CSE532
Database Design: The Entity-Relationship Model

CSE 532, Theory of Database Systems

Stony Brook University

http://www.cs.stonybrook.edu/~cse532

Database Design

- Goal: specification of database schema
- Methodology:
  - Use *E-R model* to get a high-level graphical view of essential components of enterprise and how they are related
  - Convert E-R diagram to DDL
- *E-R Model*: enterprise is viewed as a set of
  - *Entities*
  - *Relationships* among entities

Entities

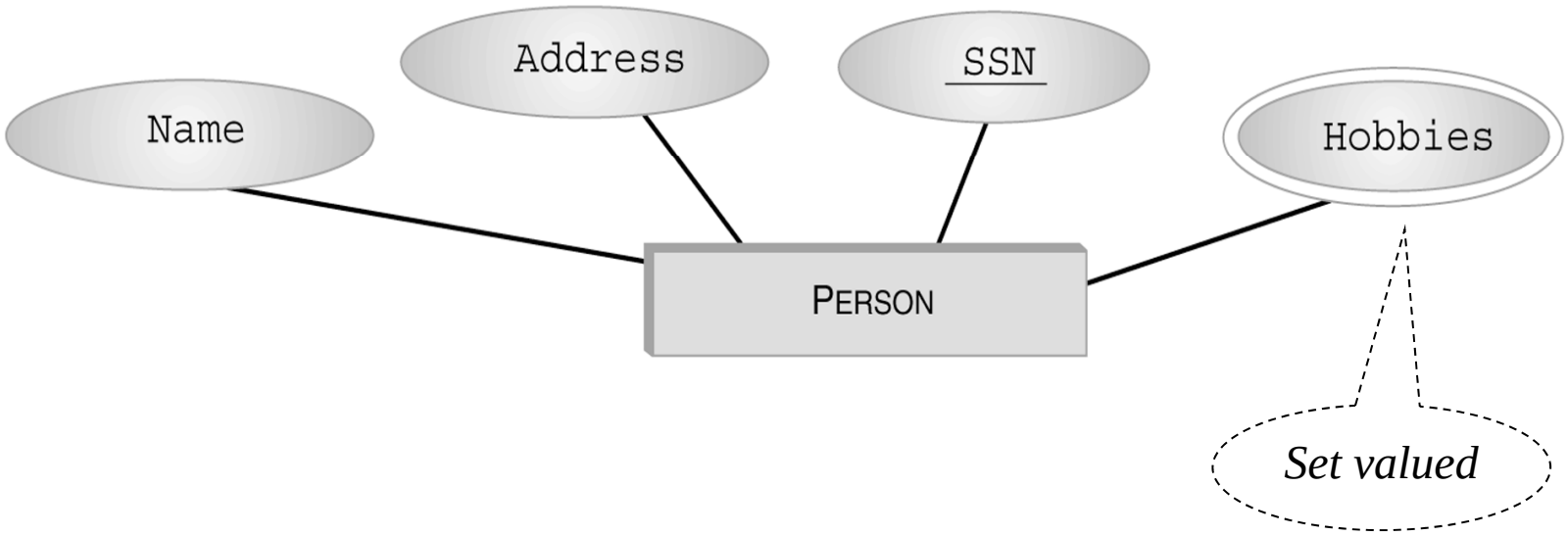
- *Entity*: an object that is involved in the enterprise
  - Ex: John, CSE305
- *Entity Type*: set of similar objects
  - Ex: students, courses
- *Attribute*: describes one aspect of an entity type
  - Ex: *name*, *maximum enrollment*

Entity Type

- Entity type described by set of attributes
  - Person: *Id*, *Name*, *Address*, *Hobbies*
- *Domain*: possible values of an attribute
  - Value can be a set (in contrast to relational model)
    - (111111, John, 123 Main St, {stamps, coins})
- *Key*: minimum set of attributes that uniquely identifies an entity (candidate key)
- *Entity Schema*: entity type name, attributes (and associated domain), key constraints

Entity Type (con't)

- Graphical Representation in E-R diagram:



Relationships

- *Relationship*: relates two or more entities
  - John ***majors in*** Computer Science
- *Relationship Type*: set of similar relationships
  - Student (entity type) related to Department (entity type) by MajorsIn (relationship type).
- Distinction:
  - *relation* (relational model) - set of tuples
  - *relationship* (E-R Model) – describes relationship between entities of an enterprise
- Both entity types and relationship types (E-R model) may be represented as relations (in the relational model)

Attributes and Roles

- *Attribute* of a relationship type describes the relationship
  - e.g., John majors in CS ***since* 2000**
    - John and CS are related
    - 2000 describes relationship - value of SINCE attribute of MajorsIn relationship type
- *Role* of a relationship type names one of the related entities
  - e.g., John is value of *Student* role, CS value of *Department* role of MajorsIn relationship type
  - (John, CS; 2000) describes a relationship

Relationship Type

- Described by set of roles and attributes
  - e.g., MajorsIn: *Student*, *Department*, *Since*
  - Here we have used as the role name (*Student*) the name of the entity type (Student) of the participant in the relationship, but there are problems (e.g., relationships can relate elements of same entity type)

Roles

- *Problem*: relationship can relate elements of same entity type
  - e.g., *ReportsTo* relationship type relates two elements of Employee entity type:
    - Bob reports to Mary since 2000
  - We do not have distinct names for the roles
  - It is not clear who reports to whom

Roles (con't)

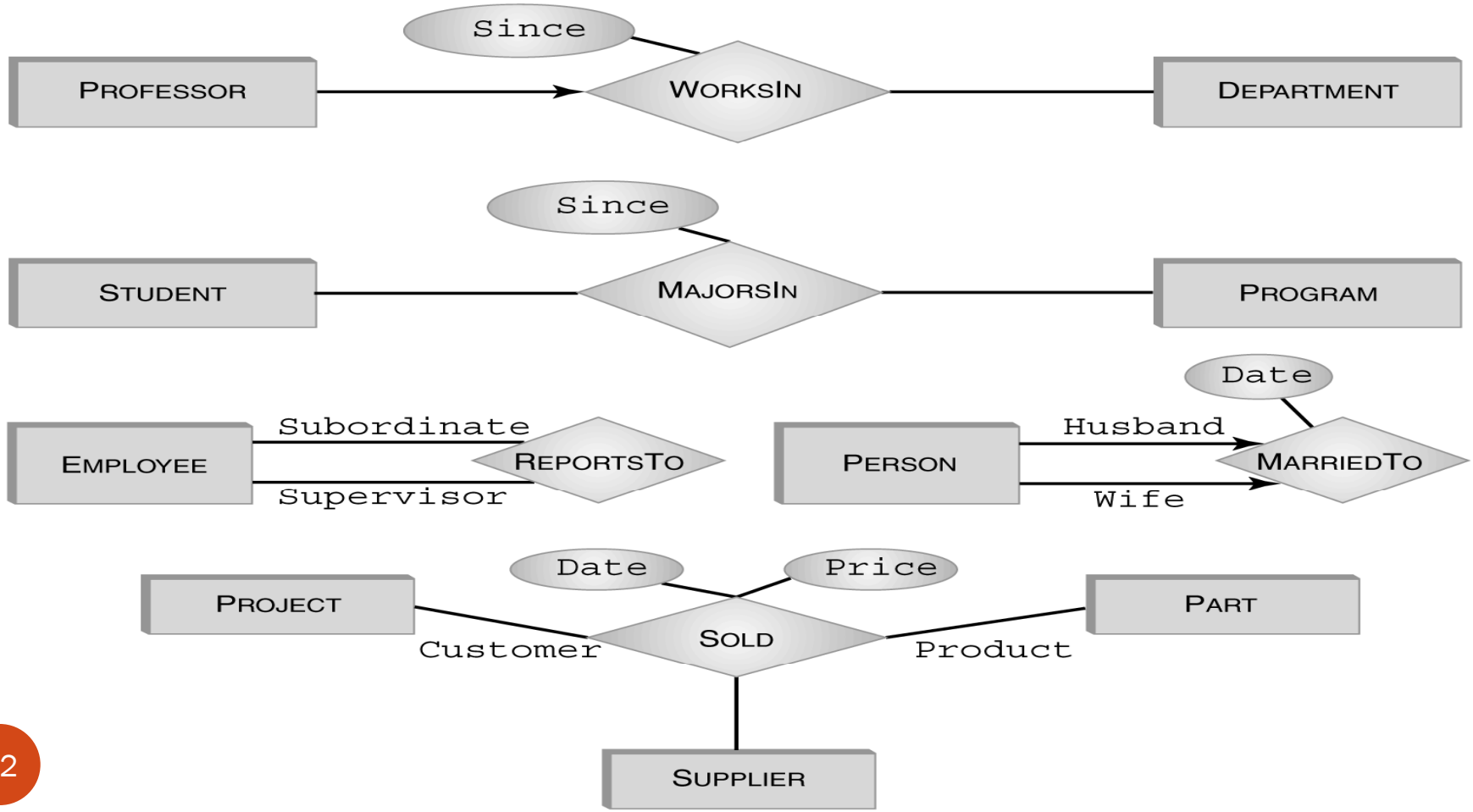
- *Solution*: role name of relationship type need not be same as name of entity type from which participants are drawn
  - ReportsTo has roles *Subordinate* and *Supervisor* and attribute *Since*
  - Values of *Subordinate* and *Supervisor* both drawn from entity type Employee

Schema of a Relationship Type

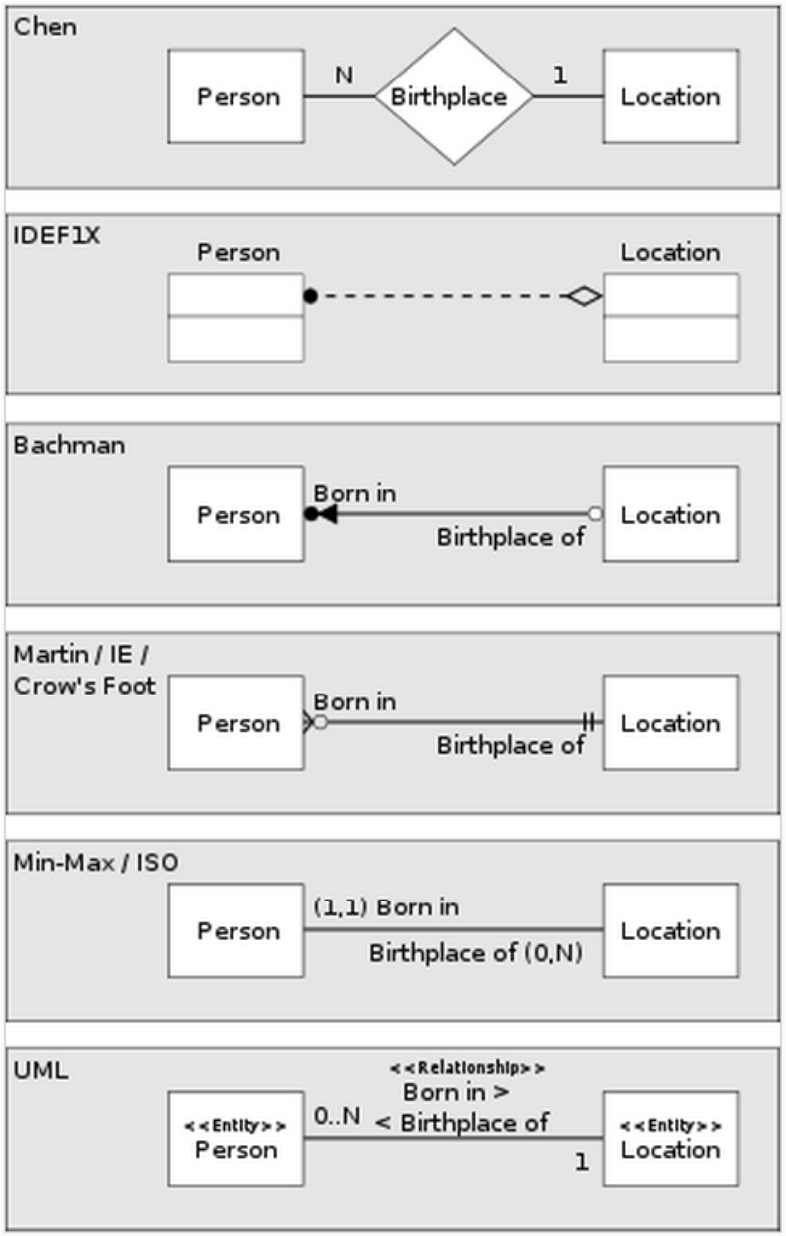
- *Role names*, R_i, and their corresponding entity sets. Roles must be single valued (number of roles = degree of relationship)
- *Attribute names*, A_j, and their corresponding domains (attributes may be set valued)
- *Key*: Minimum set of roles and attributes that uniquely identify a relationship
- *Relationship*: $<e_1, \ldots, e_n; a_1, \ldots, a_k>$
  - e_i is an entity, a value from R_i's entity set
  - a_j is a set of attribute values with elements from domain of A_j

Graphical Representation

- Roles are edges labeled with role names (omitted if role name = name of entity set). Most attributes have been omitted.

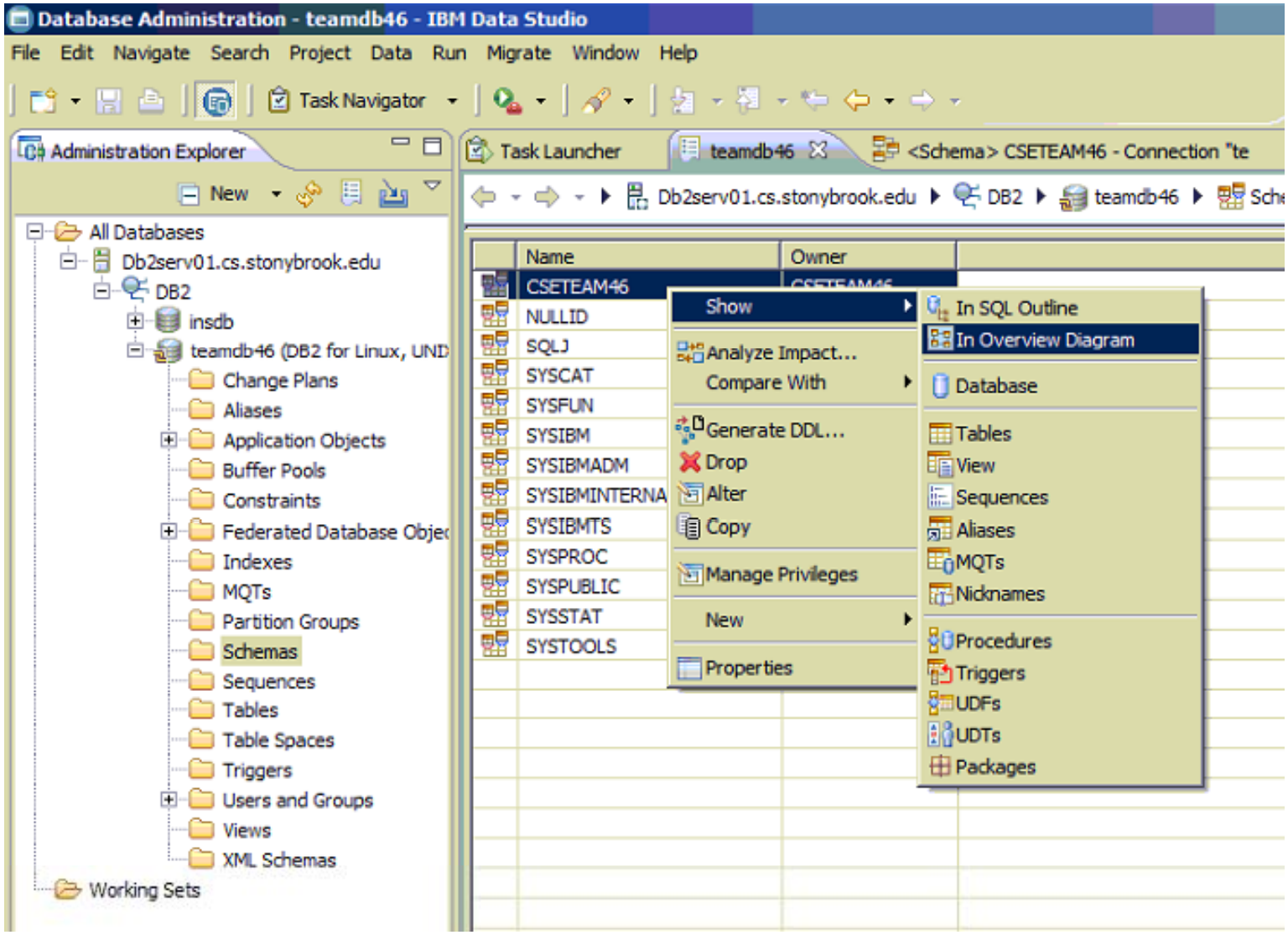


Multitude of graphical notations



Tools: IBM Data Studio, Microsoft Visio, Open ModelSphere, MySQL Workbench, RISE Editor, etc.

