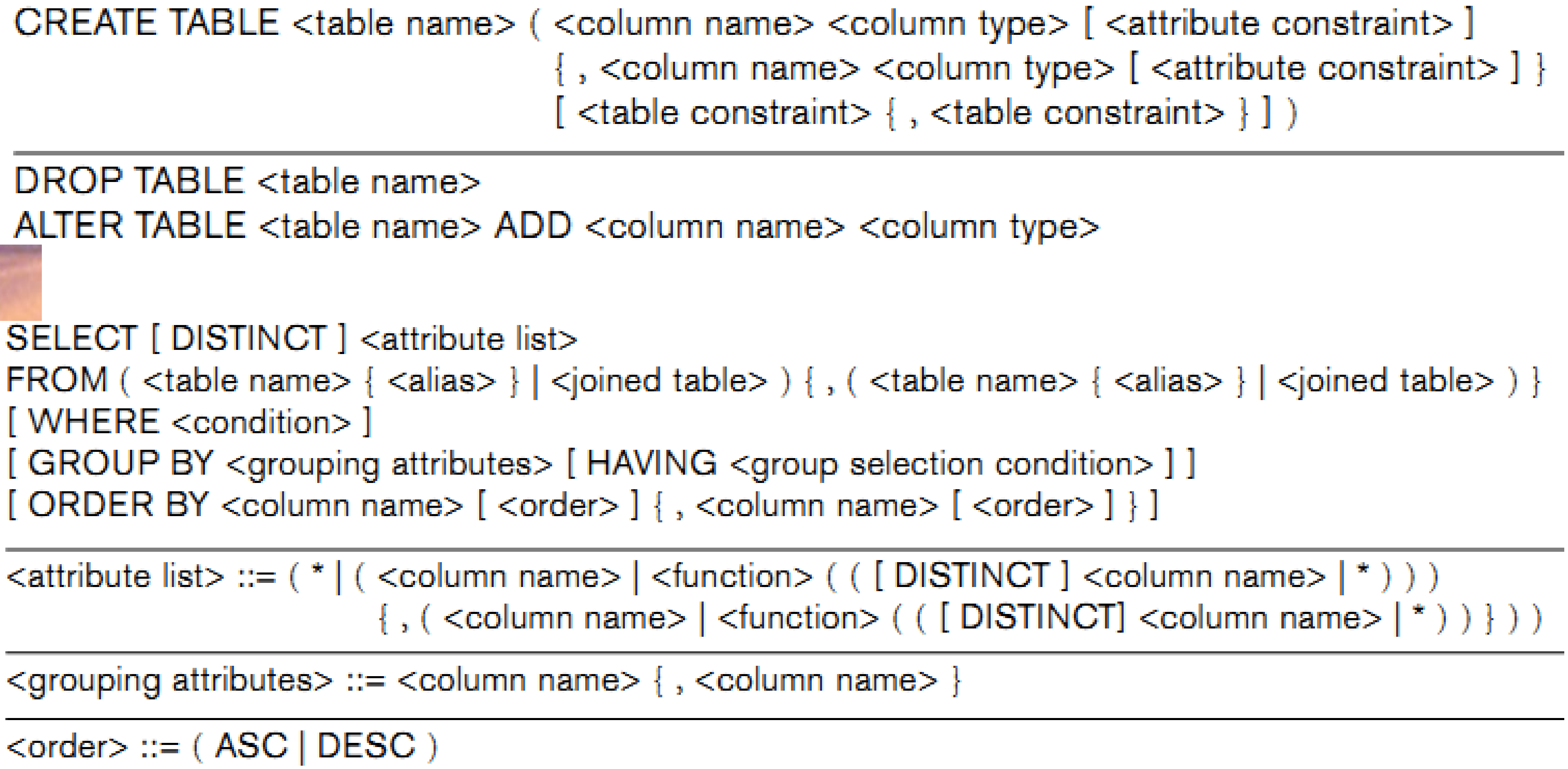
ALTER TABLE

- Used to add an attribute to one of the base relations
- The new attribute will have NULLs in all the tuples of the relation right after the command is executed; hence, the NOT NULL constraint is *not allowed* for such an attribute
- Example:

```
ALTER TABLE  EMPLOYEE  ADD COLUMN JOB VARCHAR(12);
ALTER TABLE COMPANY.EMPLOYEE DROP COLUMN Address
    CASCADE;
ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN
    Mgr_ssn DROP DEFAULT;
ALTER TABLE COMPANY.EMPLOYEE DROP CONSTRAINT
    EMPSUPERFK CASCADE;
```

View the syntax in book or google

Summary



```
CREATE TABLE <table name> ( <column name> <column type> [ <attribute constraint> ]
                            { , <column name> <column type> [ <attribute constraint> ] }
                            [ <table constraint> { , <table constraint> } ] )
```

```
DROP TABLE <table name>
ALTER TABLE <table name> ADD <column name> <column type>
```

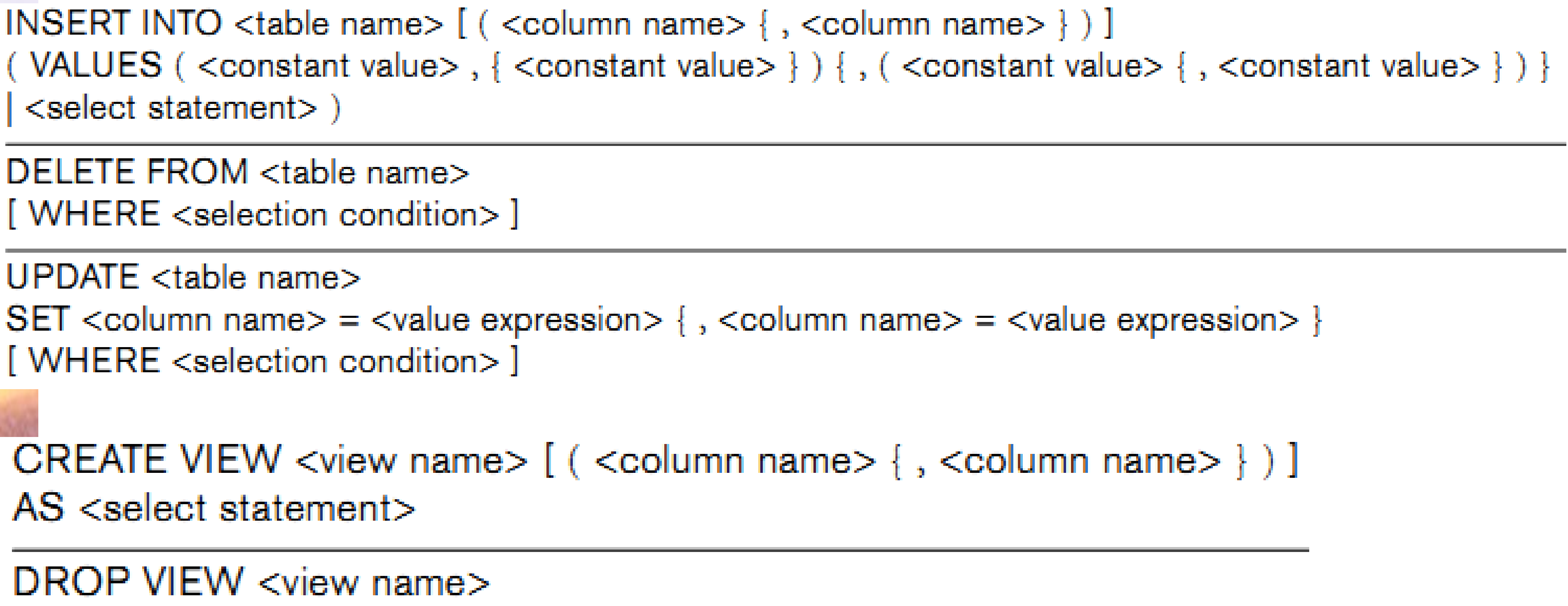
```
SELECT [ DISTINCT ] <attribute list>
FROM ( <table name> { <alias> } | <joined table> ) { , ( <table name> { <alias> } | <joined table> ) }
[ WHERE <condition> ]
[ GROUP BY <grouping attributes> [ HAVING <group selection condition> ] ]
[ ORDER BY <column name> [ <order> ] { , <column name> [ <order> ] } ]
```

```
<attribute list> ::= ( * | ( <column name> | <function> ( ( [ DISTINCT ] <column name> | * ) ) )
                     { , ( <column name> | <function> ( ( [ DISTINCT] <column name> | * ) ) } ) )
```

```
<grouping attributes> ::= <column name> { , <column name> }
```

```
<order> ::= ( ASC | DESC )
```

Summary



```
INSERT INTO <table name> [ ( <column name> { , <column name> } ) ]
( VALUES ( <constant value> , { <constant value> } ) { , ( <constant value> { , <constant value> } ) }
| <select statement> )
```

```
DELETE FROM <table name>
[ WHERE <selection condition> ]
```

```
UPDATE <table name>
SET <column name> = <value expression> { , <column name> = <value expression> }
[ WHERE <selection condition> ]
```

```
CREATE VIEW <view name> [ ( <column name> { , <column name> } ) ]
AS <select statement>
```

```
DROP VIEW <view name>
```

Chapter 4-5

Data Modeling Using the (Enhanced) Entity-Relationship (E-ER) Model

Outline

- Example Database Application (COMPANY)
- 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
- ER Diagram for COMPANY Schema
- Enhanced Entity Diagram

Example COMPANY Database

- 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.
  - Each department *controls* a number of PROJECTs*.* Each project has a name, number and is located at a single location.

Example COMPANY Database (Cont.)

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

ER Model Concepts

- Entities and Attributes
  - Entities are specific objects or things 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 a 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, subrange, enumerated type, …

Types of Attributes (1)

- Simple
  - Each entity has a single atomic value for the attribute. For example, SSN or Sex.
- 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}.

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)}.

Entity Types and Key Attributes

- Entities with the same basic attributes are grouped or typed into **an entity type**. For example, the EMPLOYEE entity type or the PROJECT entity type.
- 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.
- A key attribute may be composite. For example, VehicleTagNumber is a key of the CAR entity type with components (Number, State).
- An entity type may have more than one key. For example, the CAR entity type may have two keys:
  – VehicleIdentificationNumber (popularly called VIN) and
  – VehicleTagNumber (Number, State), also known as license_plate number.

ENTITY SET corresponding to the ENTITY TYPE CAR

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$_3$

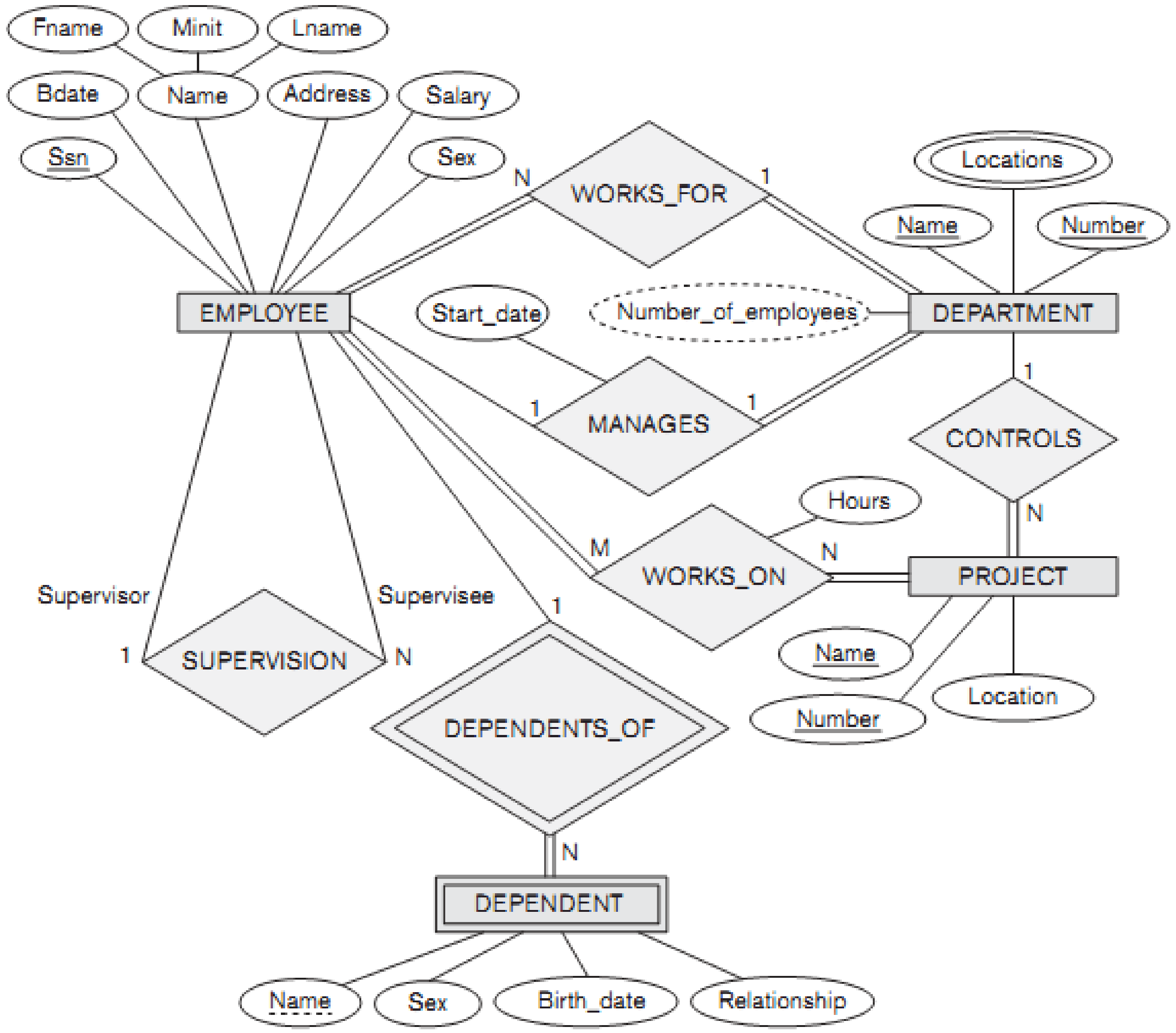
((VSY 720, TEXAS), TD729, Buick LeSabre, 4-door, 2003, (white, blue))

•

•

•

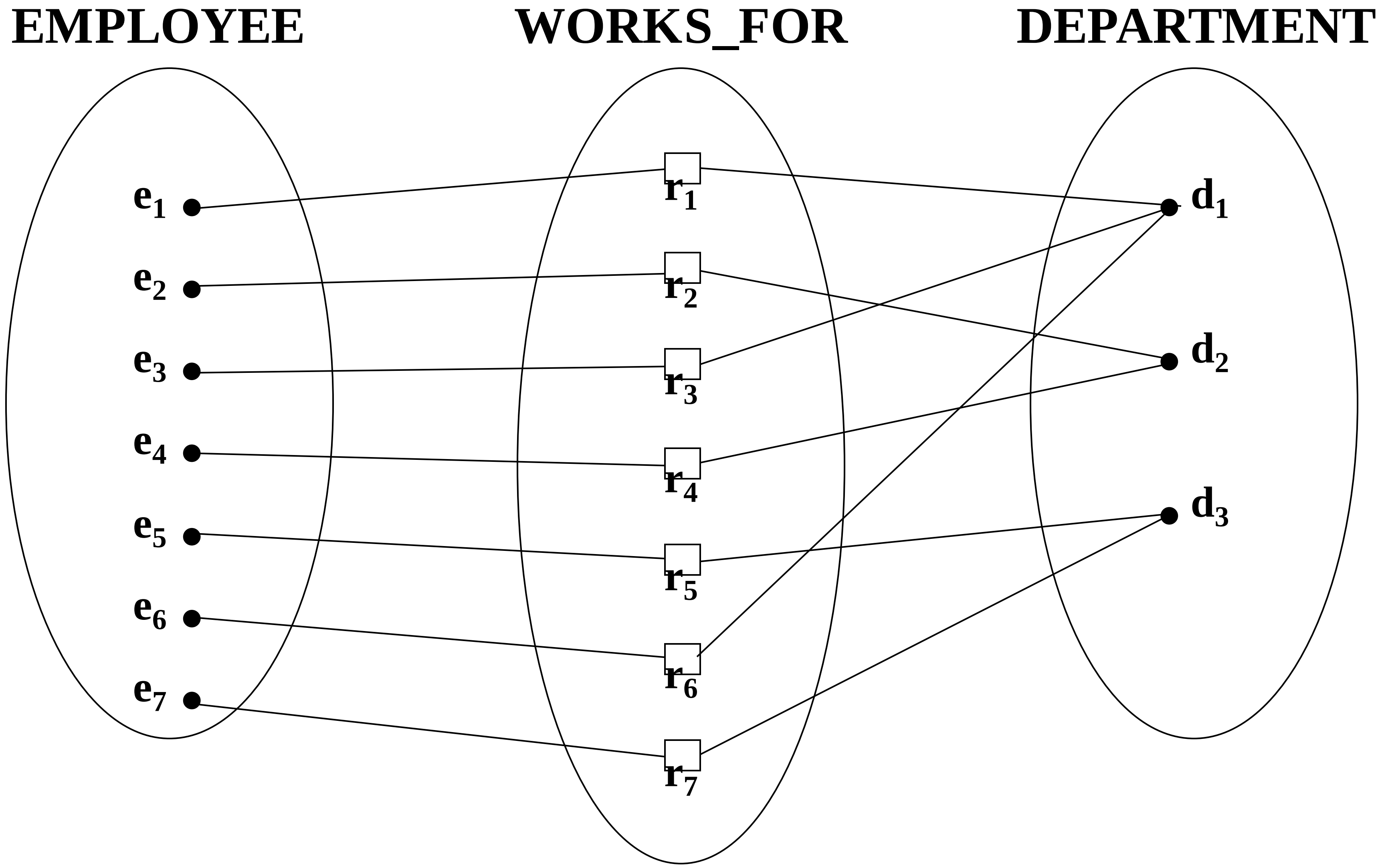
ER DIAGRAM – Entity Types are:
EMPLOYEE, DEPARTMENT, PROJECT, DEPENDENT



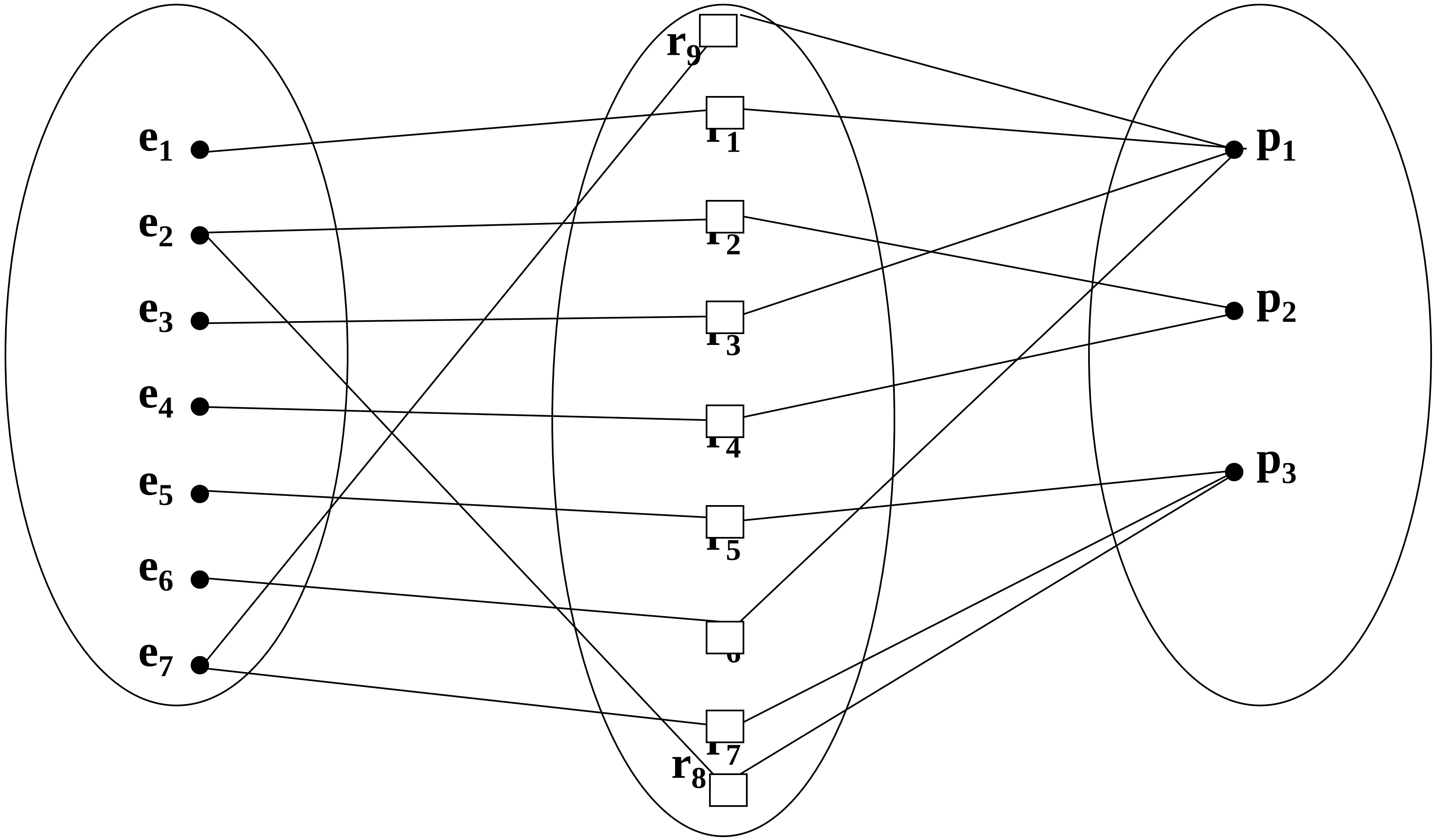
Relationships and Relationship Types (1)

- 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.

