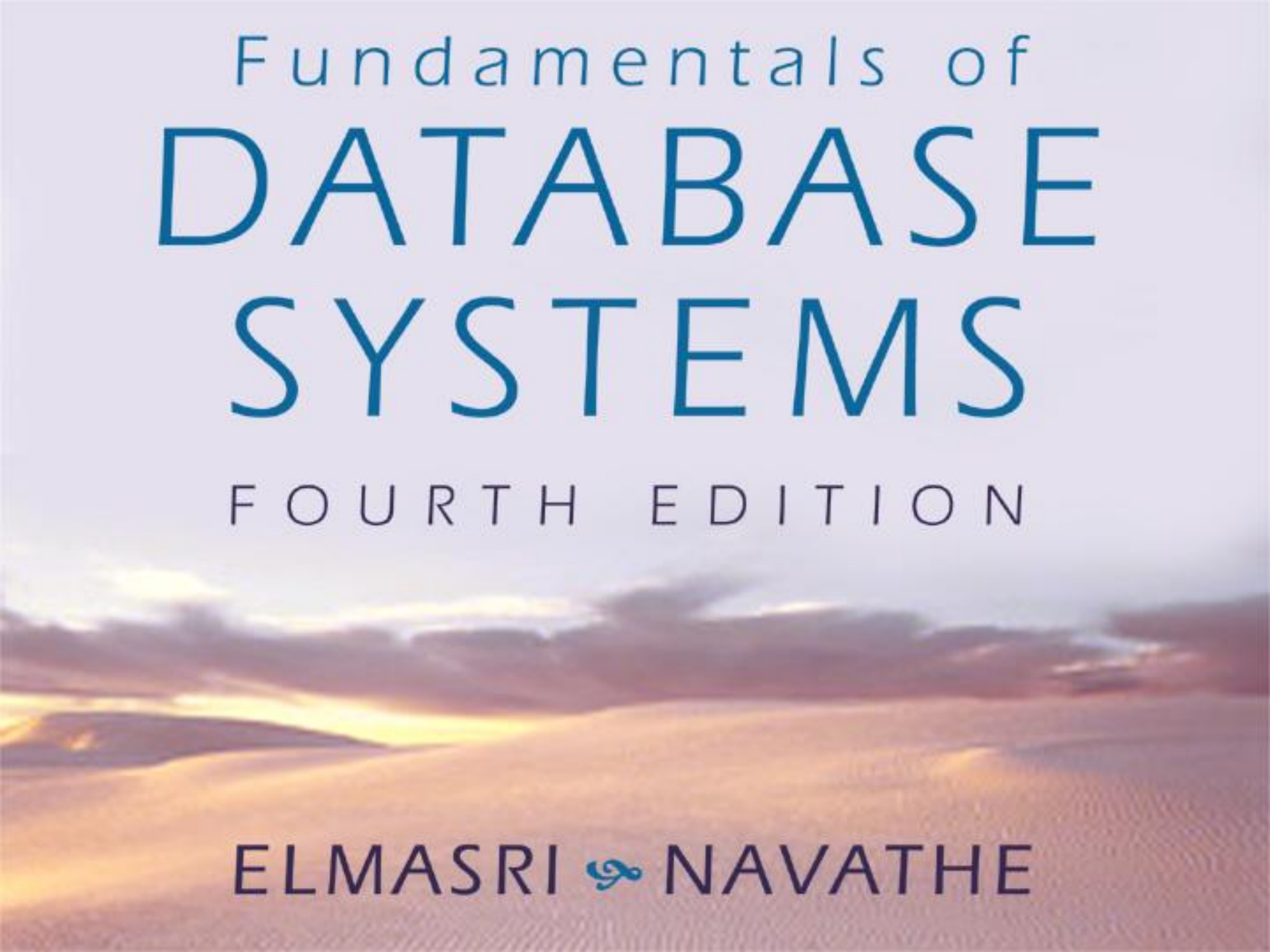
Fundamentals of DATABASE SYSTEMS

FOURTH EDITION



ELMASRI NAVATHE

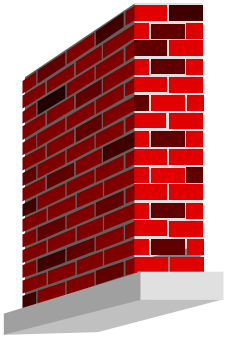
Chapter 4 - Part II

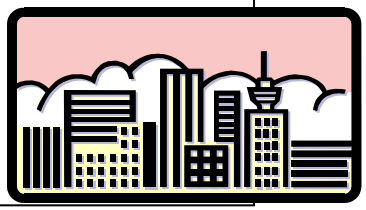
Enhanced Entity-Relationship and UML Modeling

This material may be used at the beginning of the transparencies for Chapter 4. It sets the stage for looking at modeling from a general standpoint. The details of abstraction mentioned here are covered in Section 4.8, but the constraints are previously discussed in Section 3.4 already.

THE BASICS

- Fundamental Principle of Modeling:
  - Data Abstraction
- Basic Process of Modeling
  - Define building blocks for
  - holding groups of data
  - Use rules of a data model to establish
  - relationships among blocks
- Add constraints - structural/ semantic





Part 1: Fundamentals of Data Modeling

1 Inputs to Data Modeling
2 The Process of Modeling
3 Data Modeling Abstractions
4 Classification
5 Aggregation
6 Identification
7 Generalization
8 Coverage Constraints in Generalization
9 Cardinality and Participation Constraints

Inputs to Data Modeling

- Using the products of requirements analysis
- Verbal and written communication among users and designers
- Knowledge of meaning of data
  - Existing Programs
  - Existing Files
  - Existing Documents
  - Existing Reports
- Application Planning / Documentation and Design

Overall Process of Modeling

- Abstraction
- Use of some modeling discipline (Data Model)
- Use of a representation technique
  - Language
  - Diagramming
  - Tools
- Analysis of business rules/semantic constraints (these are typically beyond the capability of the data model)

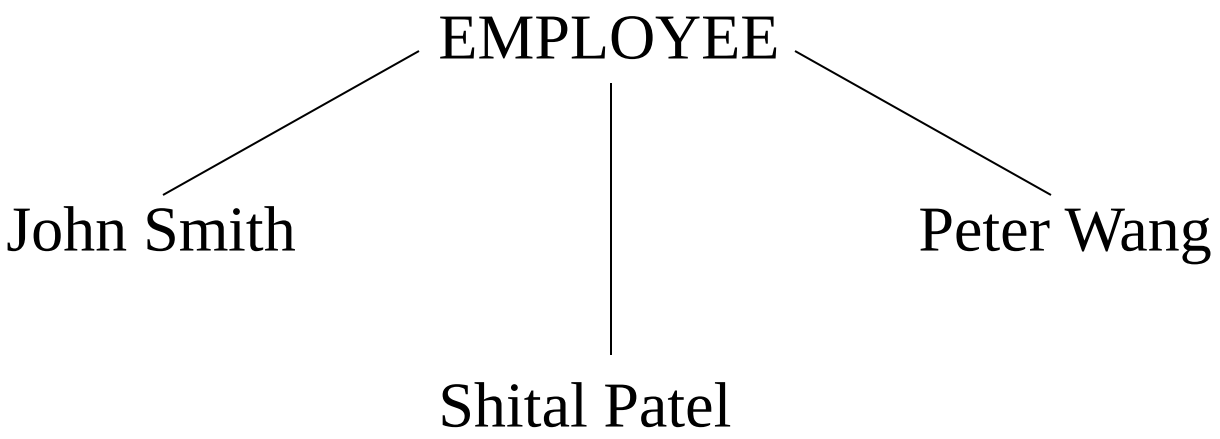
Types of Abstractions

Classification	A is a member of class B
Aggregation	B,C,D are aggregated into A A is made of/composed of B,C,D
Generalization	B,C,D can be generalized into A, **B is-an A, C is- an A, D is-an A**
Specialization	A can be specialized into B,C,D B,C,D are special cases of A

Classification Abstraction

- Relationship between a class and its members

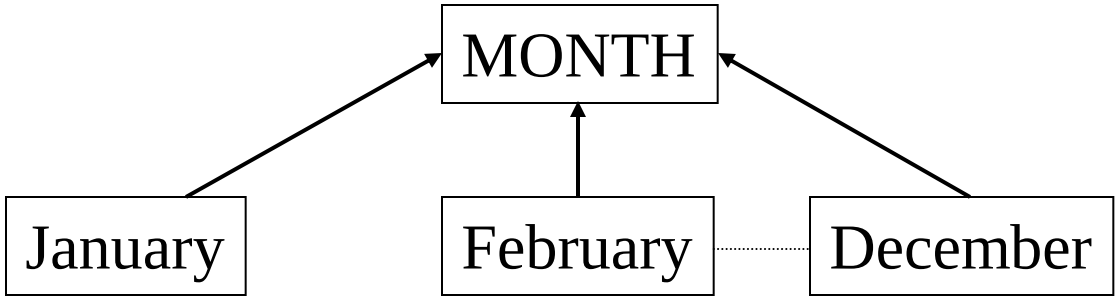
  John Smith, Shital Patel, and Peter Wang are all employees. They are all members of a class: EMPLOYEE class



Each individual employee is a member of the class EMPLOYEE

Classification Abstraction (contd.)