IBM Data Studio ER tool



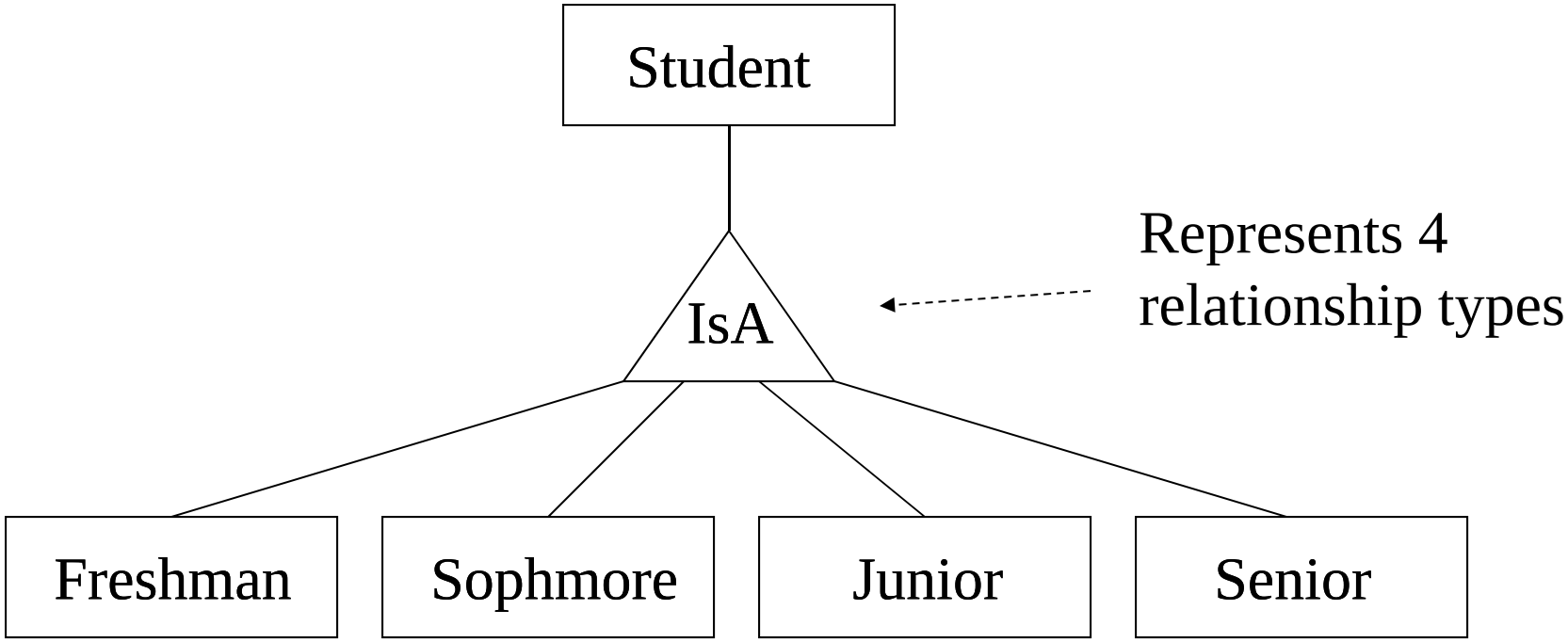
Single-role Key Constraint

- If, for a particular participant entity type, each entity participates in *at most* one relationship, corresponding role is a key of relationship type
  - E.g., *Professor* role is unique in WorksIn
- Representation in E-R diagram: arrow

Entity Type Hierarchies

- One entity type might be subtype of another
  - Freshman is a subtype of Student
- A relationship exists between a Freshman entity and the corresponding Student entity
  - e.g., Freshman John is related to Student John
- This relationship is called *IsA*
  - Freshman IsA Student
  - The two entities related by IsA are always descriptions of the same real-world object

IsA



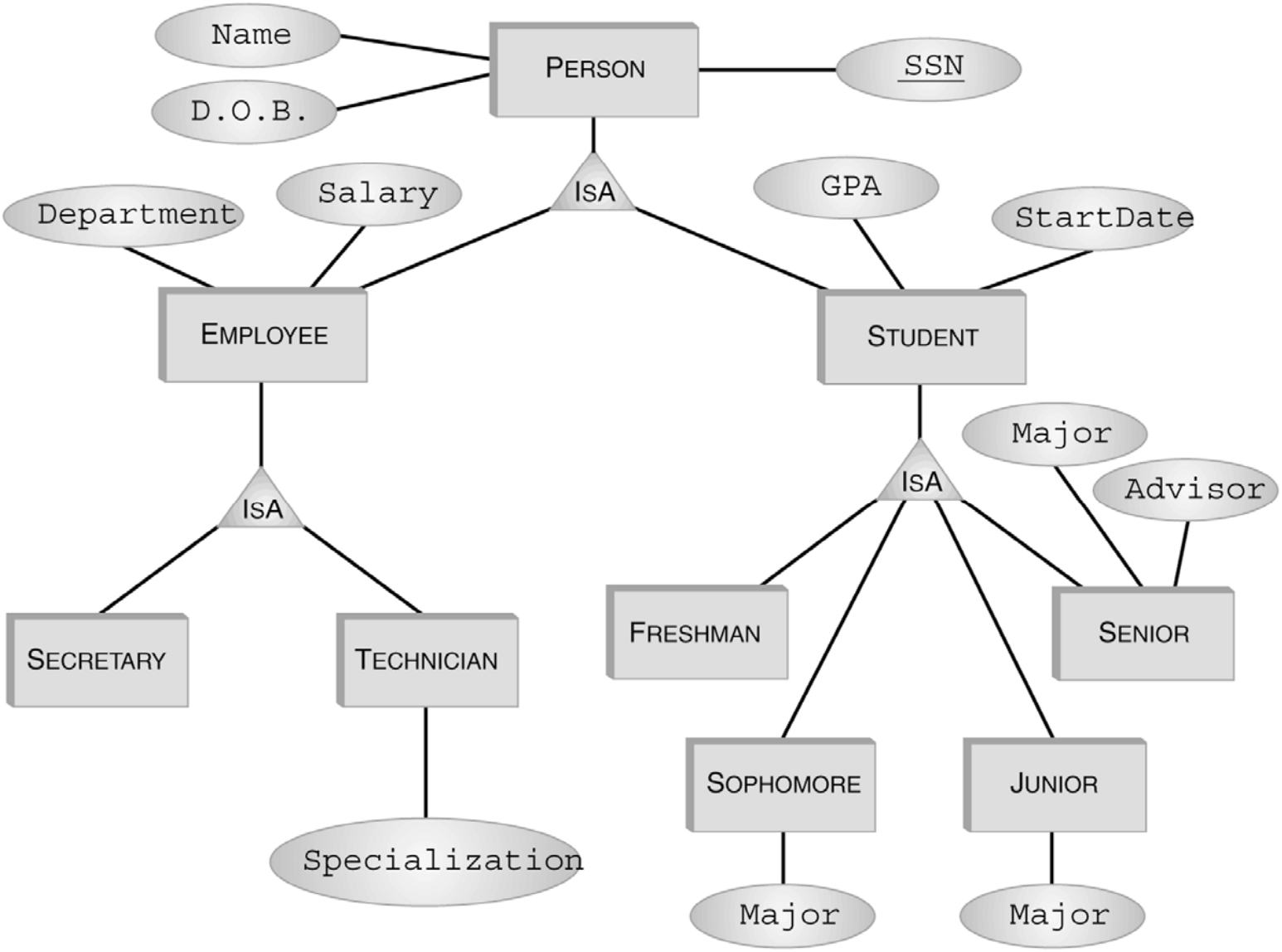
Properties of IsA

- *Inheritance* - Attributes of supertype apply to subtype.
  - E.g., *GPA* attribute of Student applies to Freshman
  - Subtype *inherits* all attributes of supertype.
  - Key of supertype is key of subtype
- *Transitivity* - Hierarchy of IsA
  - Student is subtype of Person, Freshman is subtype of Student, so Freshman is also a subtype of Student

Advantages of IsA

- Can create a more concise and readable E-R diagram
  - Attributes common to different entity sets need not be repeated
  - They can be grouped in one place as attributes of supertype
  - Attributes of (sibling) subtypes can be different

IsA Hierarchy - Example

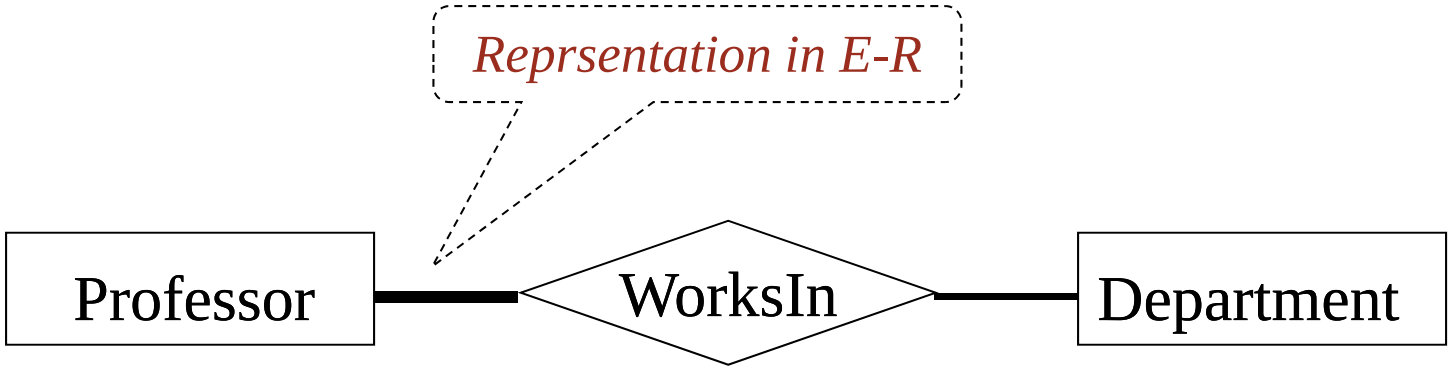


Constraints on Type Hierarchies

- Might have associated constraints:
  - *Covering constraint*: Union of subtype entities is equal to set of supertype entities
    - Employee is either a secretary or a technician (or both)
  - *Disjointness constraint*: Sets of subtype entities are disjoint from one another
    - Freshman, Sophomore, Junior, Senior are disjoint set

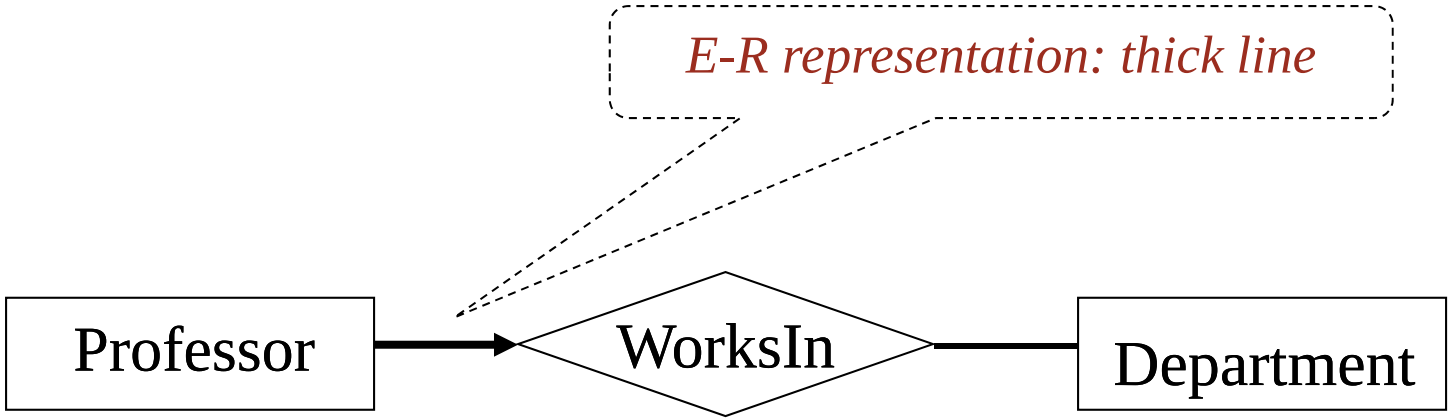
Participation Constraint

- If every entity participates in *at least* one relationship, a *participation constraint* holds:
  - A participation constraint of entity type E having role ρ in relationship type R states that for *e* in E there is an *r* in R such that $\rho(r) = e$.
  - e.g., every professor works in *at least* one department



Participation *and* Key Constraint

- If every entity participates in *exactly* one relationship, both a participation and a key constraint hold:
  - e.g., every professor works in *exactly one* department



Cardinality constraints

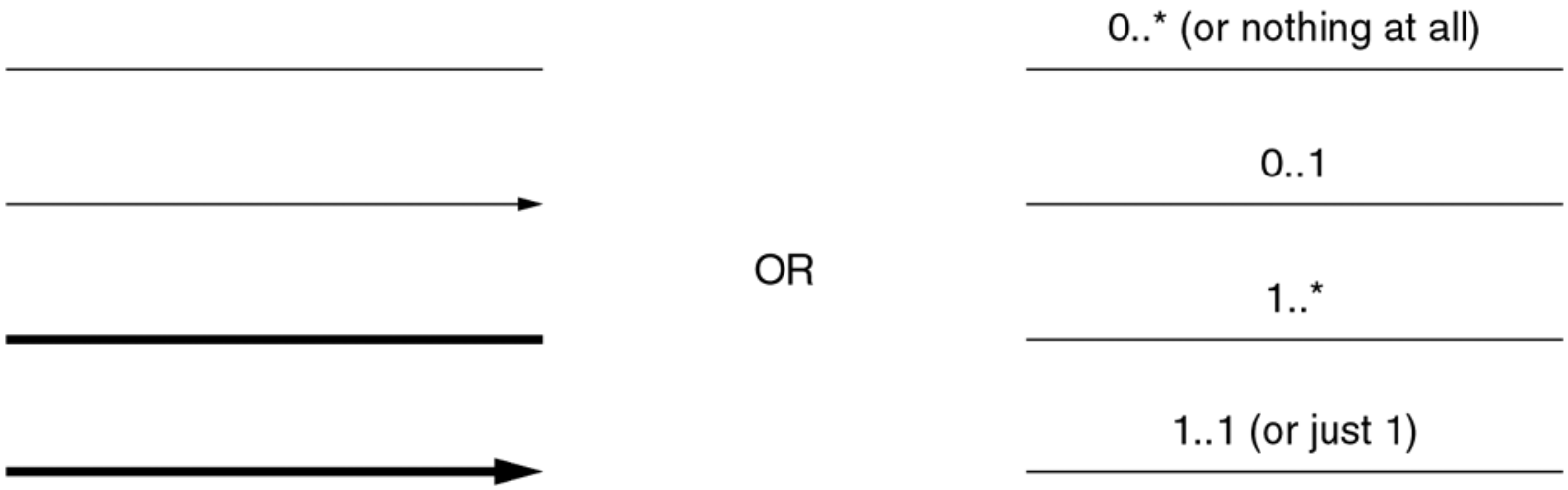


FIGURE 4.9 Line-based representation vs. cardinality constraints.