Example relationship instances of the WORKS_FOR relationship between EMPLOYEE and DEPARTMENT



Example relationship instances of the WORKS_ON relationship between EMPLOYEE and PROJECT

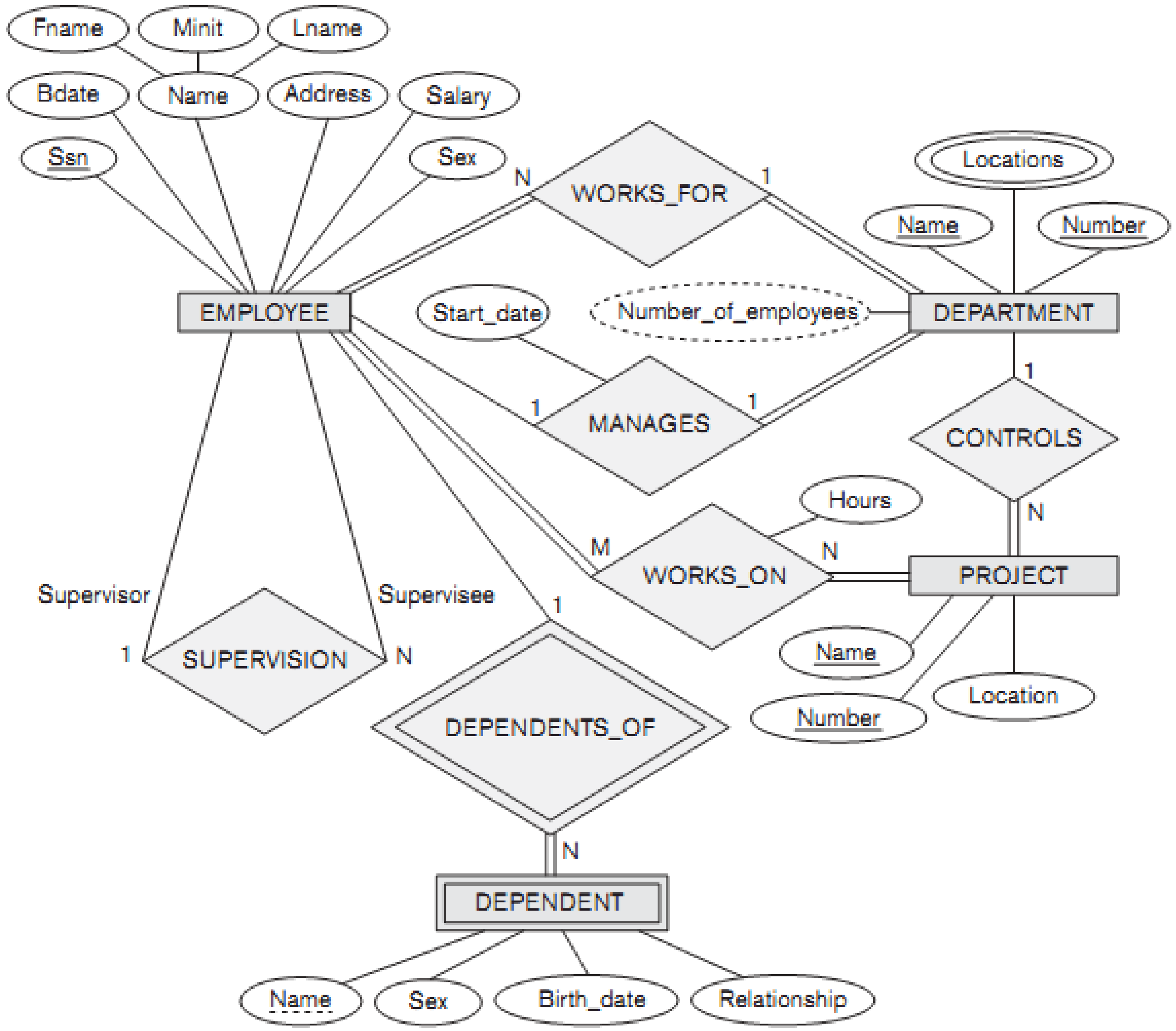


Relationships and Relationship Types (2)

- More than one relationship type can **exist with the same participating entity types.** For example, MANAGES and WORKS_FOR are distinct relationships between EMPLOYEE and DEPARTMENT, but with different meanings and different relationship instances.

ER DIAGRAM – Relationship Types are:

WORKS_FOR, MANAGES, WORKS_ON, CONTROLS, SUPERVISION, DEPENDENTS_OF



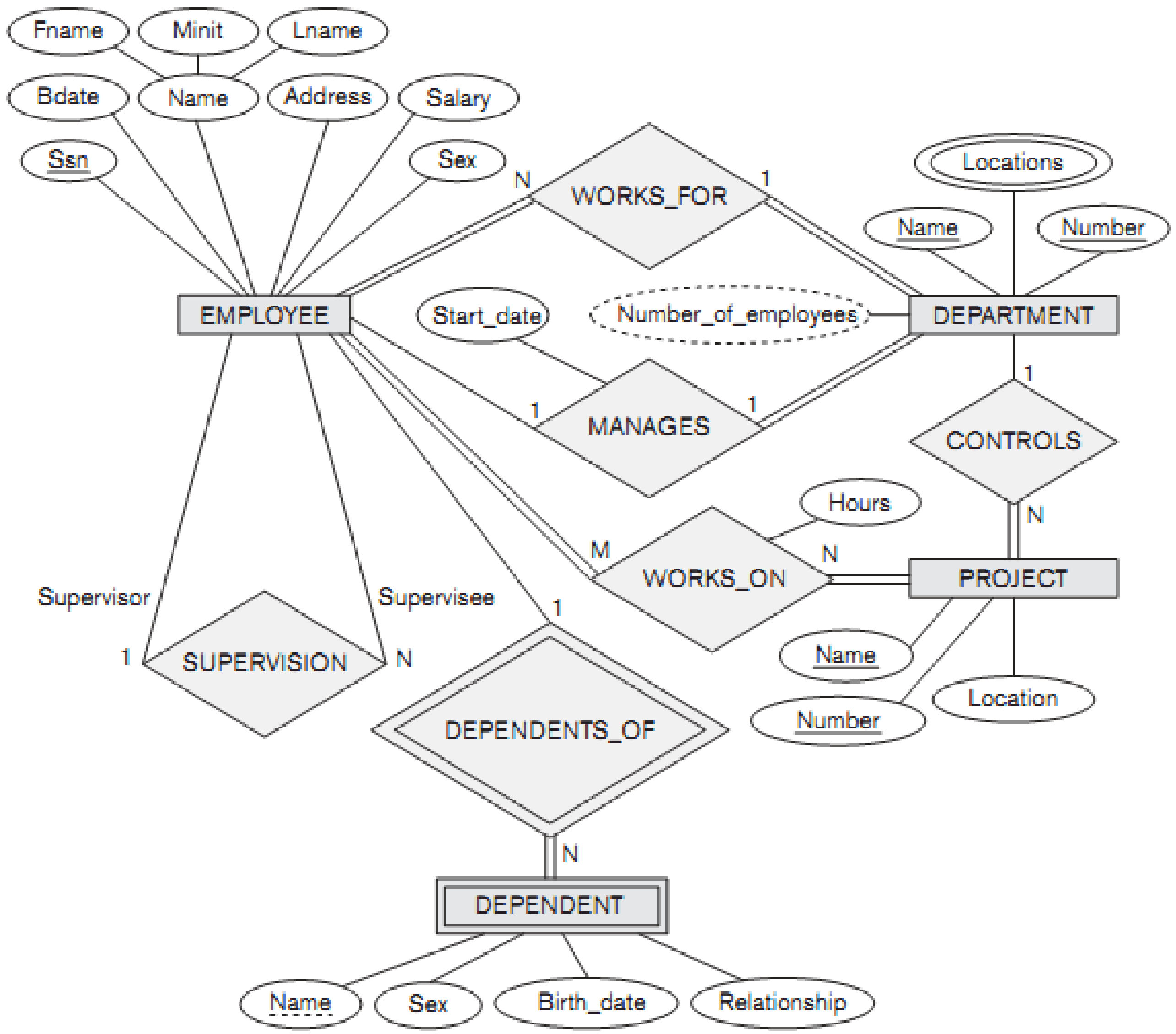
Weak Entity Types

- An entity that **does not have** a key attribute
- 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 entity type

Example:

Suppose that a DEPENDENT entity is identified by the dependent's first name and birhtdate, *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

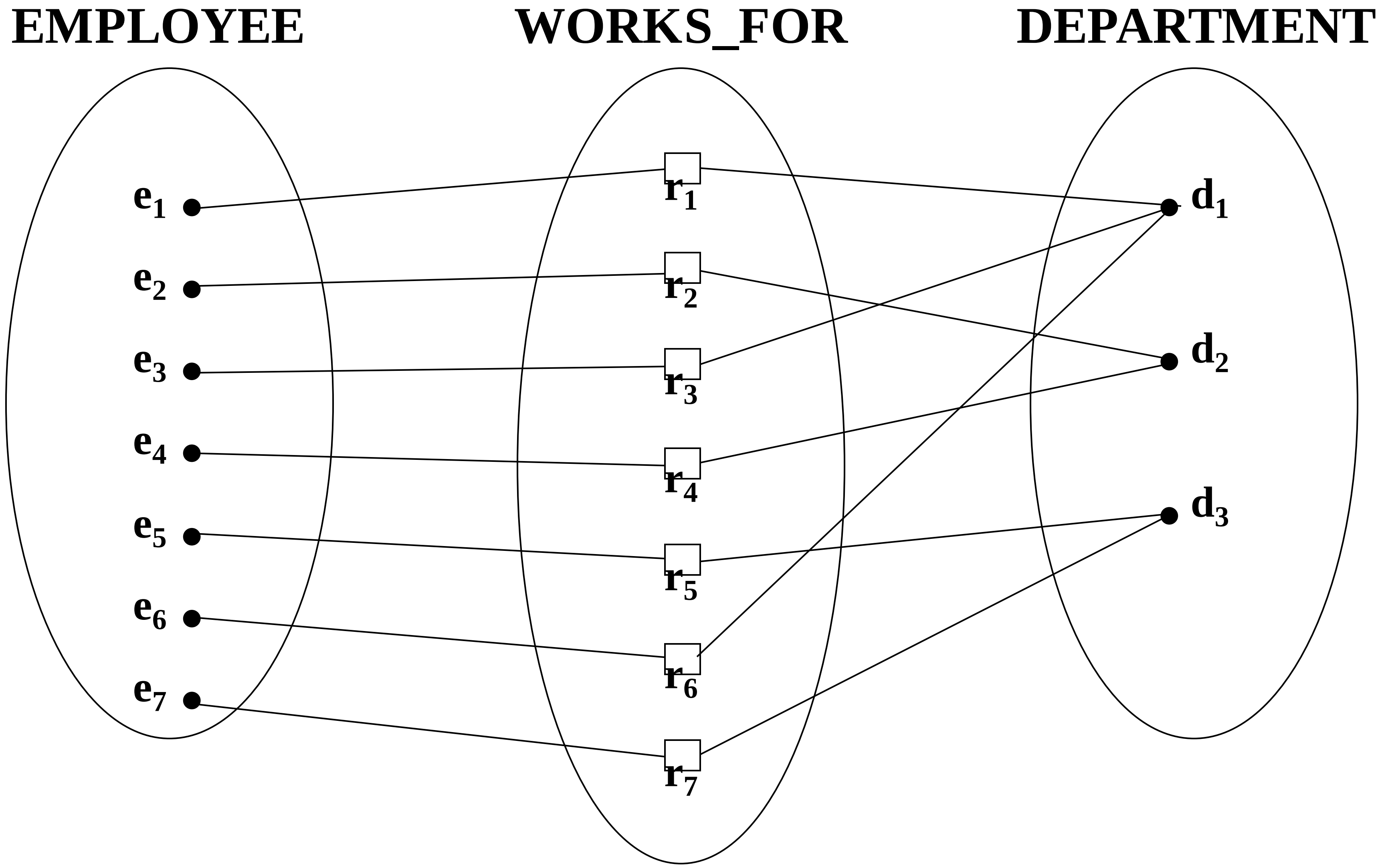
Weak Entity Type is: DEPENDENT
Identifying Relationship is: DEPENDENTS_OF



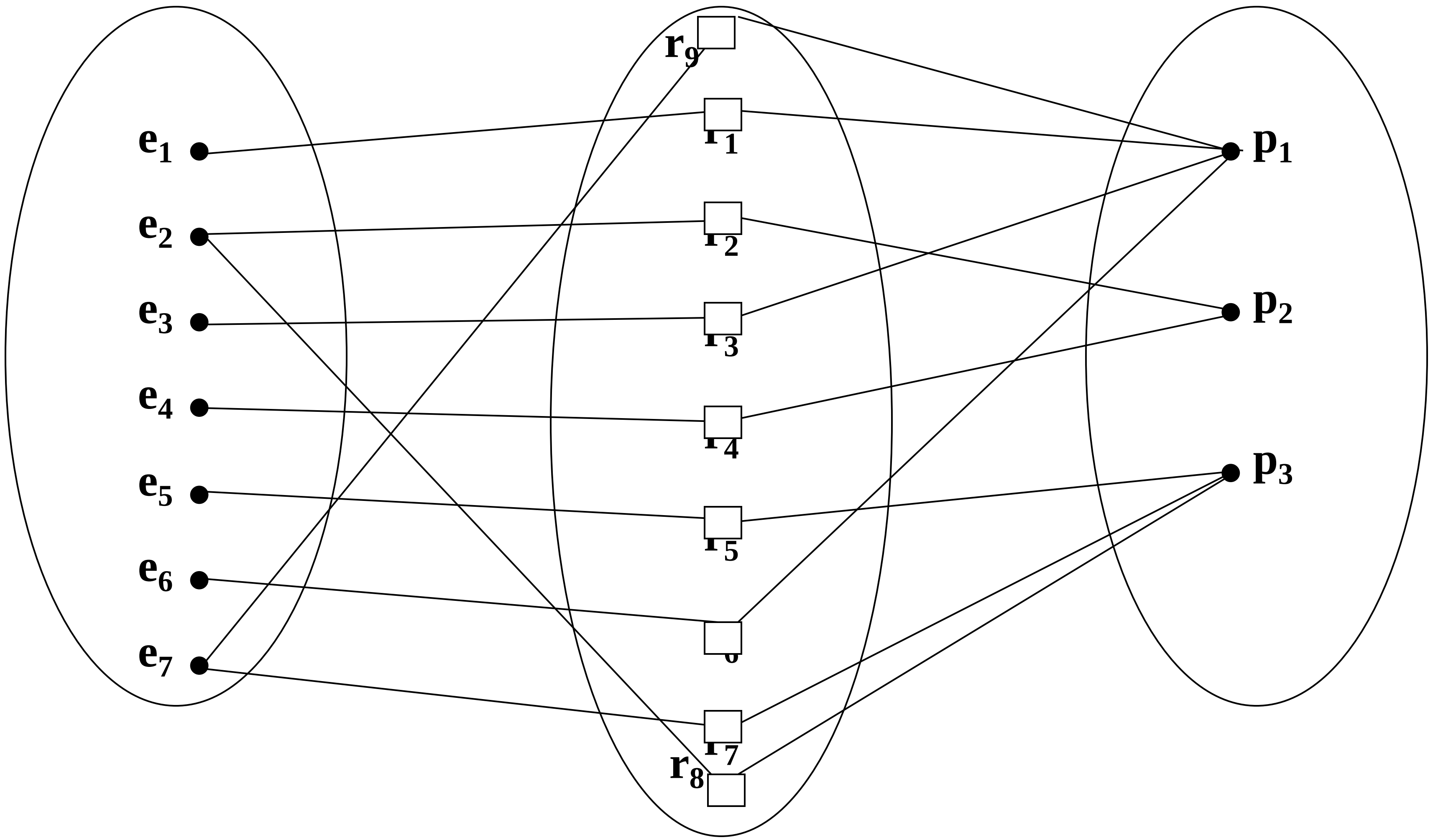
Constraints on Relationships

- Constraints on Relationship Types
  - (Also known as ratio constraints)
  - Maximum Cardinality
    - One-to-one (1:1)
    - One-to-many (1:N) or Many-to-one (N:1)
    - Many-to-many
  - Minimum Cardinality (also called **participation constraint** or existence dependency constraints)
    - zero (optional participation, not existence-dependent)
    - one or more (mandatory, existence-dependent)

Many-to-one (N:1) RELATIONSHIP



Many-to-many (M:N) RELATIONSHIP



Structural Constraints – one way to express semantics of relationships

Structural constraints on relationships:

