Conceptual Modeling of Databases with Entity-Relationship Diagrams and Unified Modeling Language

CSE 305 – Principles of Database Systems

Paul Fodor

Stony Brook University

http://www.cs.stonybrook.edu/~cse305

Database Design

- We interviewed the users and we prepared a detailed Software Specification Document
  - What is next?
  - We are ready to begin designing the database portion of the system.
    - The main issue in database design is to provide an accurate model of the enterprise in the form of a relational database that can be efficiently accessed by concurrently executing transactions.

Database Design

- Our Goal: specification of database schema
  - The design process should be performed according to a *well-defined methodology* and be evaluated according to a set of objective criteria.
  - Design methodologies for relational databases:
    - the entity-relationship (E-R)
    - the UML class diagrams

Database Design

- Database design is typically a two-stage process:
  - The initial phase is based on the E-R or UML methodology.
    - Convert E-R diagram to DDL
  - Followed by refinement using the *relational normalization theory*, which provides objective criteria for evaluating alternative designs.

Database Design

- An entity-relationship (E-R) diagram is a graphical representation of the entities, relationships, and constraints that make up a given design (developed by Peter Chen in 1976)
  - Use *E-R model* to get a high-level graphical view of essential components of enterprise and how they are related

Database Design

- In the *E-R Model*, the enterprise is viewed as a set of:
  - *Entities*
    - Entities can be thought of as nouns.
      - Examples: a computer, an employee, a song.
  - *Relationships* among entities.
    - A relationship captures how entities are related to one another.
    - Relationships can be thought of as verbs, linking two or more nouns.
    - Examples: an *owns* relationship between a company and a computer, a *supervises* relationship between an employee and a department, a *performs* relationship between an artist and a song.

Entities and Entity Types

- ***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
  - Every attribute of an entity specifies a particular property of that entity.
  - Example: an employee entity might have a Social Security Number (SSN) attribute.

Entity Type

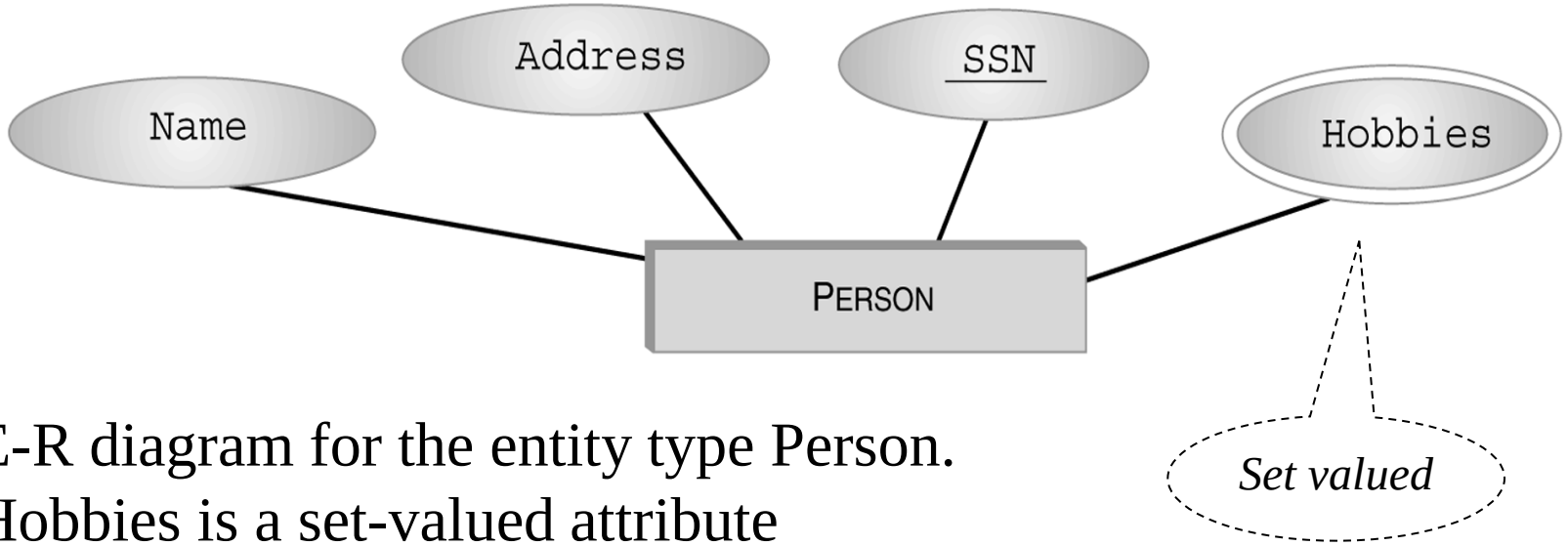
- An Entity type described by set of attributes
  - E.g.: Person: Id, Name, Address, Hobbies
- ***Domain***: possible values of an attribute
- *An attribute value can be a set* (in contrast to relational model)
  - (111111, John, 123 Main St, {stamps, coins})
- ***Key***: a minimum set of attributes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

Entity Type

- Graphical Representation in E-R diagram
  - There are many notation styles: Chen style, Bachman Style, Martin Style, crow foot ☺, EERD
    - we will use the original Chen style
  - *Entity types are represented in E-R diagrams as rectangles, and their attributes are represented as ovals.*

Entity Type

- Graphical Representation in E-R diagram:



E-R diagram for the entity type Person.
Hobbies is a set-valued attribute
SSN is underlined to indicate that it is a key

Relationships and Relationship Types

- ***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

- An ***attribute*** of a relationship type '*describes*' the relationship
  - e.g., John majors in CS since 2016
    - John and CS are related
    - 2016 describes relationship - value of SINCE attribute of MajorsIn relationship type
- A *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; 2016) describes a relationship

Relationship Type

- Described by set of attributes and roles
  - 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 relationship 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
    - e.g., Bob reports to Mary since 2016
  - If we do not have distinct names for the roles, then it is not clear who reports to whom

Roles

- 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

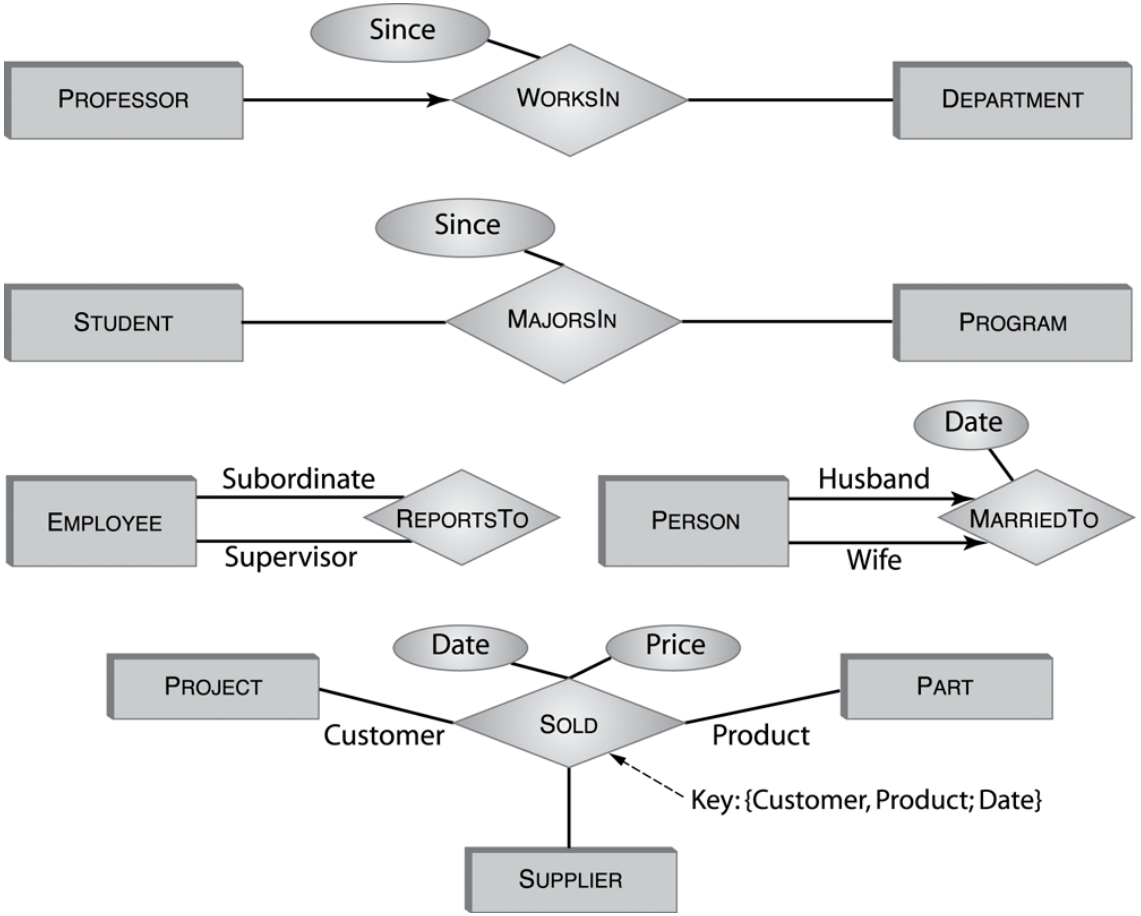
- The *schema of a relationship type* includes:
  - A list of attributes along with their corresponding domains (adverbs in natural language).
    - An attribute can be single-valued or set-valued.
  - A list of roles along with their corresponding entity types.
    - Unlike attributes, roles are always single-valued.
  - A set of constraints.

Schema of a Relationship Type

- Formally, a relationship type contains:
  - 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
  - Key: Minimum set of roles and attributes that uniquely identify a relationship
- A *relationship* (or *relationship instance*): $<e_1, \ldots, e_n; a_1, \ldots a_k>$
  - e_i is an entity (i.e., 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 the role name = name of entity set).



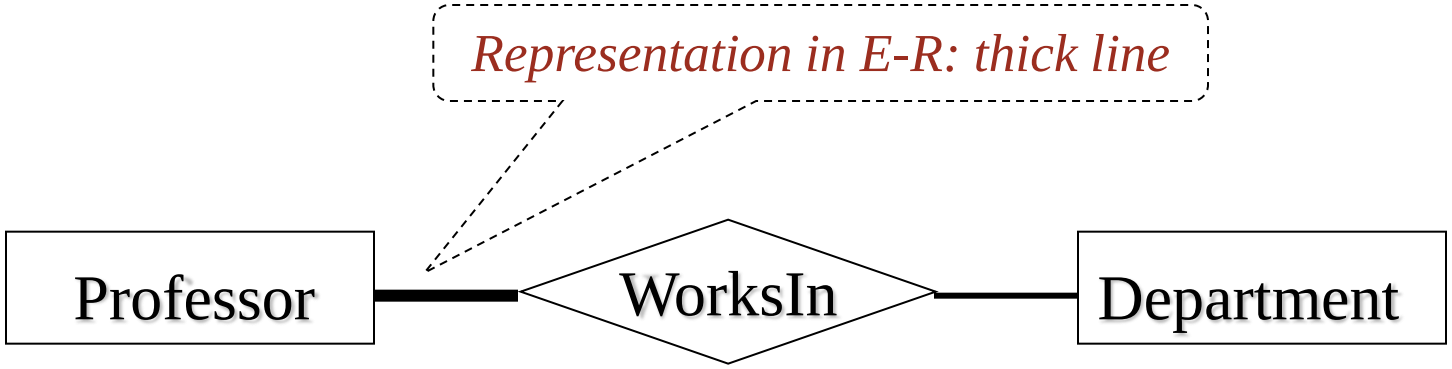
Most attributes have been omitted.