Non-exclusive part-of relationships

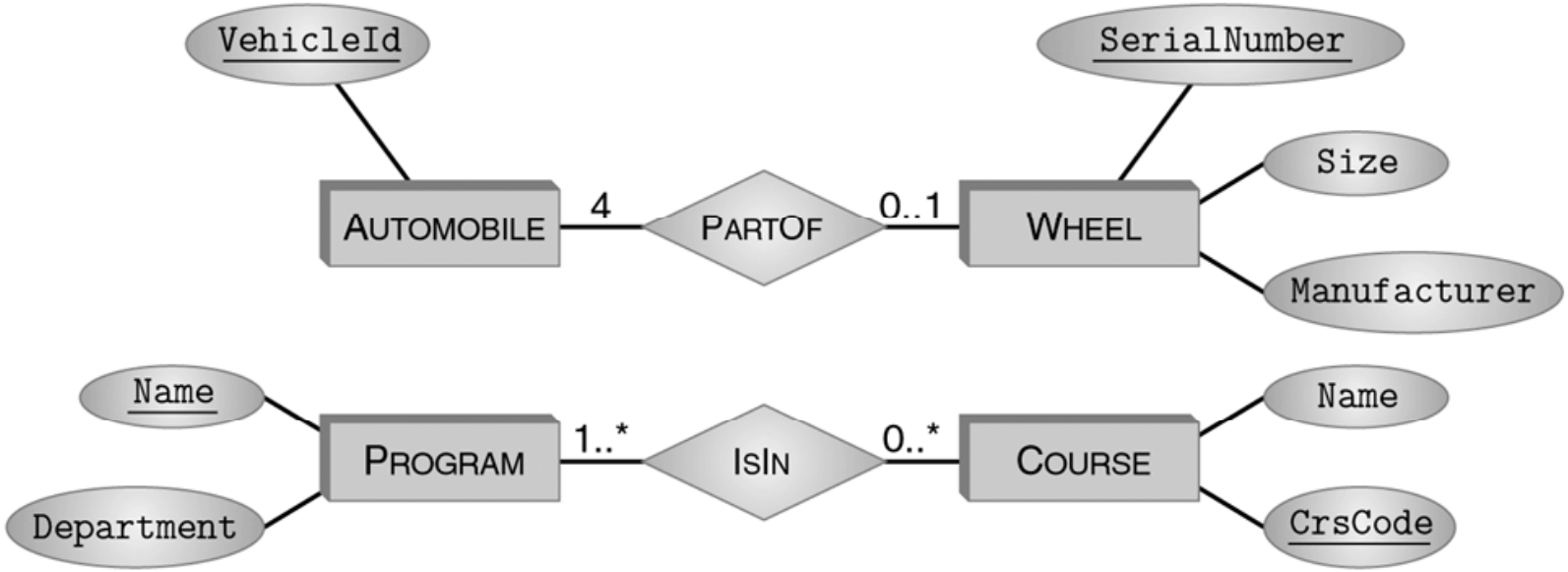


FIGURE 4.10 Non-exclusive part-of relationship in E-R.

Weak entities: exclusive part-of relationships

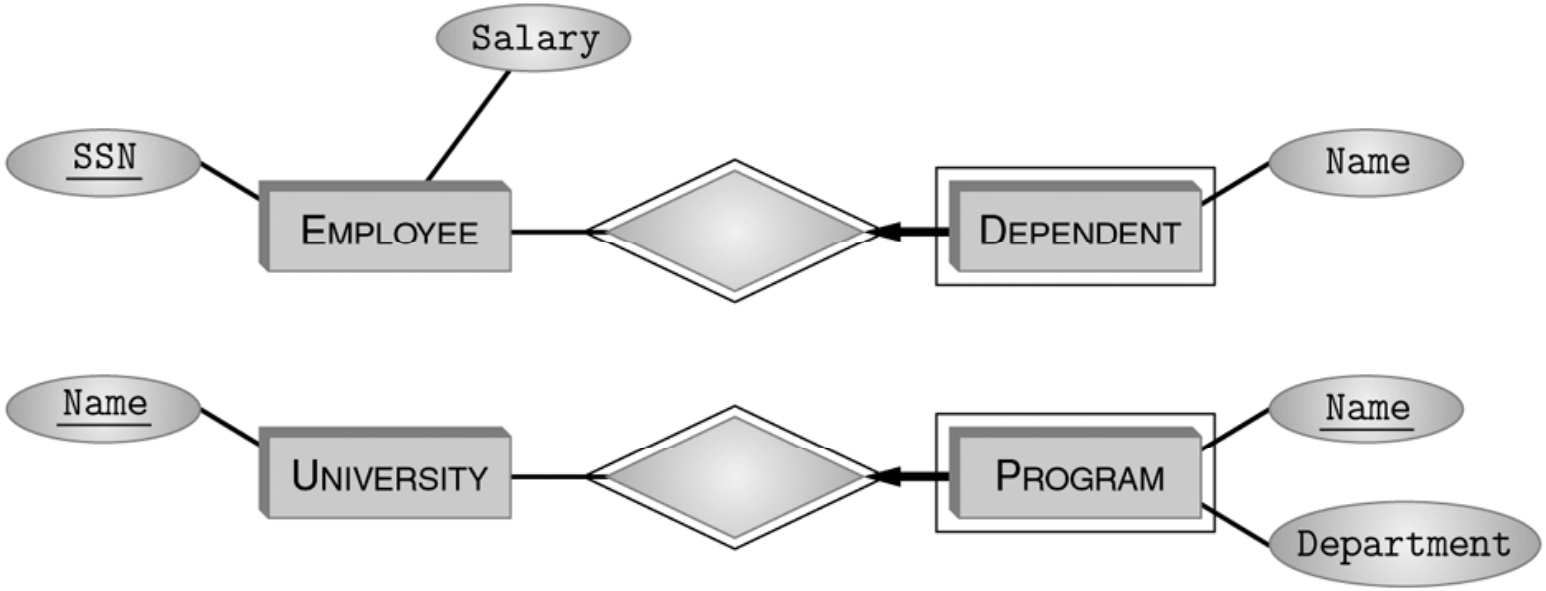


FIGURE 4.11 Exclusive part-of relationship in E-R: weak entities.

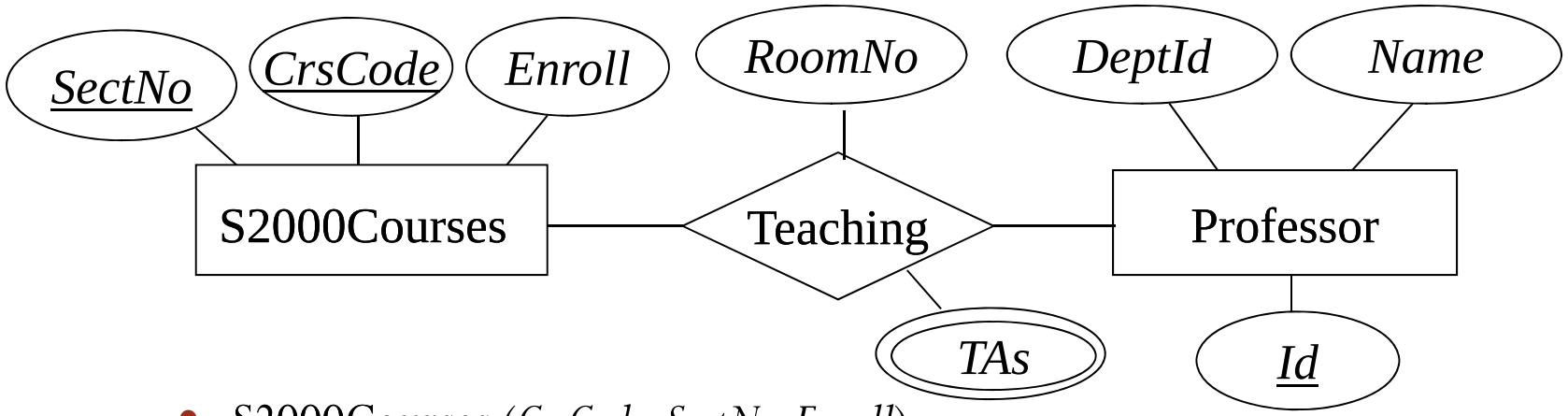
Representation of Entity Types in the Relational Model

- An entity type corresponds to a relation
- Relation's attributes = entity type's attributes
  - *Problem*: entity type can have set valued attributes, e.g.,

    Person: *Id*, *Name*, *Address*, *Hobbies*
  - *Solution*: Use several rows to represent a single entity
    - (111111, John, 123 Main St, stamps)
    - (111111, John, 123 Main St, coins)
  - Problems with this solution:
    - Redundancy
    - Key of entity type (Id) not key of relation
    - Hence, the resulting relation must be further transformed (Chapter 6)

Representation of Relationship Types in the Relational Model

- Typically, a relationship becomes a relation in the relational model
- Attributes of the corresponding relation are
  - Attributes of relationship type
  - For each role, the primary key of the entity type associated with that role
- *Example*:



  - S2000Courses (*CrsCode*, *SectNo*, *Enroll*)
  - Professor (*Id*, *DeptId*, *Name*)
  - Teaching (*CrsCode*, *SecNo*, *Id*, *RoomNo*, *TAs*)

Representation of Relationship Types in the Relational Model

- Candidate key of corresponding table = candidate key of relation
  - Except when there are set valued attributes
  - Example: Teaching (*CrsCode, SectNo, Id, RoomNo, TAs*)
    - Key of relationship type = (*CrsCode, SectNo*)
    - Key of relation = (*CrsCode, SectNo, TAs*)

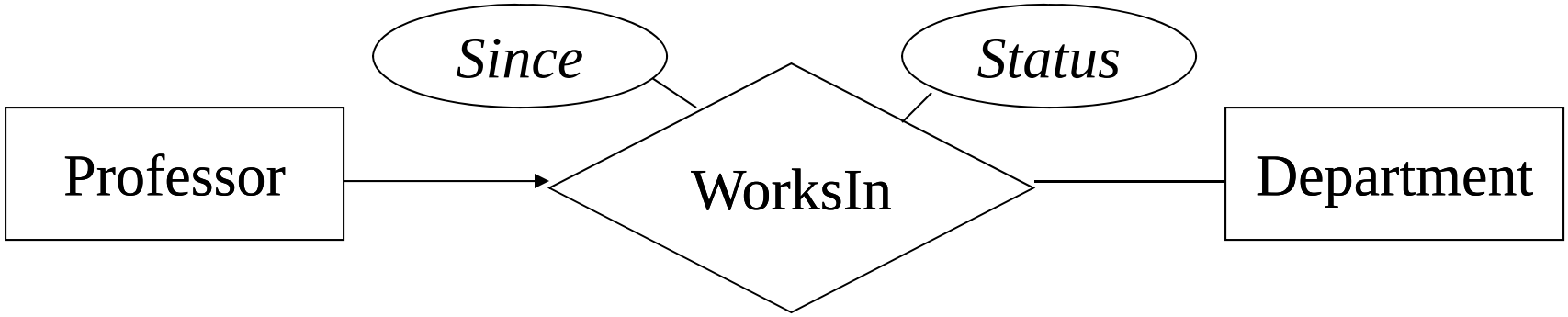
Set valued

CrsCode	*SectNo*	*Id*	*RoomNo*	*TAs*
CSE305	1	1234	Hum 22	Joe
CSE305	1	1234	Hum 22	Mary

Representation in SQL

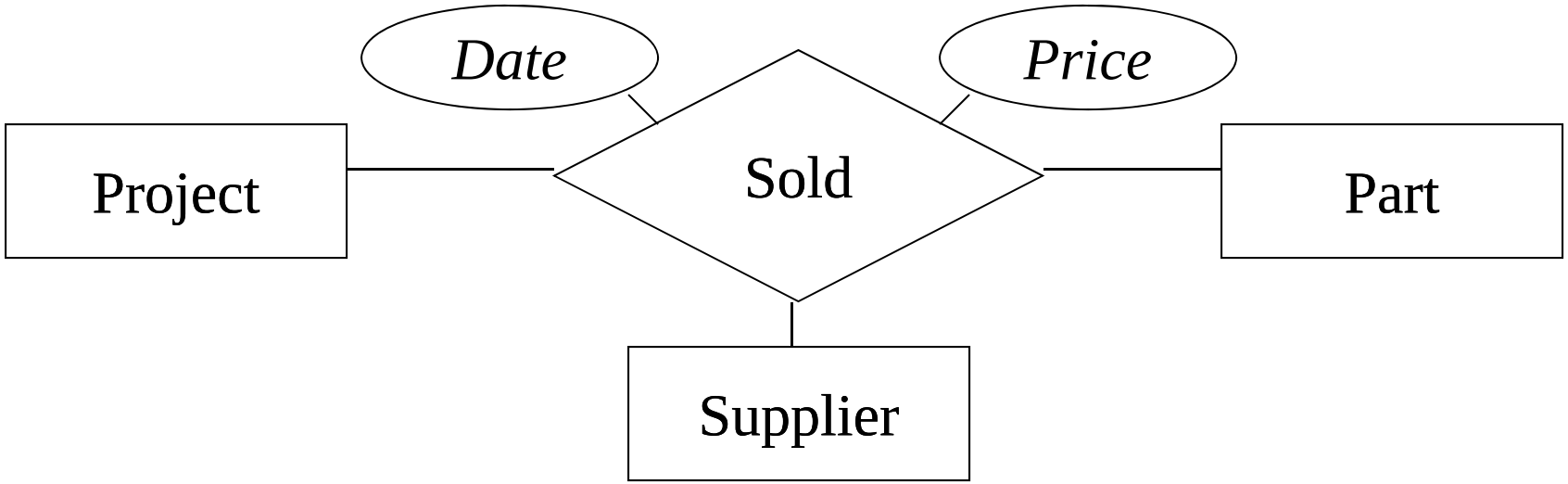
- Each role of relationship type produces a **foreign key** in corresponding relation
  - Foreign key references table corresponding to entity type from which role values are drawn

Example 1



CREATE TABLE WorksIn (
Since DATE, -- *attribute*
Status CHAR (10), -- *attribute*
ProfId INTEGER, -- *role (key of* Professor*)*
DeptId CHAR (4), -- *role (key of* Department*)*
PRIMARY KEY (*ProfId*), -- *since a professor works in at most one department*
FOREIGN KEY (*ProfId*) REFERENCES Professor (*Id*),
FOREIGN KEY (*DeptId*) REFERENCES Department)

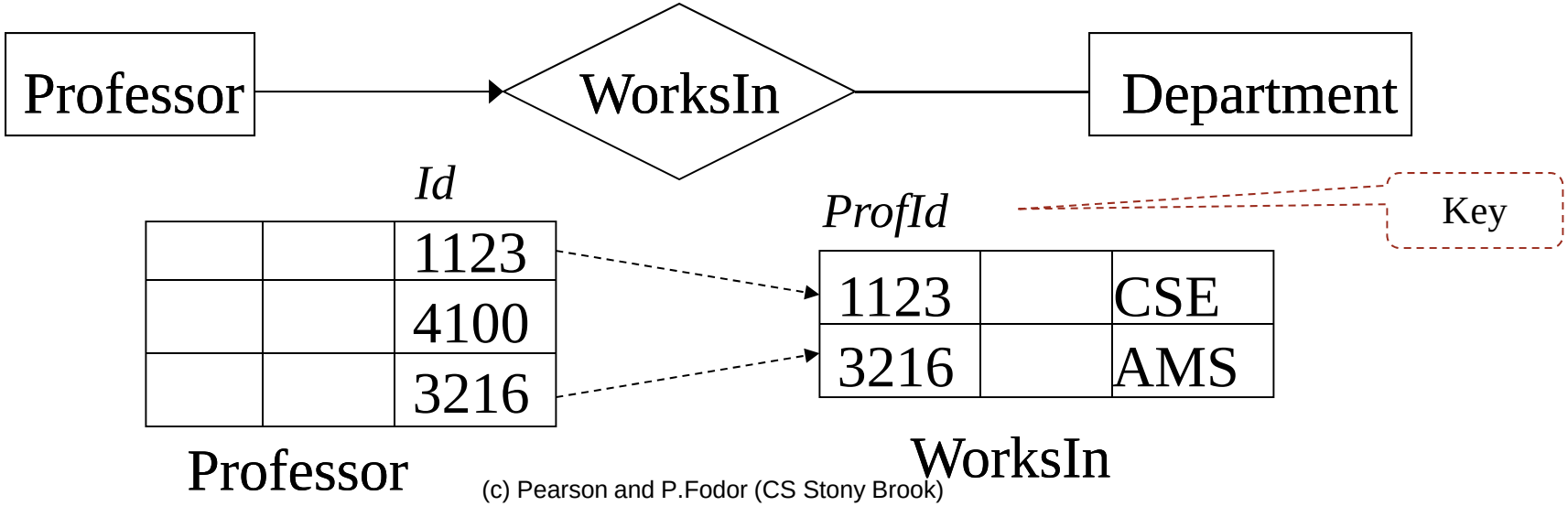
Example 2




```
CREATE TABLE  Sold  (
   Price  INTEGER,                -- attribute
   Date  DATE,                    -- attribute
   ProjId  INTEGER,               -- role
   SupplierId  INTEGER,           -- role
   PartNumber  INTEGER,           -- role
   PRIMARY KEY (ProjId, SupplierId, PartNumber, Date),                    ????
   FOREIGN KEY (ProjId) REFERENCES Project,
   FOREIGN KEY (SupplierId) REFERENCES Supplier (Id),
   FOREIGN KEY (PartNumber) REFERENCES Part (Number)  )
```

Representation of Single Role Key Constraints in the Relational Model

- *Relational model representation*: key of the relation corresponding to the entity type is key of the relation corresponding to the relationship type
  - *Id* is primary key of Professor; *ProfId* is key of WorksIn. Professor 4100 does not participate in the relationship.
  - Cannot use foreign key in Professor to refer to WorksIn since some professors may not work in any dept. (But *ProfId* is a foreign key in WorksIn that refers to Professor.)