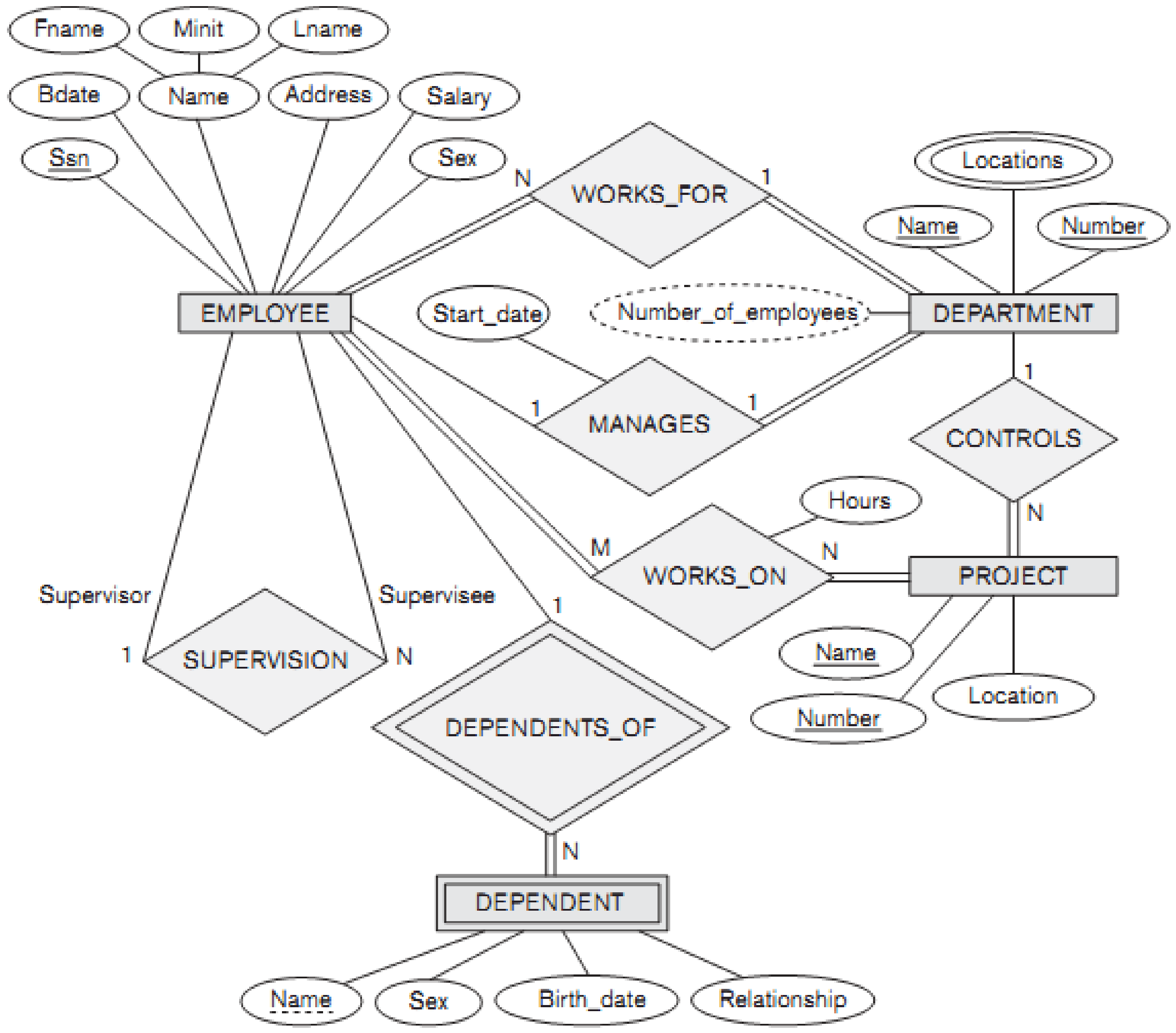
- **Cardinality ratio** (of a binary relationship): 1:1, 1:N, N:1, or M:N

  SHOWN BY PLACING APPROPRIATE NUMBER ON THE LINK.

- **Participation constraint** (on each participating entity type): total (called *existence dependency*) or partial.

  SHOWN BY DOUBLE LINING THE LINK

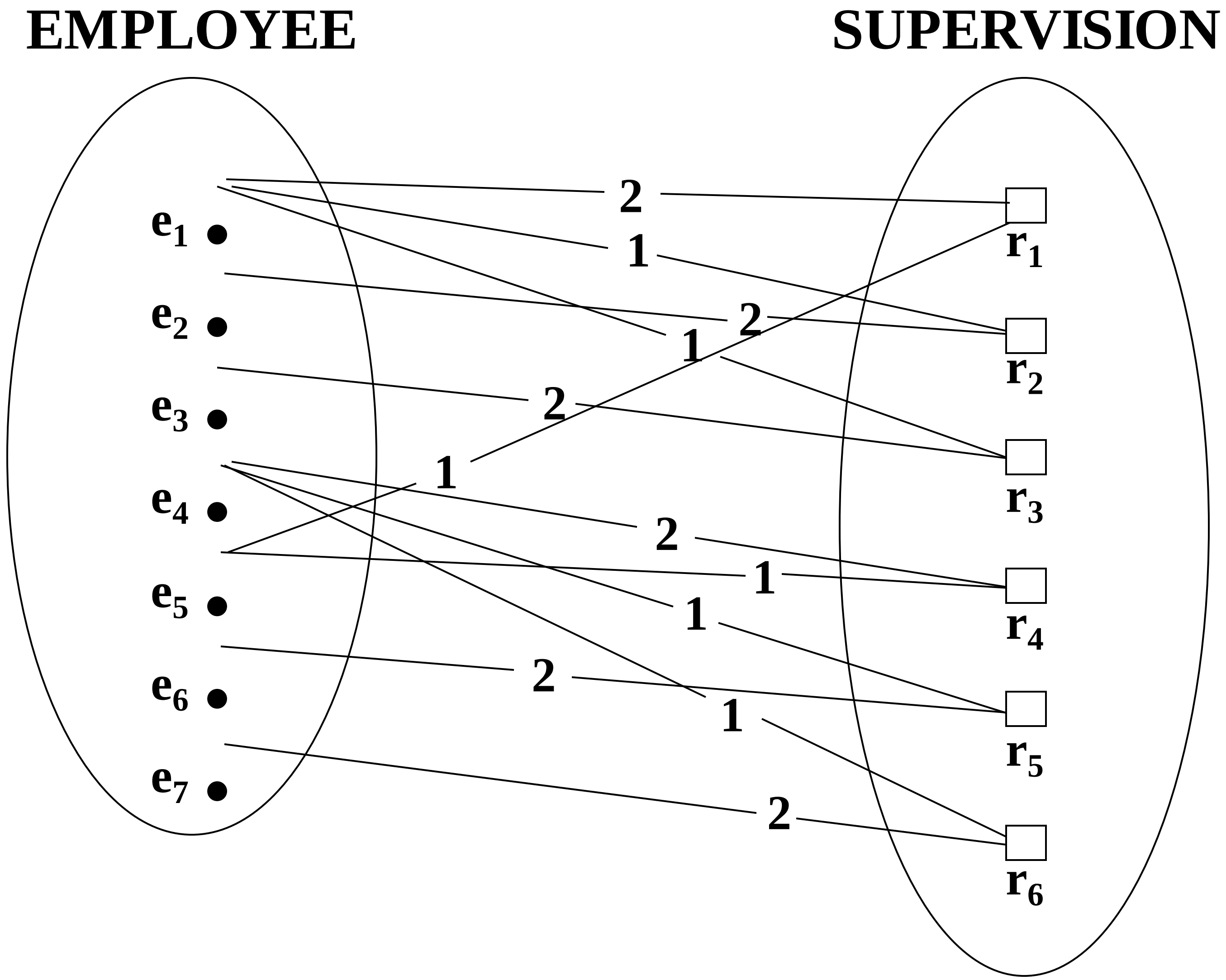
NOTE: These are easy to specify for Binary Relationship Types.


Fname
Minit
Lname
Bdate
Name
Address
Salary
Ssn
Sex
N
WORKS_FOR
1
Locations
Name
Number
EMPLOYEE
Start_date
Number_of_employees
DEPARTMENT
1
MANAGES
1
1
CONTROLS
Hours
N
M
WORKS_ON
N
PROJECT
Supervisor
Supervisee
1
1
SUPERVISION
N
Name
Number
Location
DEPENDENTS_OF
N
DEPENDENT
Name
Sex
Birth_date
Relationship

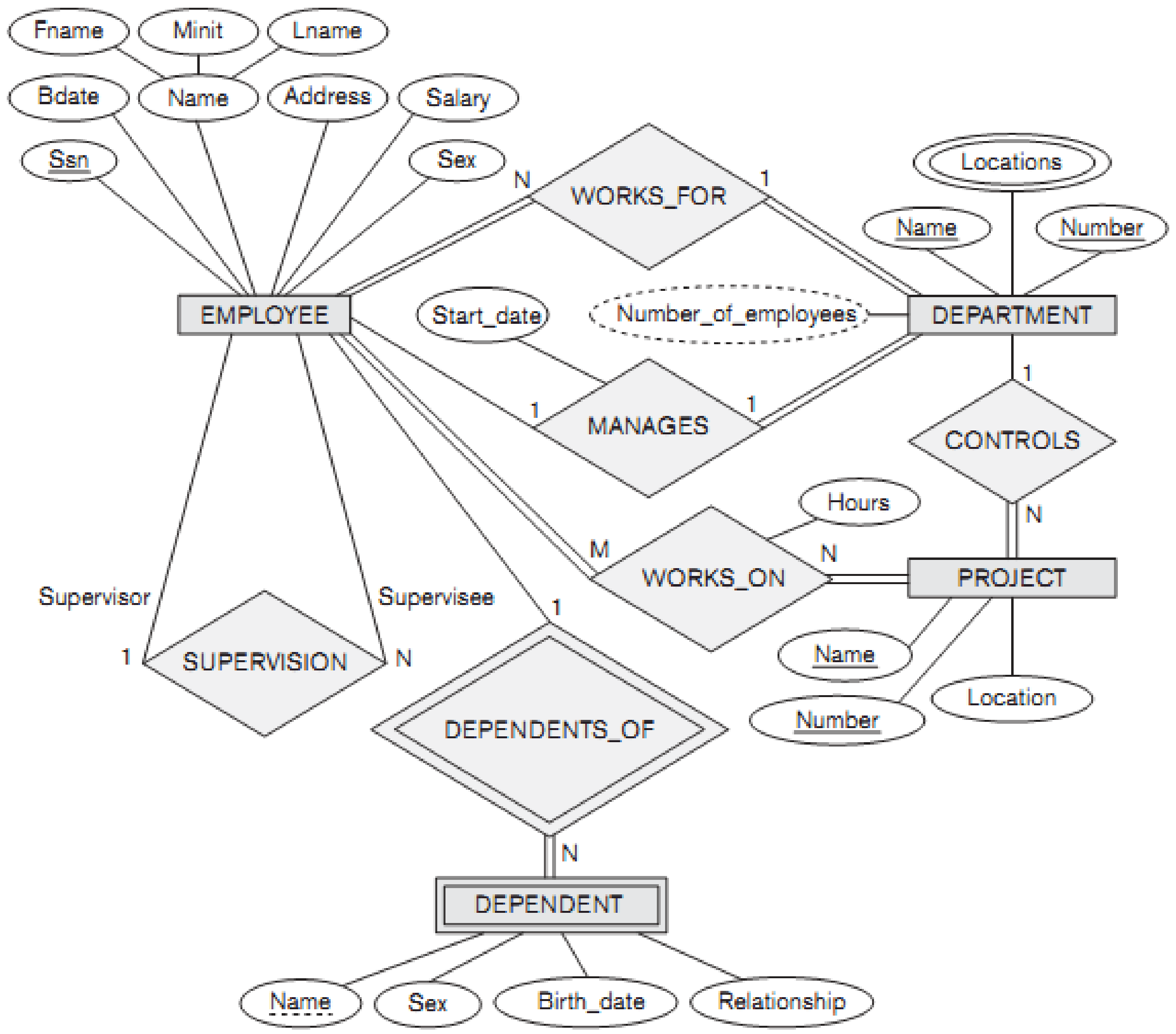
Relationships and Relationship Types (3)

- We can also have 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



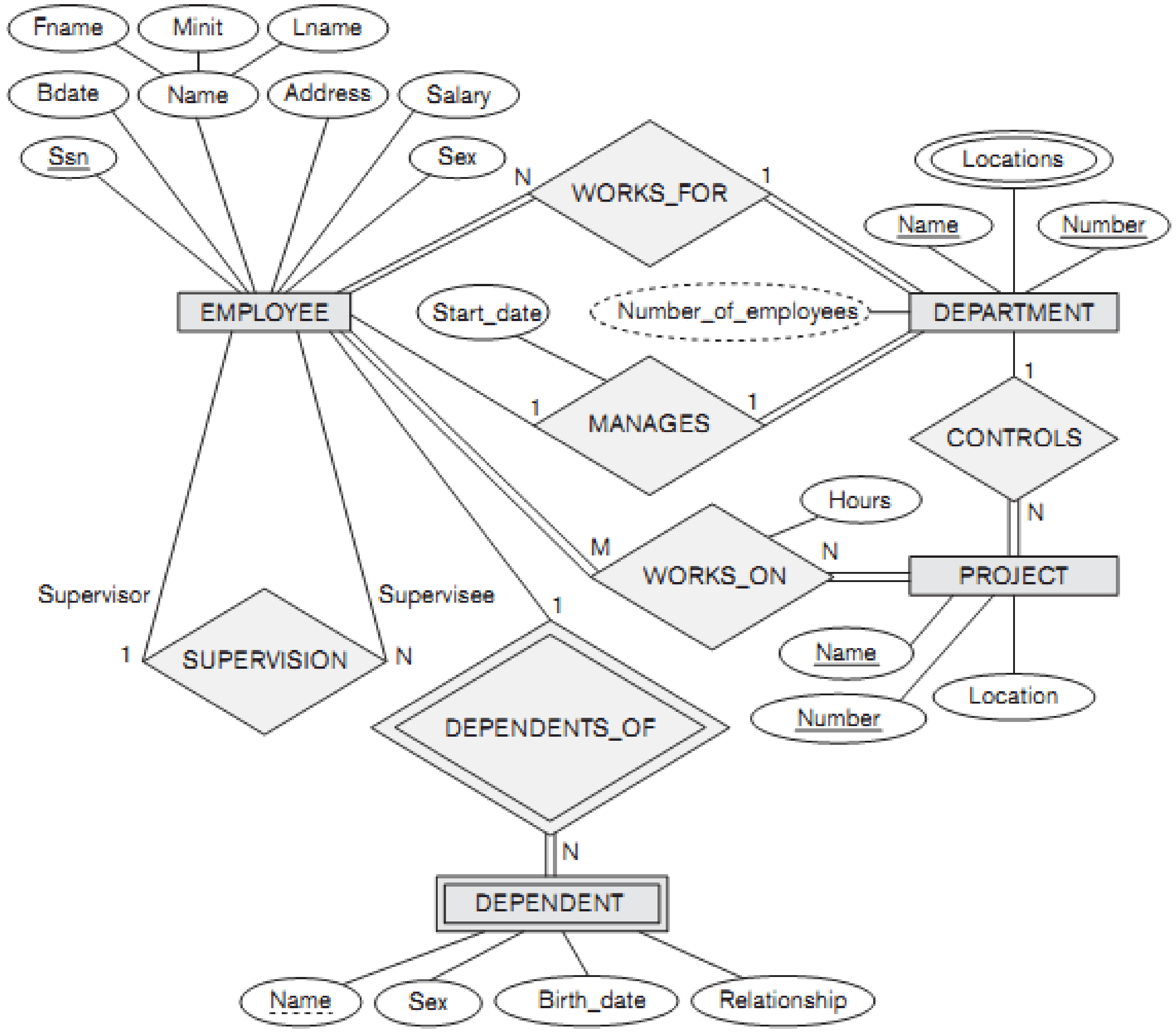
Recursive Relationship Type is: SUPERVISION (participation role names are shown)



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.

Attribute of a Relationship Type is: Hours of WORKS ON



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
- 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.

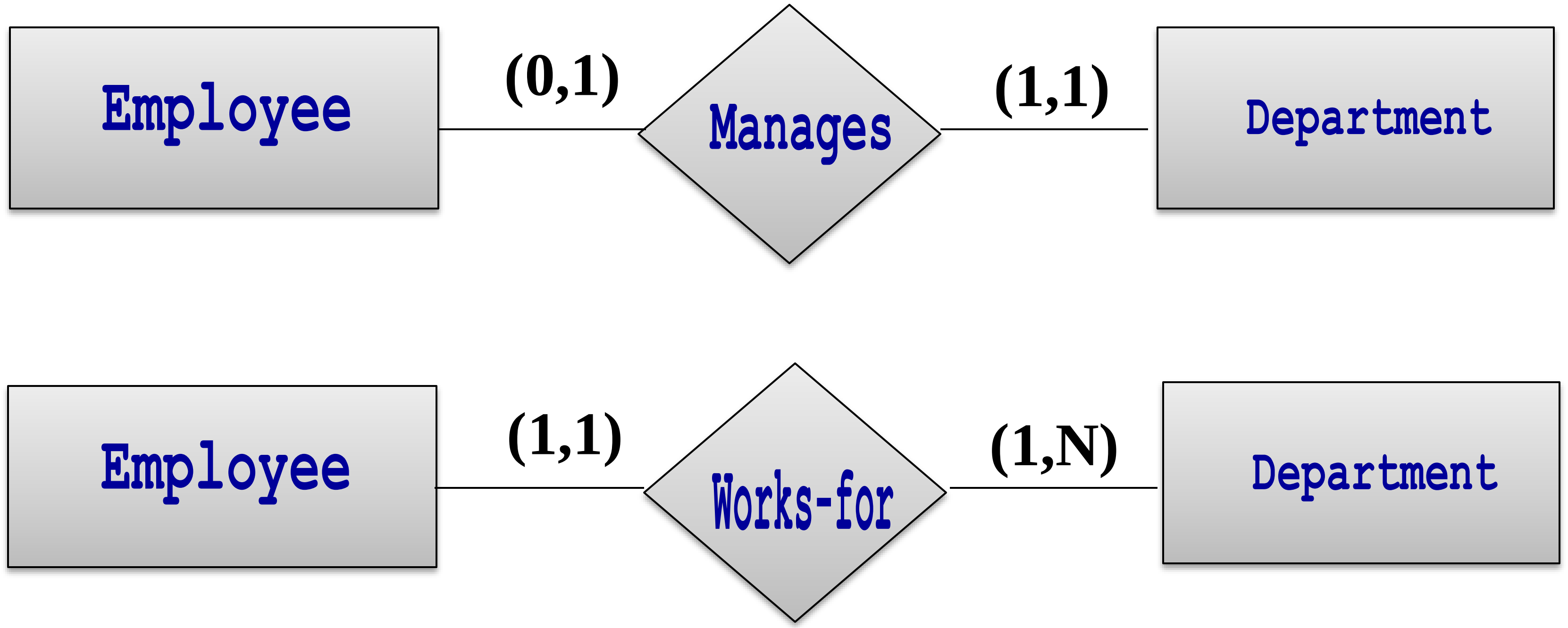
  Specify (0,1) for participation of EMPLOYEE in MANAGES