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

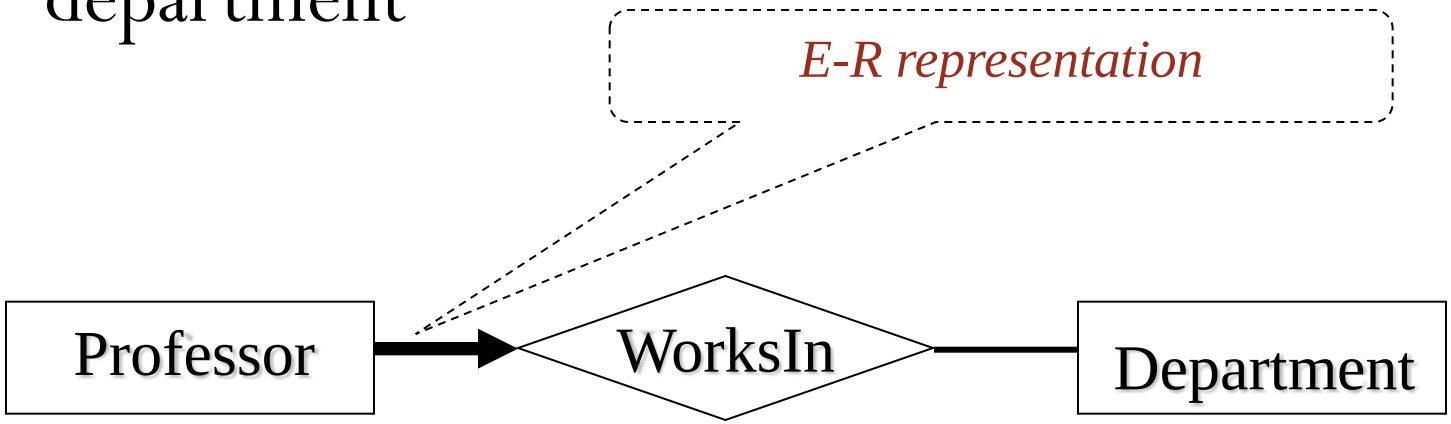
Participation Constraints

- 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 ρ(r) = e.
  - e.g., every professor works in at least one department



Participation and Key Constraints

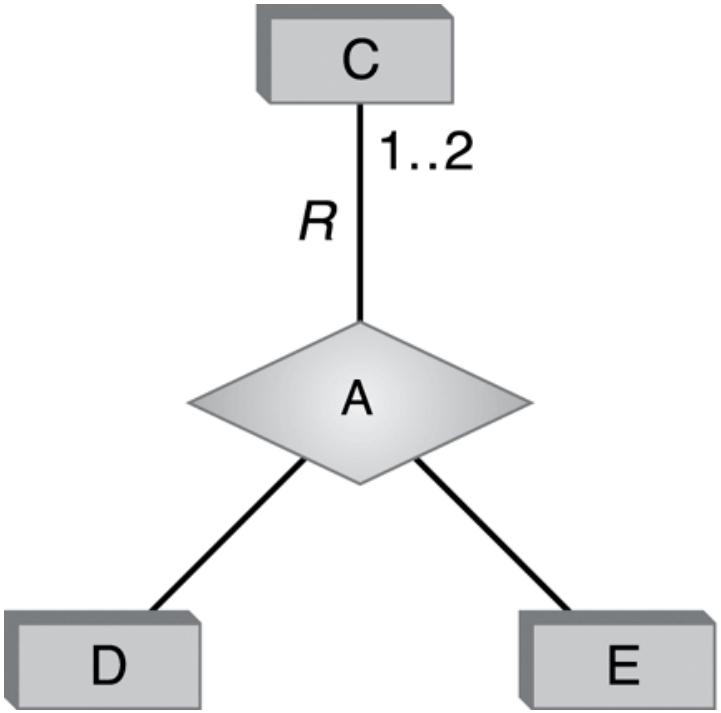
- If every entity participates in exactly one relationship, both a participation and a key constraint hold:
  - e.g., every professor works in exactly one department

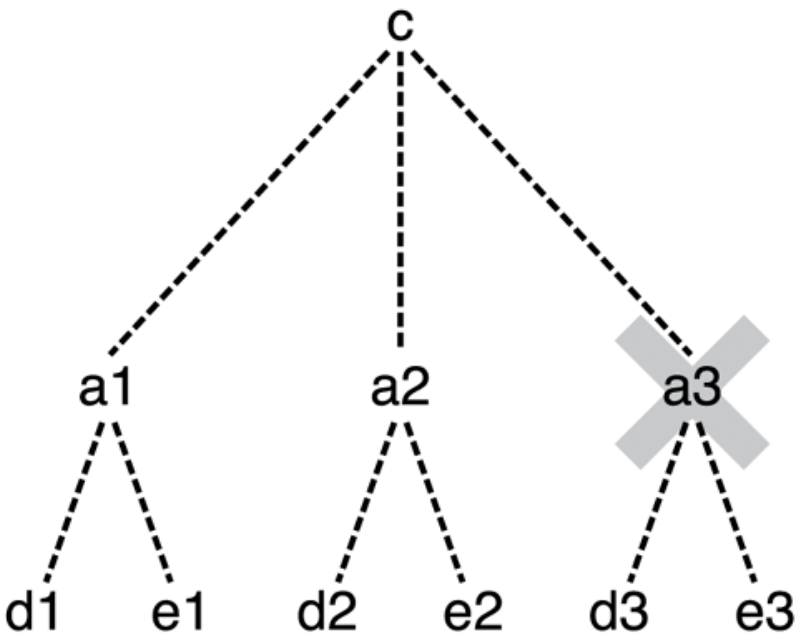


Cardinality constraints

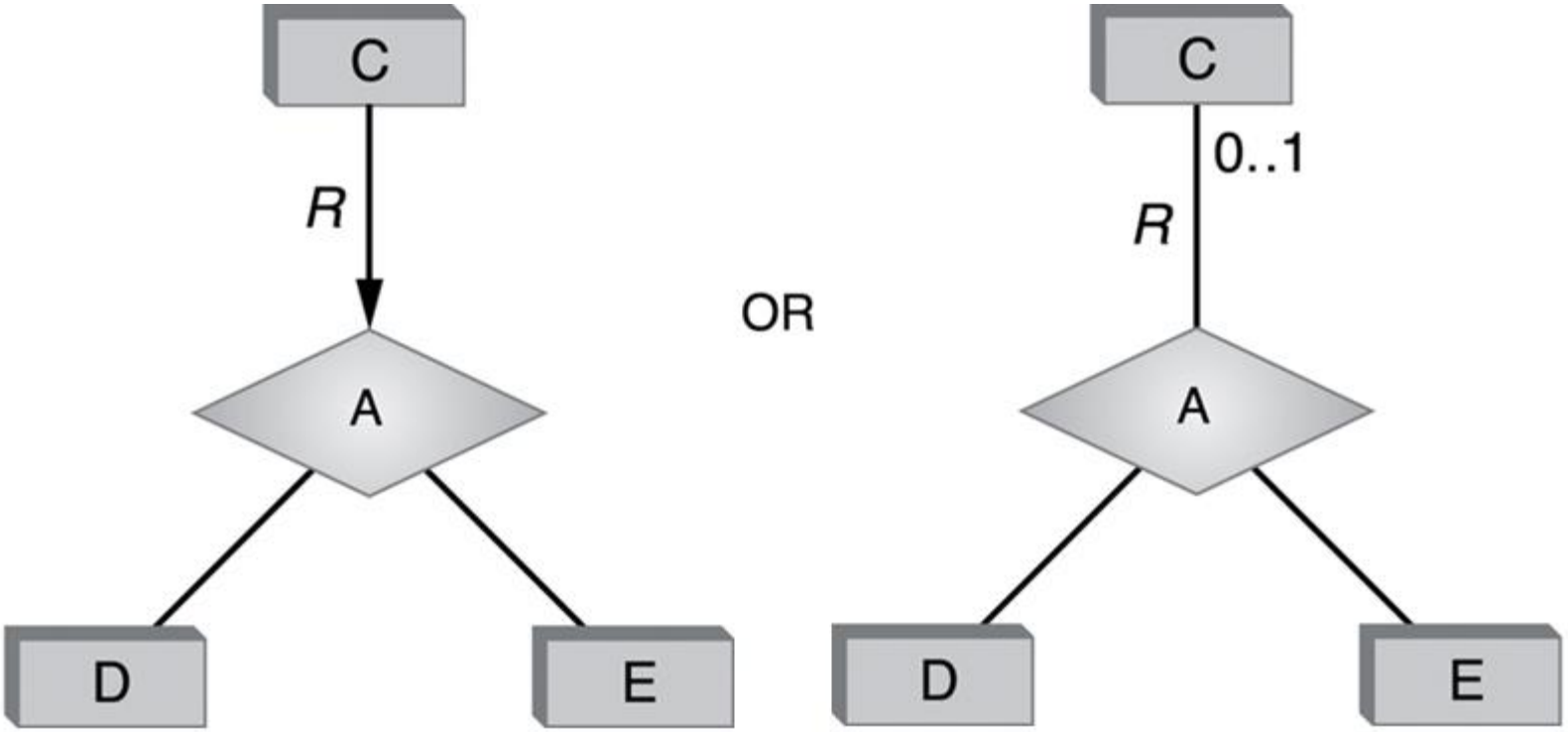
- Let C be an entity type and A be a relationship type that is connected to C via a role, R.
  - A cardinality constraint on the role R is a statement of the form min..max attached to R and it restricts the number of relationship instances of type A in which a single entity of type C can participate in role R to be a number in the interval min..max (with end points included).

Cardinality constraints

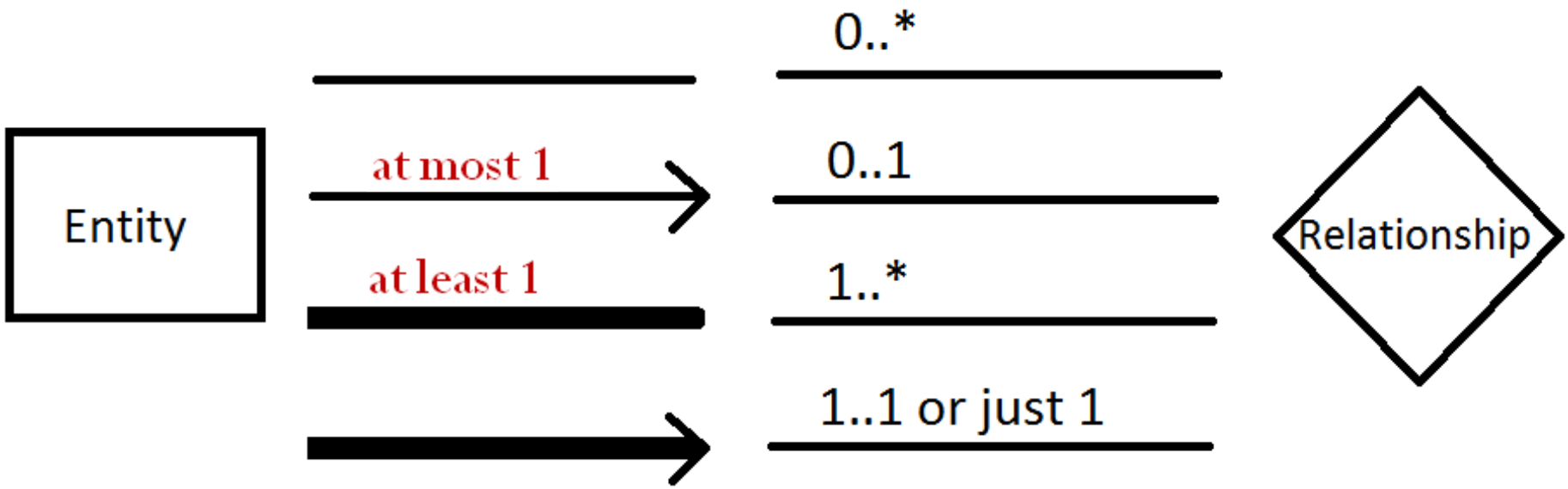




Two ways to represent single-role key constraints



Two ways to represent single-role key constraints



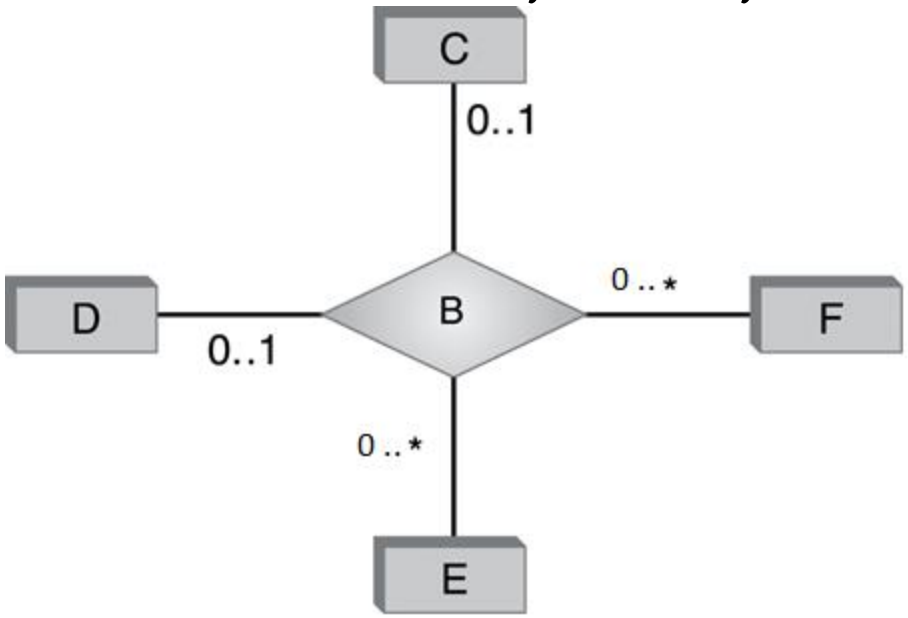
Cardinality constraints

- max can be the * symbol, which represents infinity

Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only *n* (where *n* > 1)
0..n	Zero to *n* (where *n* > 1)
1..n	One to *n* (where *n* > 1)