Exhaustive enumeration of members:



January, February etc. are members of the class "MONTH"

Represents "member-of" relationship

In object-oriented modeling :

MONTH : an Object type or class

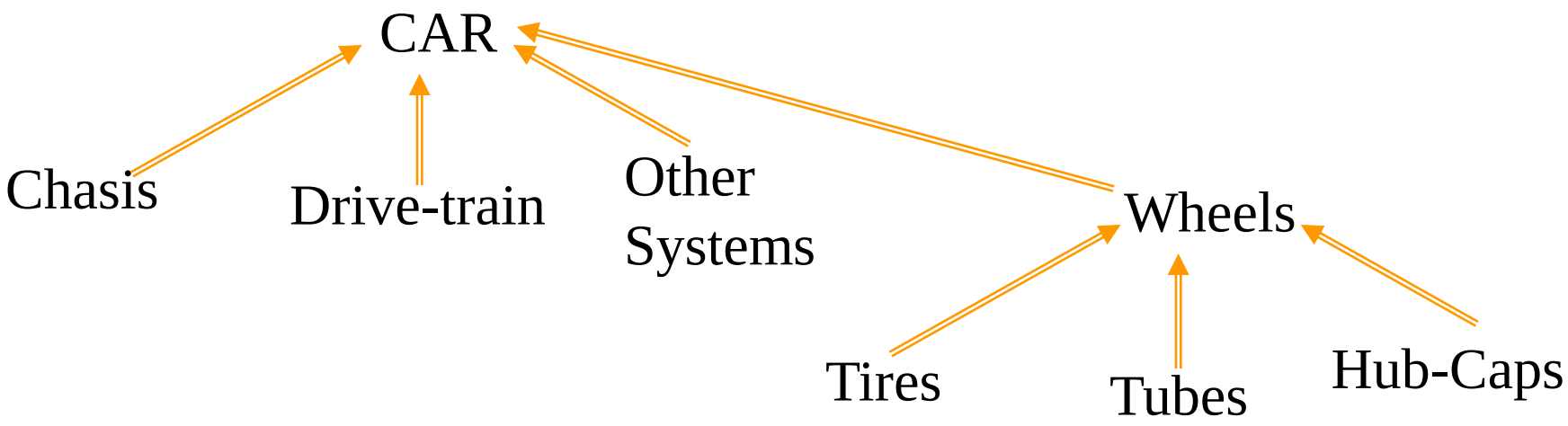
January … December : objects that belong to class MONTH

Classification - Class Properties

- Collection of similar entities or concepts into a higher level concept
- **EMPLOYEE class collects all employees into one class**
- A class has properties called “class properties”
- EMPLOYEE class has **class properties -** e.g., average salary, total number of employees
- Each member has values for own properties (e.g. name, address, salary): called **member properties**

Aggregation Abstraction

Defines a new class from a **set of classes which are identified as <u>components of</u> the root class**



⟶ represents IS-PART-OF (component) relationship

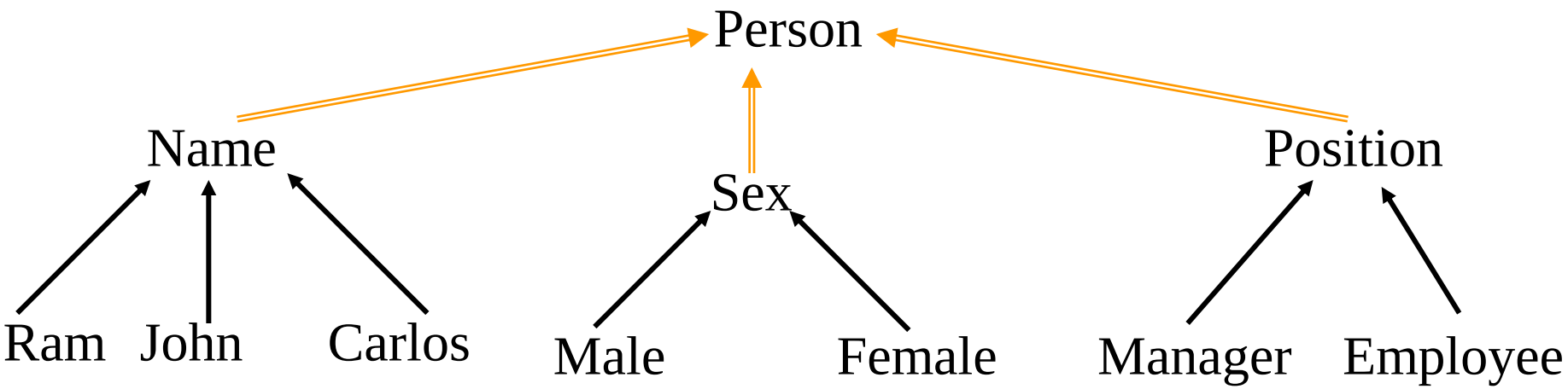
Root class: CAR
Component Classes: Chassis, Drive-Train, Other Systems, Wheels
Root class: Wheels
Component Classes: Tires, Tubes, Hub-Caps

Classification and Aggregation

Classification and Aggregation are used to build schemas

Example: class Person

Representation:



Name, Sex, and Position aggregate into Person. They are classes themselves.

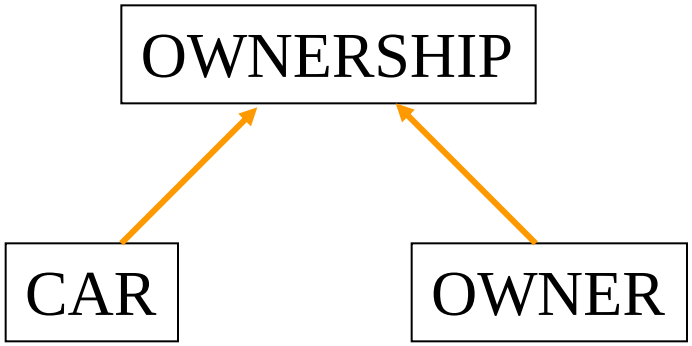
Ram, John, Carlos are classified into Name or Name is a classification of Ram, John, Carlos

Two Contexts for Aggregation

Aggregate two or more classes into a higher level concept. It may be considered a relationship or association between them.

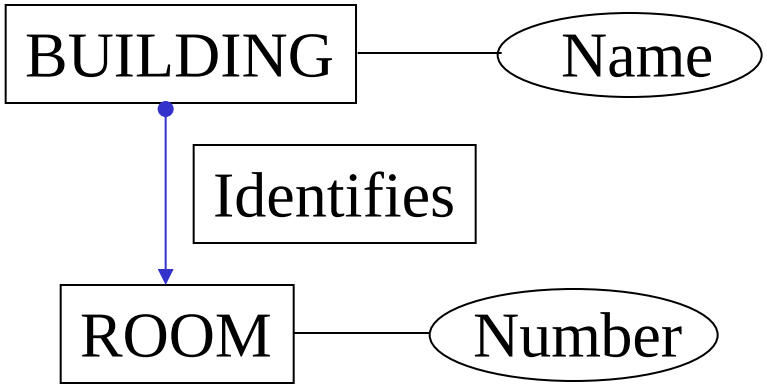
Context1: CAR is an aggregate (composition) of Chassis, Drive-train, Other Systems, Wheels.