  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 relationship constraints



COMPANY ER Schema Diagram using (min, max) notation

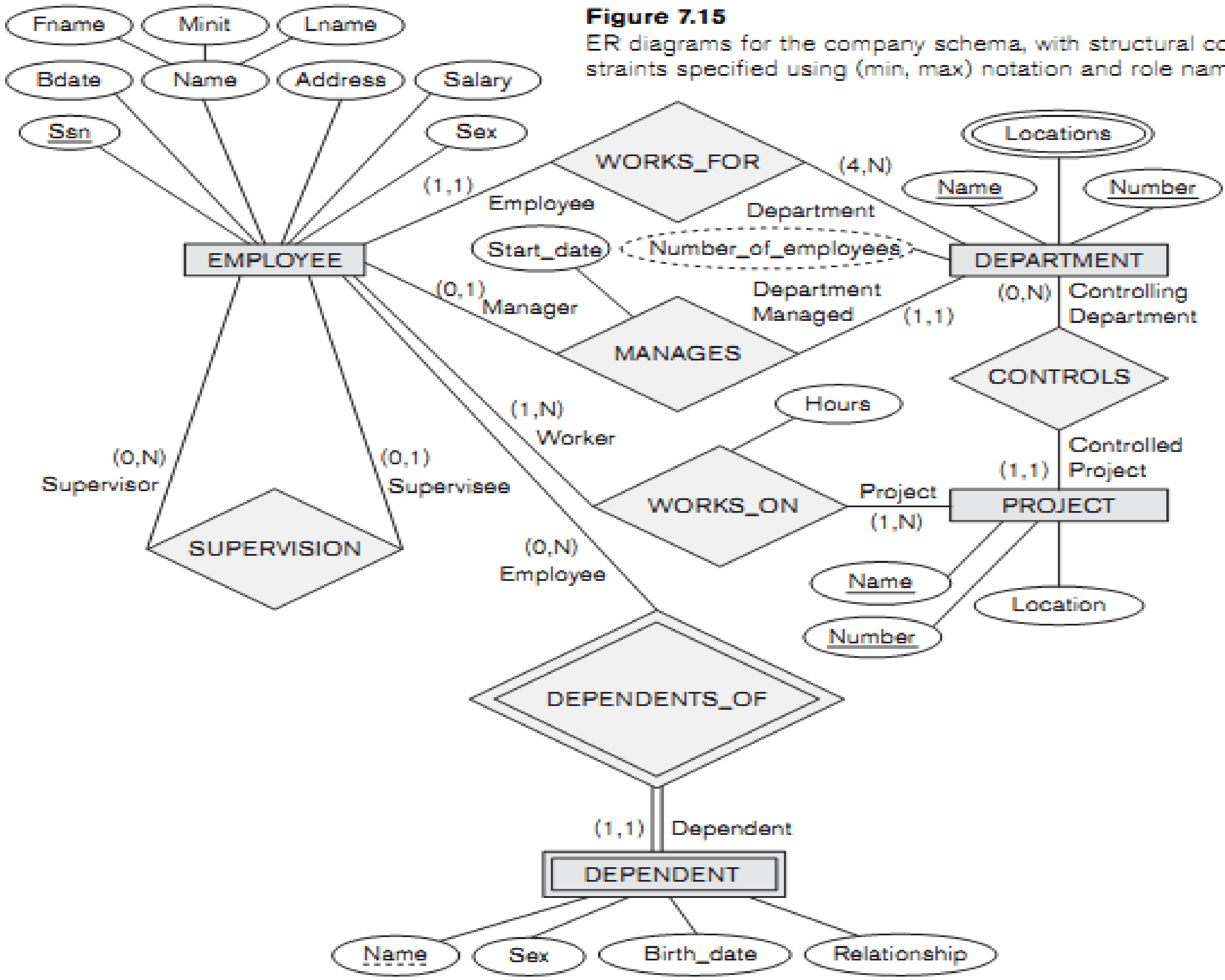


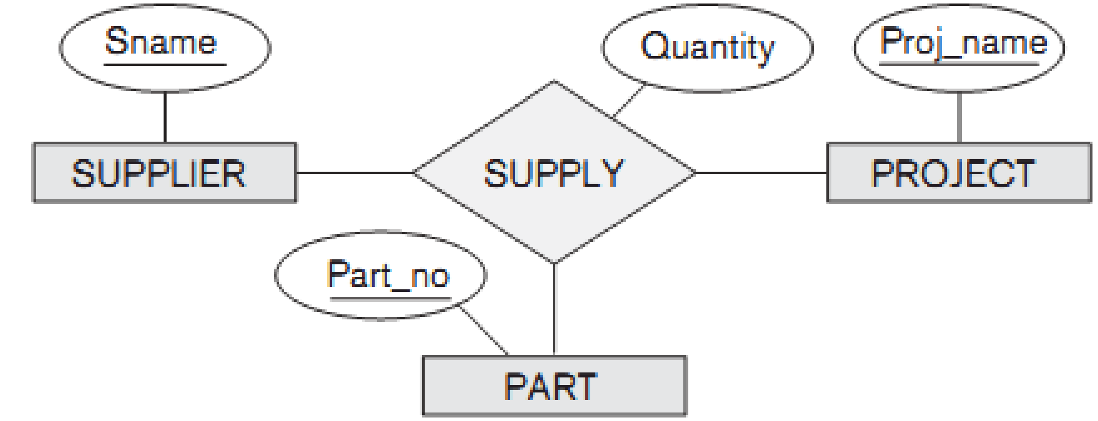
Figure 7.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.

Relationships of Higher Degree

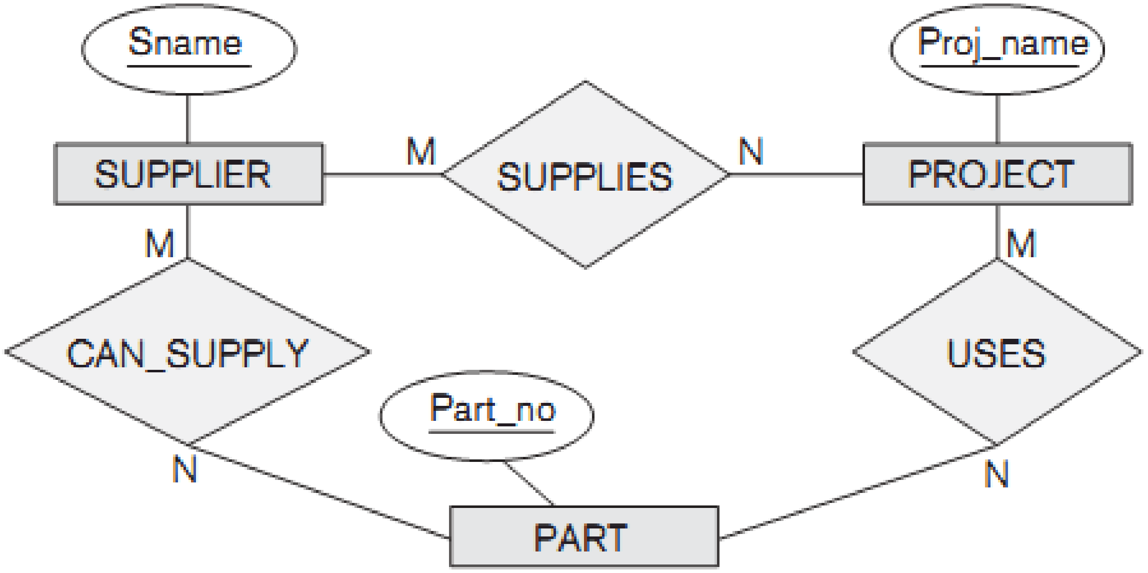
- Relationship types of degree 2 are called **binary**
- Relationship types of degree 3 are called **ternary** and of degree n are called **n-ary**
- In general, an n-ary relationship *is not* equivalent to n binary relationships

Relationships of Higher Degree

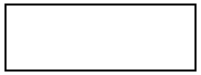
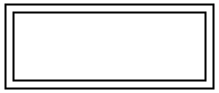
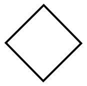
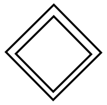
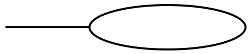
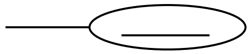
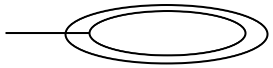
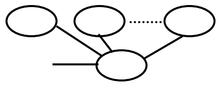
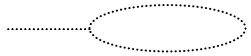
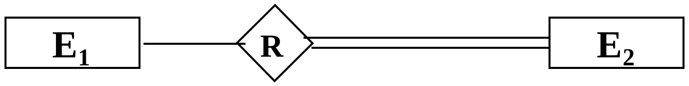
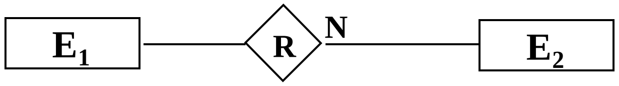
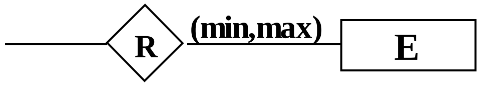
(a)



(b)



SUMMARY OF ER-DIAGRAM NOTATION FOR ER SCHEMAS

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

Enhanced-ER (EER) Model Concepts

- Includes all modeling concepts of basic ER
- Additional concepts: subclasses/superclasses, specialization/generalization, categories, attribute inheritance
- The resulting model is called the enhanced-ER or Extended ER (E2R or EER) model
- It is used to model applications more completely and accurately if needed
- It includes some object-oriented concepts, such as inheritance

Subclasses and Superclasses (1)

- An entity type may have additional meaningful subgroupings of its entities
- Example: EMPLOYEE may be further grouped into SECRETARY, ENGINEER, MANAGER, TECHNICIAN, SALARIED_EMPLOYEE, HOURLY_EMPLOYEE,…
  - Each of these groupings 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.
- Example: EMPLOYEE/SECRETARY, EMPLOYEE/TECHNICIAN

Subclasses and Superclasses (2)

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

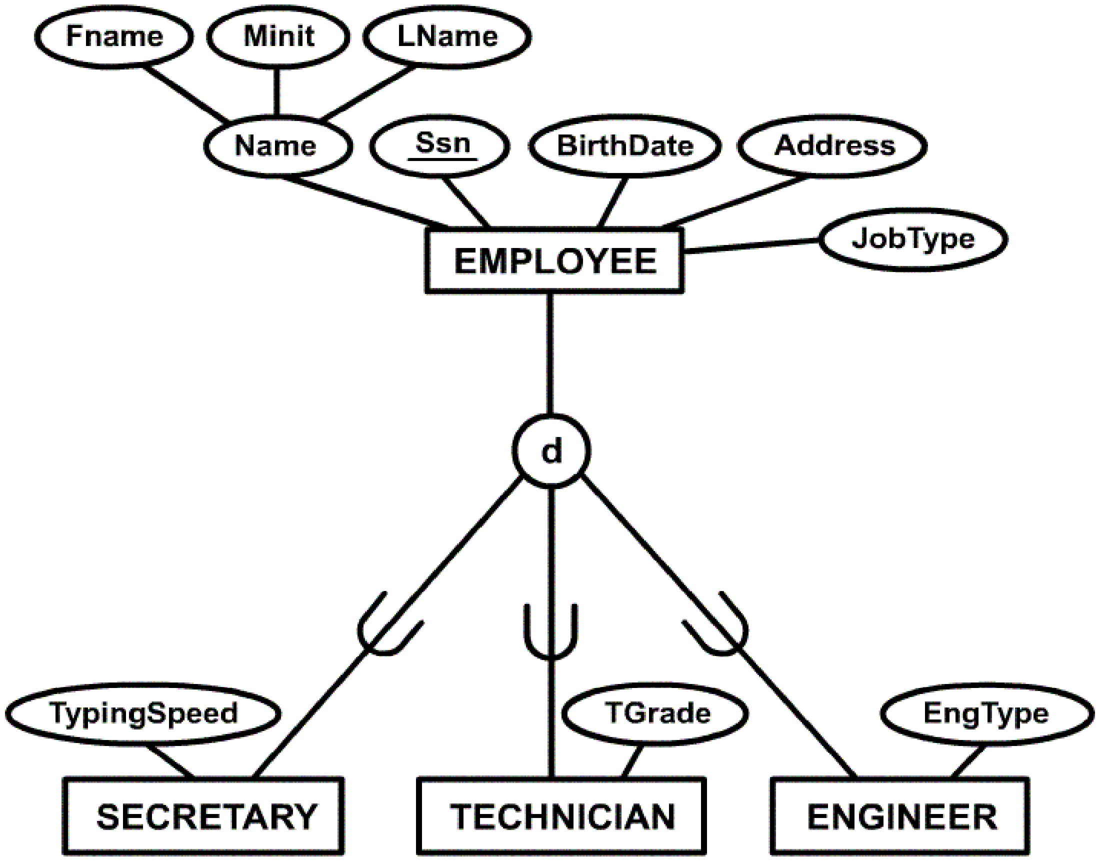
Attribute Inheritance in Superclass / Subclass Relationships

- An entity that is member of a subclass *inherits* all attributes of the entity as a member of the superclass
- It also inherits all relationships

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*.
  - May have several specializations of the same superclass
- Example: Another specialization of EMPLOYEE based in *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 attributes. For example, TypingSpeed of SECRETARY
  - The subclass can participate in specific relationship types

Example of a Specialization



Example of a Specialization

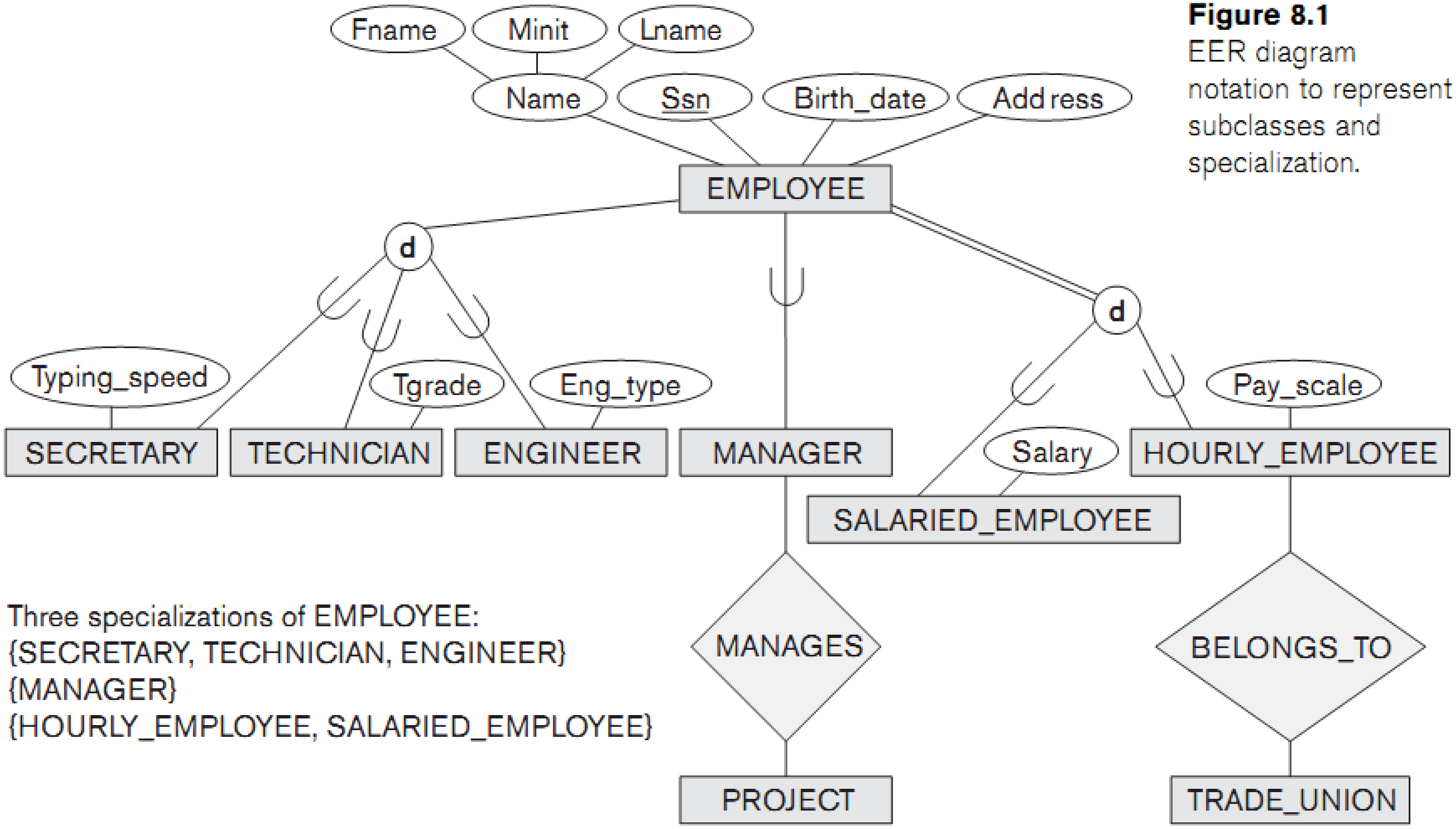
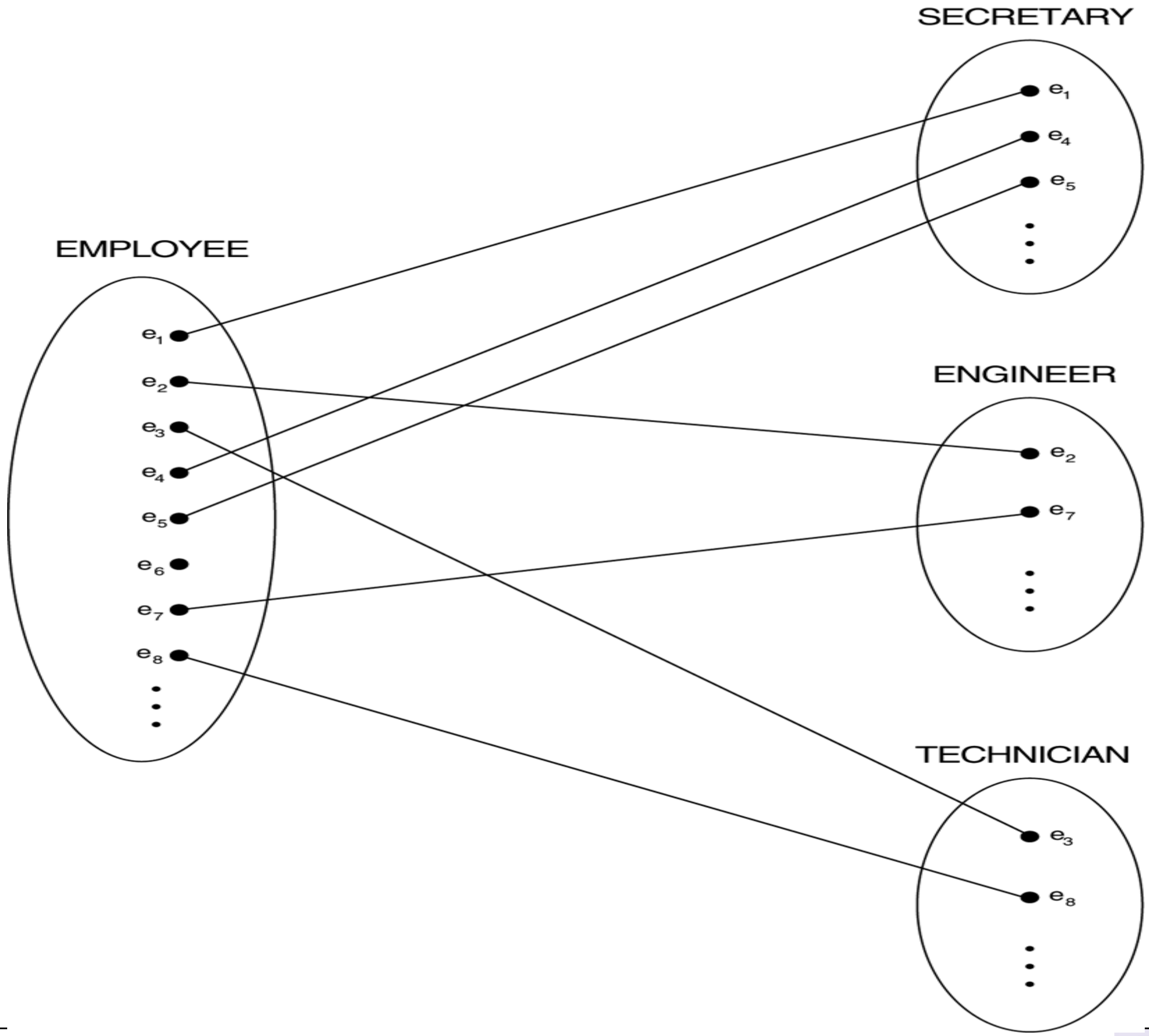


Figure 8.1
EER diagram notation to represent subclasses and specialization.