Cardinality constraints

- Many-to-one, one-to-one, and many-to-many correspondences:

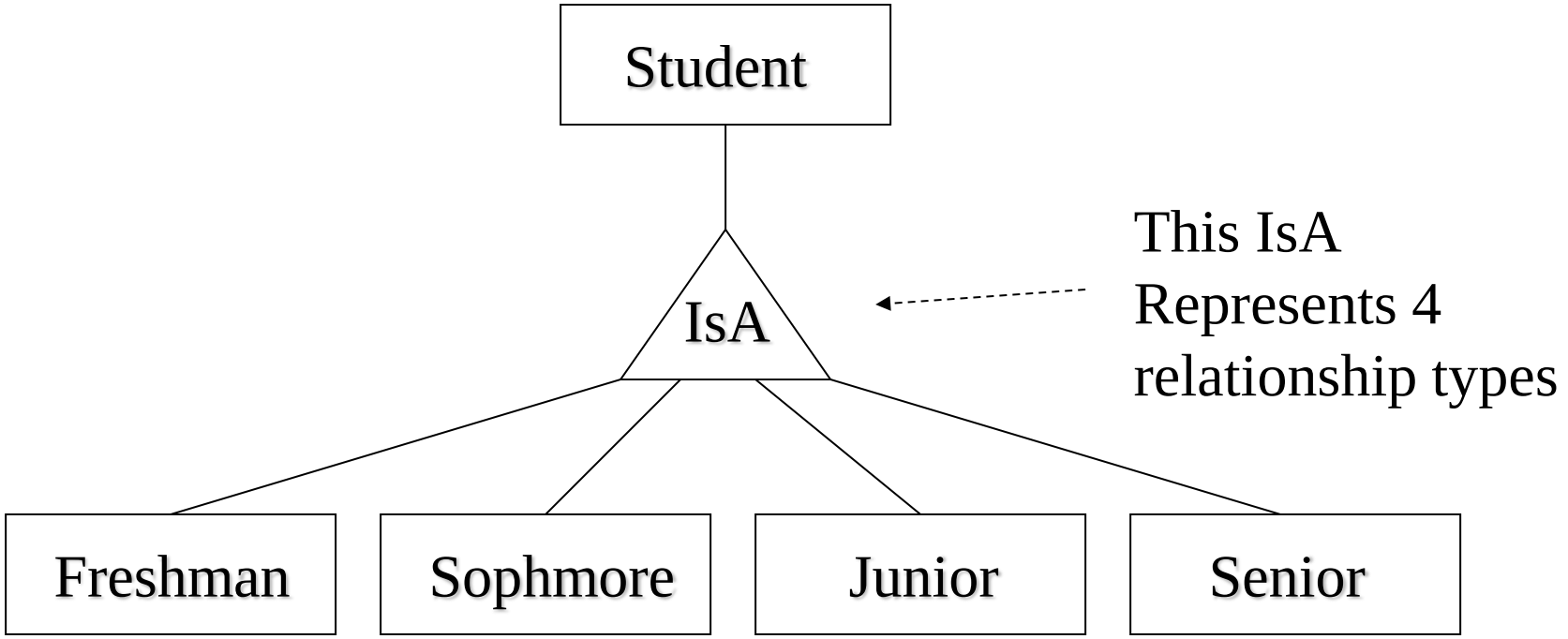


  - A *one-to-one* correspondence between the entity types C and D: an entity of type C can be associated with at most one entity of type D, and vice versa
  - A *one-to-many* correspondences of E to C and D: an entity of type E can be associated with any number (including zero) of entities of types C and D
  - A *many-to-many* between E and F, meaning that an E-entity can be associated with any number of F-entities

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

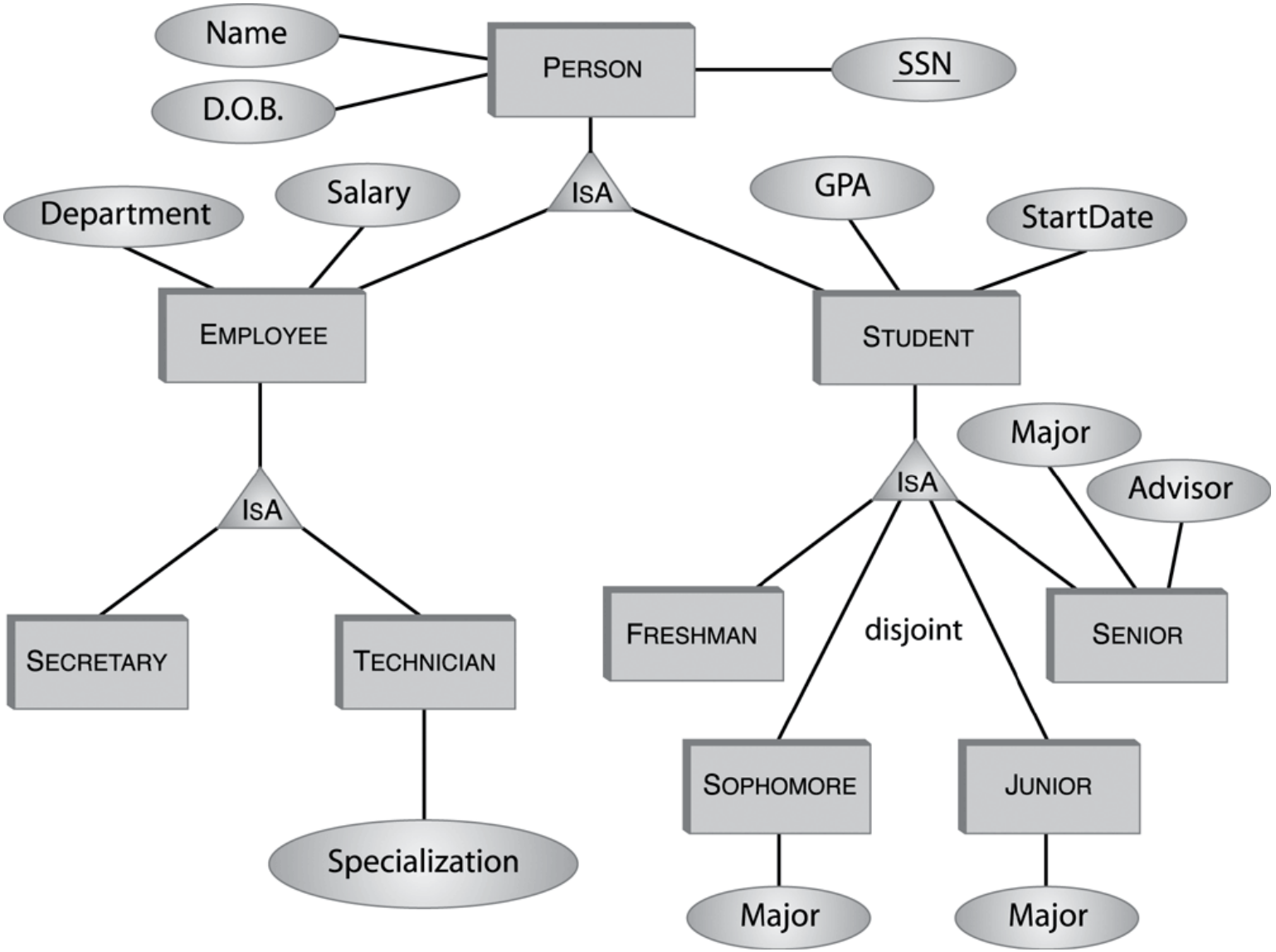
IsA



Properties of IsA

- Inheritance - Attributes of supertype apply to subtype.
  - E.g., the 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

IsA Hierarchy Example



Advantages of IsA

- Can create a more concise and readable E-R diagrams
  - 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

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 sets

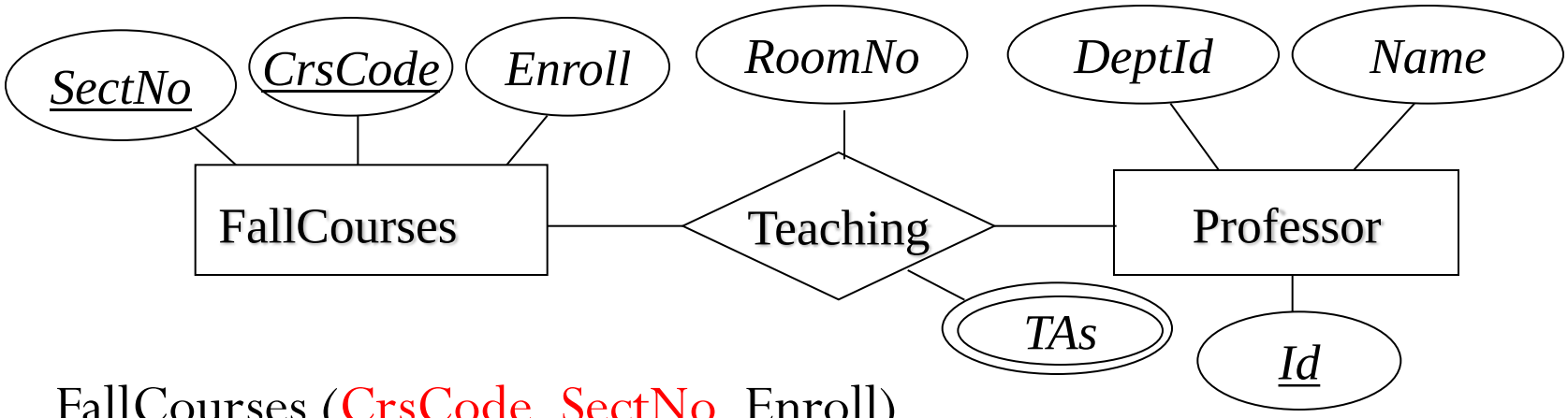
From E-R Diagrams to Relational Database Schemas

- 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

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



FallCourses (CrsCode, SectNo, Enroll)

Professor (Id, DeptId, Name)

Teaching (CrsCode, SecNo, Id, RoomNo, TAs)

Representation of Relationship Types in the Relational Model

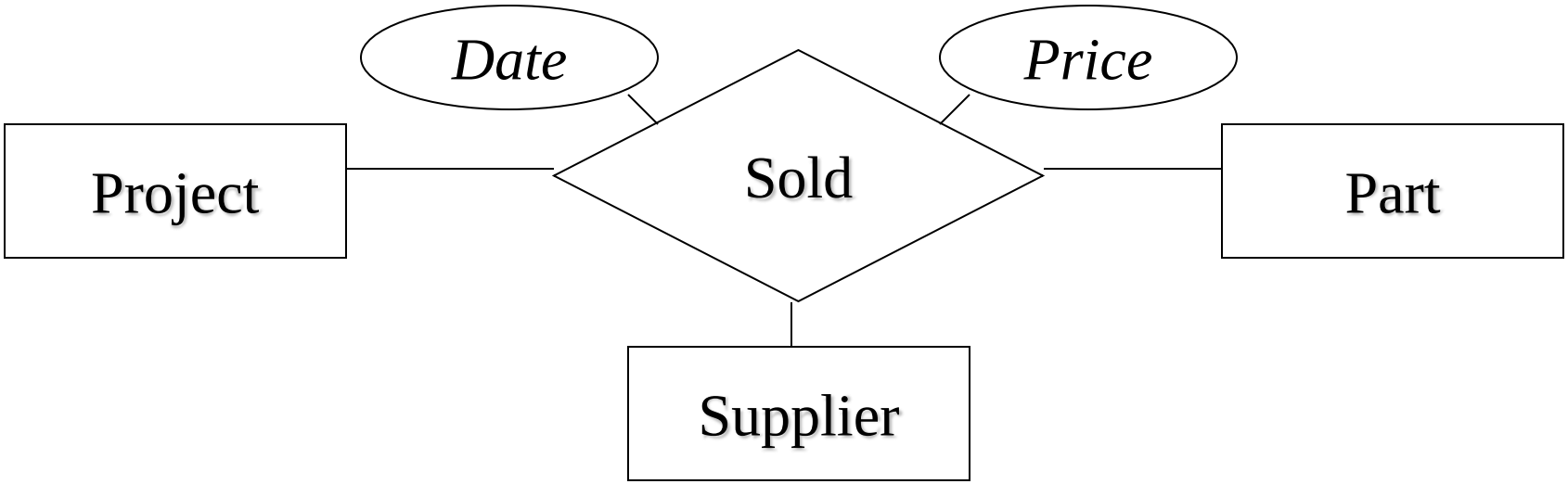
- Candidate key of corresponding table = candidate key of the relationship
  - Except when there are set valued attributes
    - Example: Teaching (CrsCode, SectNo, Id, RoomNo, TAs)
      - Key of relationship type = (CrsCode, SectNo)
      - Key of relation = (CrsCode, SectNo, TAs)