Context 2: OWNERSHIP is an aggregate (relationship) of CAR and OWNER



Identification

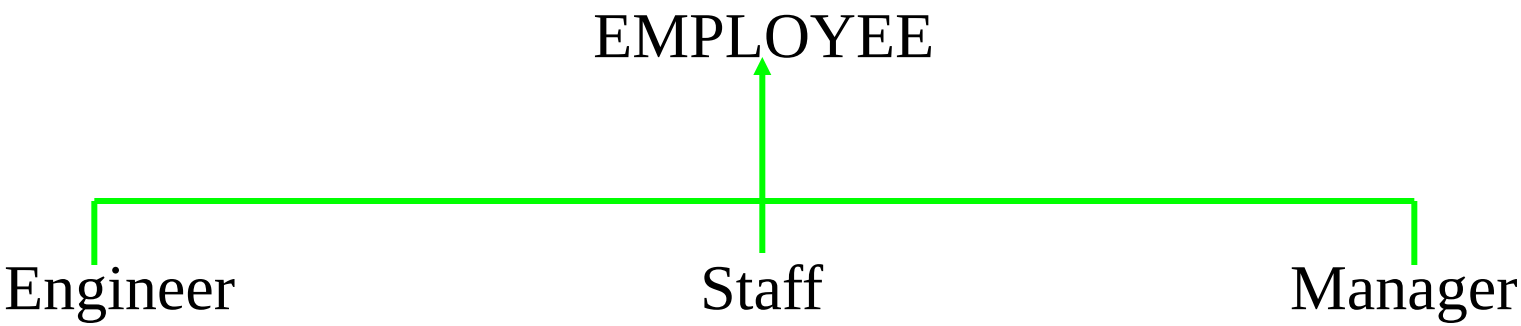
Identifies one concept (an instance of it) from another concept.



Generalization Abstraction

Defines a set-subset relationship between a class and a set of member classes.

Establishes a mapping (or a relationship) from the **generic class** to the **member class** (or subclass, or subset class).



GENERIC CLASS: EMPLOYEE

MEMBER CLASS: Engineer, Staff, Manager

Implies that all properties associated with the Employee class are inherited by the three leaf classes.

Data Abstraction (contd.)

Process of hiding (suppressing) unnecessary details so that the high level concept can be made more visible.

This enables programmers, designers, etc., To communicate easily and to understand the application's data and functional requirements easily.

TYPES OF ABSTRACTION

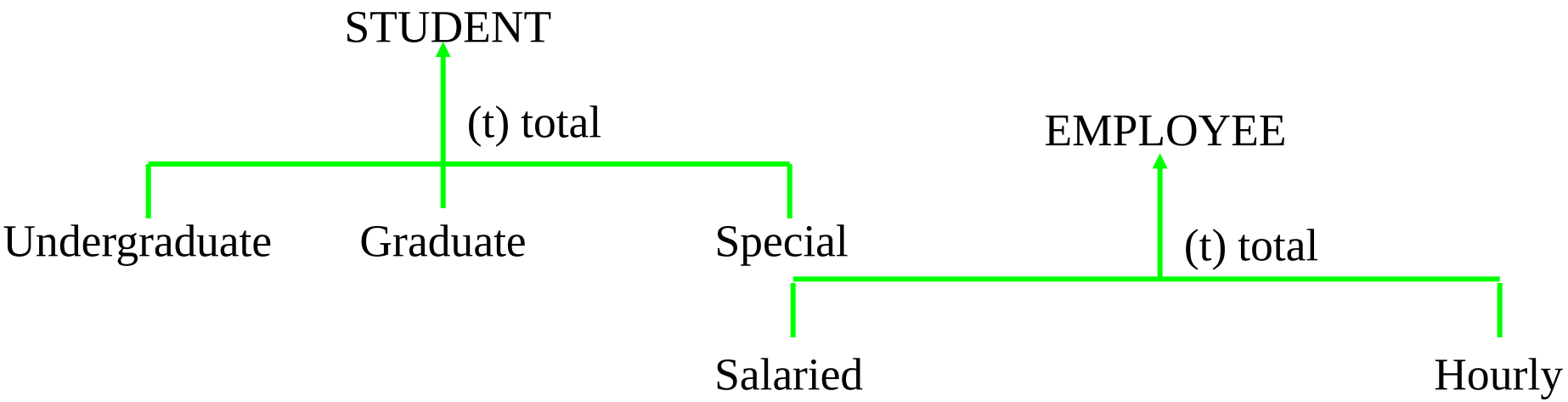
Classification: IS-A-MEMBER-OF
Aggregation: IS-MADE-OF, IS-ASSOCIATED-WITH
Composition: IS-MADE-OF (similar to aggregation)
(A COMPRISES B,C,D)
Identification: IS-IDENTIFIED-BY
Generalization: IS-A IS-LIKE IS-KIND-OF

Coverage Constraints for Generalization Abstraction

TYPE 1 : Total vs. Partial coverage

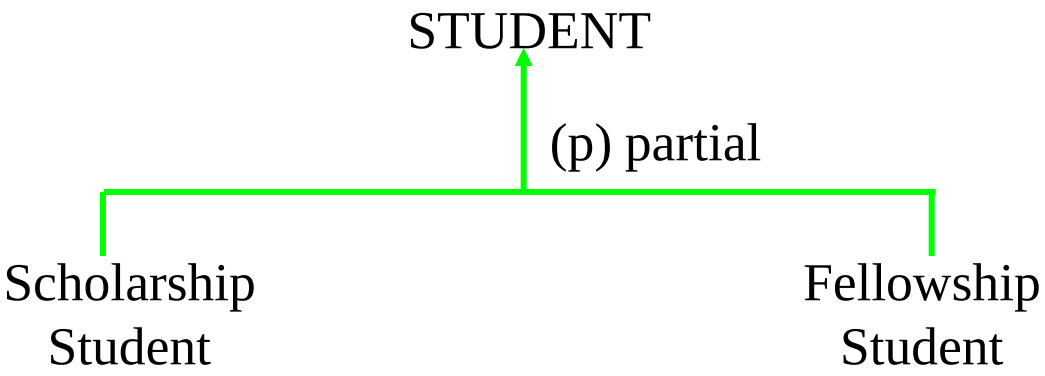
Total: The coverage is total if each member of the generic class is mapped to at least one member among the member classes

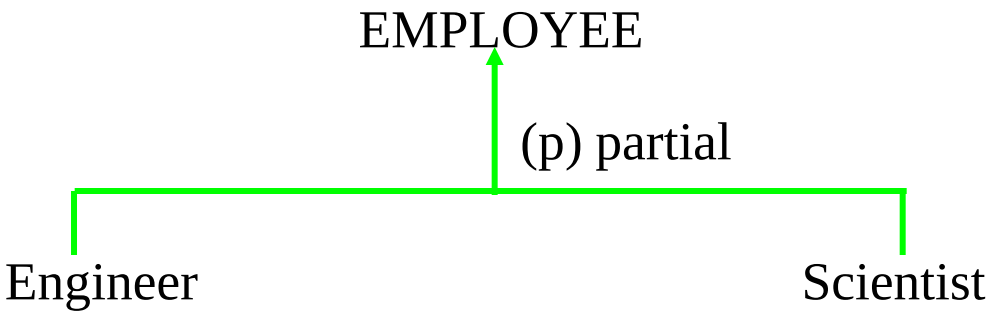
Partial: The coverage is partial if there are some member(s) of the generic class that cannot be mapped to any member among the member classes



Coverage Constraints for Generalization Abstraction (contd.)