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

Instances of a specialization



Generalization

- 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

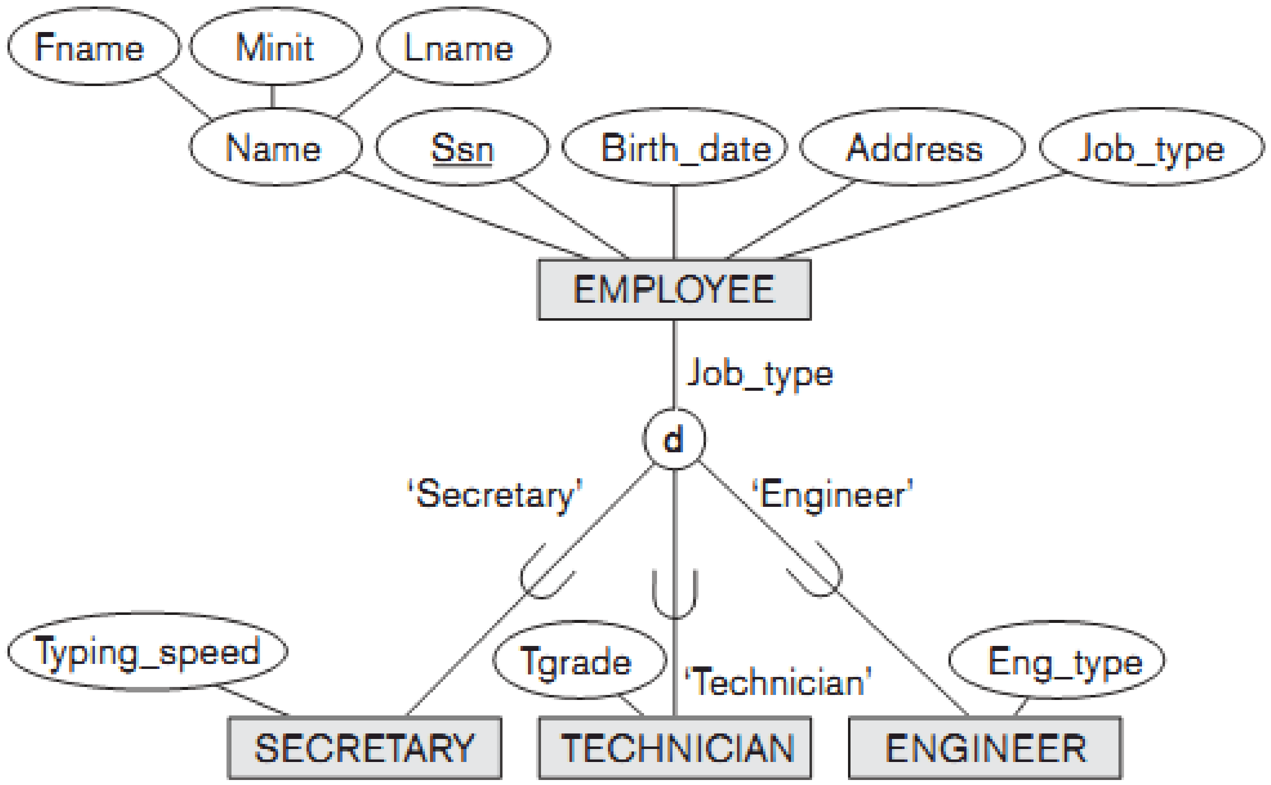
Generalization and Specialization

- Diagrammatic notation 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 will not use this notation because the decision as to which process is followed in a particular situation is often subjective.
- Data Modeling with Specialization and Generalization
  - A superclass or subclass represents a set of entities
  - Shown in rectangles in EER diagrams (as are entity types)
  - Sometimes, all entity sets are simply called classes, whether they are entity types, superclasses, or subclasses

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
- 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

Example of disjoint partial Specialization



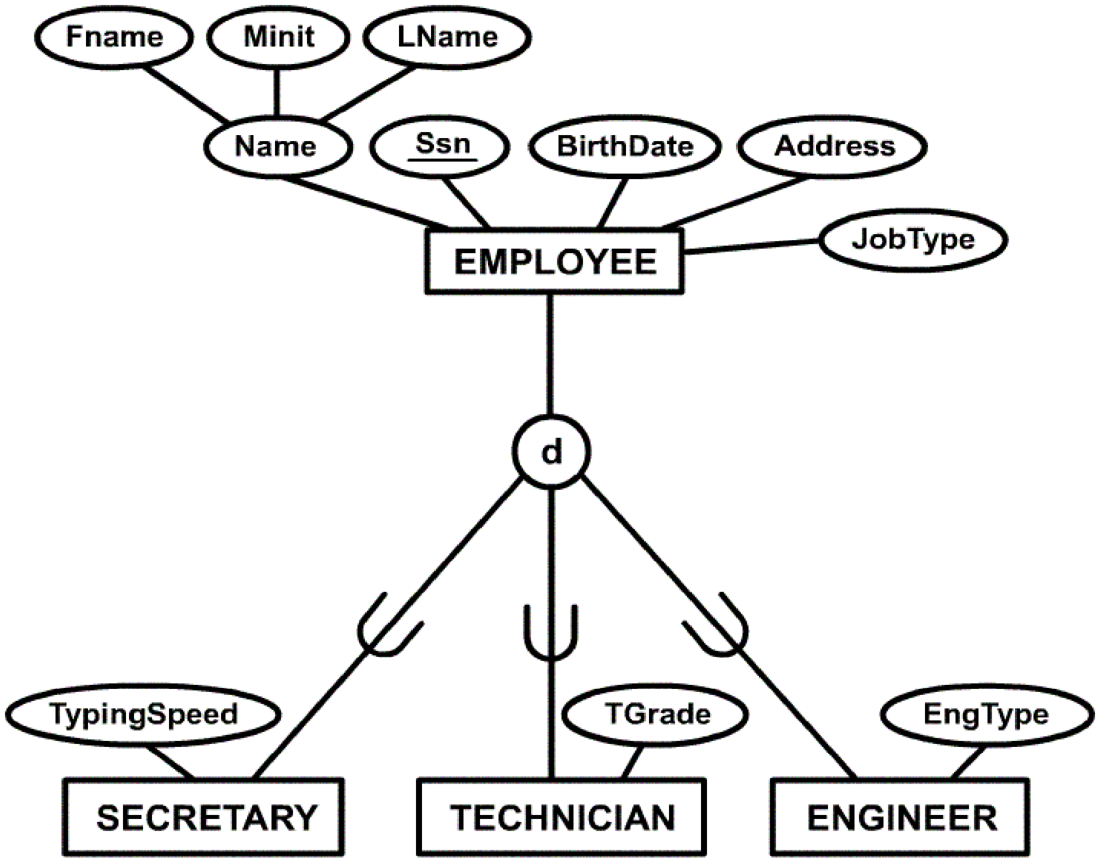
Constraints on Specialization and Generalization (2)

- Two other constraints apply to a specialization/generalization:
- **Disjointness Constraint**:
  - Specifies that the subclasses of the specialization must be disjointed (an entity can be a member of at most one of the subclasses of the specialization)
  - Specified by d in EER diagram
  - If not disjointed, overlap; that is the same entity may be a member of more than one subclass of the specialization
  - Specified by o in EER diagram
- **Completeness 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

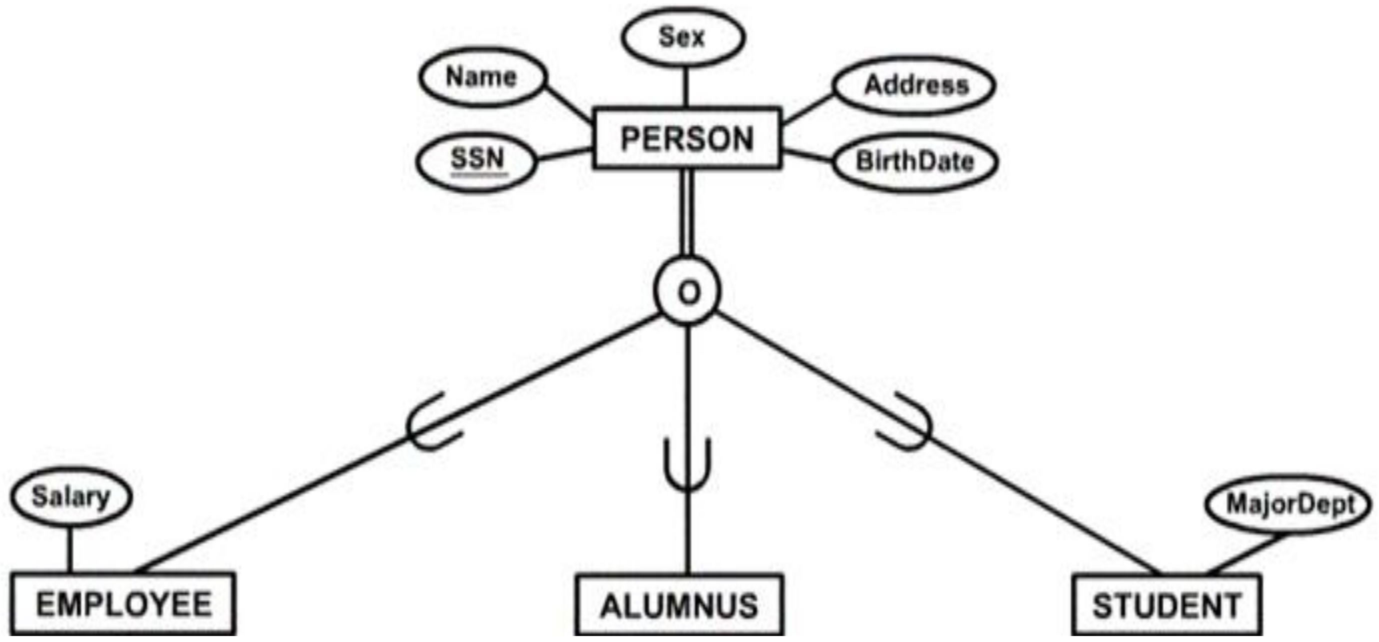
Constraints on Specialization and Generalization (3)

- 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



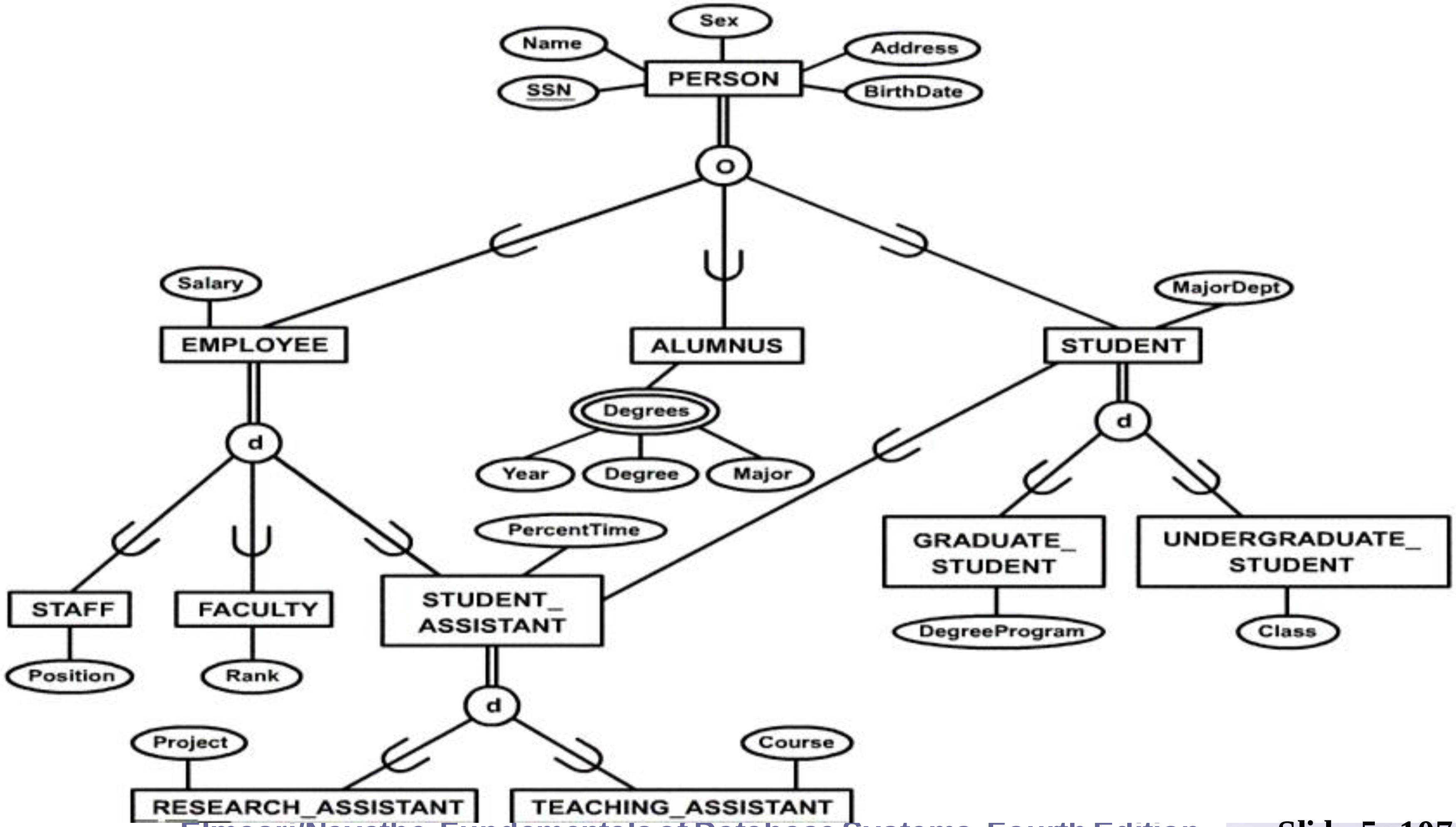
Example of overlap total Specialization



Specialization / Generalization Hierarchies, Lattices and Shared Subclasses

- A subclass may itself have further subclasses specified on it that forms a hierarchy or a lattice
- Hierarchy has a constraint that **every subclass has only one superclass** (called *single inheritance*)
- In a lattice, **a subclass can be subclass of more than one superclass** (called *multiple inheritance*)
- 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**
- Can have specialization hierarchies or lattices, or generalization hierarchies or lattices
- In specialization, start with an entity type and then define subclasses of the entity type by successive specialization (top down conceptual refinement process)
- In generalization, start with many entity types and generalize those that have common properties (bottom up conceptual synthesis process)
- In practice, the combination of two processes is employed

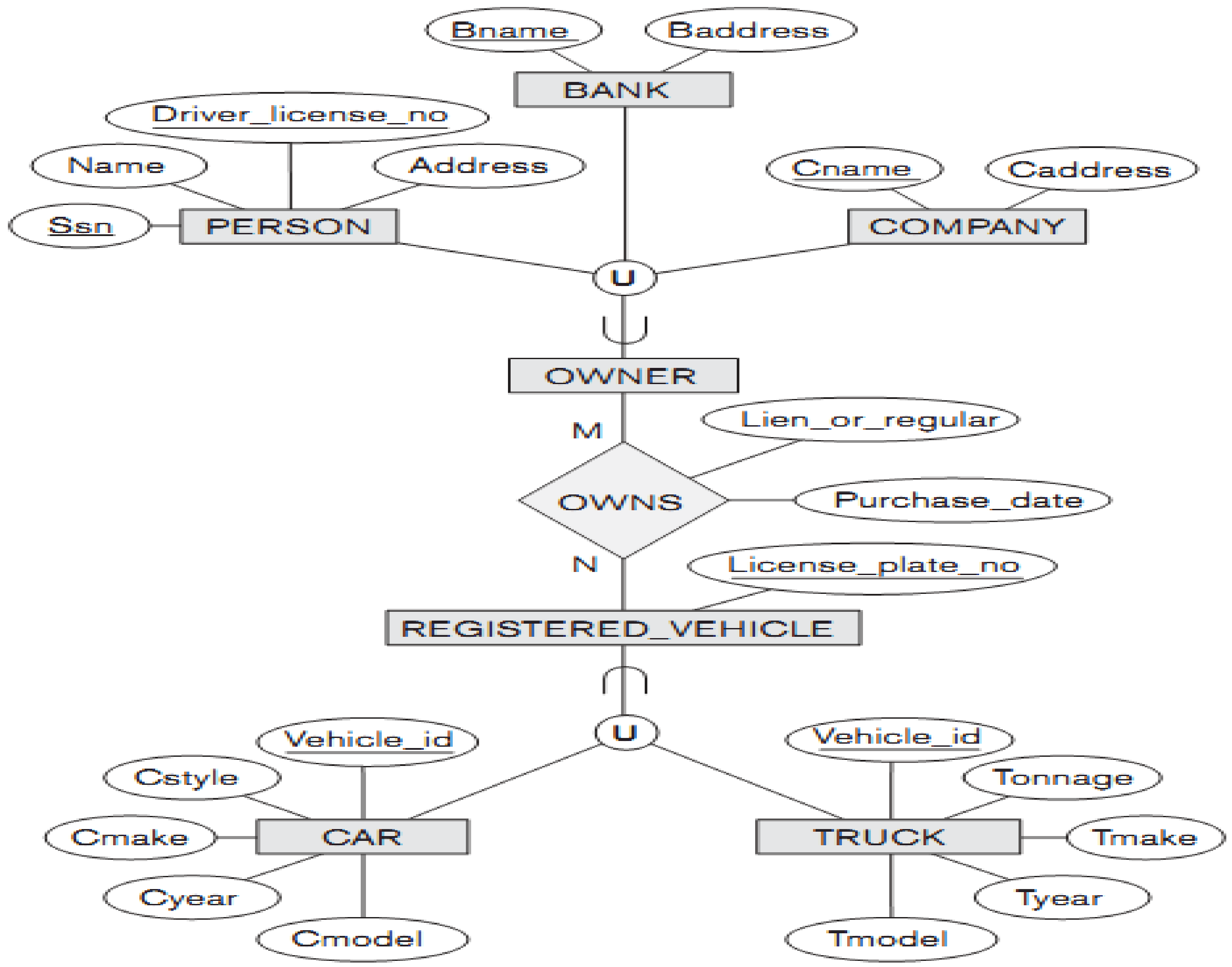
Specialization / Generalization Lattice Example (UNIVERSITY)



Categories (UNION TYPES)

- All of the superclass/subclass relationships we have seen thus far have a single superclass
- **A shared subclass is subclass in more than one distinct superclass/subclass relationships, where each relationships has a single superclass (multiple inheritance)**
- In some cases, need to model a single superclass/subclass relationship with more than one superclass
- Superclasses represent different entity types
- Such a subclass is called a **category or UNION TYPE**
- Example: Database for vehicle registration, vehicle owner can be a person, a bank (holding a lien on a vehicle) or a company.
  - Category (subclass) OWNER is a subset of the union of the three superclasses COMPANY, BANK, and PERSON
  - A category member must exist in at least one of its superclasses
- Note: The difference from shared subclass, which is **subset of the intersection of its superclasses** (shared subclass member must exist in all of its superclasses).

Example of categories (UNION TYPES)



Formal Definitions of EER Model (1)

- Class C: A set of entities; could be entity type, subclass, superclass, category.
- Subclass S: A class whose entities must always be subset of the entities in another class, called the superclass C of the superclass/subclass (or IS-A) relationship S/C: $S \subseteq C$
- Specialization Z: Z = {S1, S2,…, Sn} a set of subclasses with same superclass G; hence, G/Si 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:

    $S1 \cup S2 \cup \ldots \cup Sn = G$;

    Otherwise, Z is partial.
  - Z is disjoint if we always have:

    $Si \cap S2$ empty-set for $i \neq j$;

    Otherwise, Z is overlapping.