Set valued

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

Representation in SQL

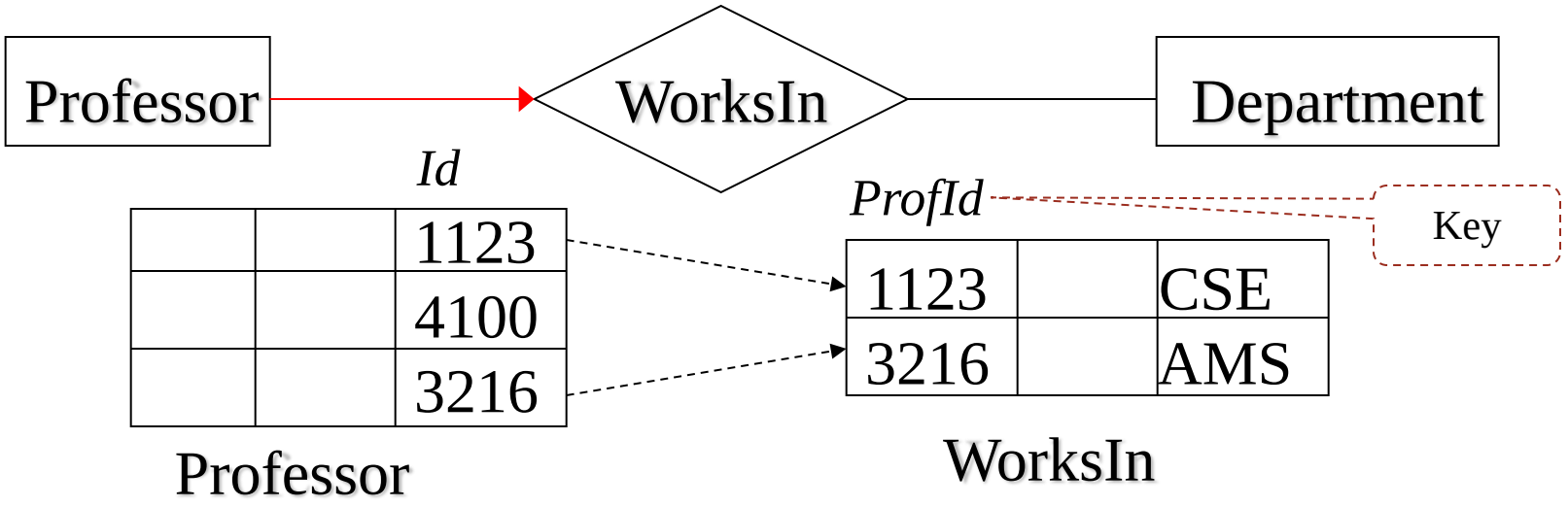
- Each role of relationship type produces a foreign key in corresponding relation
  - Foreign key references table corresponding to entity type from which role values are drawn

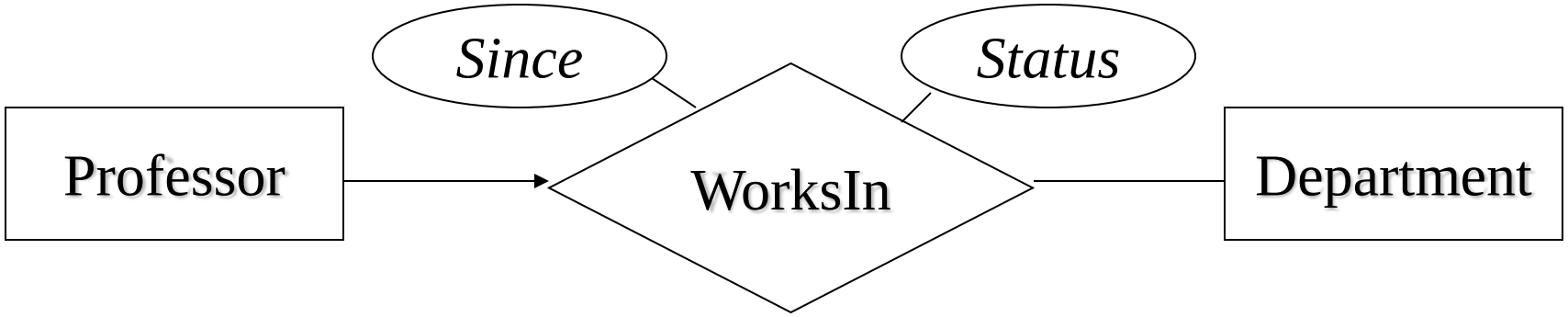



```
CREATE TABLE  Sold  (
    Price  INTEGER,                -- attribute
    Date  DATE,                    -- attribute
    ProjId  INTEGER,               -- role
    SupplierId  INTEGER,           -- role
    PartNumber  INTEGER,           -- role
    PRIMARY KEY (ProjId, SupplierId, PartNumber, Date),
    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: the key of the relation corresponding to the entity type is the 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.
  - ProfId is a foreign key in WorksIn that refers to Professor.





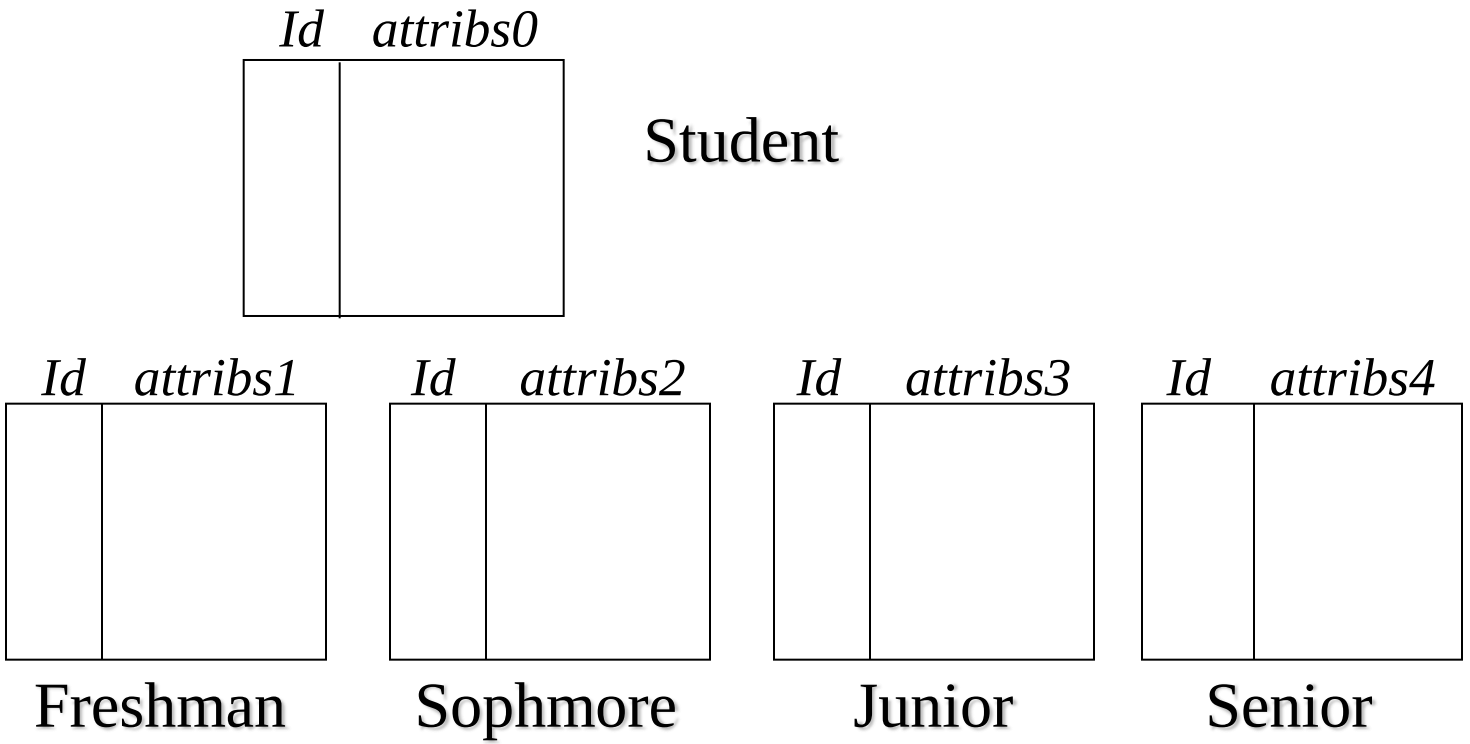

```
CREATE TABLE WorksIn  (
    Since DATE,                -- attribute
    Status  CHAR (10),         -- attribute
    ProfId  INTEGER,           -- role (key of  Professor)
    DeptId  CHAR (4),          -- role (key of  Department)
    PRIMARY KEY (ProfId),  -- since a professor works in at most one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

Representing Type Hierarchies in the Relational Model

- Supertypes and subtypes can be realized as separate relations
  - We 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.



FOREIGN KEY *Id* REFERENCES Student
in all Freshman, Sophomore, Junior, and Senior

Type Hierarchies and the Relational Model

- Advantage: any redundancy is eliminated if IsA is not disjoint
  - Example: 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

- Example: Every professor works in at least one department
  - inclusion dependency in the relational model:
    - Professor (Id) references WorksIn (ProfId)

  - in SQL:
    - If ProfId is a key in WorksIn (i.e., every professor works in exactly one department) then it is easy:

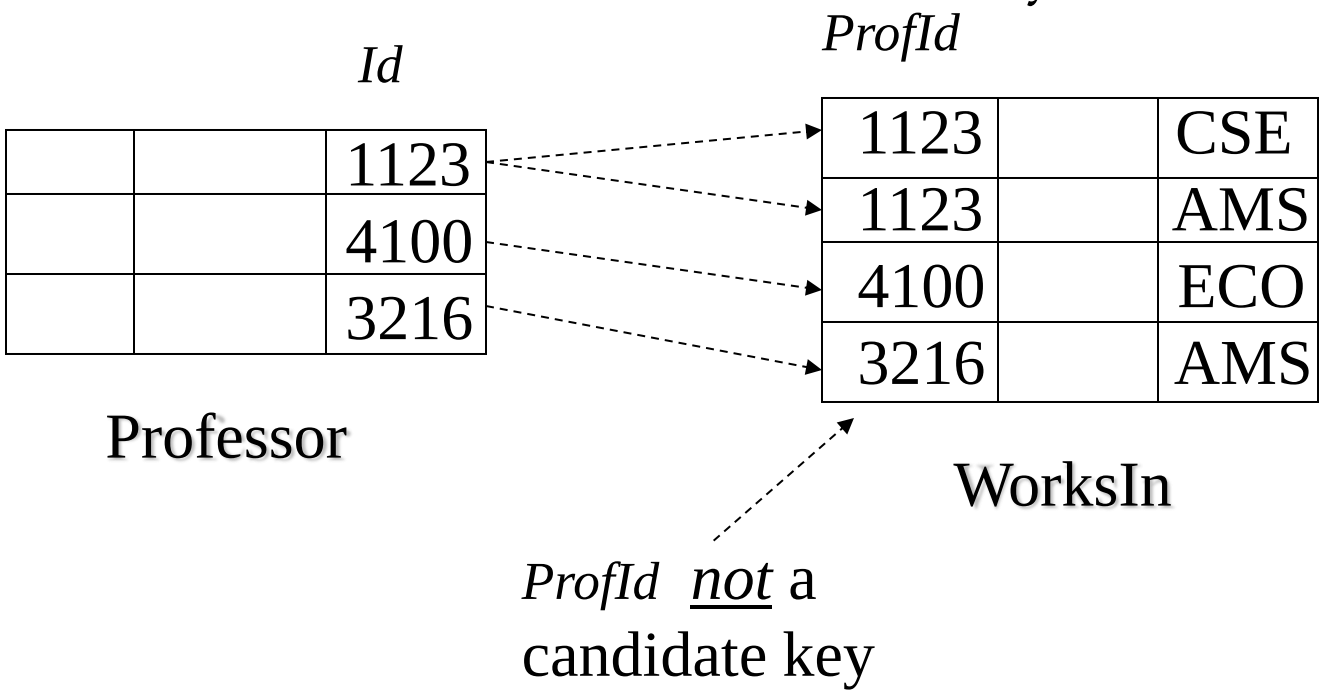
      FOREIGN KEY Id REFERENCES WorksIn (ProfId)