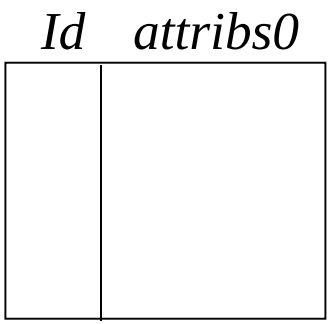


Representing Type Hierarchies in the Relational Model

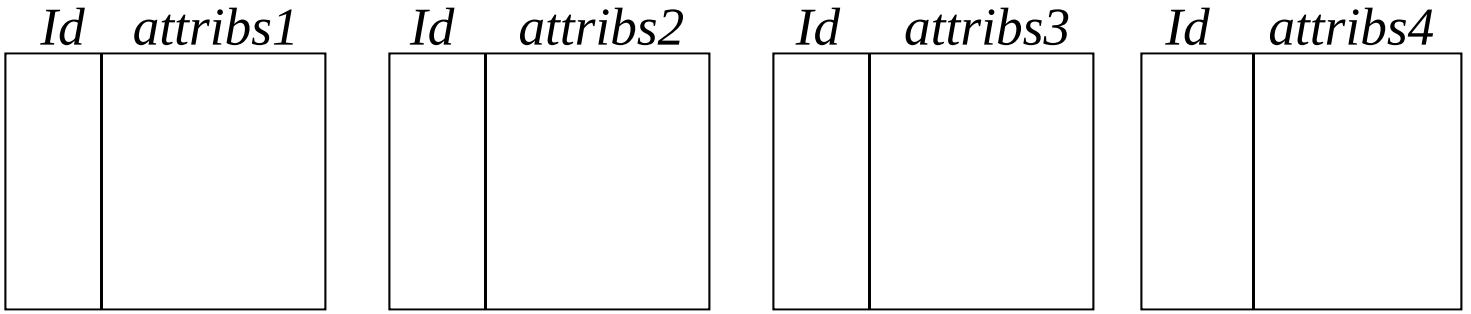
- Supertypes and subtypes can be realized as separate relations
  - Need a way of identifying subtype entity with its (unique) related supertype entity
    - *Choose a candidate key and make it an attribute of all entity types in hierarchy*

Type Hierarchies and the Relational Model

Translated by adding the primary key of supertype to all subtypes. Plus foreign key from subtypes to the supertype.



Student



Freshman Sophmore Junior Senior

FOREIGN KEY *Id* REFERENCES Student

in Freshman, Sophomore, Sunior, Senior

Type Hierarchies and the Relational Model

- Redundancy eliminated if IsA is not disjoint
  - For individuals who are both employees and students, Name and DOB are stored only once

Person

SSN	*Name*	*DOB*
1234	Mary	1950

Employee

SSN	*Department*	*Salary*
1234	Accounting	35000

Student

SSN	*GPA*	*StartDate*
1234	3.5	1997

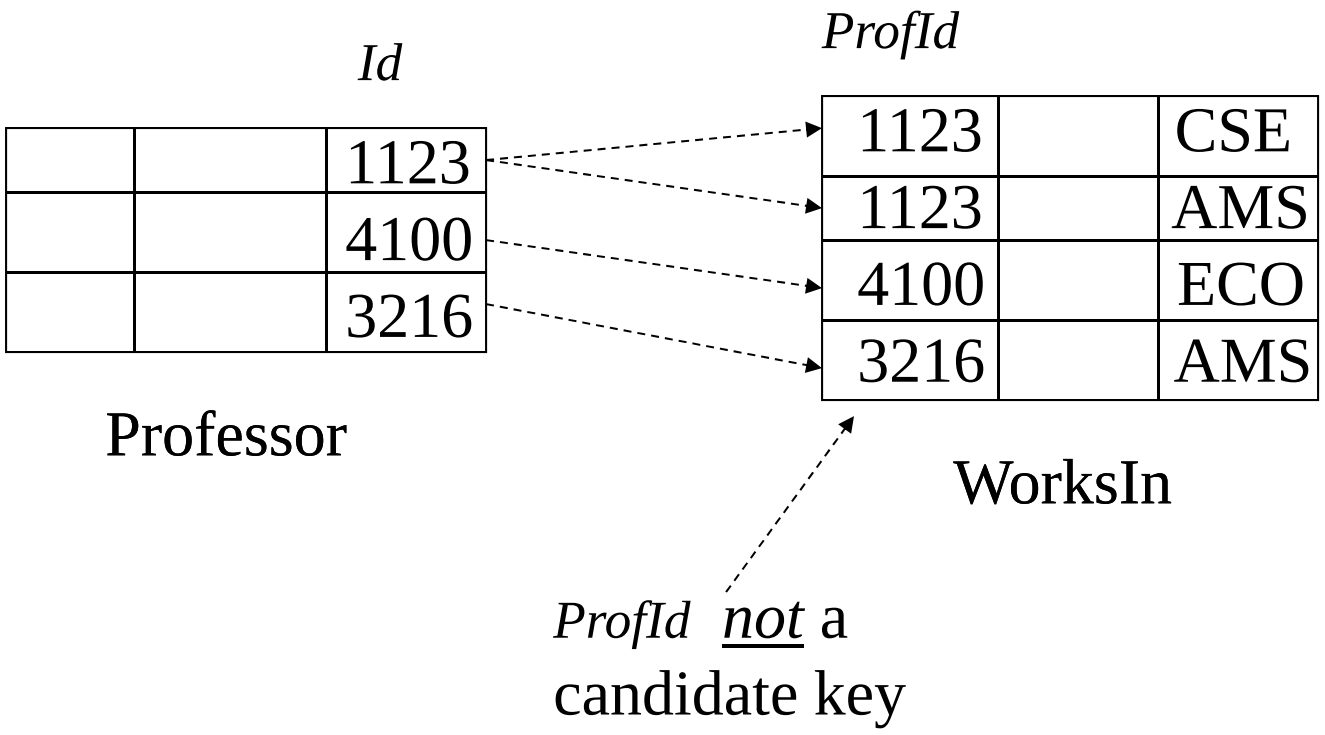
Representing Participation Constraints in the Relational Model

- *Inclusion dependency*: Every professor works in *at least* one dep't.
  - in the relational model: (easy)
    - Professor (*Id*) references WorksIn (*ProfId*)
  - in SQL:
    - Simple case: *If ProfId is a key in* **WorksIn** (i.e., every professor works in *exactly one* department) then it is easy:
      - FOREIGN KEY *Id* REFERENCES WorksIn (*ProfId*)
    - General case – *ProfId* is *not* a key in **WorksIn**, so can't use foreign key constraint (not so easy):

```
CREATE ASSERTION  ProfsInDepts
   CHECK ( NOT EXISTS (
      SELECT  *  FROM  Professor P
      WHERE NOT EXISTS (
         SELECT  *  FROM  WorksIn W
         WHERE   P.Id = W.ProfId ) ) )
```

Representing Participation Constraints in the Relational Model

- Example (can't use foreign key in Professor if ProfId is not a candidate key in WorksIn)



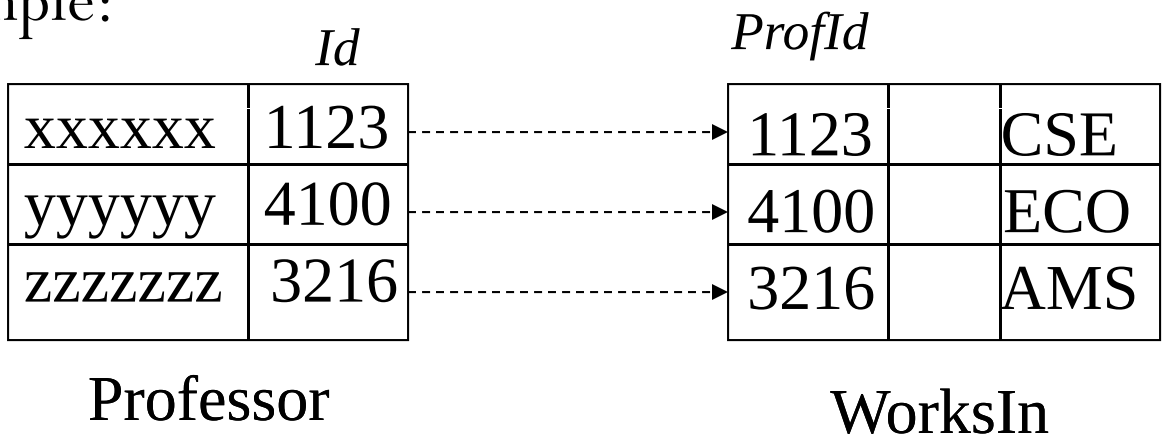
Representing Participation *and* Key Constraint in SQL

- If both participation and key constraints apply, use foreign key constraint in entity table (but beware: if candidate key in entity table is not primary, presence of nulls violates participation constraint).

```
CREATE TABLE Professor  (
   Id   INTEGER,
        ……
   PRIMARY KEY (Id),      -- Id can't be null
   FOREIGN KEY (Id) REFERENCES WorksIn  (ProfId)
                                --all professors participate
)
```

Participation and Key Constraint in the Relational Model

- Example:



Participation and Key Constraint in the Relational Model

- Alternative solution if both key and participation constraints apply: merge the tables representing the entity and relationship sets
  - Since there is a 1-1 and onto relationship between the rows of the entity set and the relationship sets, might as well put all the attributes in one table

Participation and Key Constraint in the Relational Model

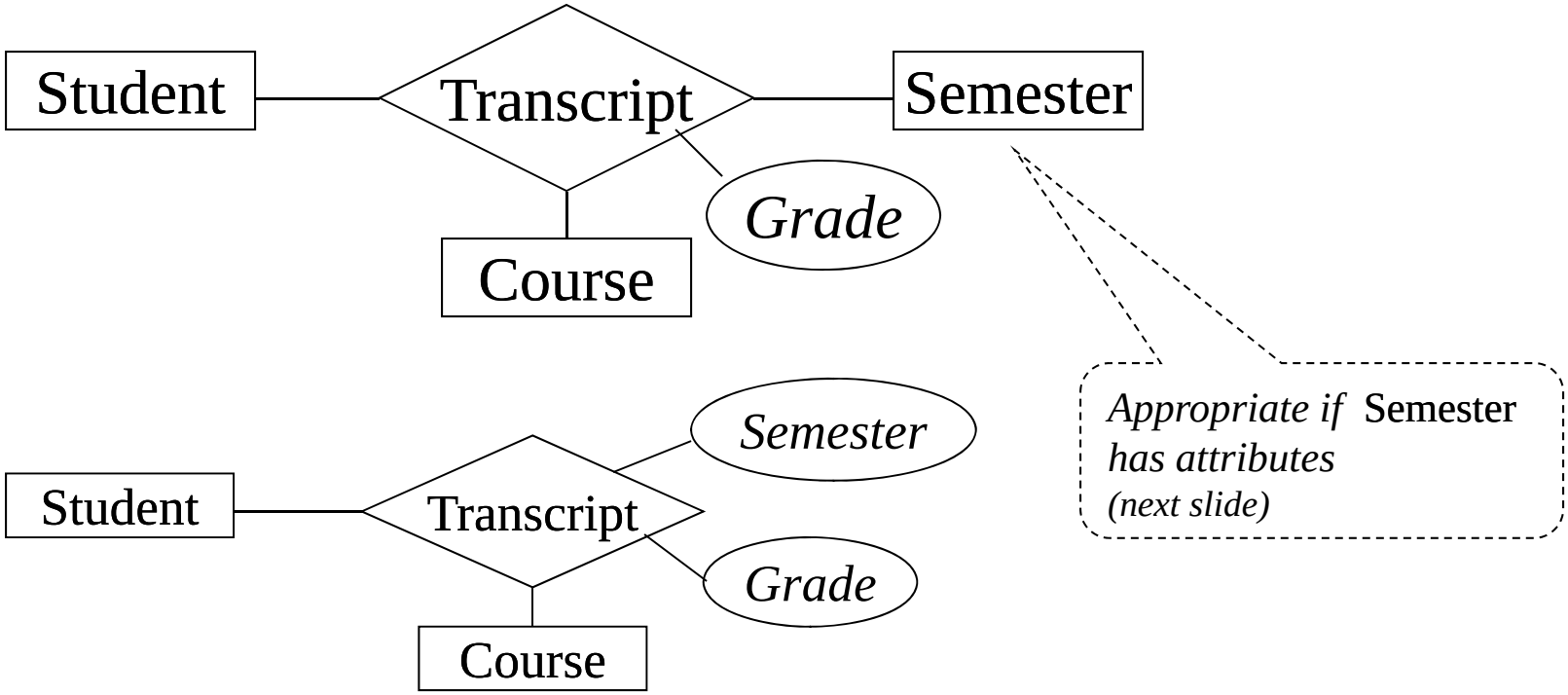
- Example

Name	*Id*		*DeptId*
xxxxxxx	1123		CSE
yyyyyyy	4100		ECO
zzzzzzzz	3216		AMS

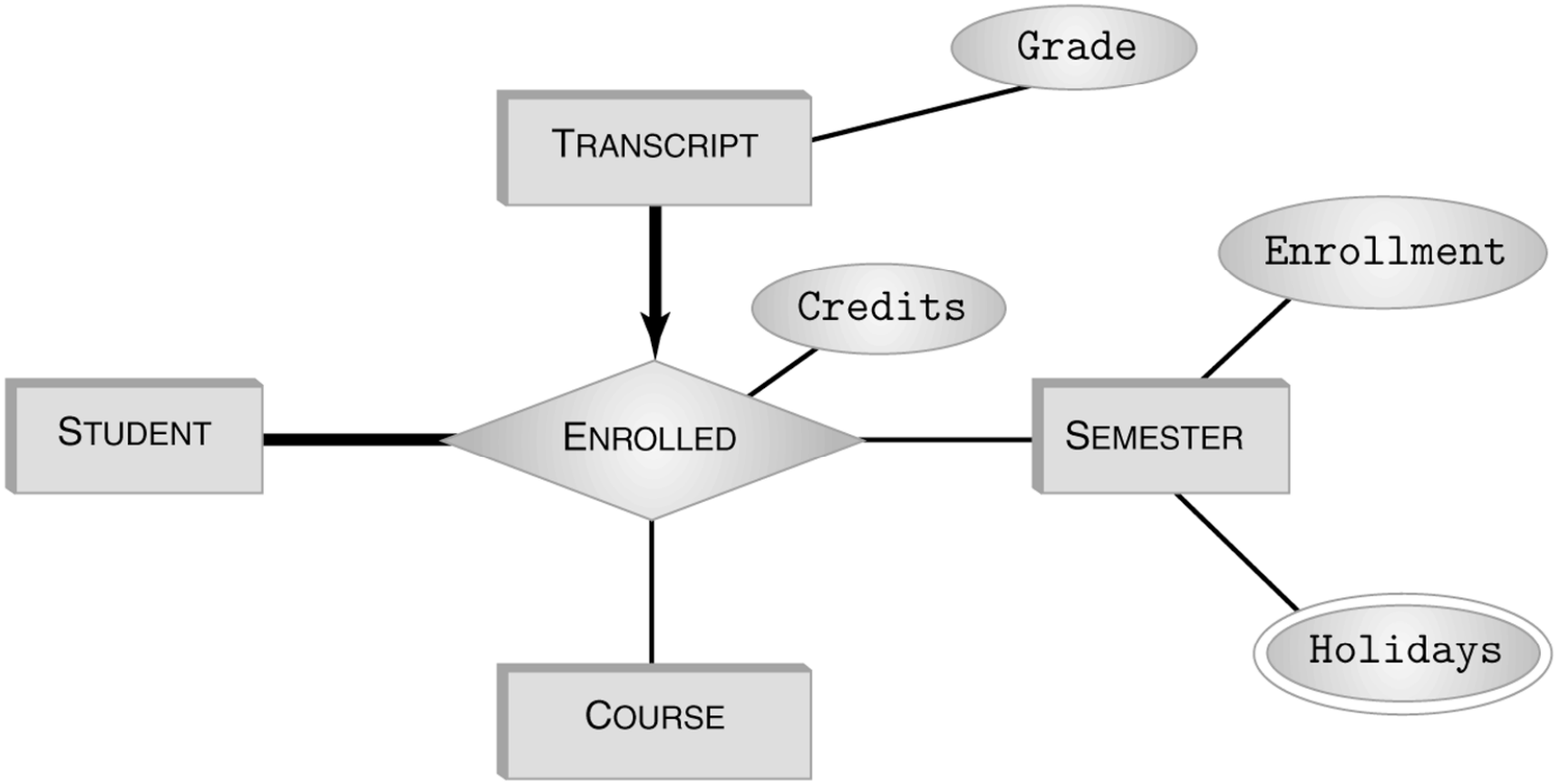
Prof_WorksIn

Entity or Attribute?