Formal Definitions of EER Model (2)

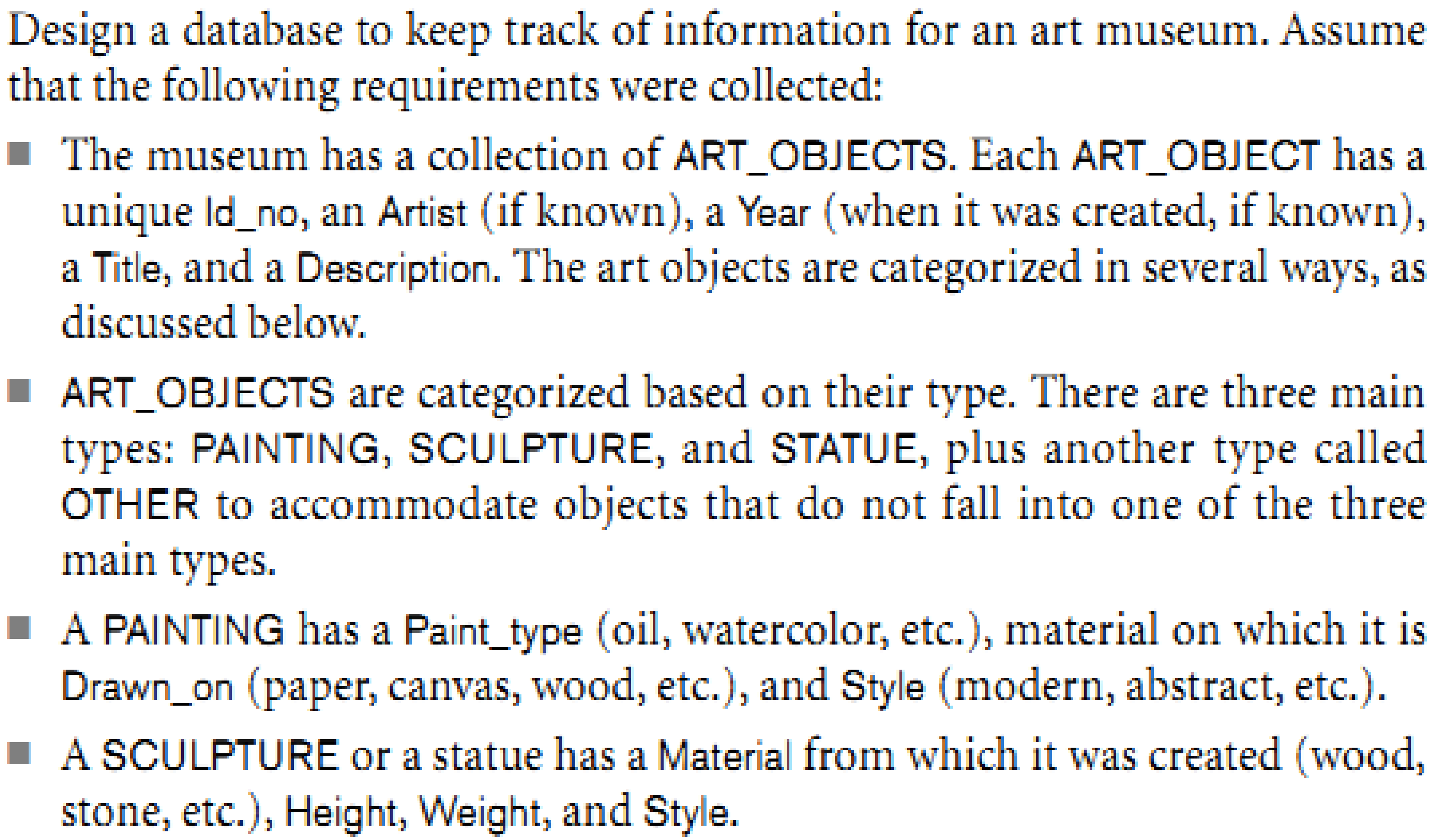
- Subclass S of C is predicate defined if predicate p on attributes of C is used to specify membership in S; that is, S = C[p], where C[p] is the set of entities in C that satisfy p
- A subclass not defined by a predicate is called user-defined
- Attribute-defined specialization: if a predicate A = ci (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 (3)

- Category or UNION type T
  - A class that is a subset of the union of n defining superclasses
    D1, D2,…Dn, n>1:

    T ⊆ (D1 ∪ D2 ∪ … ∪ Dn)
    A predicate pi on the attributes of T.
  - If a predicate pi on the attributes of Di can specify entities of Di that are members of T.
  - If a predicate is specified on every Di: T = (D1[p1] ∪ D2[p2] ∪…∪ Dn[pn]
  - Note: The definition of relationship type should have 'entity type' replaced with 'class'.

Case Study



Design a database to keep track of information for an art museum. Assume that the following requirements were collected:

- The museum has a collection of ART_OBJECTS. Each ART_OBJECT has a unique Id_no, an Artist (if known), a Year (when it was created, if known), a Title, and a Description. The art objects are categorized in several ways, as discussed below.
- ART_OBJECTS are categorized based on their type. There are three main types: PAINTING, SCULPTURE, and STATUE, plus another type called OTHER to accommodate objects that do not fall into one of the three main types.
- A PAINTING has a Paint_type (oil, watercolor, etc.), material on which it is Drawn_on (paper, canvas, wood, etc.), and Style (modern, abstract, etc.).
- A SCULPTURE or a statue has a Material from which it was created (wood, stone, etc.), Height, Weight, and Style.

- An art object in the OTHER category has a Type (print, photo, etc.) and Style.
- ART_OBJECTs are categorized as either PERMANENT_COLLECTION (objects that are owned by the museum) and BORROWED. Information captured about objects in the PERMANENT_COLLECTION includes Date_acquired, Status (on display, on loan, or stored), and Cost. Information captured about BORROWED objects includes the Collection from which it was borrowed, Date_borrowed, and Date_returned.
- Information describing the country or culture of Origin (Italian, Egyptian, American, Indian, and so forth) and Epoch (Renaissance, Modern, Ancient, and so forth) is captured for each ART_OBJECT.
- The museum keeps track of ARTIST information, if known: Name, DateBorn (if known), Date_died (if not living), Country_of_origin, Epoch, Main_style, and Description. The Name is assumed to be unique.
- Different EXHIBITIONS occur, each having a Name, Start_date, and End_date. EXHIBITIONS are related to all the art objects that were on display during the exhibition.
- Information is kept on other COLLECTIONS with which the museum interacts, including Name (unique), Type (museum, personal, etc.), Description, Address, Phone, and current Contact_person.

Draw an EER schema diagram for this application. Discuss any assumptions you make, and that justify your EER design choices.

Chapter 6

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

Chapter Outline

- **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 Relation 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.

- **Mapping EER Model Constructs to Relations**

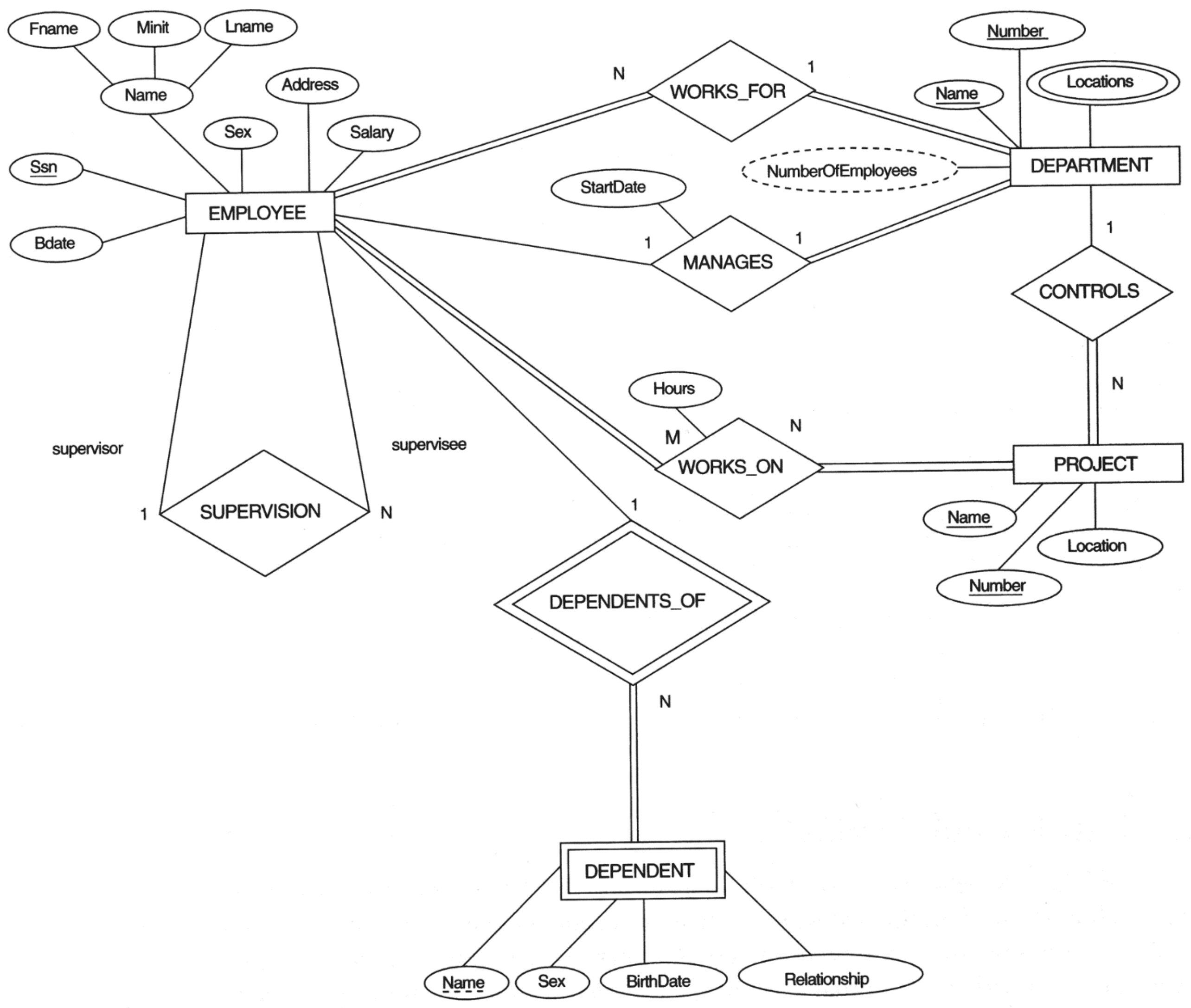
  Step 8: Options for Mapping Specialization or Generalization.
  Step 9: Mapping of Union Types (Categories).

ER-to-Relational Mapping Algorithm

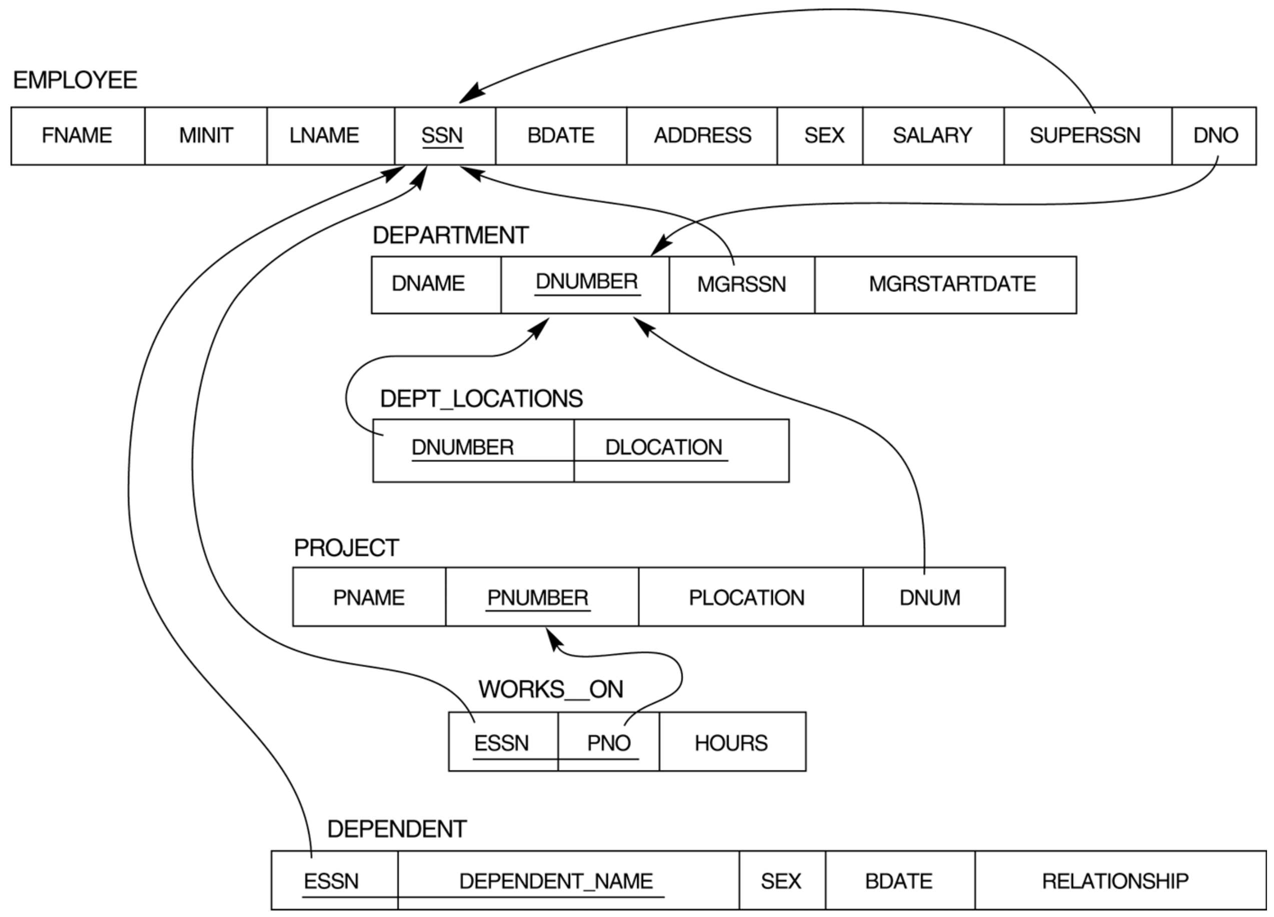
- **Step 1: Mapping of Regular Entity Types.**
  - For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E.
  - Choose one of the key attributes of E as the primary key for R. If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R.

  Example: We create the relations EMPLOYEE, DEPARTMENT, and PROJECT in the relational schema corresponding to the regular entities in the ER diagram. SSN, DNUMBER, and PNUMBER are the primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT as shown.

The ER conceptual schema diagram for the COMPANY database.



Result of mapping the COMPANY ER schema into a relational schema.



ER-to-Relational Mapping Algorithm (cont)

- **Step 2: Mapping of Weak Entity Types**
  - For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes (or simple components of composite attributes) of W as attributes of R.
  - In addition, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).
  - The primary key of R is the *combination of* the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any.

  Example: Create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT. Include the primary key SSN of the EMPLOYEE relation as a foreign key attribute of DEPENDENT (renamed to ESSN).

  The primary key of the DEPENDENT relation is the combination {ESSN, DEPENDENT_NAME} because DEPENDENT_NAME is the partial key of DEPENDENT.

ER-to-Relational Mapping Algorithm (cont)

- **Step 3: Mapping of Binary 1:1 Relation Types**

  For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. There are three possible approaches:

  (1) Foreign Key approach: Choose one of the relations-S, say-and include a foreign key in S the primary key of T. It is better to choose an entity type with *total participation* in R in the role of S.

  Example: 1:1 relation MANAGES is mapped by choosing the participating entity type DEPARTMENT to serve in the role of S, because its participation in the MANAGES relationship type is total.

  (2) Merged relation option: An alternate mapping of a 1:1 relationship type is possible by merging the two entity types and the relationship into a single relation. This may be appropriate when *both participations are total.*

  (3) Cross-reference or relationship relation option: The third alternative is to set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types.

ER-to-Relational Mapping Algorithm (cont)

- **Step 4: Mapping of Binary 1:N Relationship Types.**
  - For each regular binary 1:N relationship type R, identify the relation S that represent the participating entity type at the N-side of the relationship type.
  - Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R.
  - Include any simple attributes of the 1:N relation type as attributes of S.

  Example: 1:N relationship types WORKS_FOR, CONTROLS, and SUPERVISION in the figure. For WORKS_FOR we include the primary key DNUMBER of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it DNO.

ER-to-Relational Mapping Algorithm (cont)

- **Step 5: Mapping of Binary M:N Relationship Types.**
  - For each regular binary M:N relationship type R, *create a new relation* S to represent R.
  - Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; *their combination will form the primary key* of S.
  - Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S.

Example: The M:N relationship type WORKS_ON from the ER diagram is mapped by creating a relation WORKS_ON in the relational database schema. The primary keys of the PROJECT and EMPLOYEE relations are included as foreign keys in WORKS_ON and renamed PNO and ESSN, respectively.

Attribute HOURS in WORKS_ON represents the HOURS attribute of the relation type. The primary key of the WORKS_ON relation is the combination of the foreign key attributes {ESSN, PNO}.

ER-to-Relational Mapping Algorithm (cont)

- **Step 6: Mapping of Multivalued attributes.**
  - For each multivalued attribute A, create a new relation R. This relation R will include an attribute corresponding to A, plus the primary key attribute K-as a foreign key in R-of the relation that represents the entity type of relationship type that has A as an attribute.
  - The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.

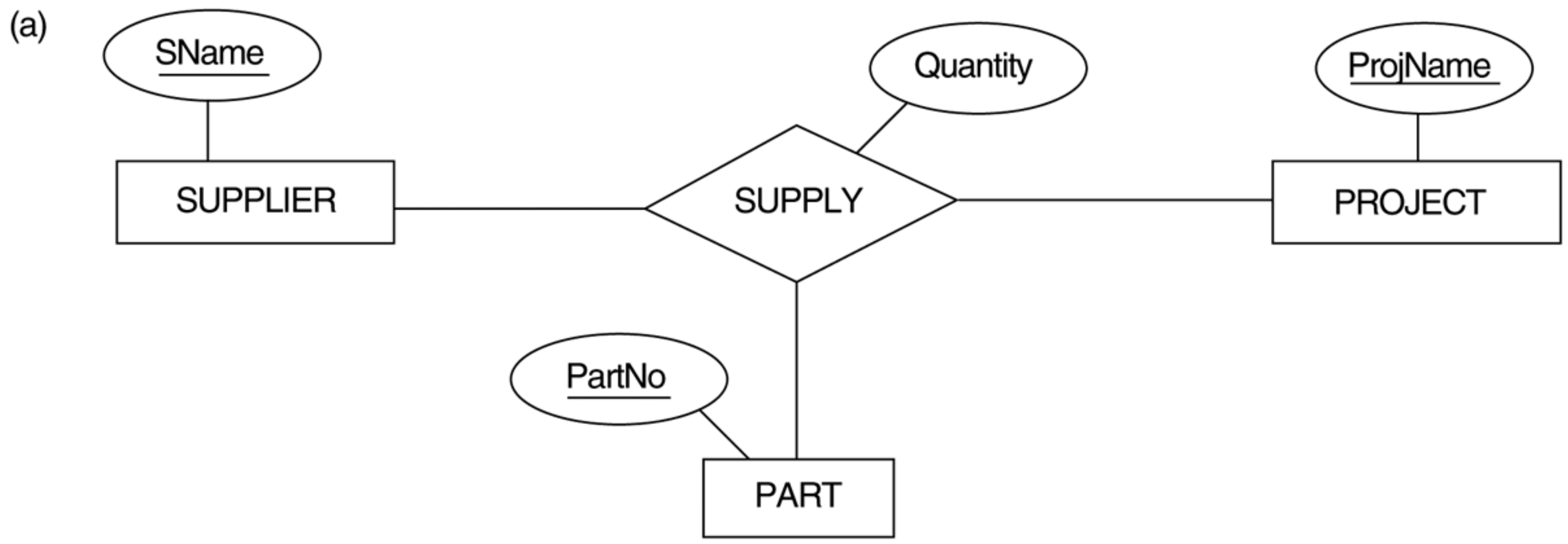
Example: The relation DEPT_LOCATIONS is created. The attribute DLOCATION represents the multivalued attribute LOCATIONS of DEPARTMENT, while DNUMBER-as foreign key-represents the primary key of the DEPARTMENT relation. The primary key of R is the combination of {DNUMBER, DLOCATION}.