    - General case: ProfId is not a key in WorksIn, so can't use foreign key constraint:

```
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

- General case Example cont. :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 single-role key constraints apply, use foreign key constraint in entity table


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

	Id
xxxxxx	1123
yyyyyy	4100
zzzzzzz	3216

Professor

ProfId		
1123		CSE
4100		ECO
3216		AMS

WorksIn

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

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

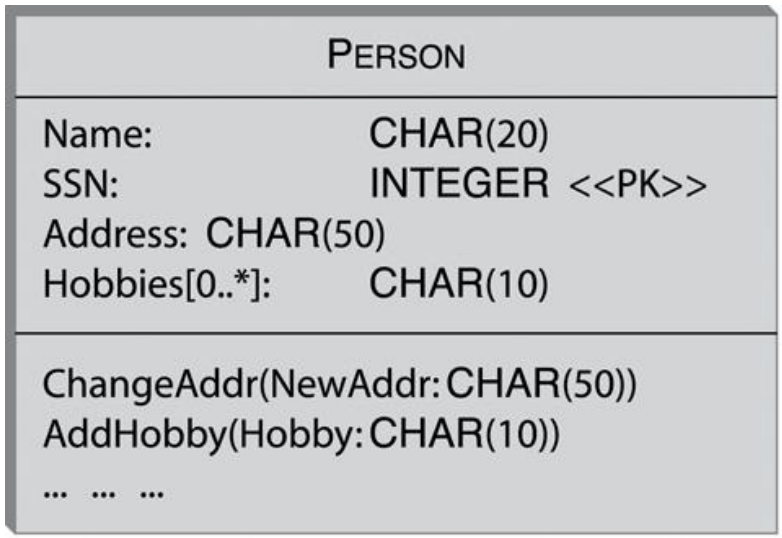
Prof_WorksIn

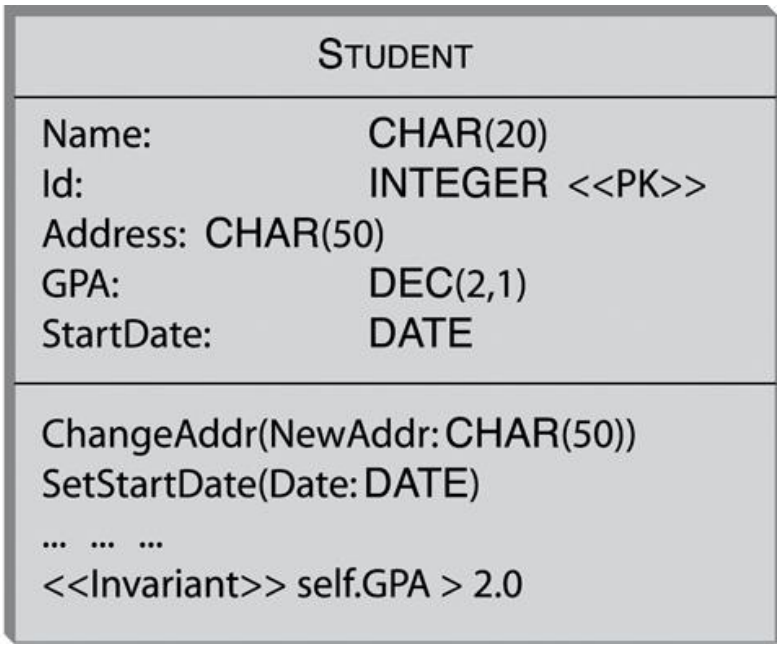
Unified Modeling Language (UML)

- 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
  - Entities are called classes.
- MySQL data model drawing tool is between ER and UML

Unified Modeling Language (UML)

- Representing Entities in UML
  - Attributes are listed under the class.





Unified Modeling Language (UML)

- Representation of Relationships
  - Without attributes: Associations without attributes

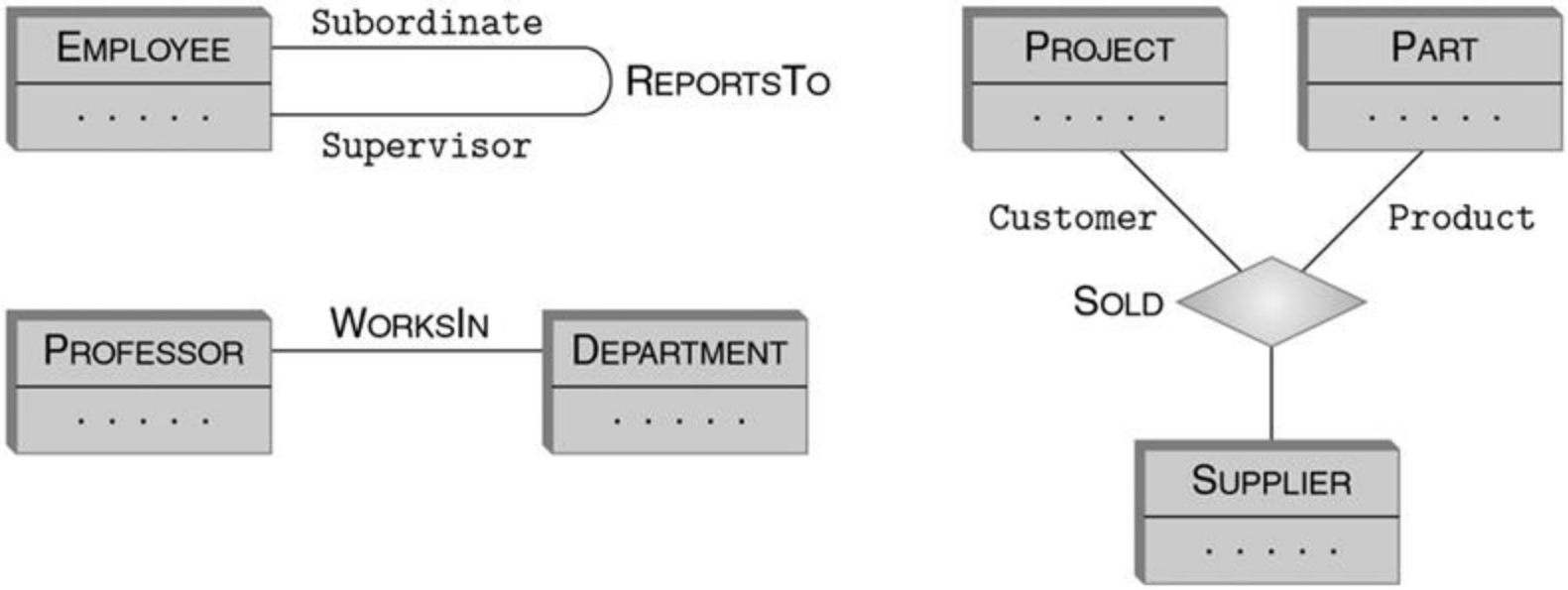
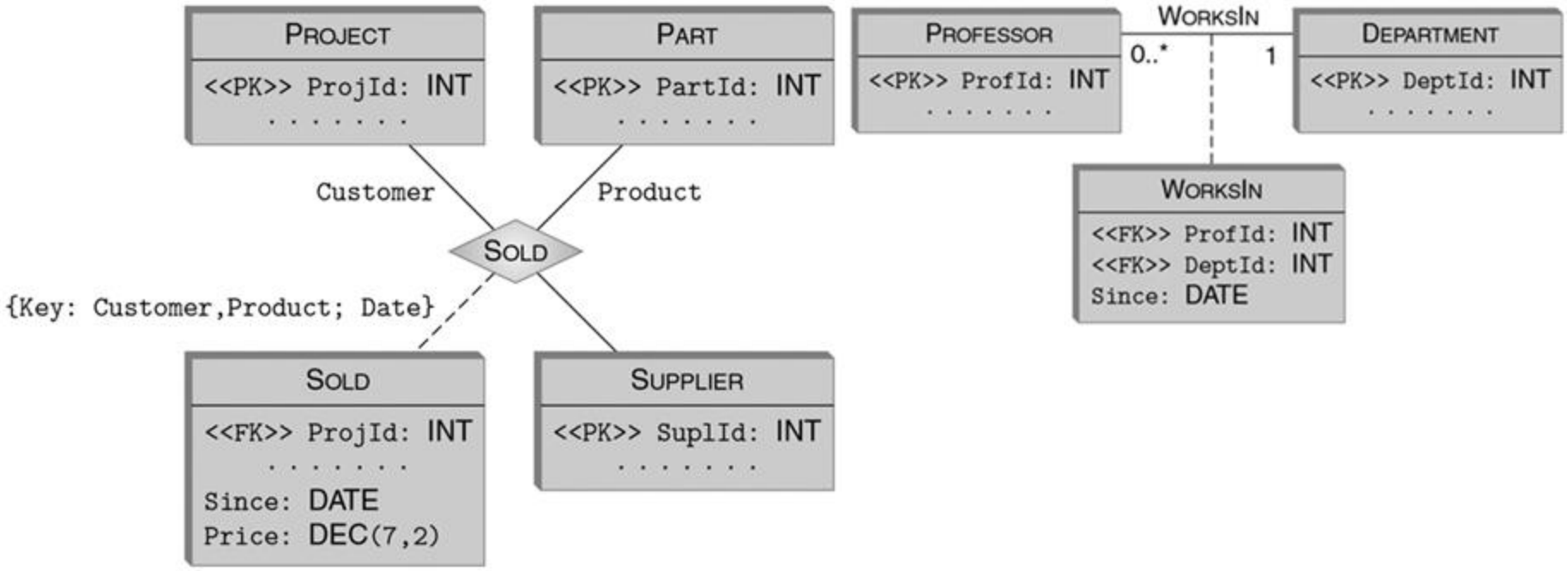


FIGURE 4.16 UML associations.

Association classes

- Representation of Relationships