- Sometimes information can be represented as either an entity or an attribute.

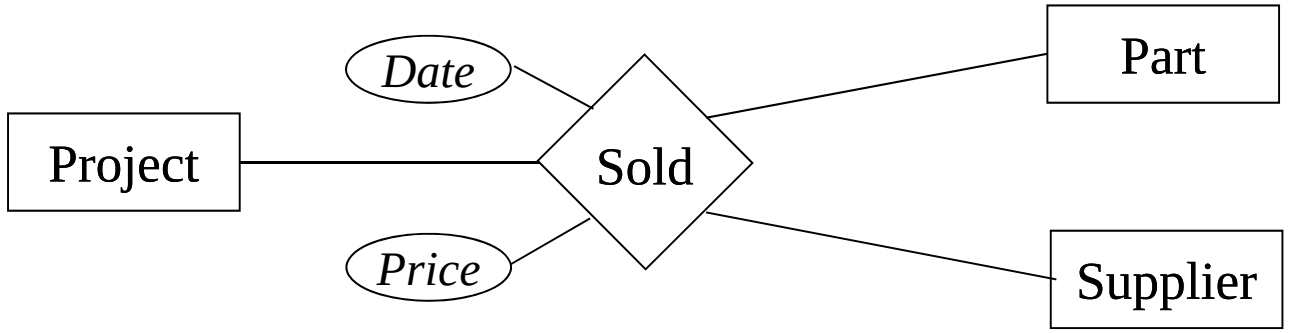


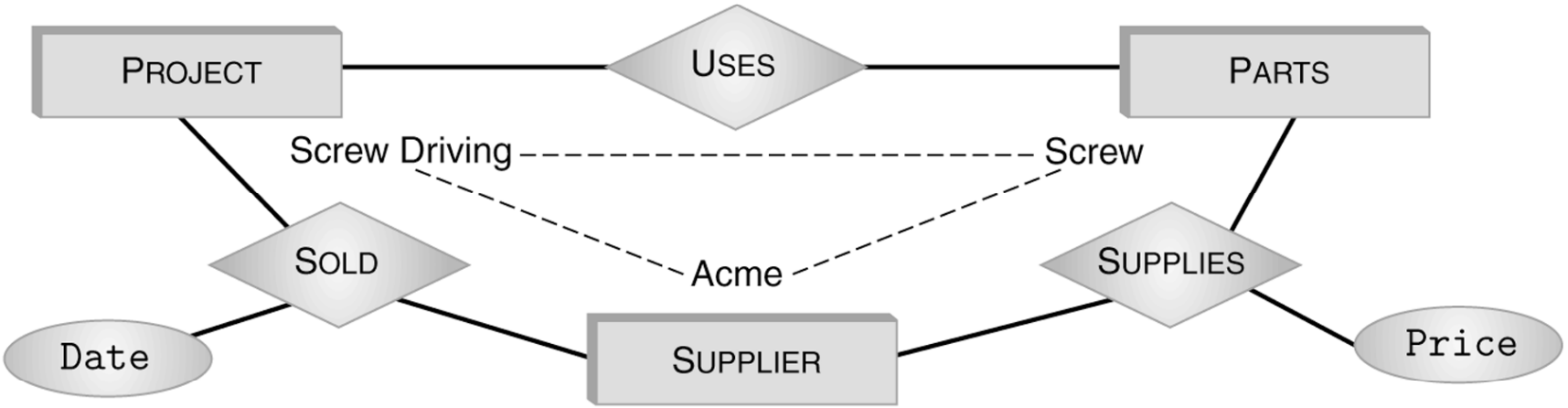
Entity or Relationship?



(Non-) Equivalence of Diagrams

- Transformations between binary and ternary relationships.





ER Diagram example

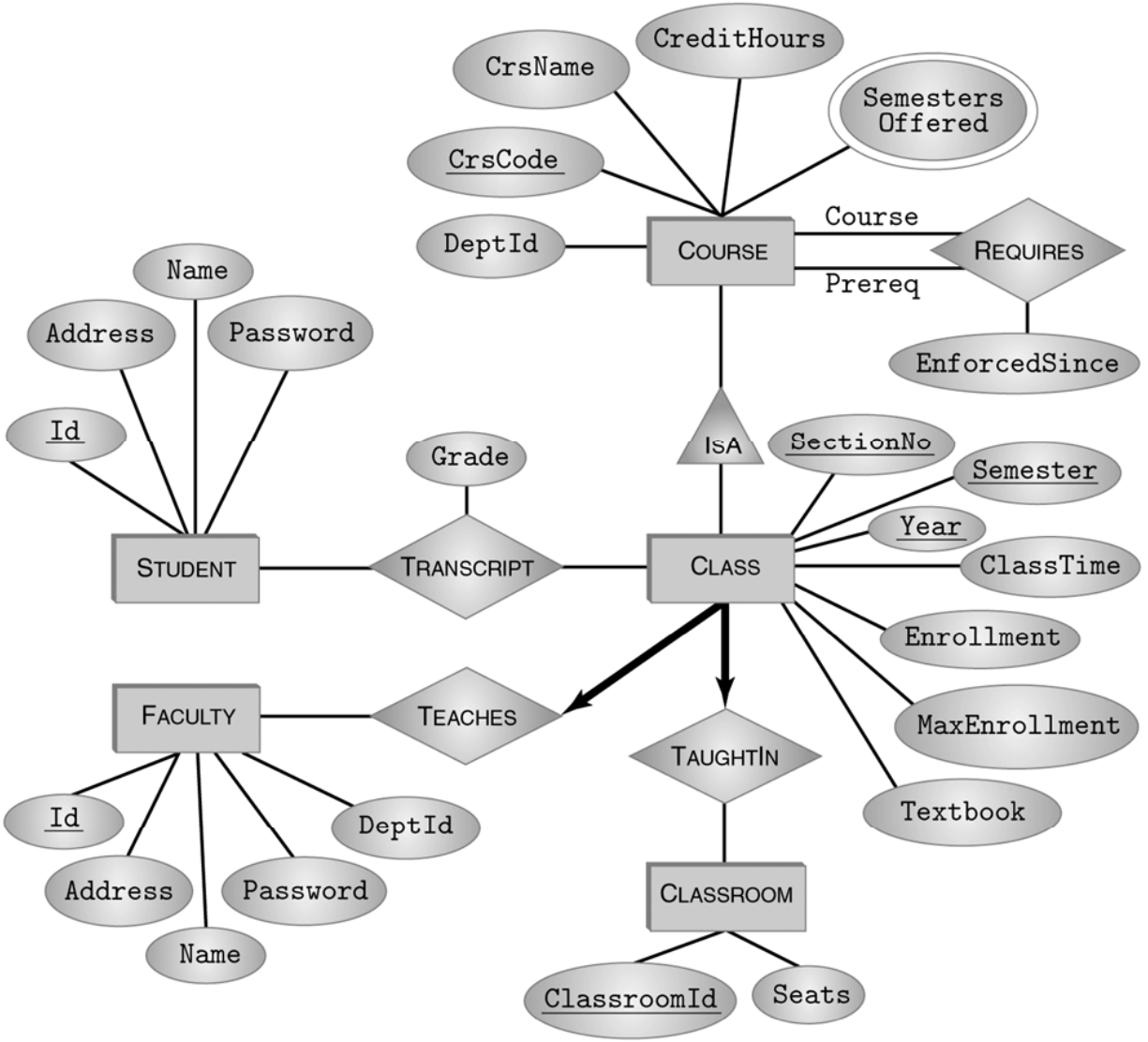


FIGURE 4.33 An E-R diagram for the Student Registration System.

FIGURE 4.34 A schema for the Student Registration System—Part 1.

```
CREATE TABLE STUDENT  (
     Id             CHAR(9),
     Name           CHAR(20) NOT NULL,
     Password       CHAR(10) NOT NULL,
     Address        CHAR(50),
     PRIMARY KEY (Id) )

CREATE TABLE FACULTY  (
     Id             CHAR(9),
     Name           CHAR(20) NOT NULL,
     DeptId         CHAR(4) NOT NULL,
     Password       CHAR(10) NOT NULL,
     Address        CHAR(50),
     PRIMARY KEY (Id) )

CREATE TABLE COURSE (
     CrsCode        CHAR(6),
     DeptId         CHAR(4) NOT NULL,
     CrsName        CHAR(20) NOT NULL,
     CreditHours    INTEGER NOT NULL,
     PRIMARY KEY (CrsCode),
     UNIQUE (DeptId, CrsName) )

CREATE TABLE WHENOFFERED (
     CrsCode        CHAR(6),
     Semester       CHAR(6),
     PRIMARY KEY (CrsCode, Semester),
     CHECK (Semester IN ('Spring','Fall') ) )

CREATE TABLE CLASSROOM  (
     ClassroomId    CHAR(3),
     Seats          INTEGER NOT NULL,
     PRIMARY KEY (ClassroomId) )
```

FIGURE 4.35 A schema for the Student Registration System—Part 2.

```
CREATE TABLE  REQUIRES  (
     CrsCode        CHAR(6),
     PrereqCrsCode  CHAR(6),
     EnforcedSince  DATE      NOT NULL,
     PRIMARY KEY (CrsCode, PrereqCrsCode),
     FOREIGN KEY (CrsCode) REFERENCES COURSE(CrsCode),
     FOREIGN KEY (PrereqCrsCode) REFERENCES COURSE(CrsCode)  )

CREATE TABLE  CLASS  (
     CrsCode        CHAR(6),
     SectionNo      INTEGER,
     Semester       CHAR(6),
     Year           INTEGER,
     Textbook       CHAR(50),
     ClassTime      CHAR(5),
     Enrollment     INTEGER,
     MaxEnrollment  INTEGER,
     ClassroomId    CHAR(3),        -- from TAUGHTIN
     InstructorId   CHAR(9),        -- from TEACHES
     PRIMARY KEY (CrsCode,SectionNo,Semester,Year),
     CONSTRAINT TIMECONFLICT
          UNIQUE (InstructorId,Semester,Year,ClassTime),
     CONSTRAINT CLASSROOMCONFLICT
          UNIQUE (ClassroomId,Semester,Year,ClassTime),
     CONSTRAINT ENROLLMENT
          CHECK (Enrollment <= MaxEnrollment AND Enrollment >= 0),
     FOREIGN KEY (CrsCode) REFERENCES COURSE(CrsCode),
     FOREIGN KEY (ClassroomId) REFERENCES CLASSROOM(ClassroomId),
     FOREIGN KEY (CrsCode, Semester)
          REFERENCES WHENOFFERED(CrsCode, Semester),
     FOREIGN KEY (InstructorId) REFERENCES FACULTY(Id)  )
```

```
CREATE TABLE  TRANSCRIPT  (
     StudId          CHAR(9),
     CrsCode         CHAR(6),
     SectionNo       INTEGER,
     Semester        CHAR(6),
     Year            INTEGER,
     Grade           CHAR(1),
     PRIMARY KEY (StudId,CrsCode,SectionNo,Semester,Year),
     FOREIGN KEY (StudId) REFERENCES STUDENT(Id),
     FOREIGN KEY (CrsCode,SectionNo,Semester,Year)
          REFERENCES CLASS(CrsCode,SectionNo,Semester,Year),
     CHECK (Grade IN ('A','B','C','D','F','I') ),
     CHECK (Semester IN ('Spring','Fall') )  )
```

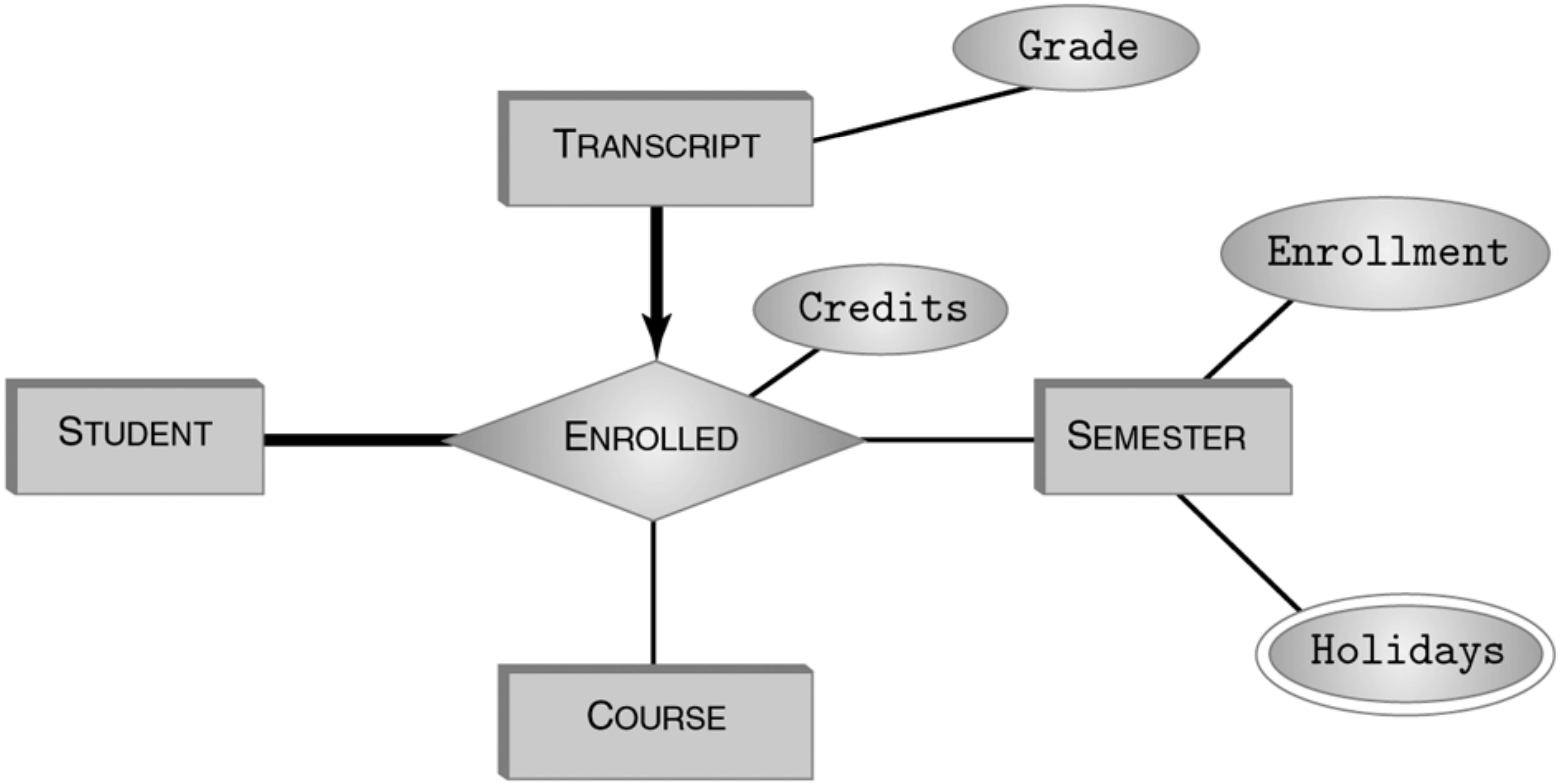
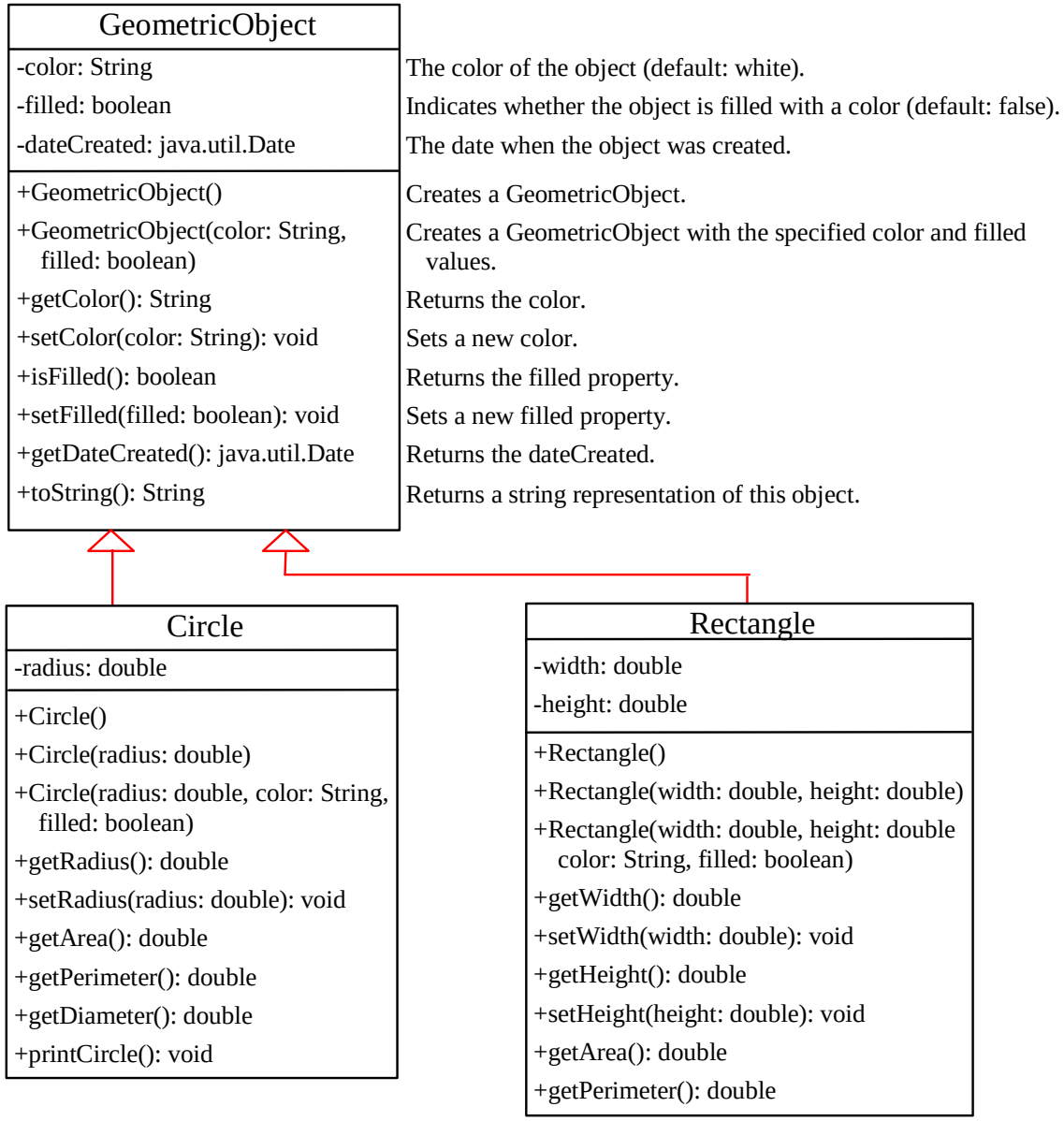