Partial Coverage Constraint examples:



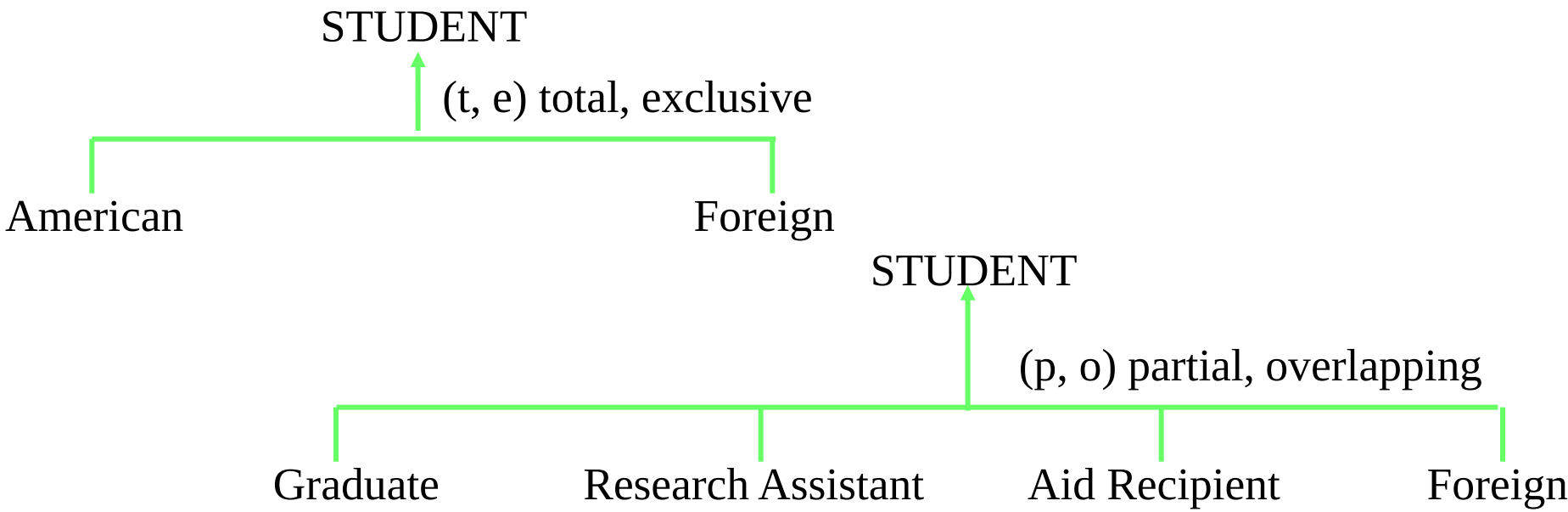


Coverage Constraints for Generalization Abstraction

TYPE 2: EXCLUSIVE VS. OVERLAPPING (Disjointedness Constraint)

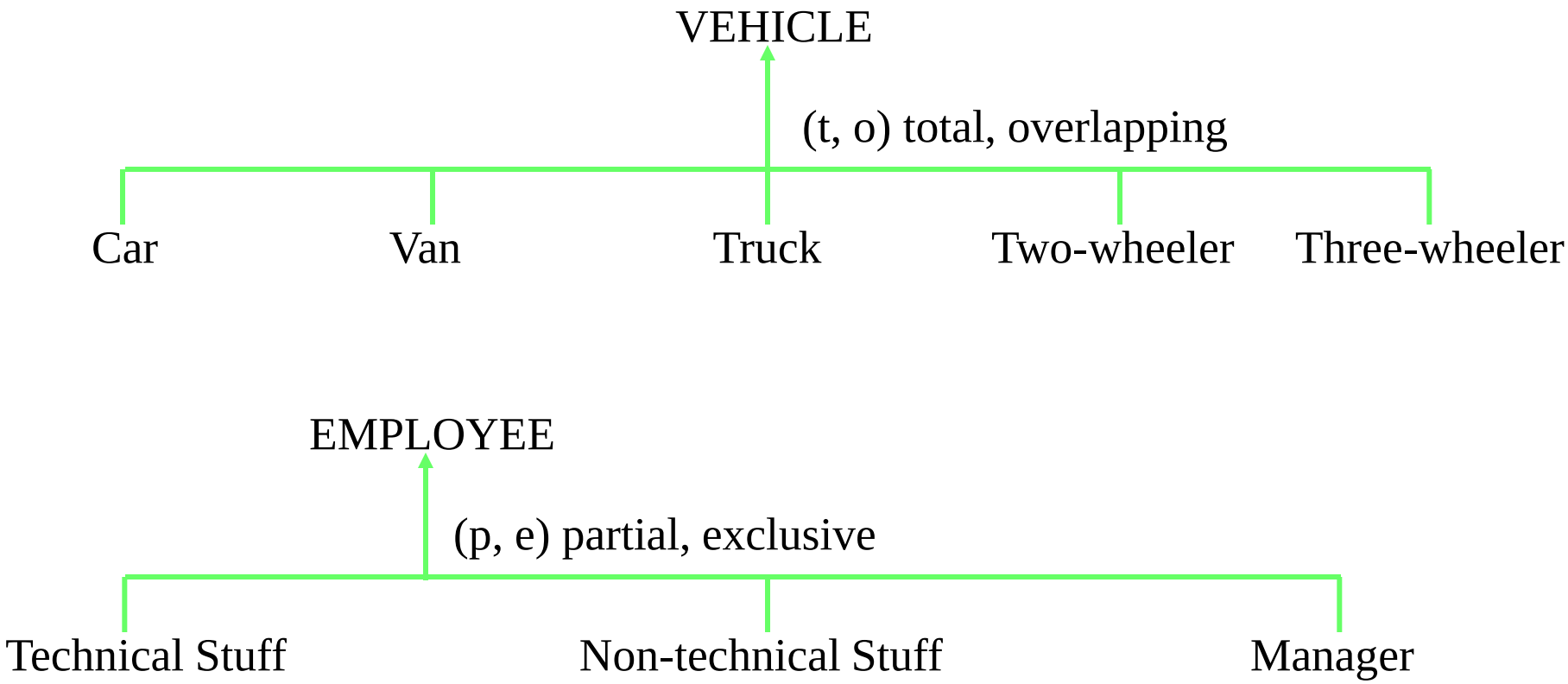
EXCLUSIVE constraint: A member of the generic class is mapped to one element of at most one subset class.

OVERLAPPING constraint: There exists some member of the generic class that can be mapped to two or more of the subset classes.



Coverage Constraints for Generalization Abstraction (contd.)

More examples of different combinations:



Cardinality Constraints

Cardinality Constraint: Quantification of the relationship between two concepts or classes (a constraint on aggregation)

MINIMUM (A,B) = n

At a minimum, one instance of A is related to at least n instances of B.

n = 0	MIN(A,B) = 0	MIN(Person, Car) = 0
n = 1	MIN(A,B) = 1	MIN(Cust, Ship-address) = 1
n = inf.	MIN(A,B) = inf.	NOT POSSIBLE
n = x (fixed)	MIN(A,B) = x	MIN(Car, Wheels) = 4

Cardinality Constraints (contd.)

MAXIMUM (A,B) = n

At a maximum, one instance of A is related to at least n instances of B.

n = 0	MAX(A,B) = 0	DOES NOT ARISE
n = 1	MAX(A,B) = 1	MAX(Cust, Ship-address) = 1
n = inf.	MAX(A,B) = inf.	MAX(Cust, Orders) = inf.
n = x (fixed)	MAX(A,B) = x	MAX(Stud, Course) = 6

Participation Constraints

MIN (A,B) = 0 **Optional Participation**

MIN (A,B) = 1 **Mandatory Participation**

MAX (A,B) = 0 **No Participation**

MIN (A,B) = x, MAX (A,B) = y **Range Constrained Participation**

Summary of Modeling Concepts

ABSTRACTIONS

- CLASSIFICATION
- AGGREGATION (COMPOSITION AND ASSOCIATION)
- IDENTIFICATION
- GENERALIZATION AND SPECIALIZATION

CONSTRAINTS

- CARDINALITY (Min. and Max)
- PARTICIPATION
- COVERAGE (Total vs. Partial, Exclusive vs. Overlapping)

Figure 4.1 EER diagram notation for representing specialization and subclasses.