  - UML doesn't have association attributes (relationship attributes) - we use association classes (see dotted lines)
  - Keys: PK, FK



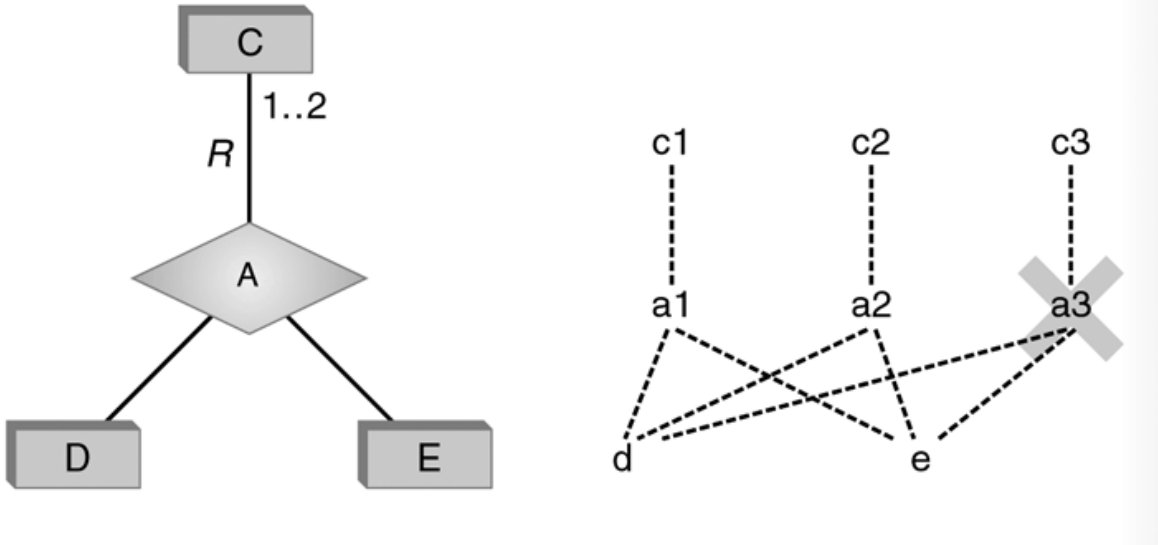
Multiplicity constraints

- A multiplicity constraint on a role, R, that connects an association type, A, with a class, C, is a range specification of the form n..m attached to R, where n ≥ 0 is a nonnegative integer and m is either the * symbol or an integer ≥ n.
  - The range gives the lower and upper bounds on the number of objects of class C that can be connected by means of an association of type A to any given set of objects that are attached to the other ends of the association (one object for each end).

Multiplicity constraints

- Each pair of objects, d ∈ D and e ∈ E, must be connected by associations of type A to at least one and at most two objects of class C.

FIGURE 4.18 The meaning of the multiplicity constraint in UML.



- Adding a3 would violate the upper bound of the multiplicity constraint because this would allow three objects of type C to be connected to a particular pair of objects (d and e) of classes D and E

Multiplicity constraints

- UML multiplicity vs. E-R cardinality constraints