ER-to-Relational Mapping Algorithm (cont)

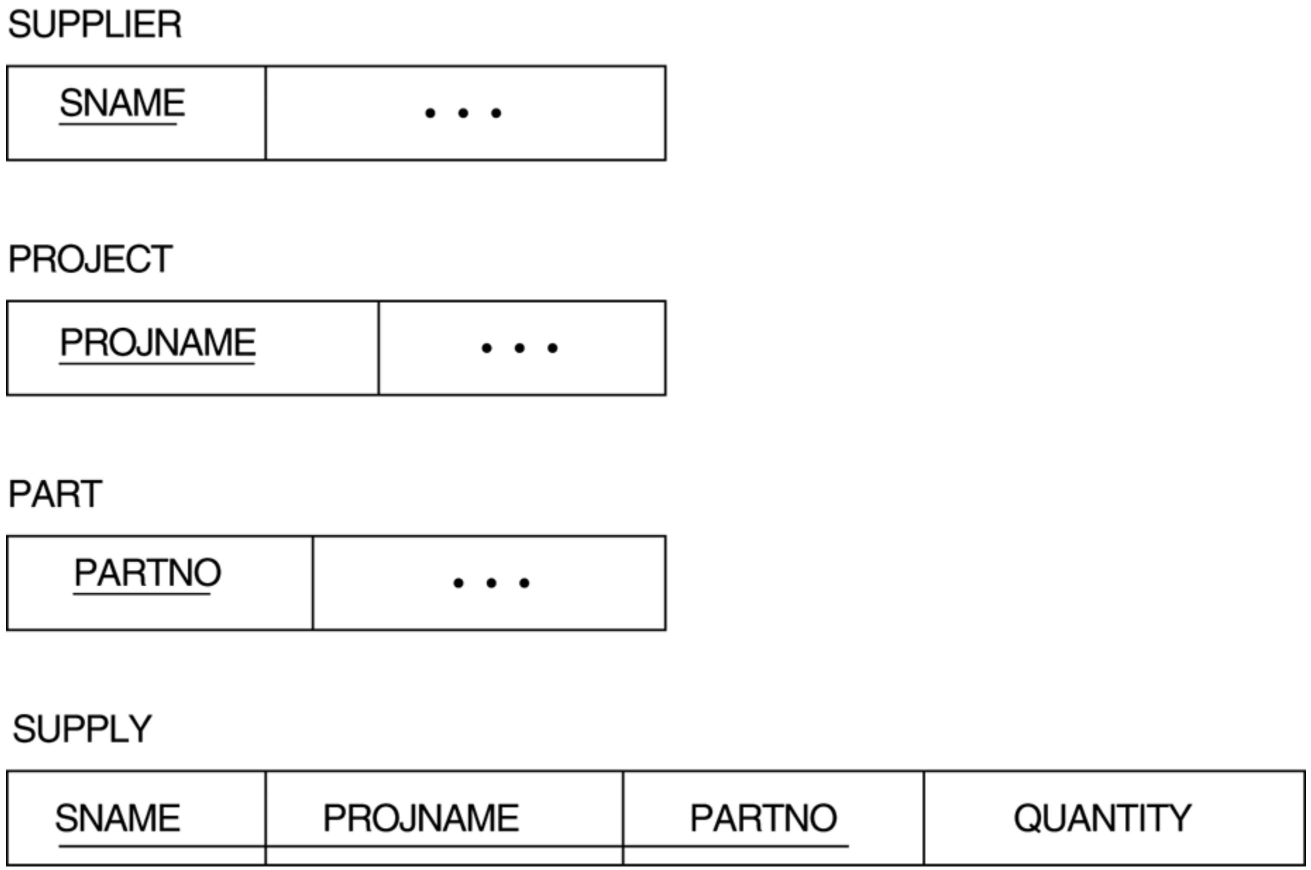
- **Step 7: Mapping of N-ary Relationship Types.**
  - For each n-ary relationship type R, where n>2, create a new relationship S to represent R.
  - Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types.
  - Also include any simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes of S.

Example: The relationship type SUPPY in the ER below. This can be mapped to the relation SUPPLY shown in the relational schema, whose primary key is the combination of the three foreign keys {SNAME, PARTNO, PROJNAME}

Ternary relationship types. (a) The SUPPLY relationship.



Mapping the *n*-ary relationship type SUPPLY



Summary of Mapping constructs and constraints

Table 7.1 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

Mapping EER Model Constructs to Relations

- **Step8: Options for Mapping Specialization or Generalization.**

  Convert each specialization with m subclasses $\{S_1, S_2, \ldots, S_m\}$ and generalized superclass C, where the attributes of C are $\{k, a_1, \ldots a_n\}$ and k is the (primary) key, into relational schemas using one of the four following options:

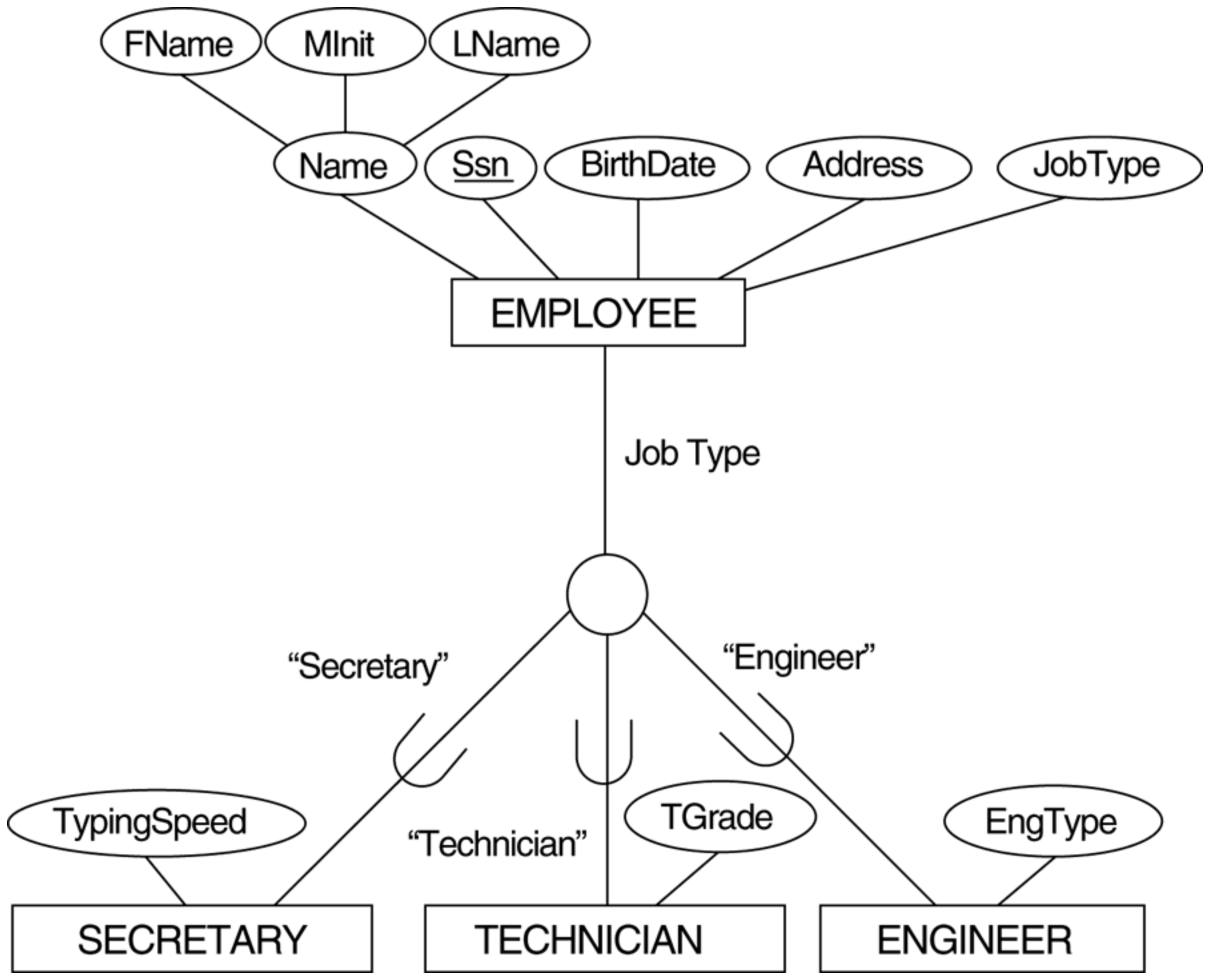
Option 8A: Multiple relations-Superclass and subclasses.

Create a relation L for C with attributes Attrs(L) = $\{k, a_1, \ldots a_n\}$ and PK(L) = k. Create a relation L_i for each subclass S_i, $1 < i < m$, with the attributesAttrs(L_i) = {k} U {attributes of S_i} and PK(L_i)=k. This option works **for any specialization** (total or partial, disjoint of over-lapping).

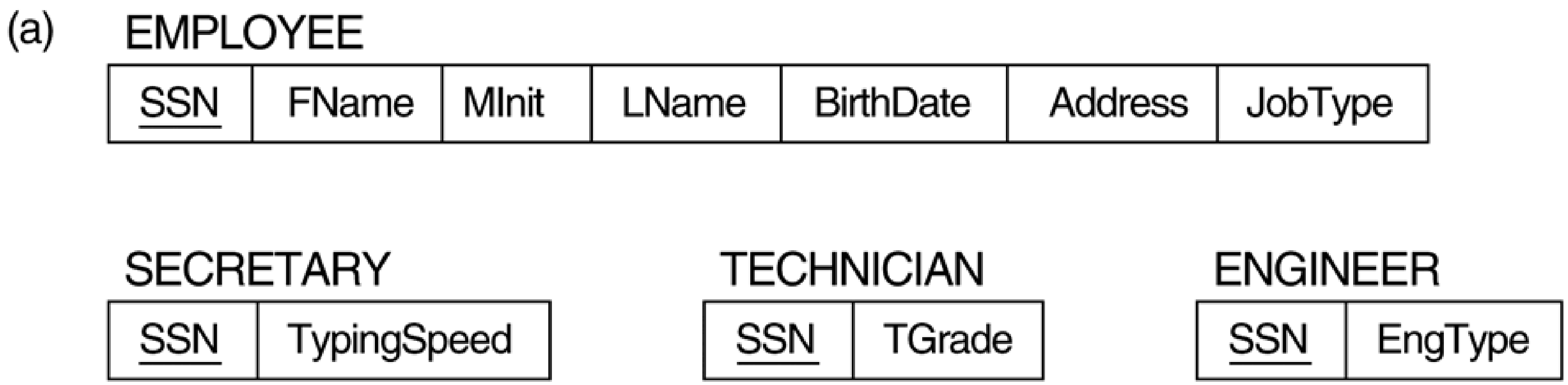
Option 8B: Multiple relations-Subclass relations only

Create a relation L_i for each subclass S_i, $1 < i < m$, with the attributes Attr(L_i) = {attributes of S_i} U $\{k, a_1 \ldots, a_n\}$ and PK(L_i) = k. This option only works for a specialization whose subclasses are **total** (every entity in the superclass must belong to (at least) one of the subclasses).

EER diagram notation for an attribute-defined specialization on JobType.



Options for mapping specialization or generalization.
(a) Mapping the EER schema using option 8A.



(a)

EMPLOYEE

SSN	FName	MInit	LName	BirthDate	Address	JobType

SECRETARY

SSN	TypingSpeed

TECHNICIAN

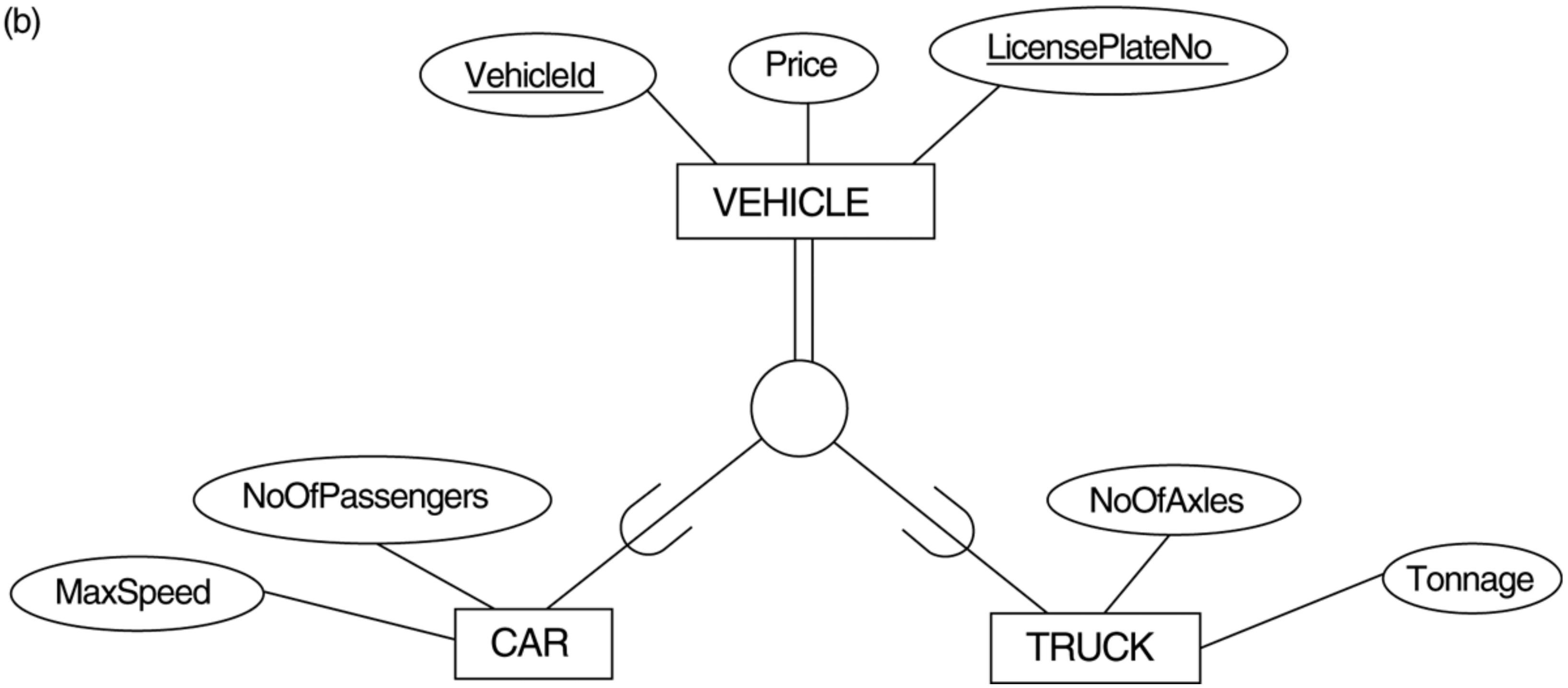
SSN	TGrade

ENGINEER

SSN	EngType

Generalization. (b) Generalizing CAR and TRUCK into the superclass VEHICLE.

(b)



Options for mapping specialization or generalization.
(b) Mapping the EER schema using option 8B.

(b) CAR

VehicleId	LicensePlateNo	Price	MaxSpeed	NoOfPassengers

TRUCK

VehicleId	LicensePlateNo	Price	NoOfAxles	

Mapping EER Model Constructs to Relations (cont)

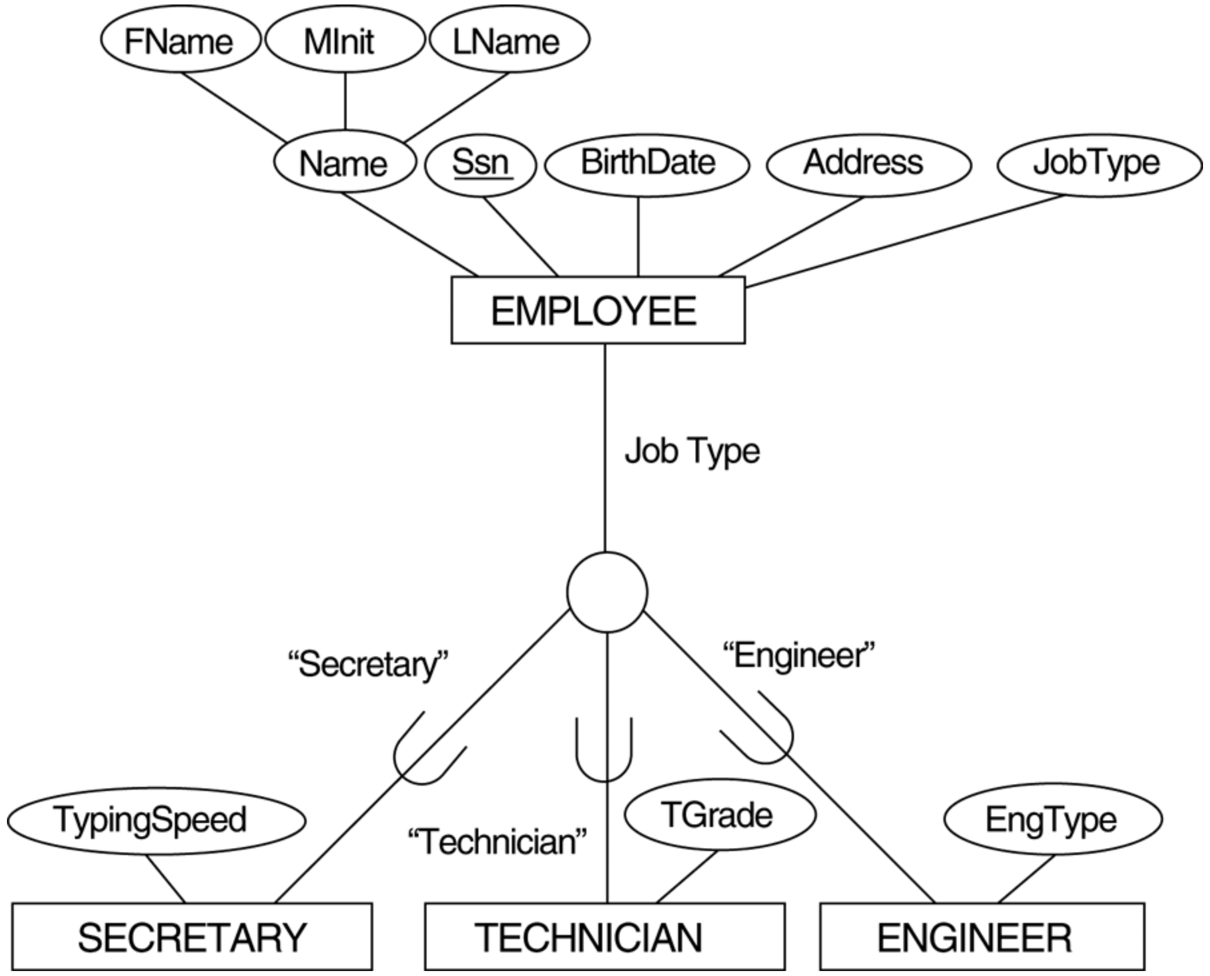
Option 8C: Single relation with one type attribute.