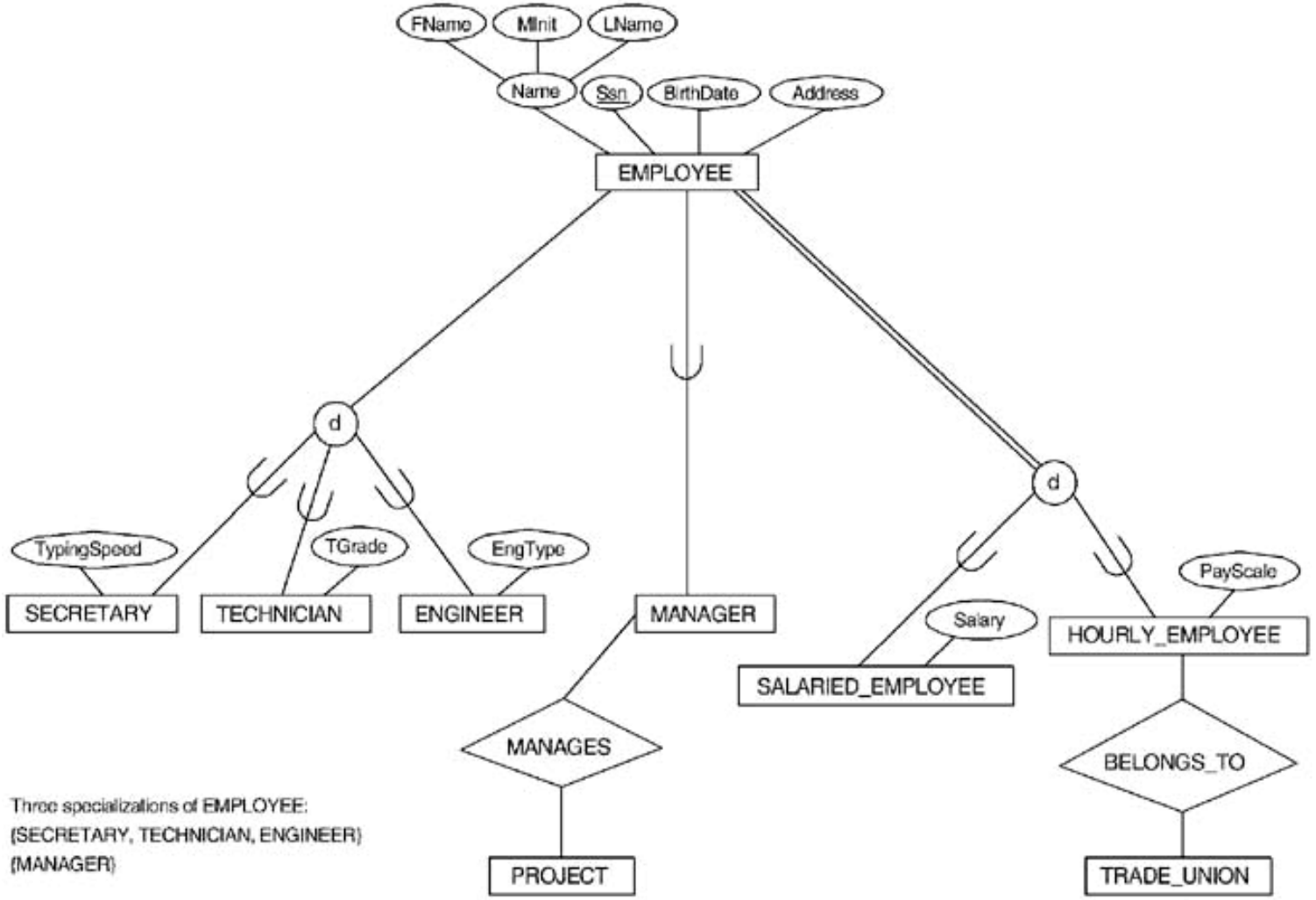


Figure 4.3 Examples of generalization. (a) Two entity types CAR and TRUCK. (b) Generalizing car and TRUCK into VEHICLE.

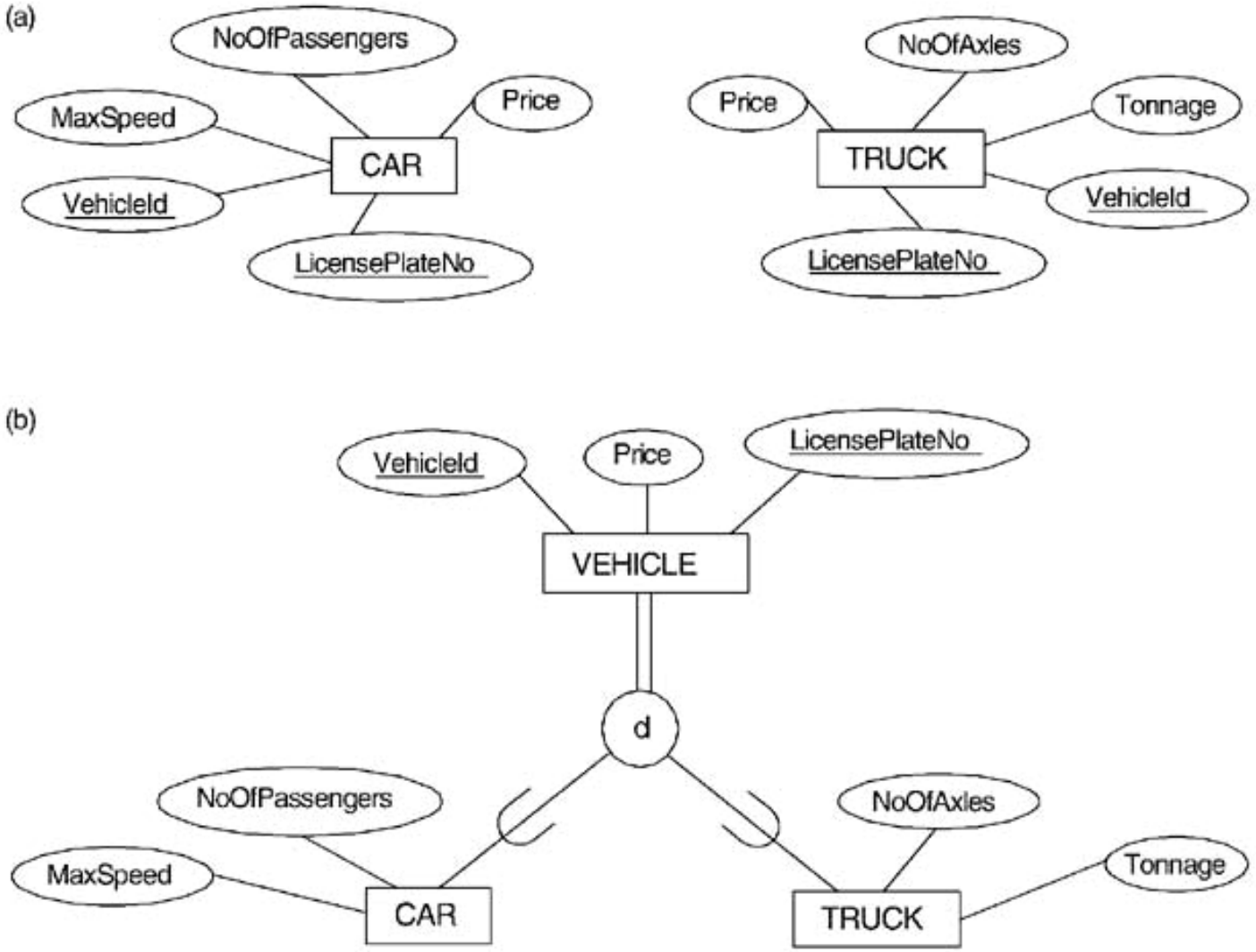


Figure 4.5 Notation for specialization with overlapping (nondisjoint) subclasses.