FIGURE 4.36 An alternative representation of the transcript information.

Unified Modeling Language (UML)

- UML unifies a number of methodologies in software engineering, business modeling and management, database design, and others.
- UML is gaining in popularity in many areas of design, including database design and programming languages.
  - UML class diagrams are a subset of UML that is suitable for conceptual modeling of databases.

UML: Superclasses and Subclasses



UML Class Diagram

- **Visibility:**
  - + Public
  - - Private
  - # Protected
  - ~ Package
  - underline Static
- **Generalization Relationship**
- **Instance Level Relationships**
  - **Association**
  - **Composition**
  - **Aggregation**
    - if the container is destroyed, its contents are not

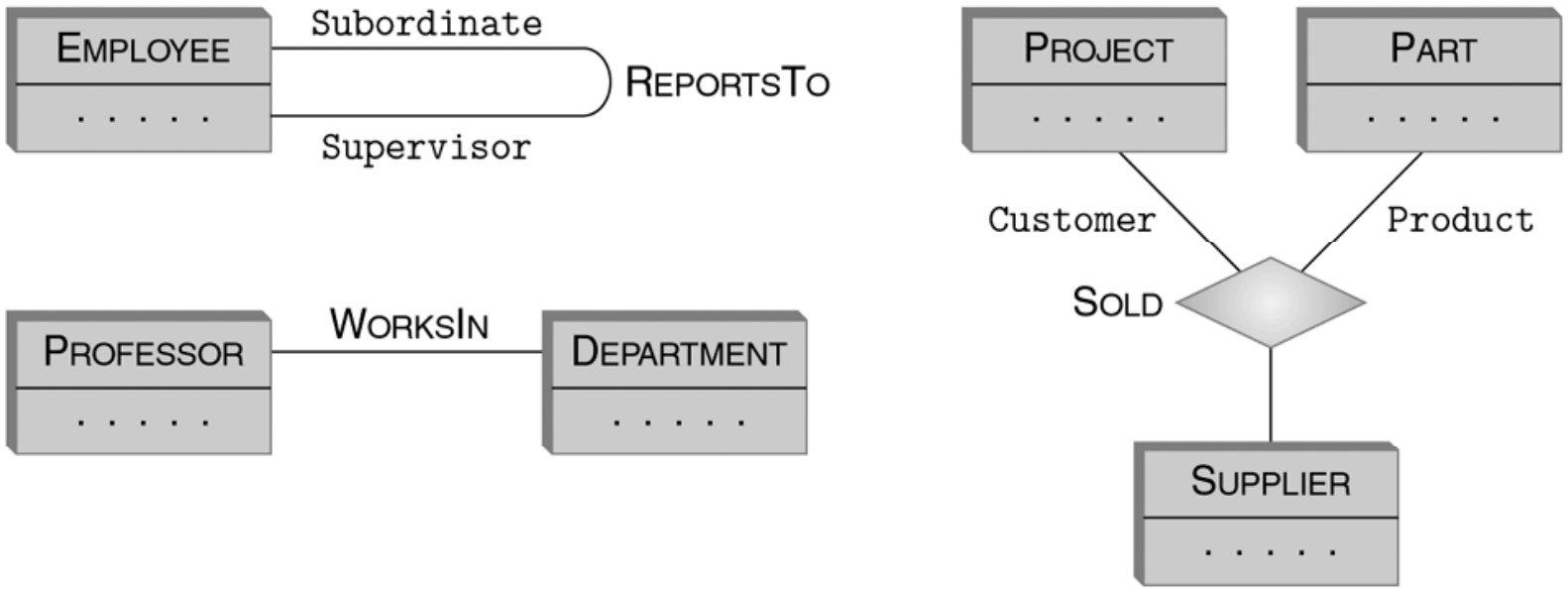


FIGURE 4.16 UML associations.

Association classes

- UML doesn't have association atributes (relationship atributes) - we use association classes (see dotted lines)

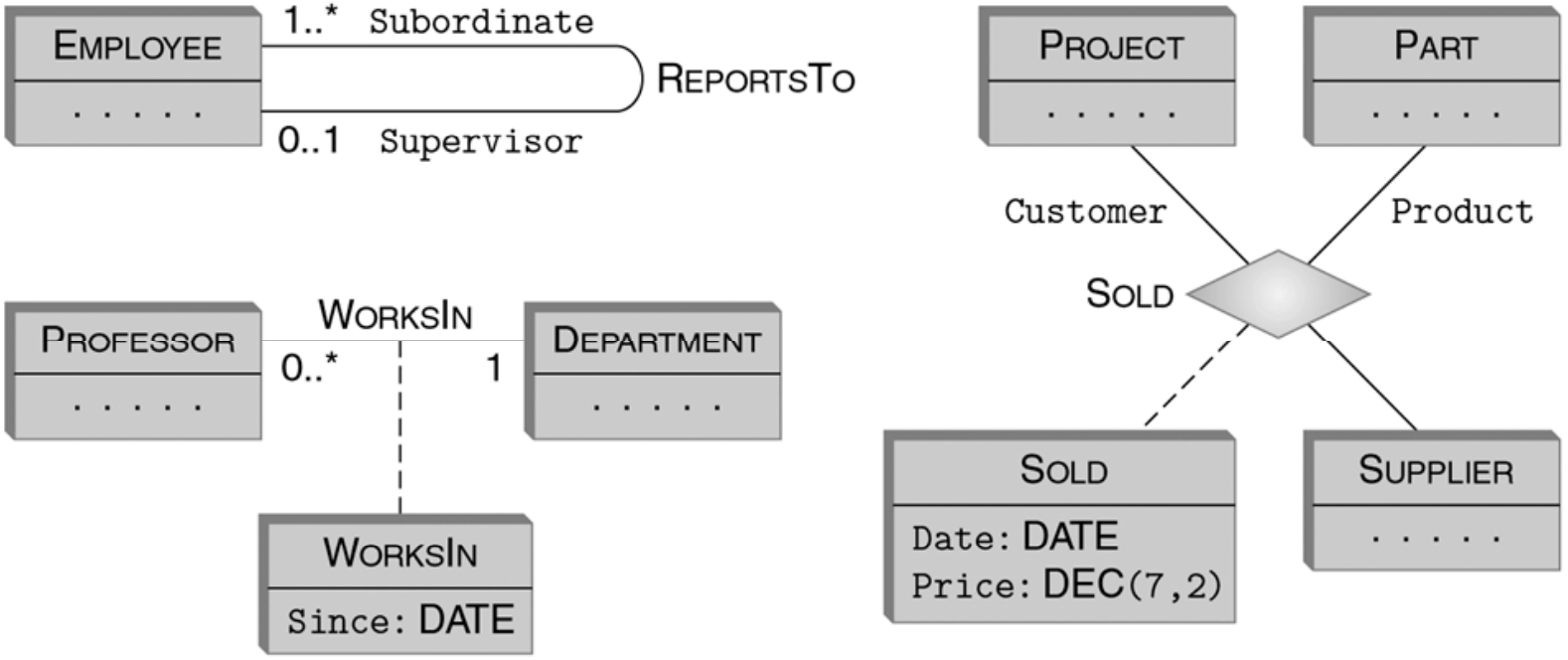


FIGURE 4.17 UML associations with association classes.

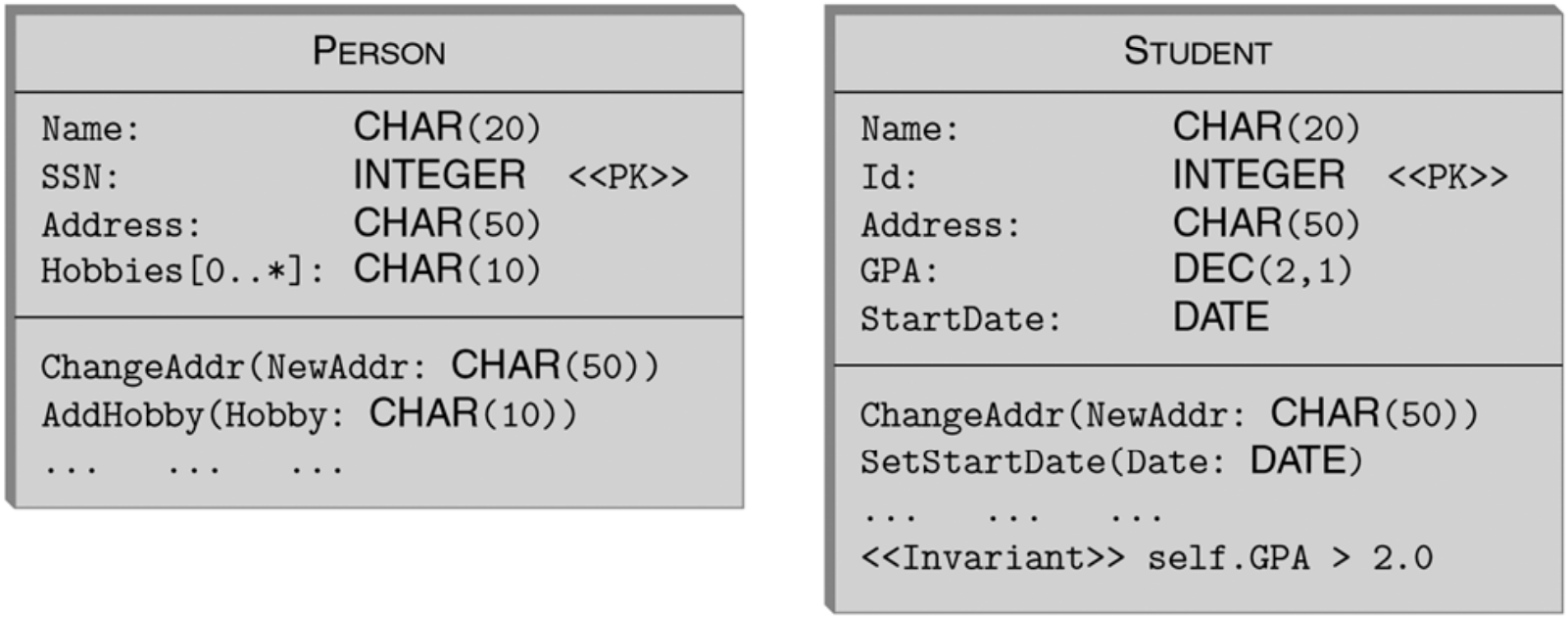
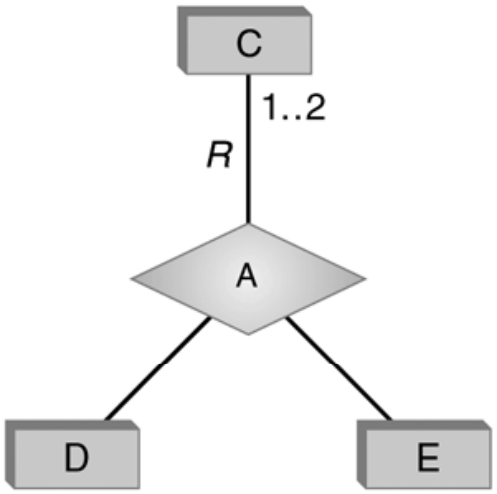


FIGURE 4.15 Examples of UML classes.

Multiplicity constraints

FIGURE 4.18 The meaning of the multiplicity constraint in UML.