  - For binary relationships, the range specification in UML and E-R appear on the opposite ends of the association/relationship.

Representing IsA Hierarchies in UML

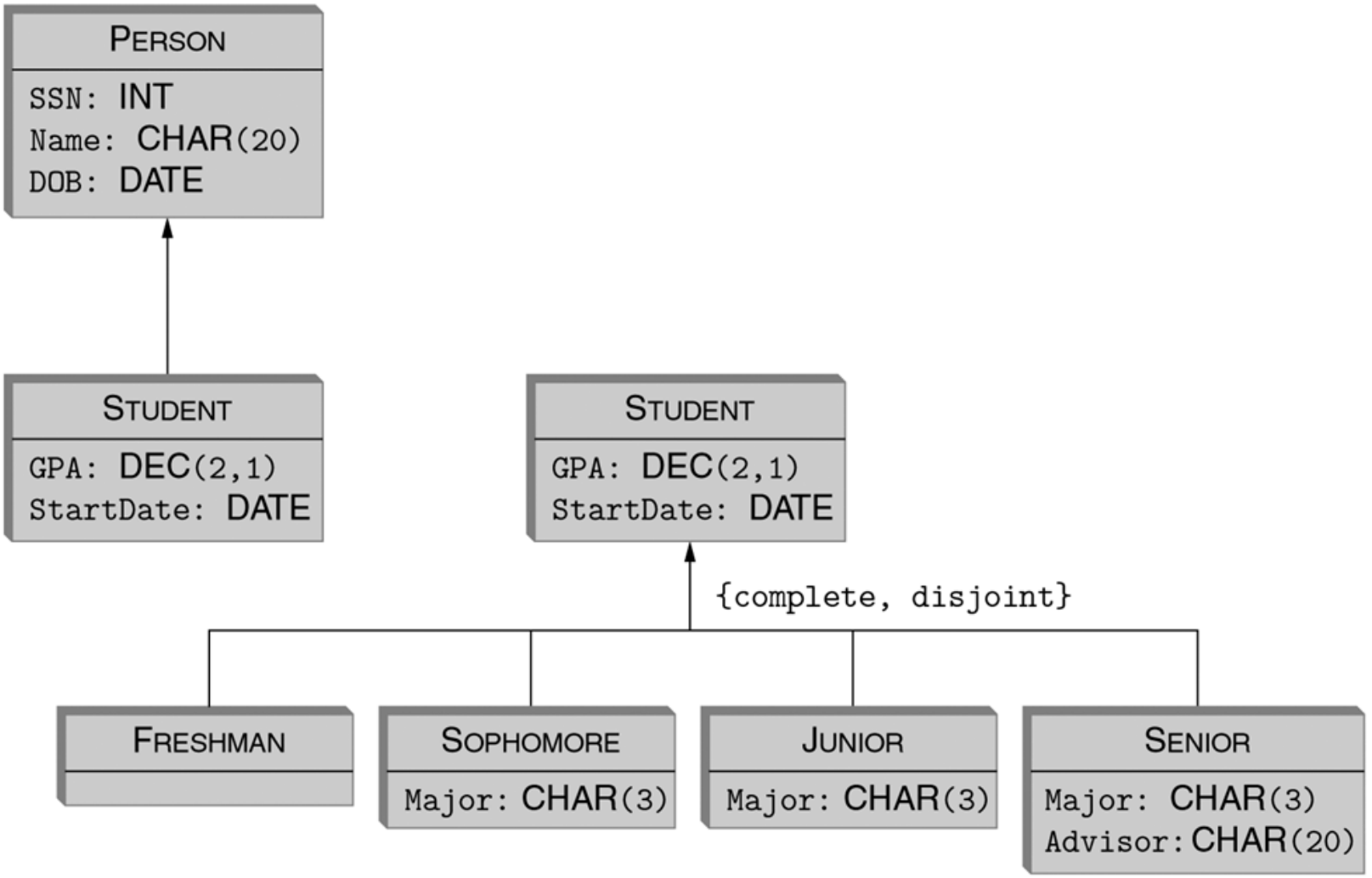
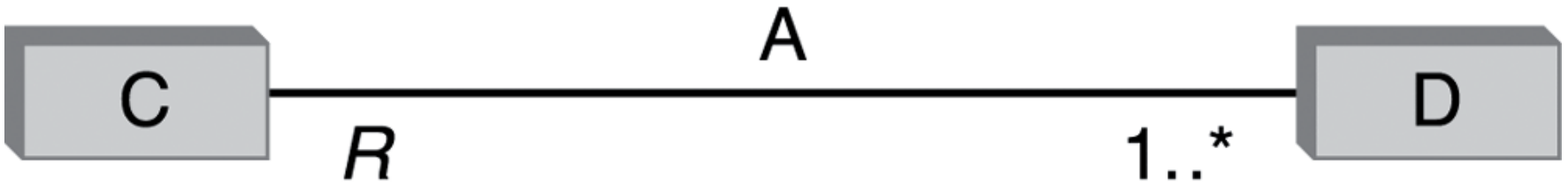


FIGURE 4.22 IsA (or generalization) hierarchies in UML.

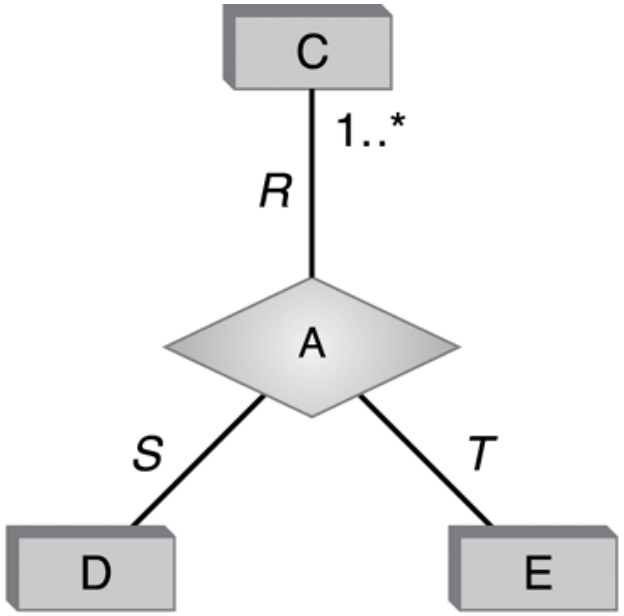
Participation constraints

- For binary relations:
  - the class C participates in a binary association A

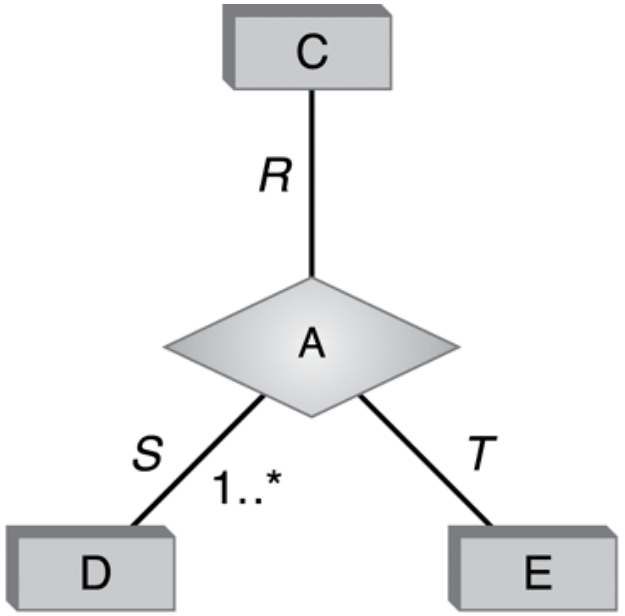


Participation constraints

- For ternary relations: not always possible.



(a) Participation in a ternary relationship in E-R



(b) Partial simulation of the same in UML

For every entity c in C there are entities d ∈ D and e ∈ E that participate in a relationship a ∈ A with c.

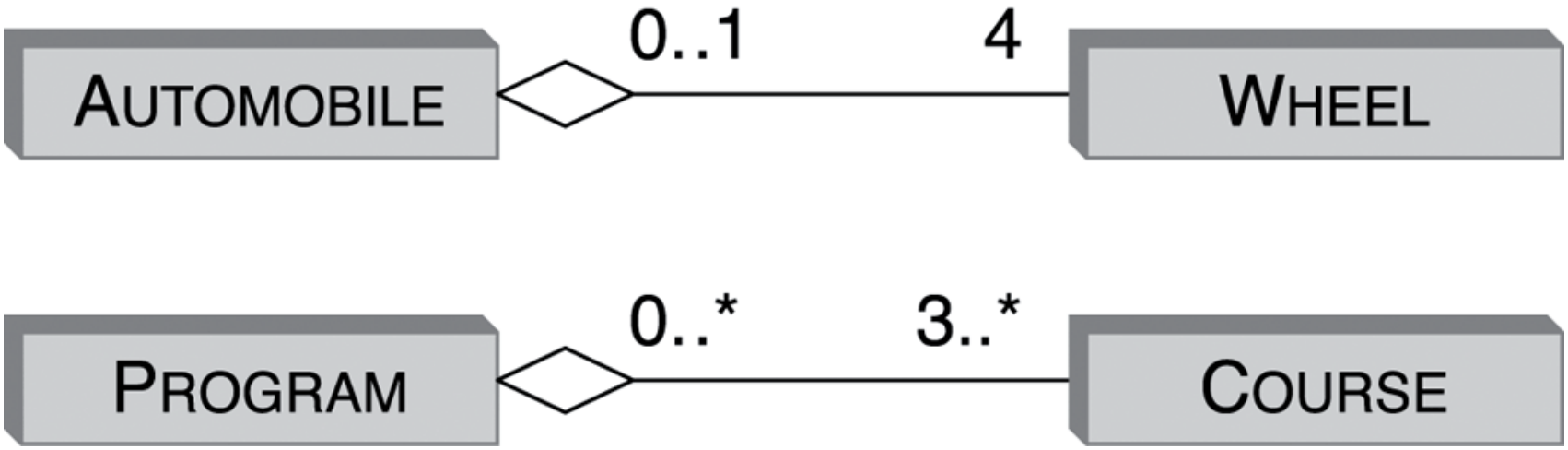
For every pair of objects c ∈ C and e ∈ E there is at least one object d ∈ D that participates in a relationship a ∈ A with c and e.

Participation constraints

- For ternary relations: not always possible.
  - One can put a constraint annotation on the role R {participates}
    - this type of annotation requires that all parties to the design understand what this means since this annotation does not have any built-in semantics in UML

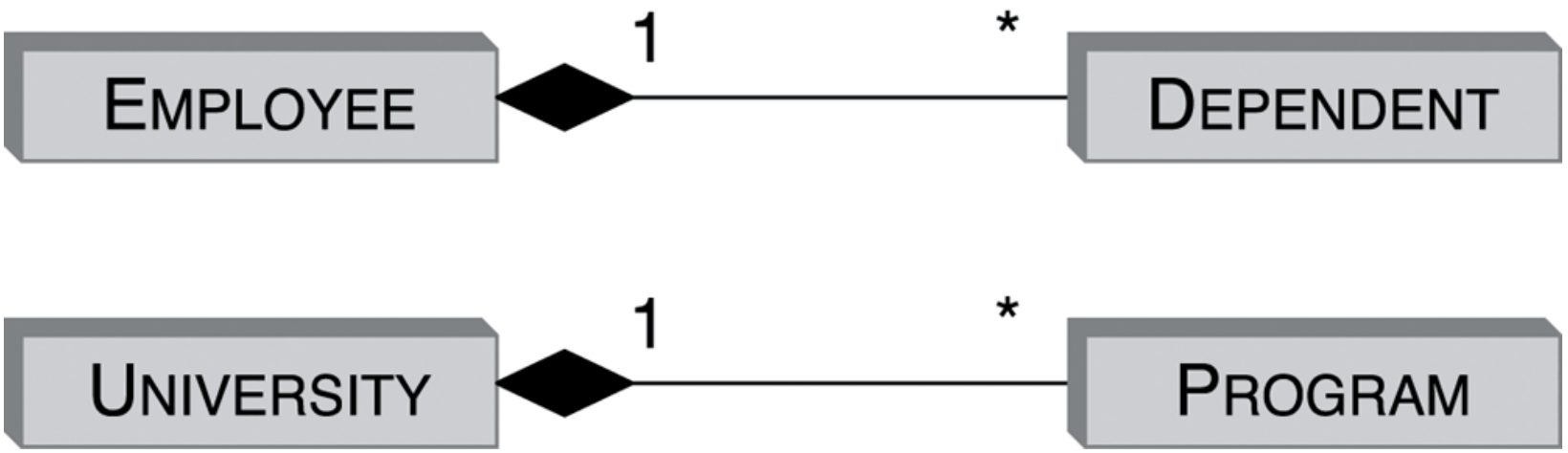
Part-of relationships