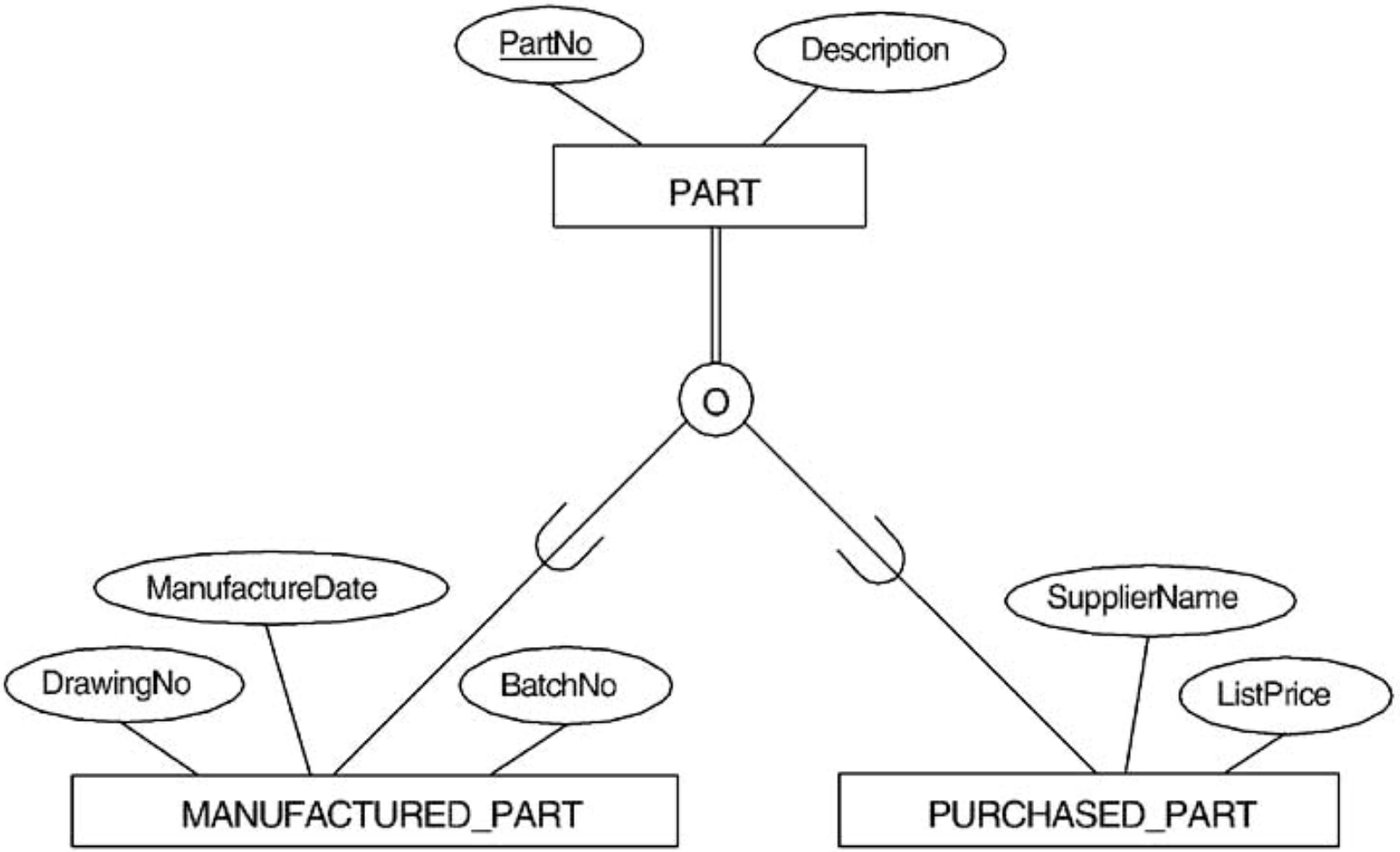


Figure 4.6 A specialization lattice with the shared subclass ENGINEERING_MANAGER.

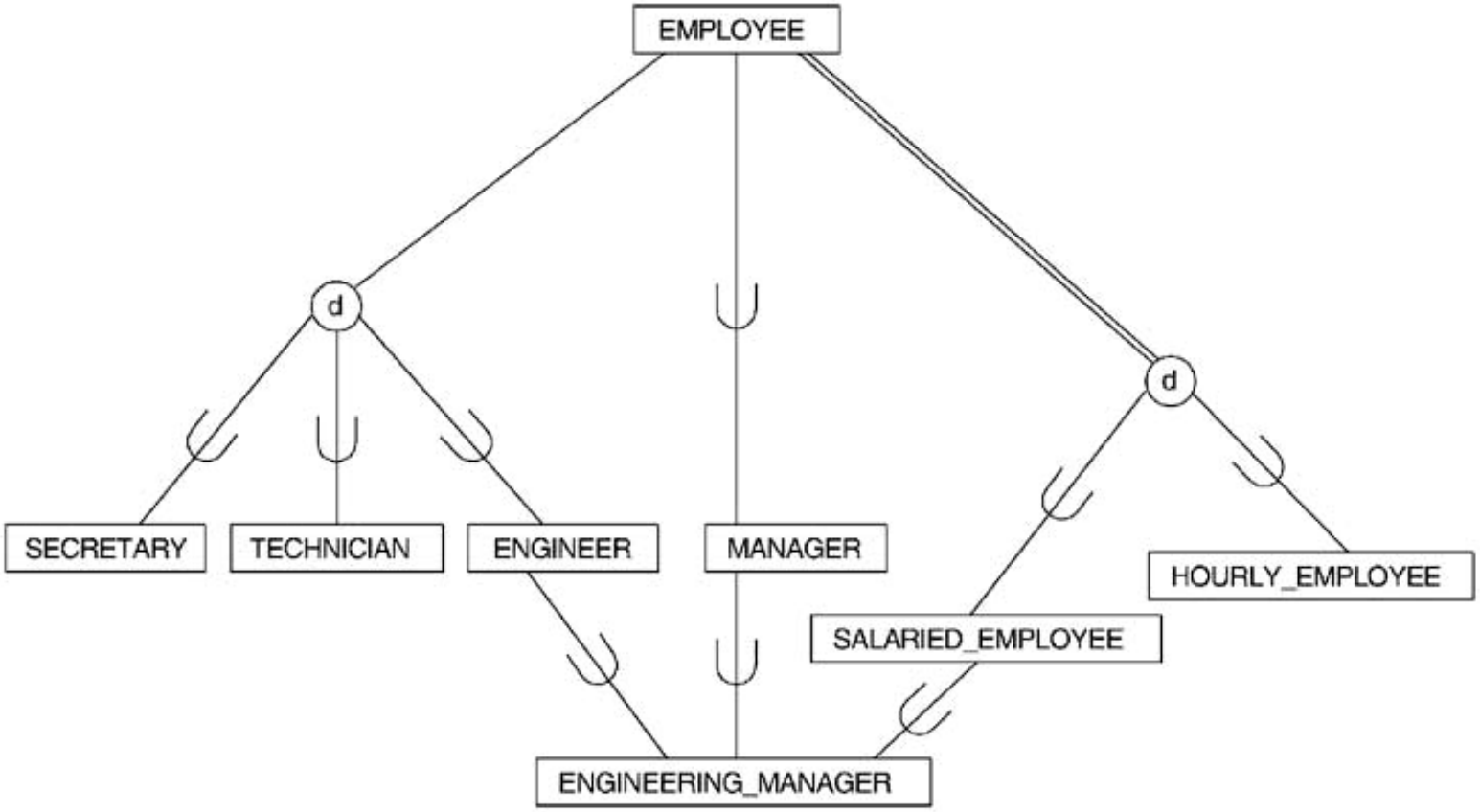
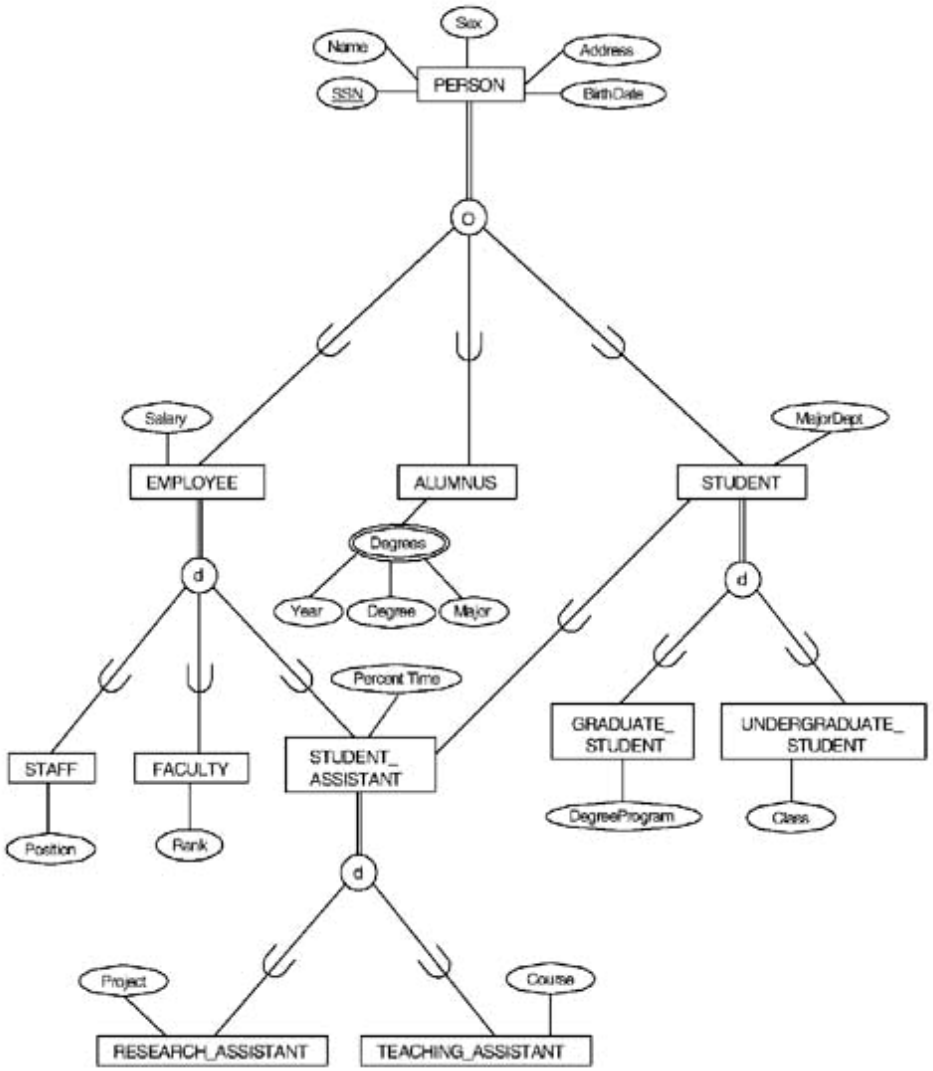