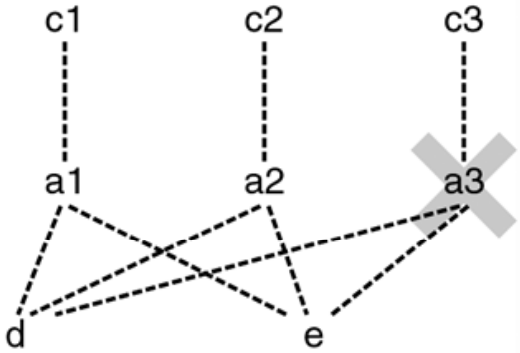
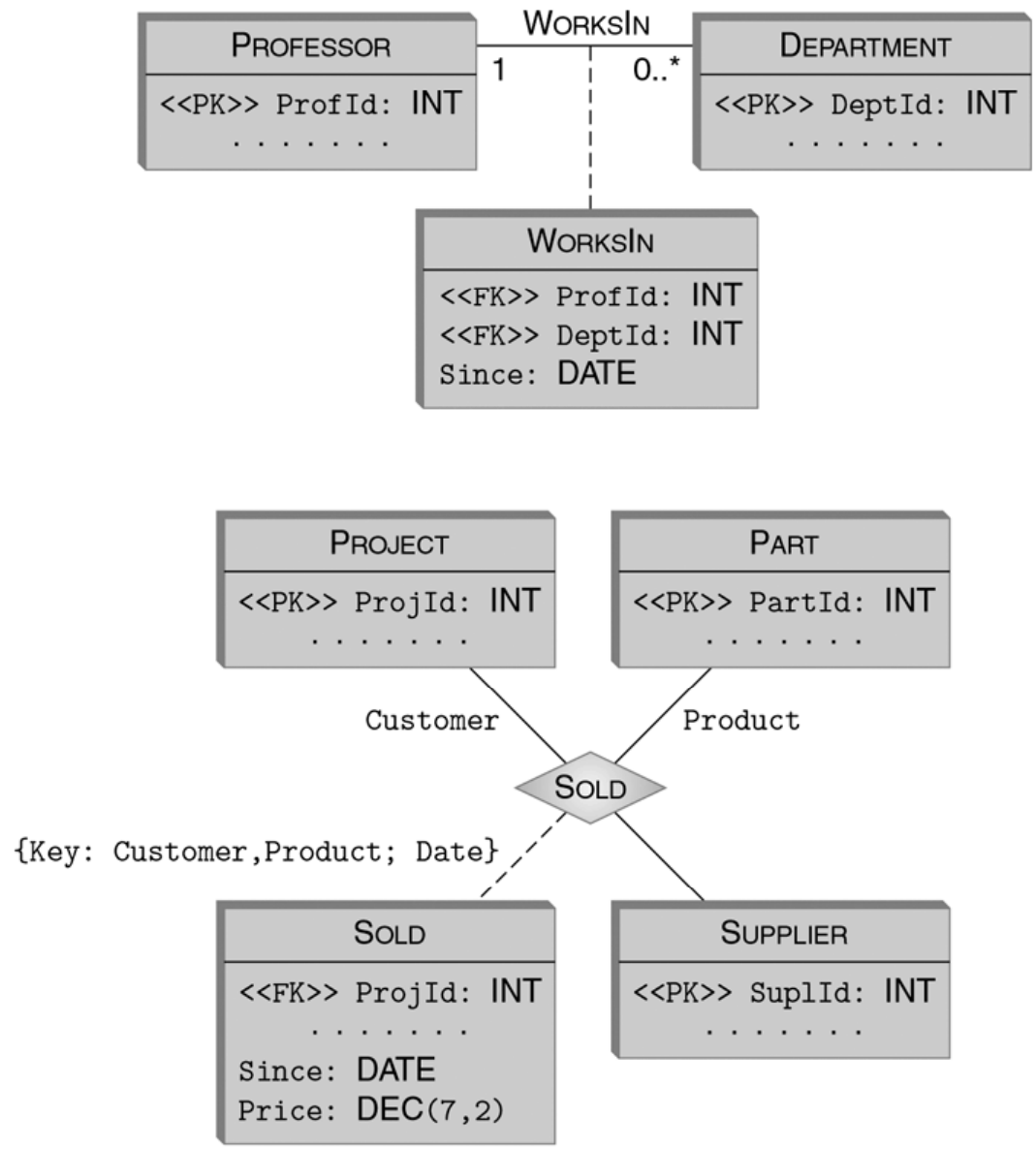


FIGURE 4.21 Foreign keys in UML.



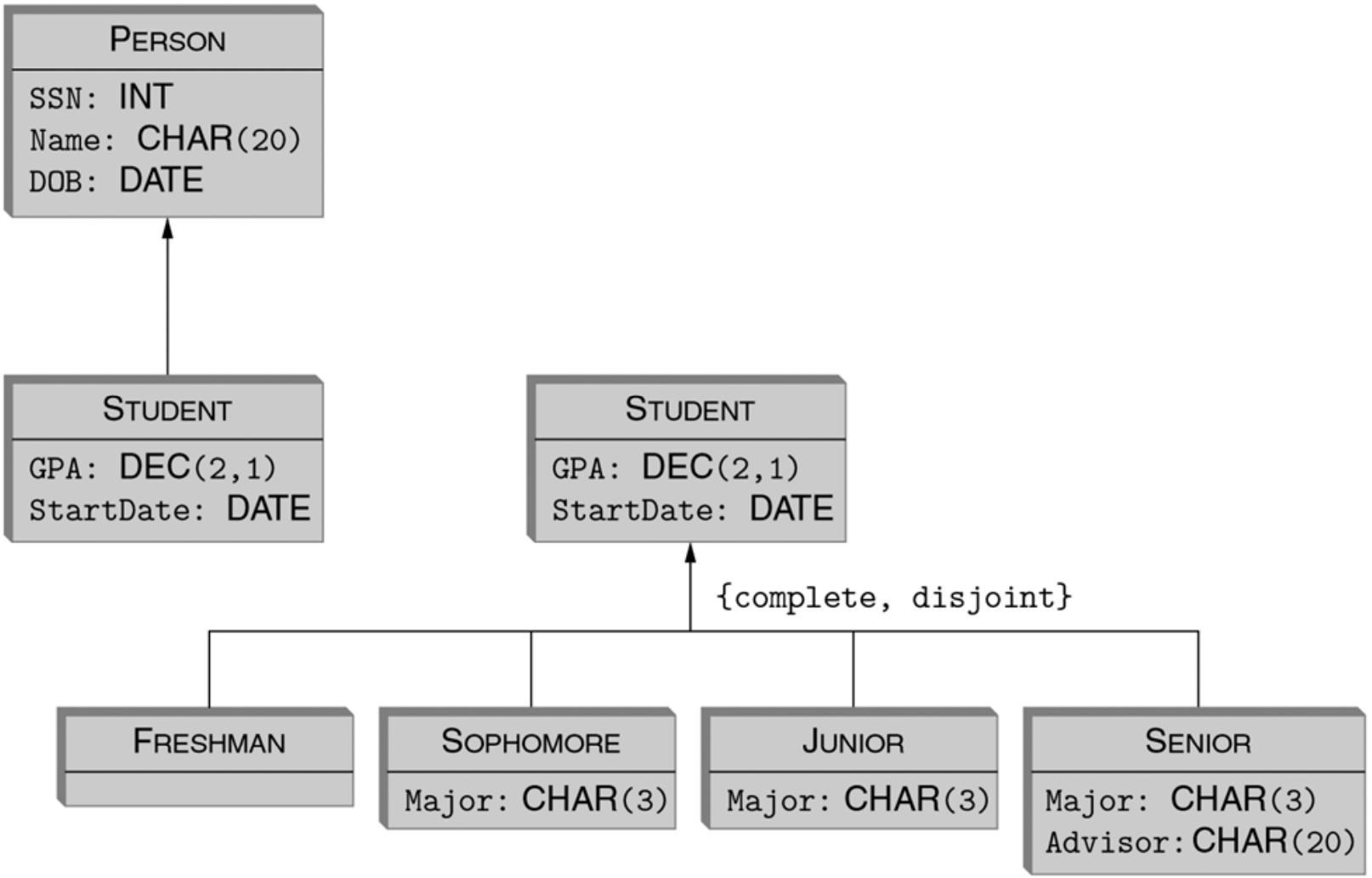


FIGURE 4.22 IsA (or generalization) hierarchies in UML.

FIGURE 4.25 Aggregation: non-exclusive part-of association in UML.

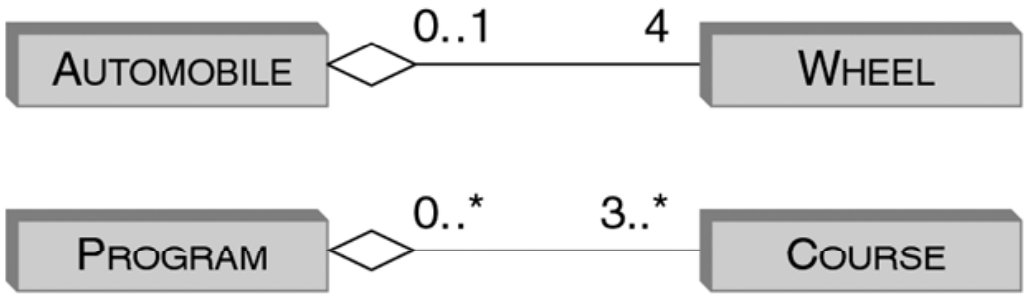
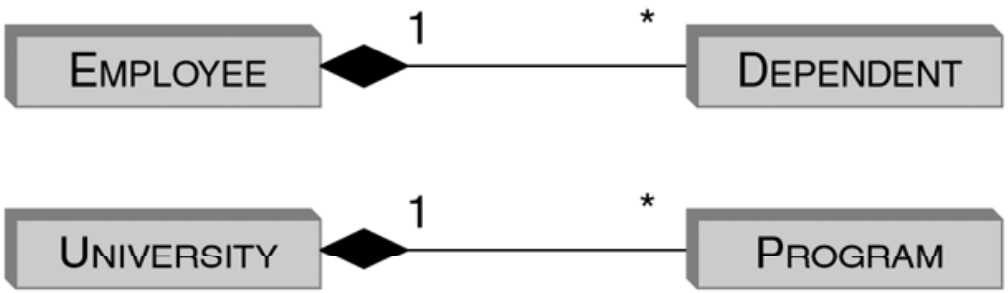


FIGURE 4.26 Composition: exclusive part-of association in UML.



UML example

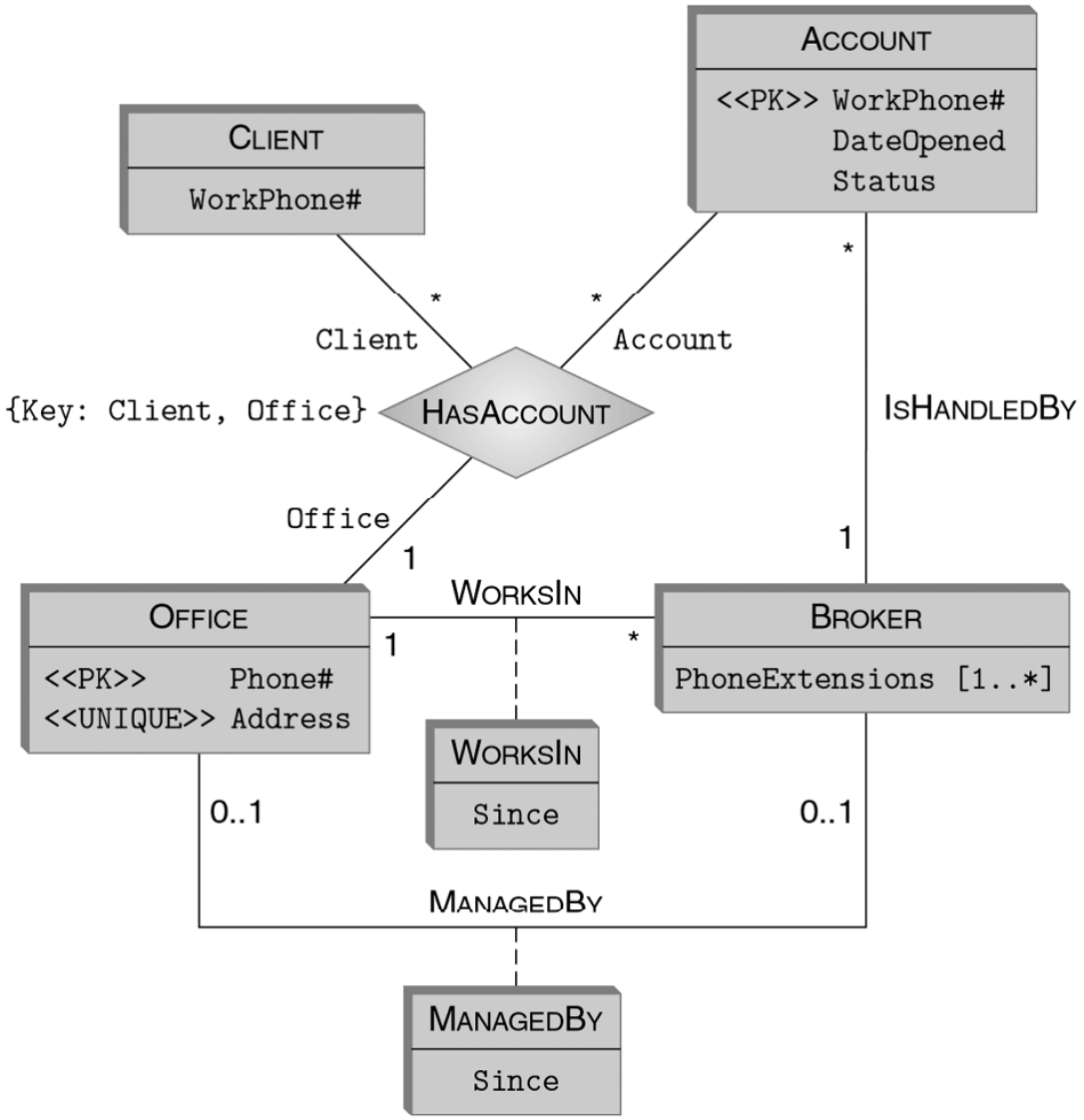


FIGURE 4.32 Client/broker information in UML.

Universal Modeling Language

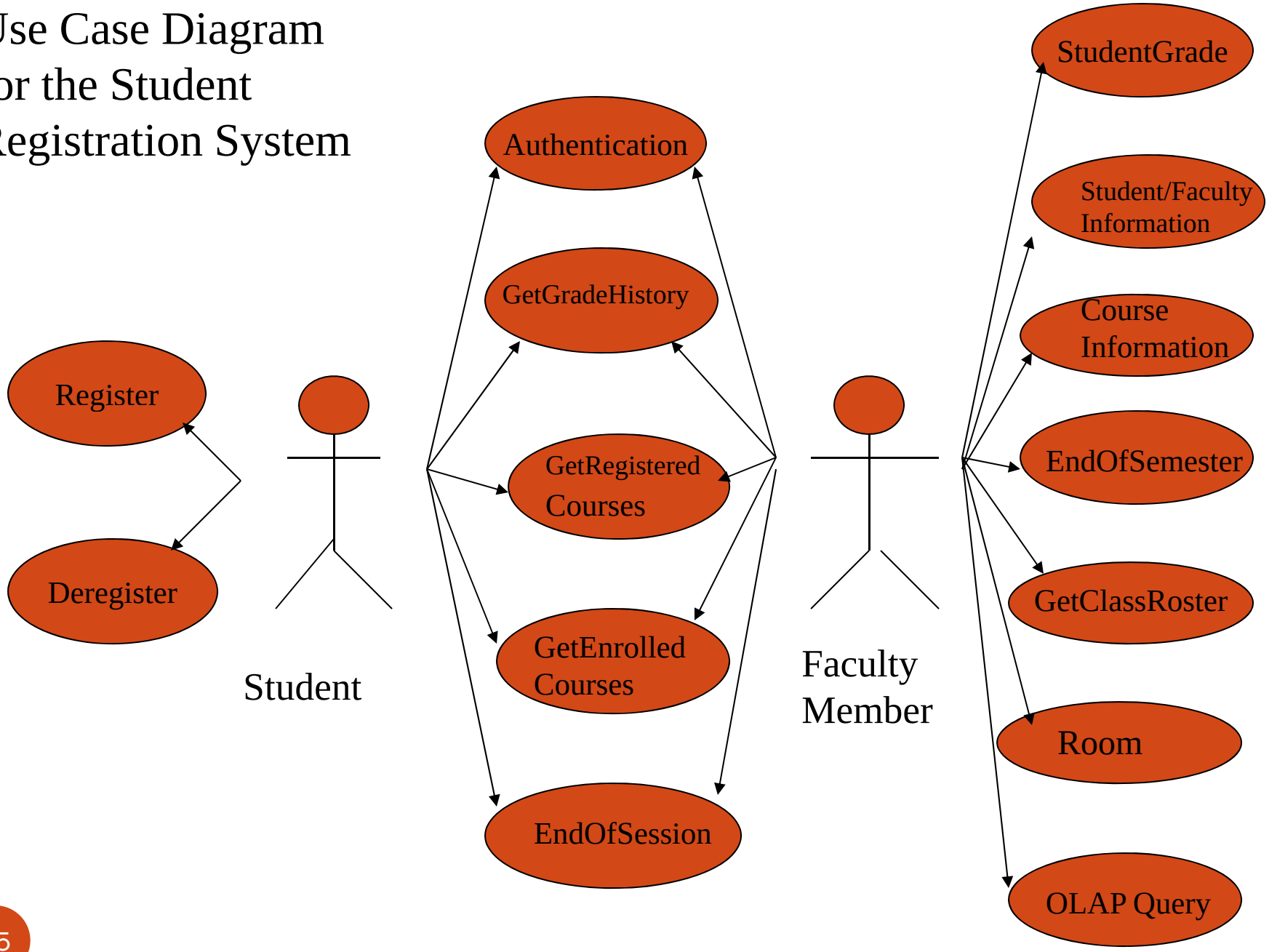
- **Supplemental material:**

- Use cases are part of the UML
  - UML is a graphic language for modeling various static and dynamic aspects of a systems behavior
  - Provides a set of diagrams, each of which models a different aspect of the system behavior.
- Because UML is graphic it is particularly appropriate for communicating between the analyst and the customer and between various members of the implementation team