Create a single relation L with attributes Attrs(L) = $\{k,a_1,\ldots a_n\}$ U {attributes of S_1} U…U {attributes of S_m} U {t} and PK(L) = k. The attribute t is called a type (or **discriminating**) attribute that indicates the subclass to which each tuple belongs

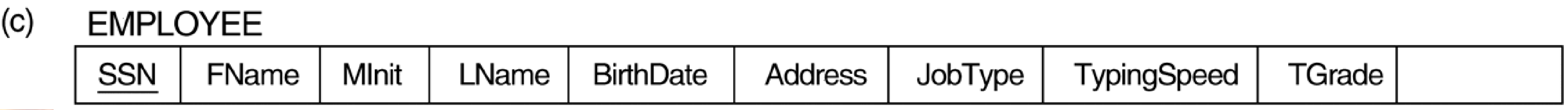
Option 8D: Single relation with multiple type attributes.

Create a single relation schema L with attributes Attrs(L) = $\{k,a_1,\ldots a_n\}$ U {attributes of S_1} U…U {attributes of S_m} U $\{t_1, t_2,\ldots,t_m\}$ and PK(L) = k. Each t_i, $1 < I < m$, is a Boolean type attribute indicating whether a tuple belongs to the subclass S_i.

EER diagram notation for an attribute-defined specialization on JobType.



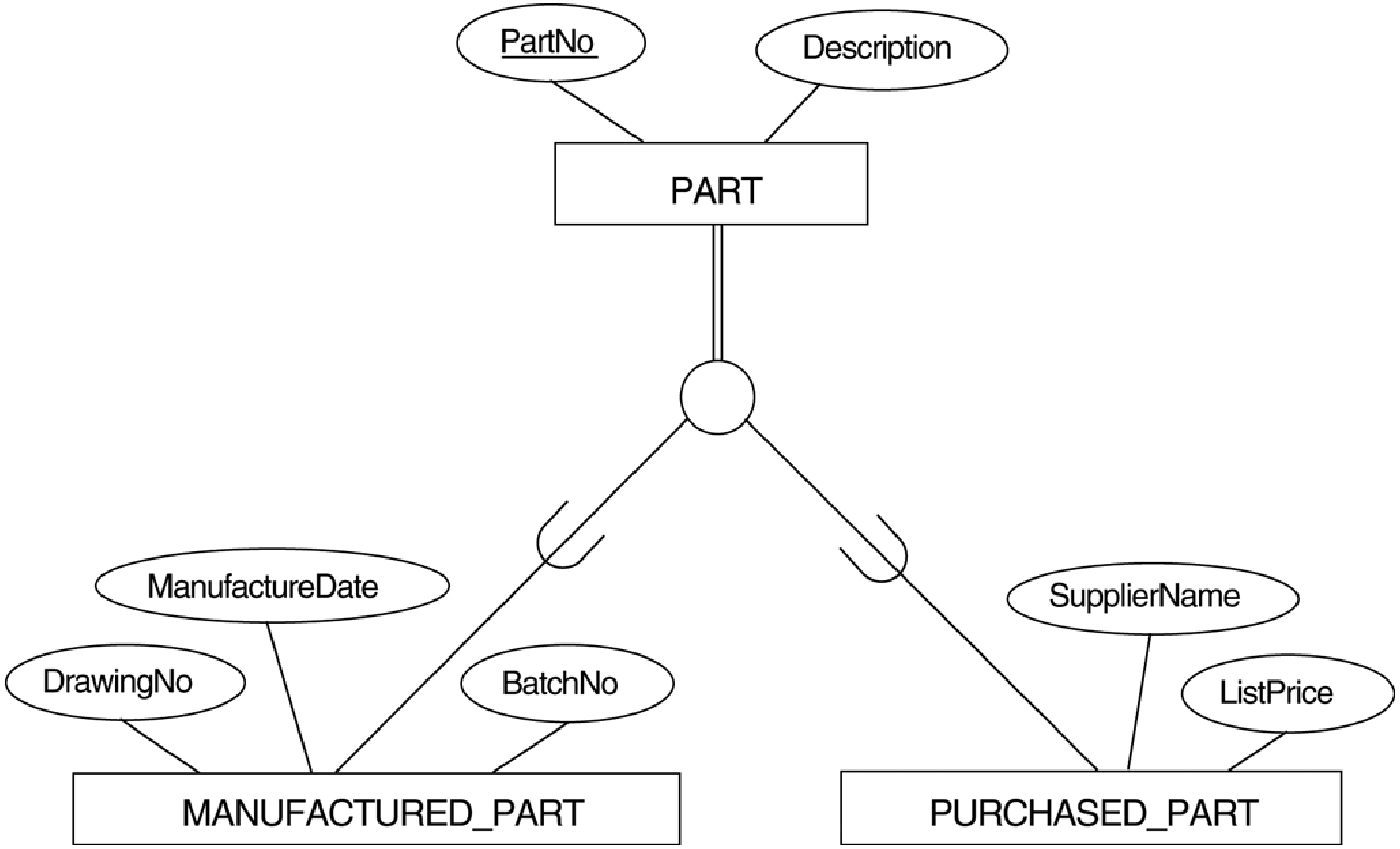
Options for mapping specialization or generalization.
(c) Mapping the EER schema using option 8C.



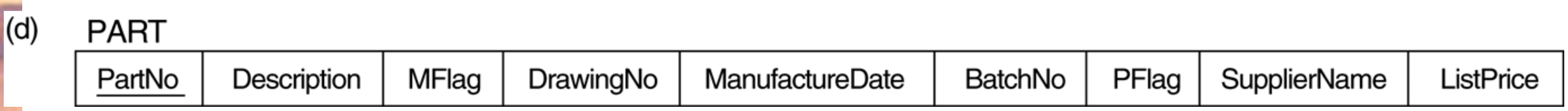
(c) EMPLOYEE

<u>SSN</u>	FName	MInit	LName	BirthDate	Address	JobType	TypingSpeed	TGrade	

EER diagram notation for an overlapping (nondisjoint) specialization.



Options for mapping specialization or generalization.
(d) Mapping using option 8D with Boolean type fields Mflag and Pflag.



(d) PART

PartNo	Description	MFlag	DrawingNo	ManufactureDate	BatchNo	PFlag	SupplierName	ListPrice

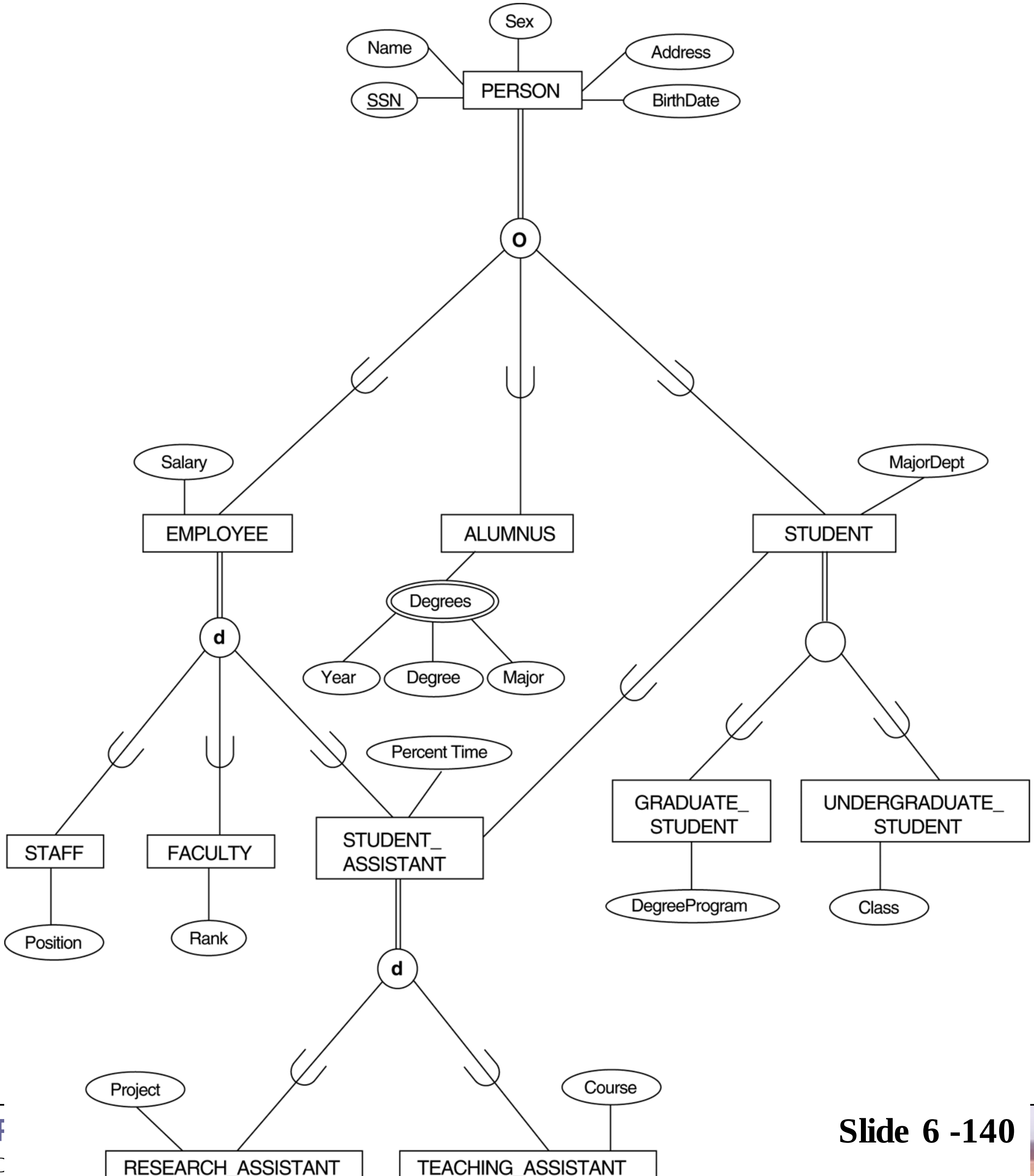
Mapping EER Model Constructs to Relations (cont)

- **Mapping of Shared Subclasses (Multiple Inheritance)**

  A shared subclass, such as STUDENT_ASSISTANT, is a subclass of several classes, indicating multiple inheritance. These classes must all have the same key attribute; otherwise, the shared subclass would be modeled as a category.

  We can apply any of the options discussed in Step 8 to a shared subclass, subject to the restriction discussed in Step 8 of the mapping algorithm. Below both 8C and 8D are used for the shared class STUDENT_ASSISTANT.

A specialization lattice with multiple inheritance for a UNIVERSITY database.



Mapping the EER specialization lattice using multiple options.

PERSON

SSN	Name	BirthDate	Sex	Address

EMPLOYEE

SSN	Salary	EmployeeType	Position	Rank	PercentTime	RAFlag	TAFlag	Project	

ALUMNUS

SSN

ALUMNUS_DEGREES

SSN	Year	Degree	

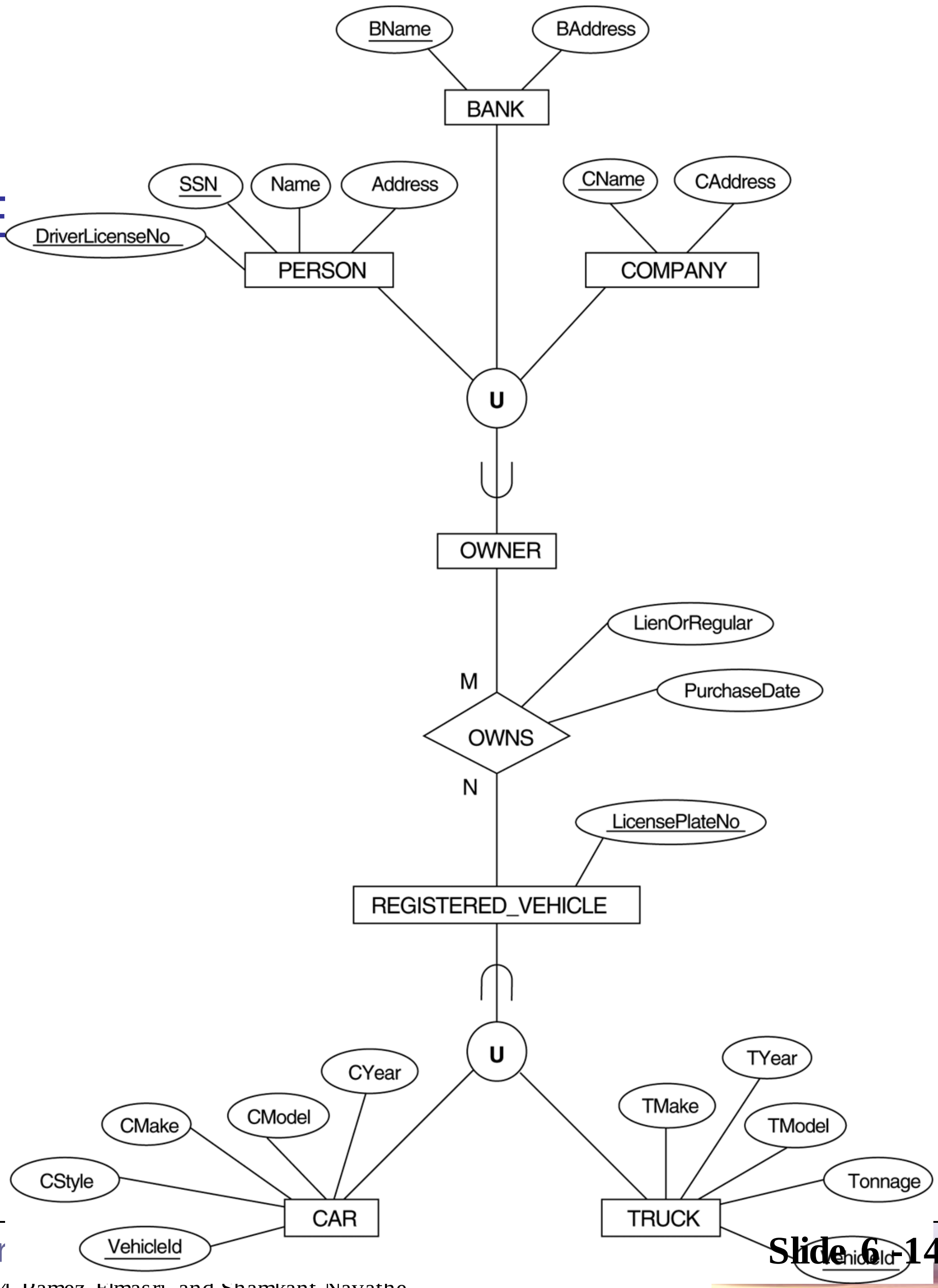
STUDENT

SSN	MajorDept	GradFlag	UndergradFlag	DegreeProgram	Class	StudAssistFlag

Mapping EER Model Constructs to Relations (cont)

- **Step 9: Mapping of Union Types (Categories).**
  - For mapping a category whose defining superclass have different keys, it is customary to specify a new key attribute, called a **surrogate key**, when creating a relation to correspond to the category.
  - In the example below we can create a relation OWNER to correspond to the OWNER category and include any attributes of the category in this relation. The primary key of the OWNER relation is the surrogate key, which we called OwnerId.

Two categories (union types): OWNER and REGISTERED_VEHICLE



Mapping the EER categories (union types) to relations.

PERSON

SSN	DriverLicenseNo	Name	Address	

BANK

BName	BAddress	OwnerId

COMPANY

CName	CAddress	OwnerId

OWNER

OwnerId

REGISTERED_VEHICLE

VehicleId	LicensePlateNumber

CAR

VehicleId	CStyle	CMake	CModel	

TRUCK

VehicleId	TMake	TModel	Tonnage	TYear

OWNS

OwnerId	VehicleId	PurchaseDate	LienOrRegular

Mapping Exercise

Exercise 7.4.

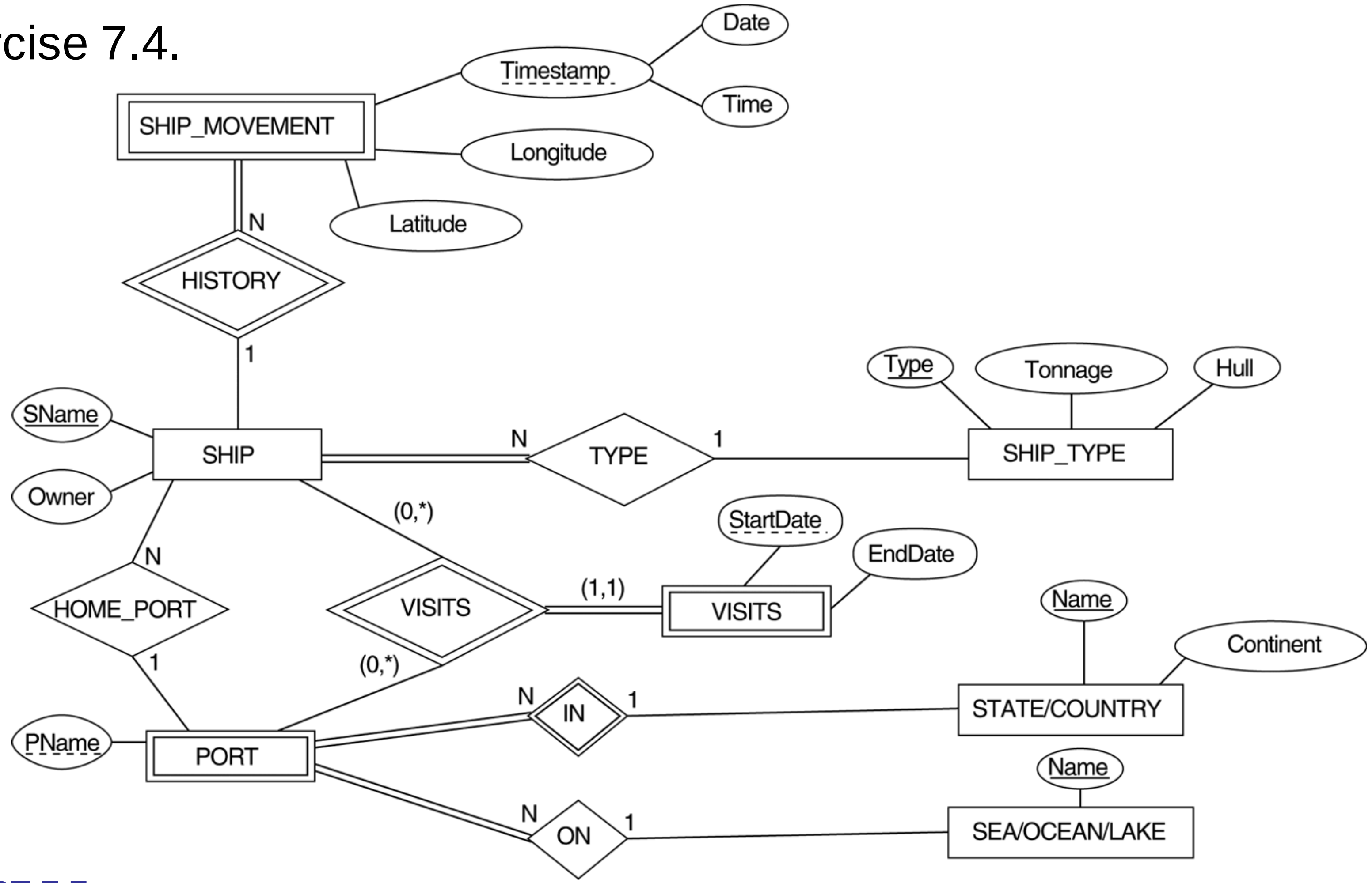


FIGURE 7.7
An ER schema for a SHIP_TRACKING database.