Figure 4.7 A specialization lattice (with multiple inheritance) for a UNIVERSITY database.



6.8.5 Aggregation

One limitation of the E-R model is that it cannot express relationships among relationships. To illustrate the need for such a construct, consider the ternary relationship *proj_guide*, which we saw earlier, between an *instructor*, *student* and *project* (see Figure 6.6).

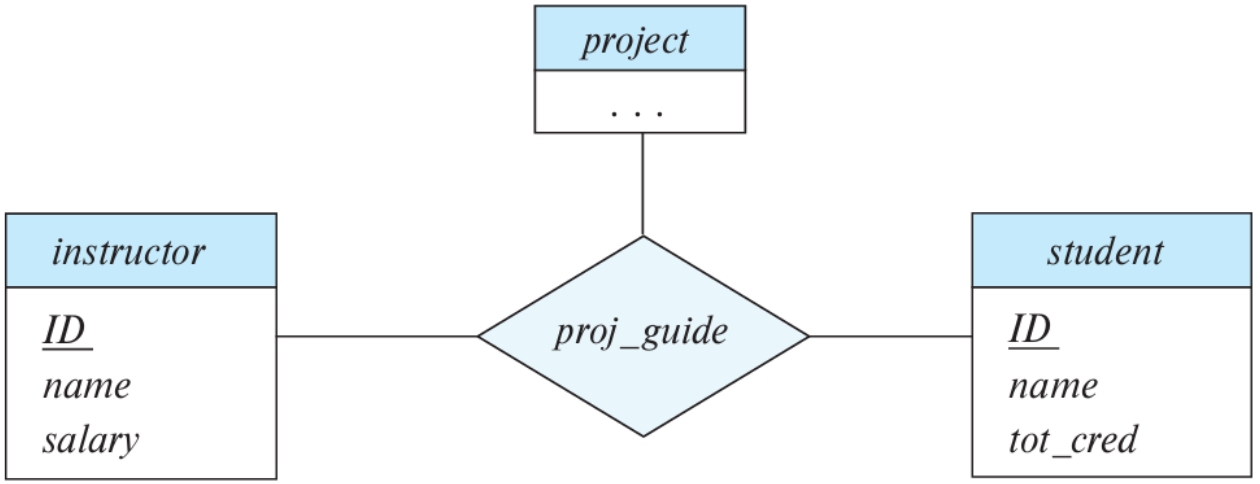


Figure 6.6 E-R diagram with a ternary relationship *proj_guide*.

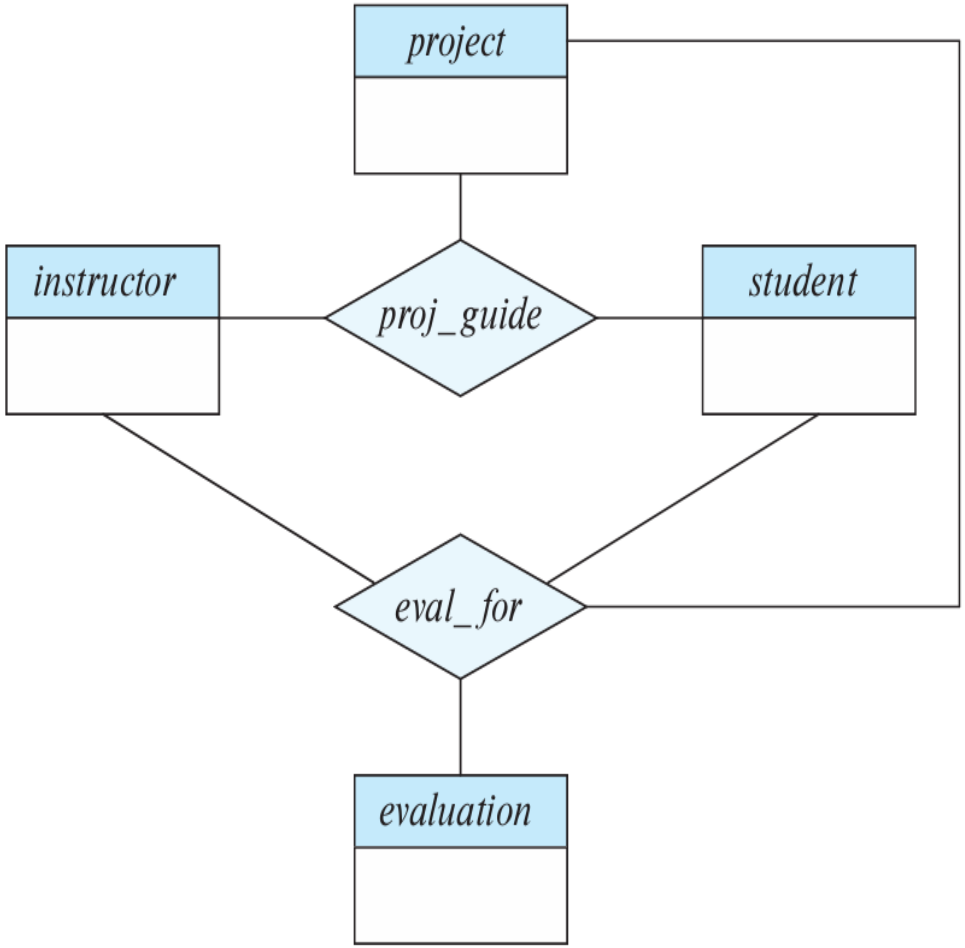


Figure 6.19 E-R diagram with redundant relationships.