Use Case Diagrams

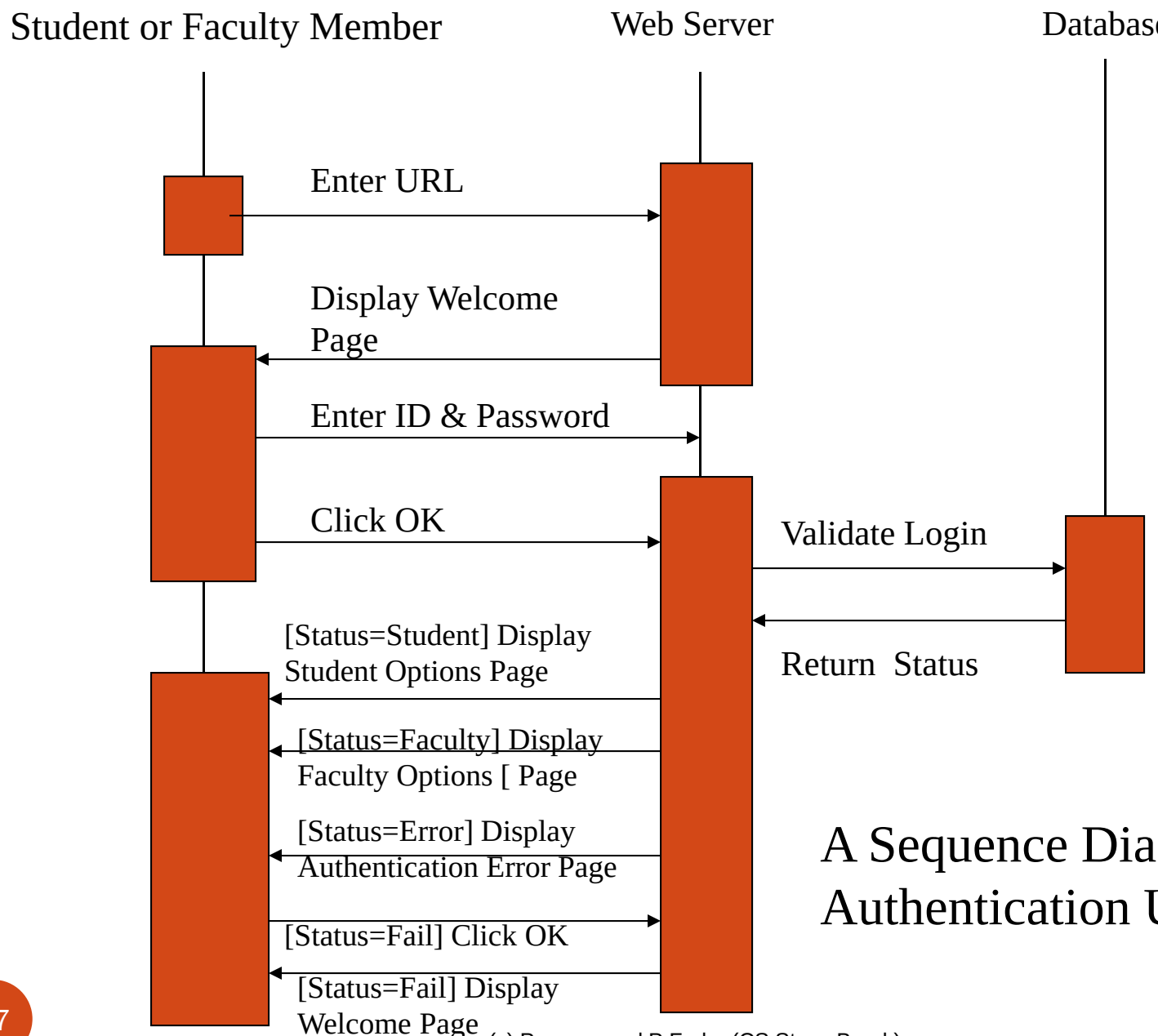
- UML provides a graphic way to display all the use cases in an application
- These diagrams can be used to communicate with the
  - Customer to determine if the current set of use cases is adequate
  - Implementors to determine what the system is supposed to do from the customer's viewpoint

Use Case Diagram for the Student Registration System



UML Sequence Diagrams

- Part of the plan for preparing the Specification Document might be to expand each use case into a UML Sequence Diagram
  - A graphic display of the temporal ordering of the interactions between the actors in a use case and the other modules in the system



A Sequence Diagram for the Authentication Use Case

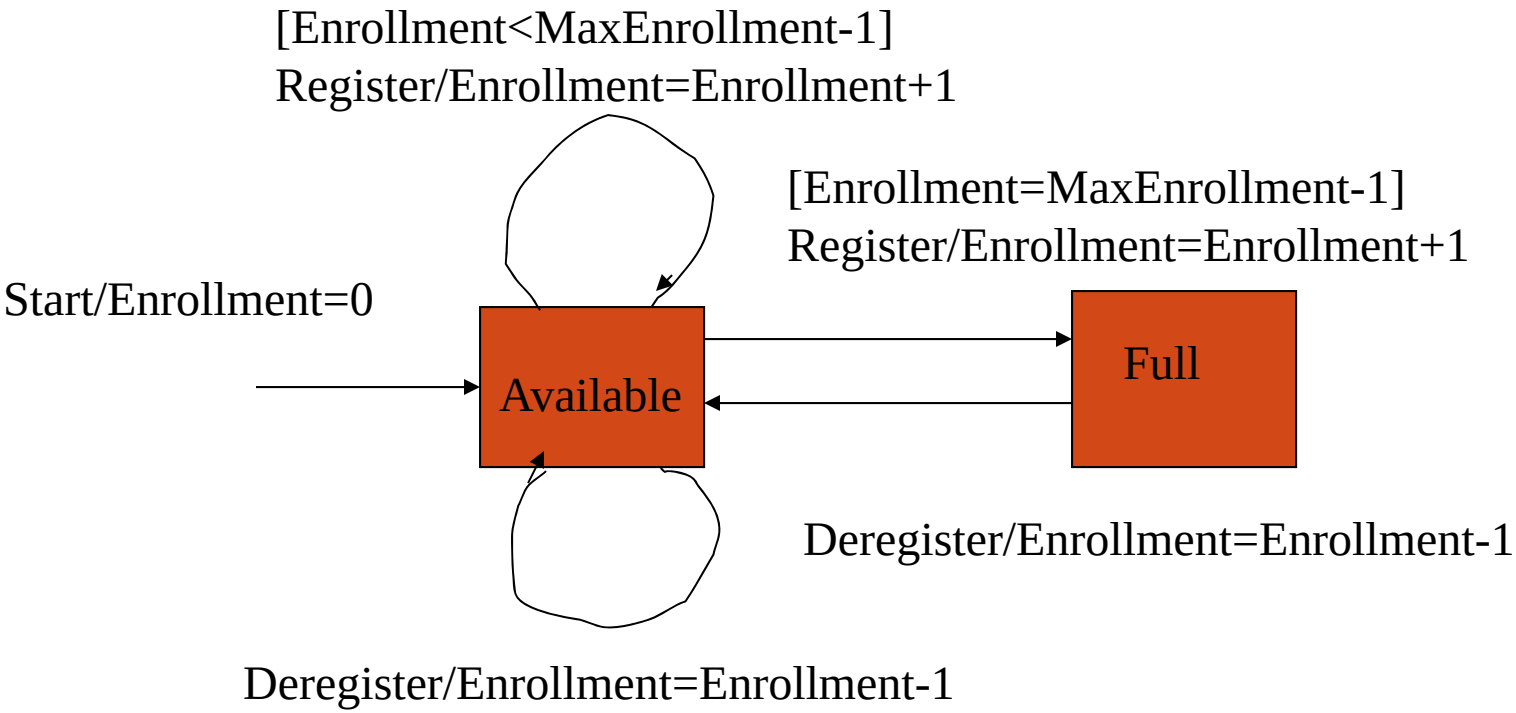
Sequence Diagrams

- The actors and pertinent modules are labelled at the top of the diagram
- Time moves downward
- The boxes show when a module or actor is active
- The horizontal lines show the actions taken by the modules or actors
  - Note the notation for conditional actions

    [status=student] Display Student Options Page

UML State Diagrams

- UML class diagrams and E-R diagrams provide *static* models of the business objects in an enterprise
- UML state diagrams can be used to model the *dynamic* behavior of those objects
  - How their internal state changes when their methods are invoked



A UML State Diagram for the Class Object

State Diagram for Class Object

- Class has two states *Available* and *Full*
  - Based on integrity constraint that class size cannot exceed *MaxEnrollment*
- Transitions between states are of the form

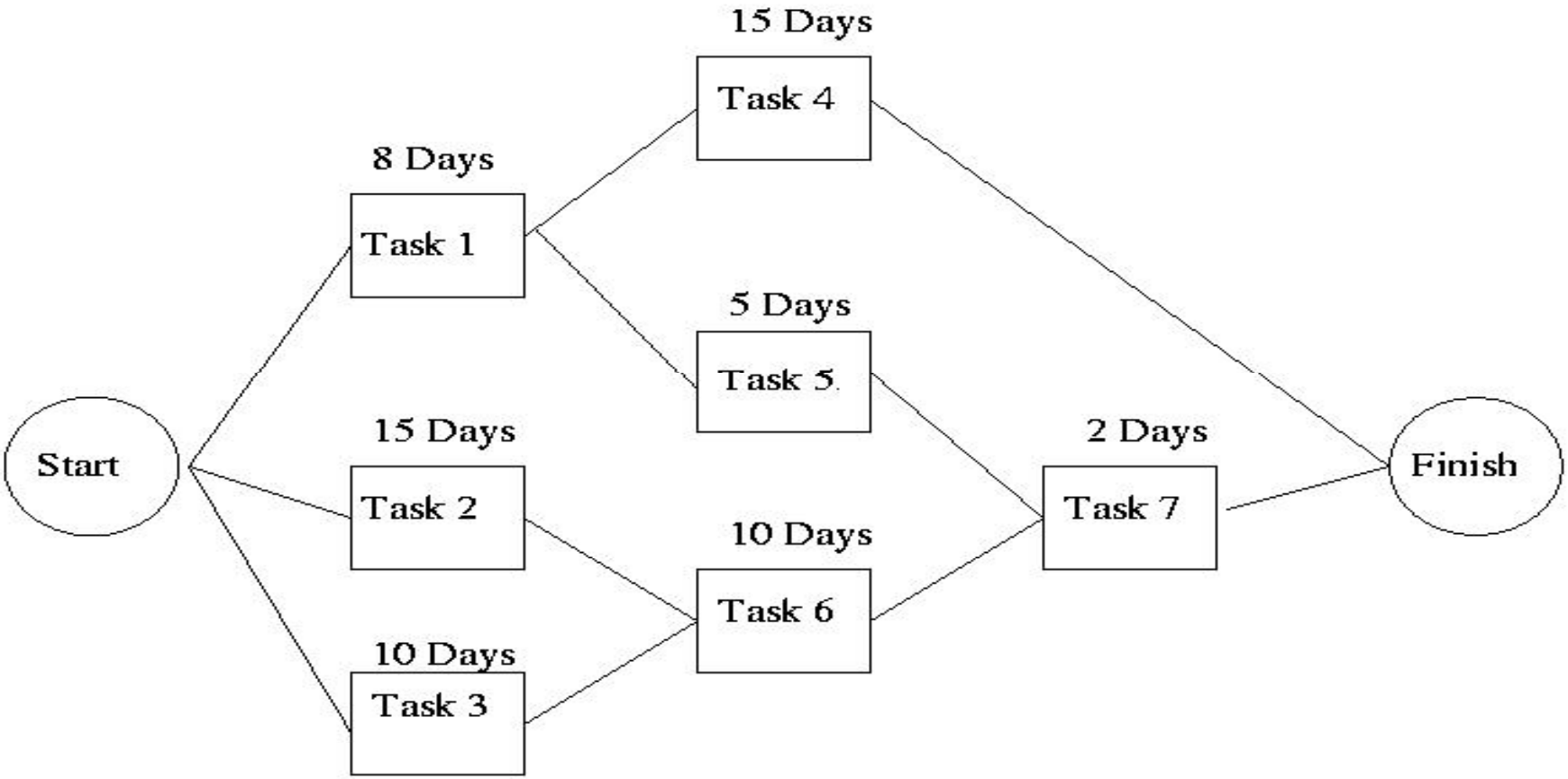
  [*guard*] *operation* / *action*
  - Guard: specifies when the transition can take place
  - Operation: the method that causes the transition
  - Action: how the internal attributes of the object change when the transaction takes place.

Uses of State Diagrams

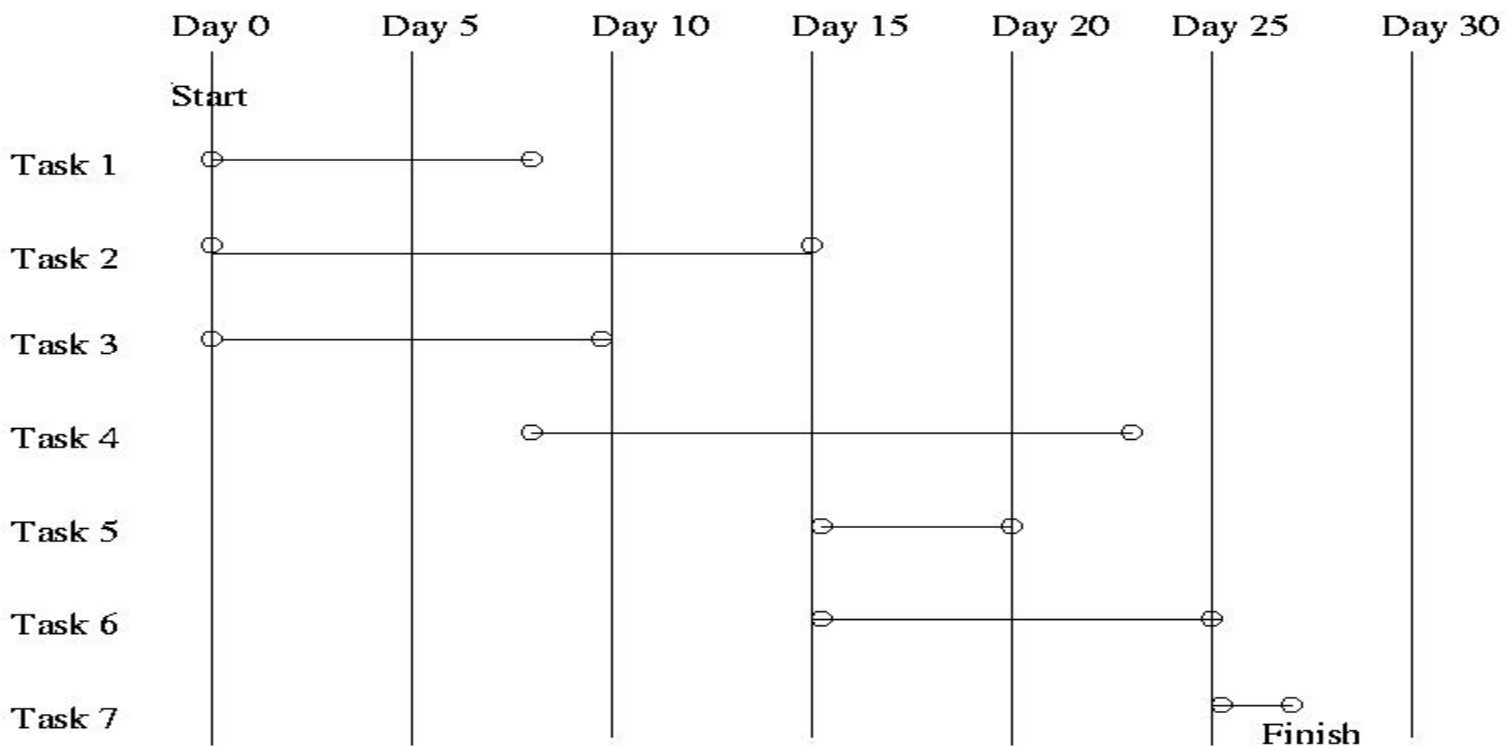
- Can be used to communicate the design to
  - Other designers
  - Coders who must implement the design
  - Test designers who must test the final system
    - A good test set must visit all the states in the diagram.

Dependency Chart (PERT Chart)

- Other visual languages: Dependency Charts



Activity Chart (Gantt Chart)



Staff Allocation Chart