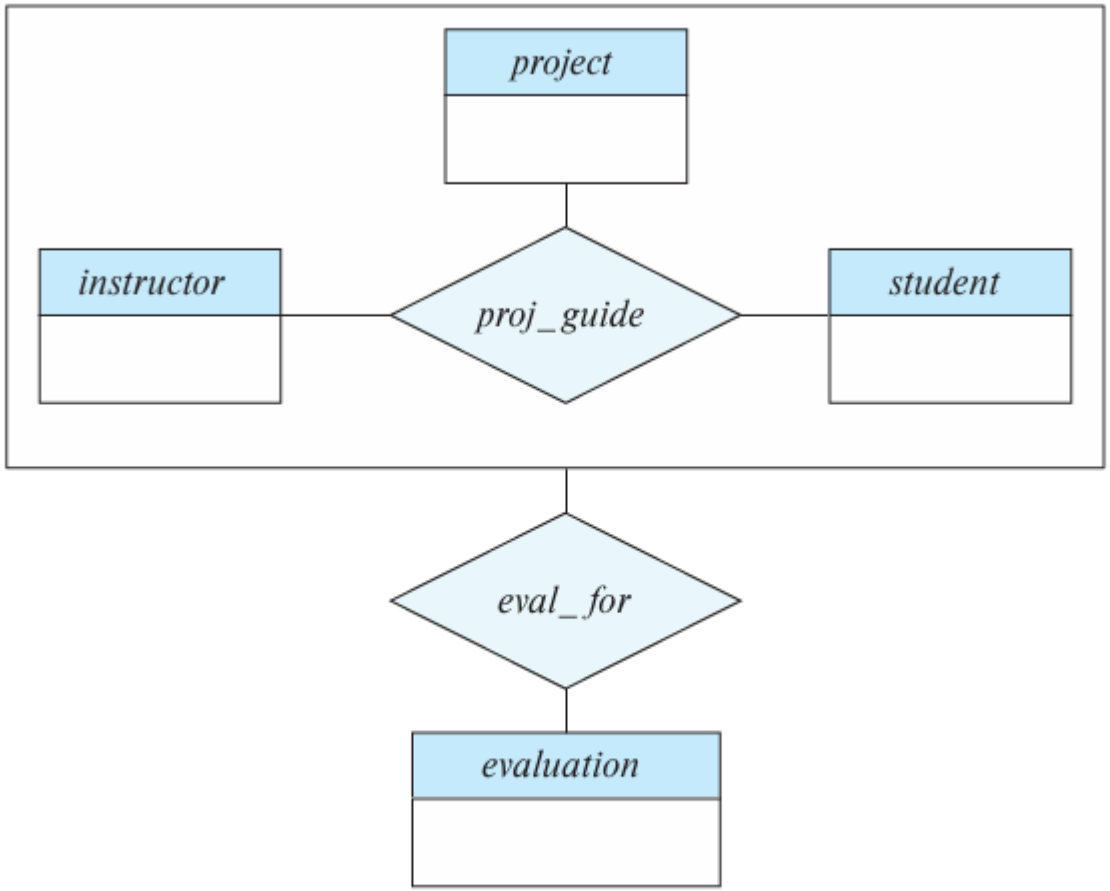


Figure 6.20 E-R diagram with aggregation.

- **Total specialization** or **generalization**. Each higher-level entity must belong to a lower-level entity set.
- **Partial specialization** or **generalization**. Some higher-level entities may not belong to any lower-level entity set.

- **Primary Keys**
- A **primary key** is a special type of candidate key in a database table that is selected to uniquely identify each record (row). It is chosen from the set of candidate keys and enforces the following properties:
- **Characteristics of a Primary Key**
- **Uniqueness**: No two rows in the table can have the same value for the primary key.
- **Non-Null**: Every row must have a value for the primary key; null values are not allowed.
- **Immutability**: The value of a primary key should not change over time.
- **Minimality**: It is the simplest combination of attributes that uniquely identifies a record.

- **Purpose of Primary Keys**

- To ensure **data integrity**.
- To uniquely identify rows in a table for operations like search, update, or delete.
- To establish **relationships** between tables (used as foreign keys in other tables).

Example 1: Simple Primary Key

Table: Students

StudentID	Name	Email
101	Alice	alice@email.com
102	Bob	bob@email.com

- **Primary Key:** `StudentID` (unique for every student).

Example 2: Composite Primary Key

Table: Orders

OrderID	ProductID	Quantity
1	1001	2
1	1002	1

- Here, neither `OrderID` nor `ProductID` is sufficient to uniquely identify a row.
- **Composite Primary Key**: `(OrderID, ProductID)` ensures each row is unique.

CandidateKeys

A **candidate key** is a minimal set of attributes in a relation (table) that can uniquely identify each tuple (row). Each candidate key satisfies the following properties:

Uniqueness: No two rows in the table can have the same values for the attributes in the candidate key.

Minimality: The key must be minimal, meaning no attribute can be removed without losing the property of uniqueness.

Properties of Candidate Keys

There can be multiple candidate keys in a table, but only one of them is chosen as the **primary key**.

A candidate key can consist of a single attribute or a combination of attributes

.

Each candidate key can **act as a primary key.**

Example 1: Single Attribute

Table: Students

StudentID	Name	Email
101	Alice	alice@email.com
102	Bob	bob@email.com

- Candidate Keys:
  - **StudentID**: Each student has a unique ID.
  - **Email**: Each student has a unique email address.

Table: Employees

EmployeeID	Name	Email	PhoneNumber
E101	Charlie	charlie@abc.com	1234567890
E102	Diana	diana@abc.com	0987654321

- Candidate Keys:
  - **EmployeeID**: Unique for each employee.
  - **Email**: Unique for each employee.
  - **PhoneNumber**: Unique for each employee.

If "Name" is not unique, it cannot be part of any candidate key.

Key Considerations

1. **Superkey**: A set of attributes that can uniquely identify a tuple. Every candidate key is a superkey, but not every superkey is a candidate key (as superkeys may not be minimal).
2. **Primary Key**: One candidate key is chosen as the primary key for the table.
3. **Alternate Keys**: All candidate keys that are not selected as the primary key.

- **Super Keys**
- A **super key** is a set of one or more attributes (columns) in a database table that can uniquely identify each row (tuple).
- Every super key satisfies the **uniqueness property**, meaning no two rows in the table will have the same combination of values for the attributes in the super key.

- **Characteristics of Super Keys**
- **Uniqueness**: A super key can uniquely identify every row in the table.
- **Redundancy**: A super key may contain additional attributes that are not necessary for uniqueness.
- **Candidate Keys**: All candidate keys are super keys, but not all super keys are candidate keys (because candidate keys are minimal).

Table: Employees

EmployeeID	Name	Email	Phone
101	Alice	alice@email.com	1234567890
102	Bob	bob@email.com	9876543210

Possible Super Keys:

1. **{EmployeeID}**: Unique for each row.
2. **{Email}**: Unique for each row.
3. **{Phone}**: Unique for each row.
4. **{EmployeeID, Name}**: Still unique but includes redundant attribute (`Name`).
5. **{EmployeeID, Email}**: Unique but redundant.

Foreign Keys

- A **foreign key** is a field (or a collection of fields) in a table that creates a relationship between two tables. It is used to ensure **referential integrity** in the database. A foreign key in one table refers to the **primary key** in another table, linking the two tables together.

- **Characteristics of a Foreign Key**
- **Referential Integrity**: Ensures that the value in the foreign key column matches a value in the referenced table's primary key column or is NULL.
- **Defines Relationships**: Establishes a parent-child relationship between two tables.
- **Multiple Foreign Keys**: A table can have multiple foreign keys referencing different parent tables.
- **Can Be NULL**: If the relationship is optional, the foreign key can have NULL values.

Key Relationships

- **Super Key**: May have extra attributes (non-minimal).
- **Candidate Key**: Minimal super key (no redundant attributes).
- **Primary Key**: A candidate key chosen to uniquely identify rows in the table.

- **Purpose of Foreign Keys**
- To enforce relationships between tables.
- To prevent actions that destroy linkages between tables.
- To maintain data integrity and consistency.

Example 1: Simple Foreign Key

Table: Departments

DeptID	DeptName
1	HR
2	Engineering
3	Marketing

Table: Employees

EmpID	EmpName	DeptID
101	Alice	1
102	Bob	2
103	Charlie	3

- **DeptID** in the `Employees` table is a foreign key that references the **DeptID** column in the `Departments` table.
- This relationship ensures that every employee belongs to a valid department.

Differences Between Super Key, Candidate Key, and Primary Key

Aspect	Super Key	Candidate Key	Primary Key
Definition	A set of attributes that uniquely identify rows.	A minimal super key.	A chosen candidate key.
Redundancy	May have extra (redundant) attributes.	No redundant attributes.	No redundant attributes.
Uniqueness	Always unique.	Always unique.	Always unique.
Example	`{EmployeeID, Name}`	`{EmployeeID}`	`{EmployeeID}`

Foreign Key vs. Primary Key

Aspect	Primary Key	Foreign Key
Purpose	Uniquely identifies rows in a table.	Establishes a relationship with another table.
Uniqueness	Must be unique for each row.	Can have duplicate values.
NULL Allowed	Not allowed.	Allowed (if relationship is optional).
Relationship	Exists within the same table.	References another table.