- non-exclusive part-of relationship = *aggregation*.
  - subparts can have independent existence

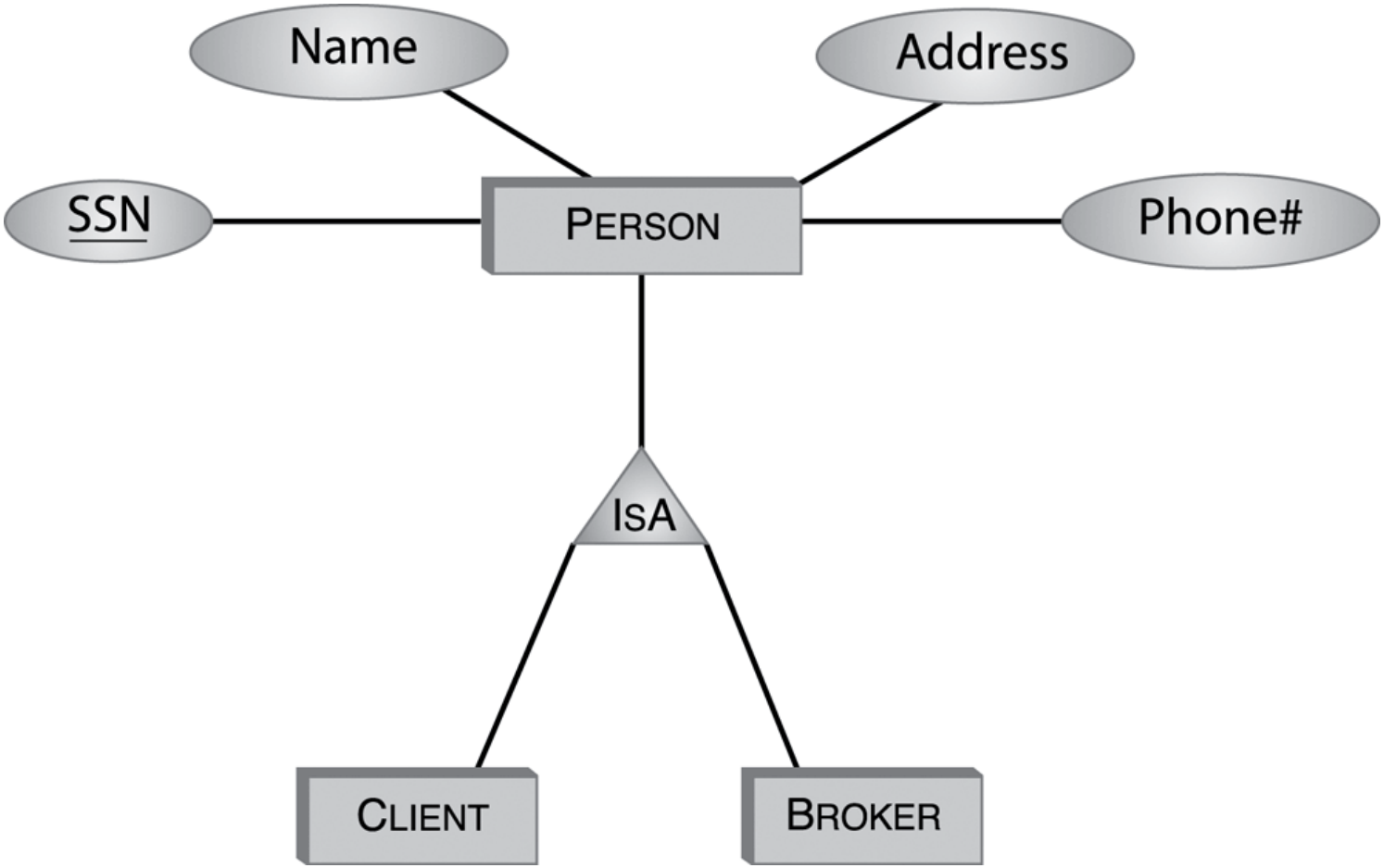


Part-of relationships

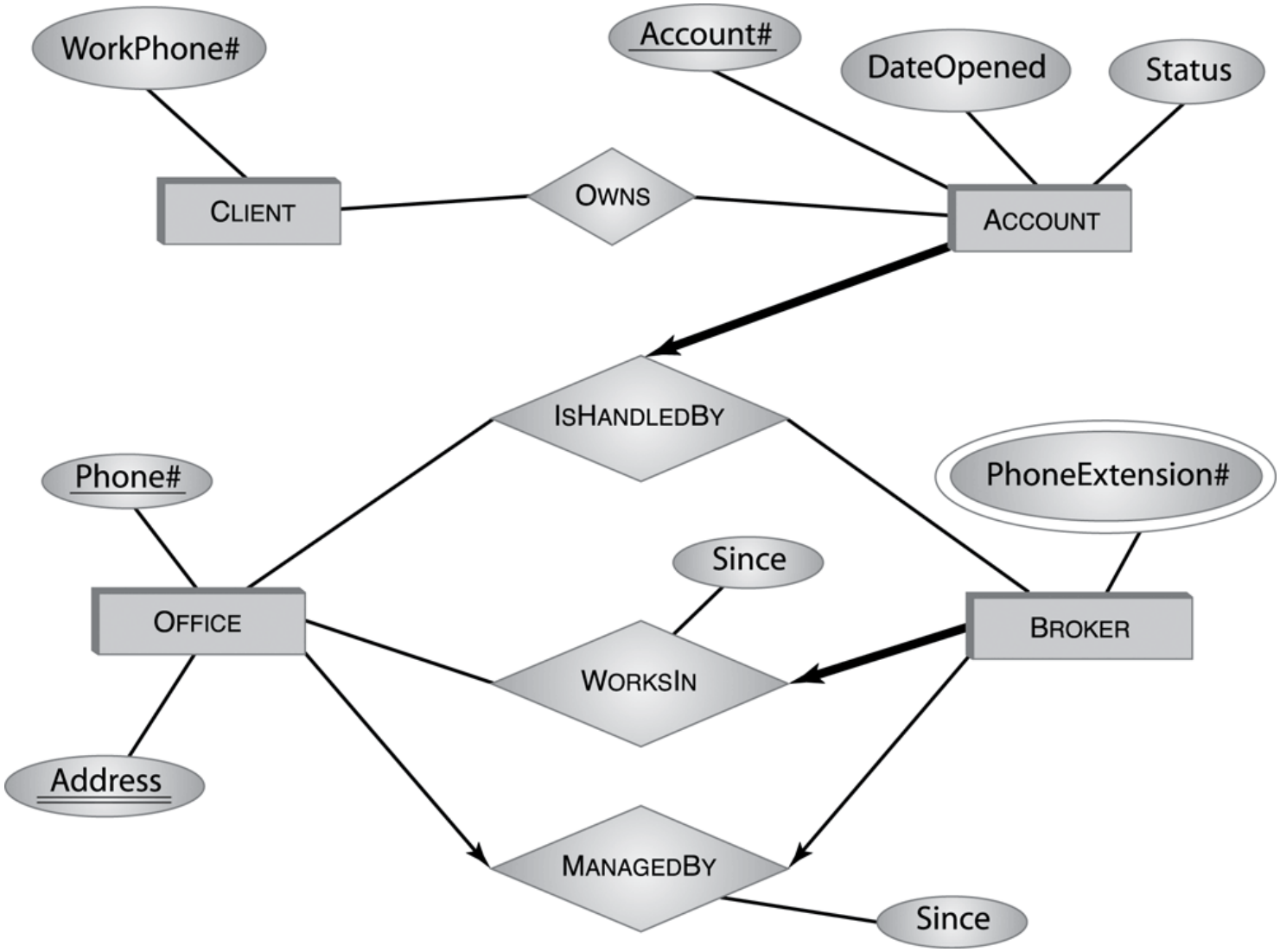
- exclusive part-of relationship = *composition*.
  - subparts must be destroyed when the master object is destroyed



A Brokerage Firm Example



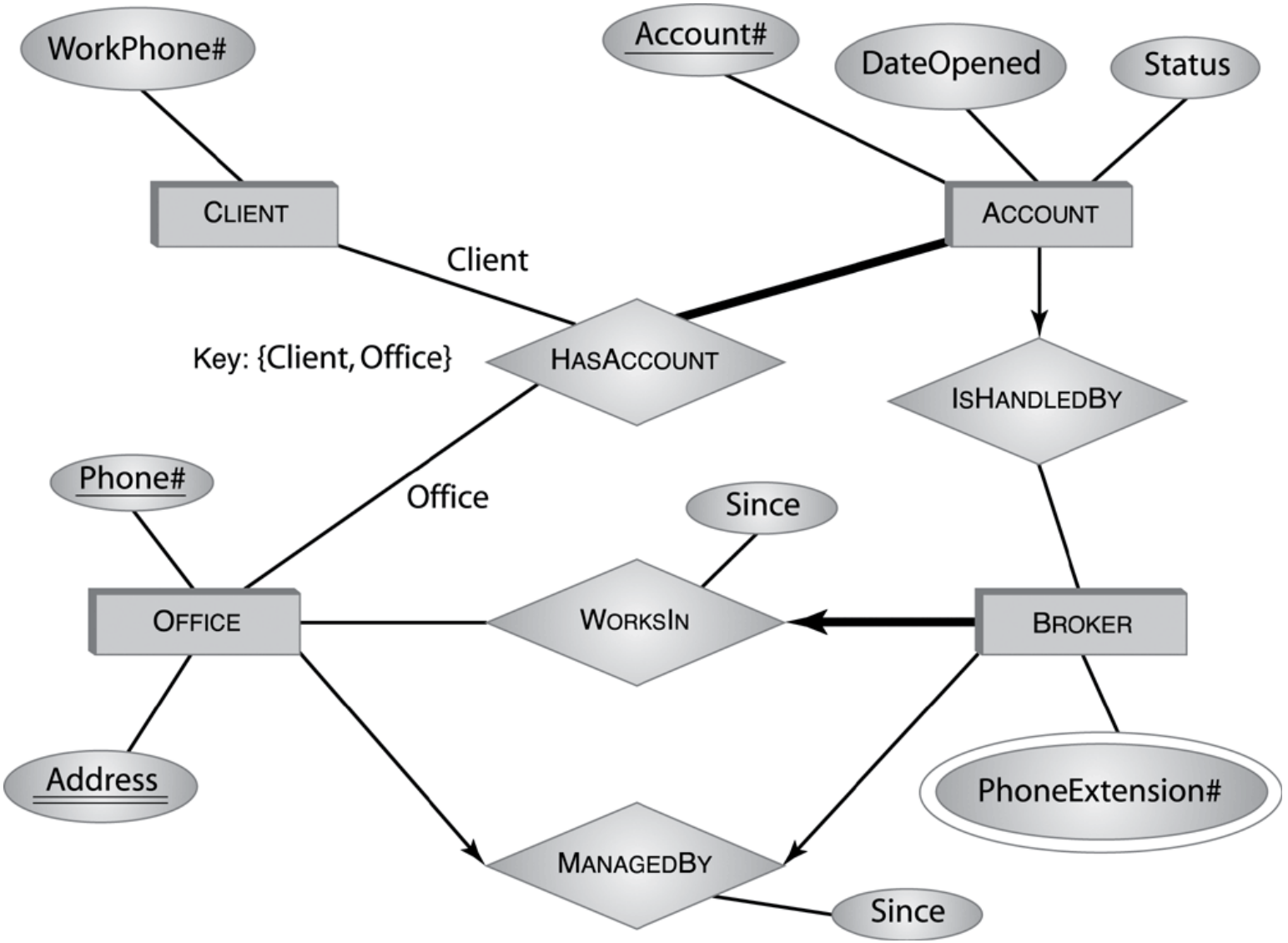
Client/broker information: first attempt.



A Brokerage Firm Example

- This E-R diagram has problems:
  - it requires every account to have a broker because the thick line from Account to IsHandledBy indicates that every account occurs in some relationship with a broker.
  - The constraint that a client cannot have two separate accounts in the same office is not represented in the diagram.

Client/broker information: second try.



Client/broker information in UML

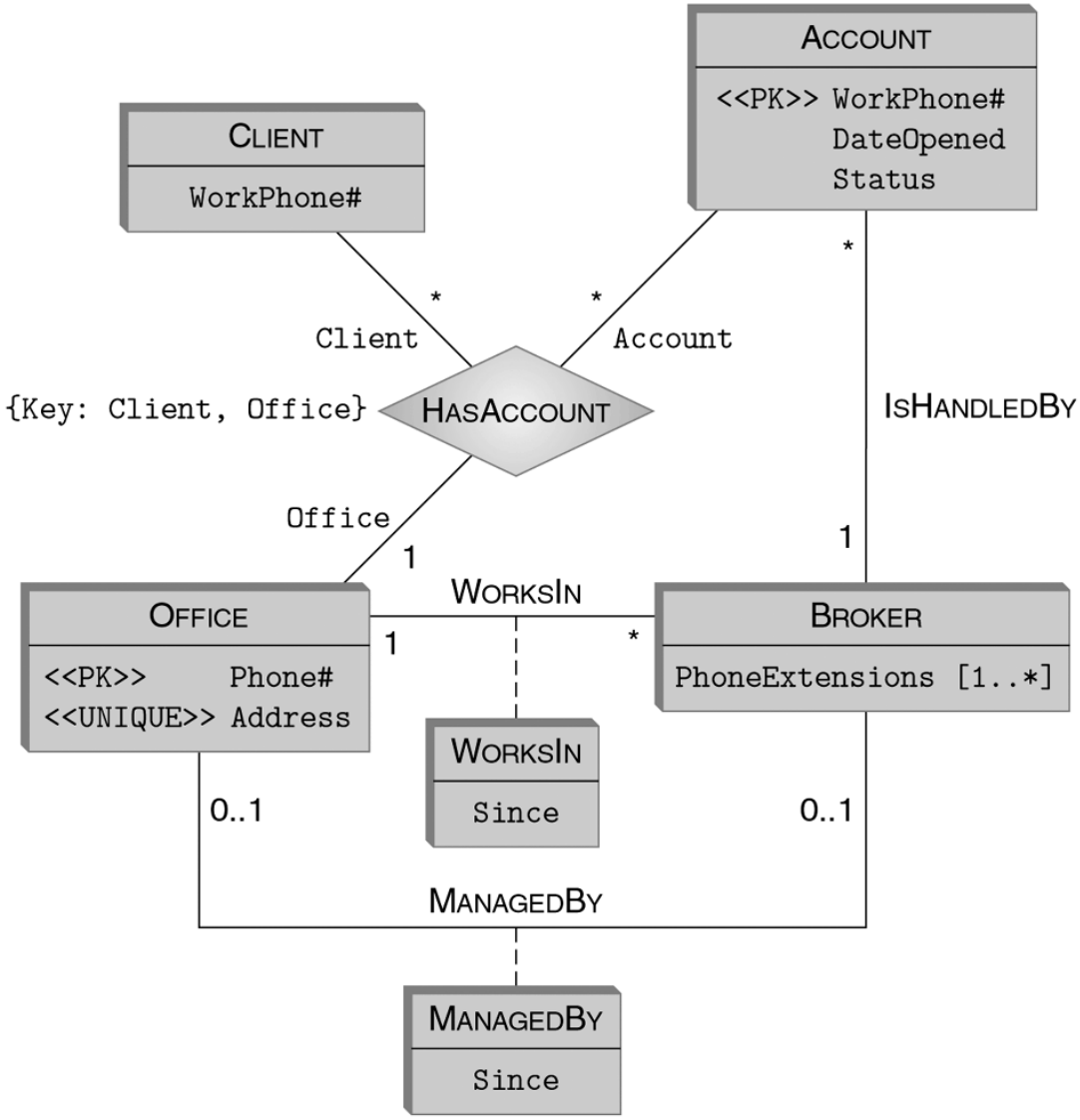
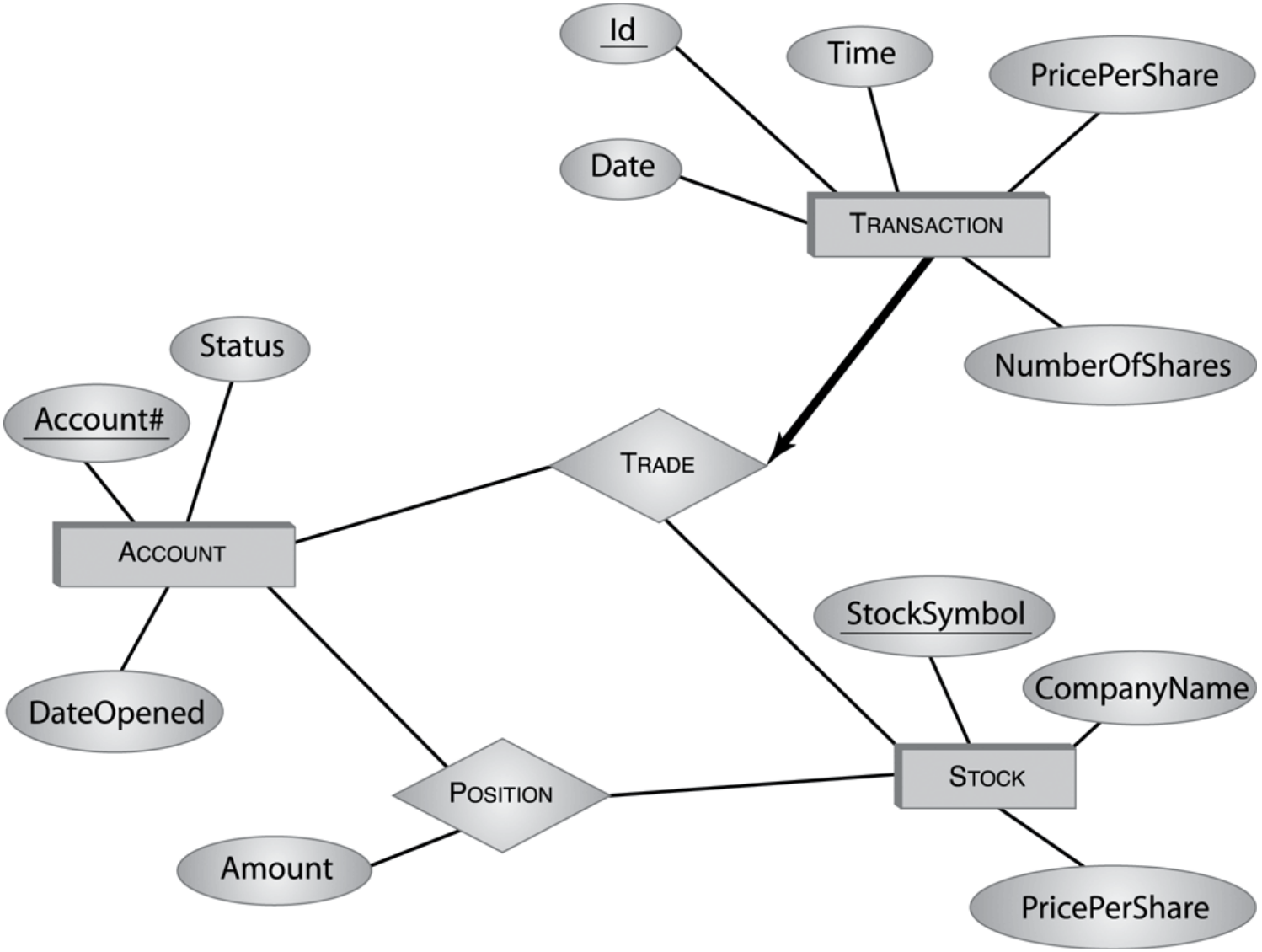


FIGURE 4.32 Client/broker information in UML.

Trading information



A database design for the Student Registration System

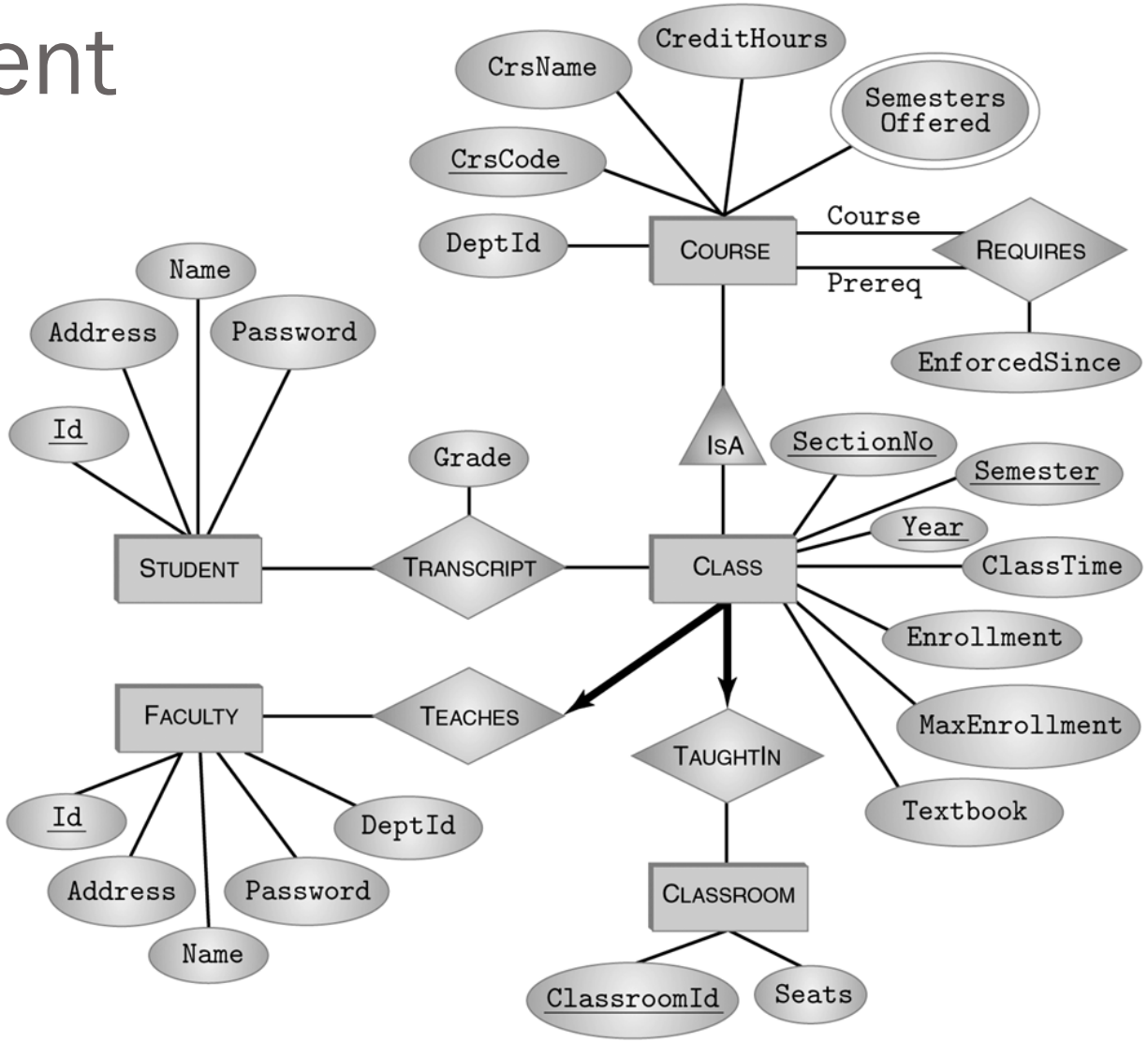


FIGURE 4.33 An E-R diagram for the Student Registration System.

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') )   )
```

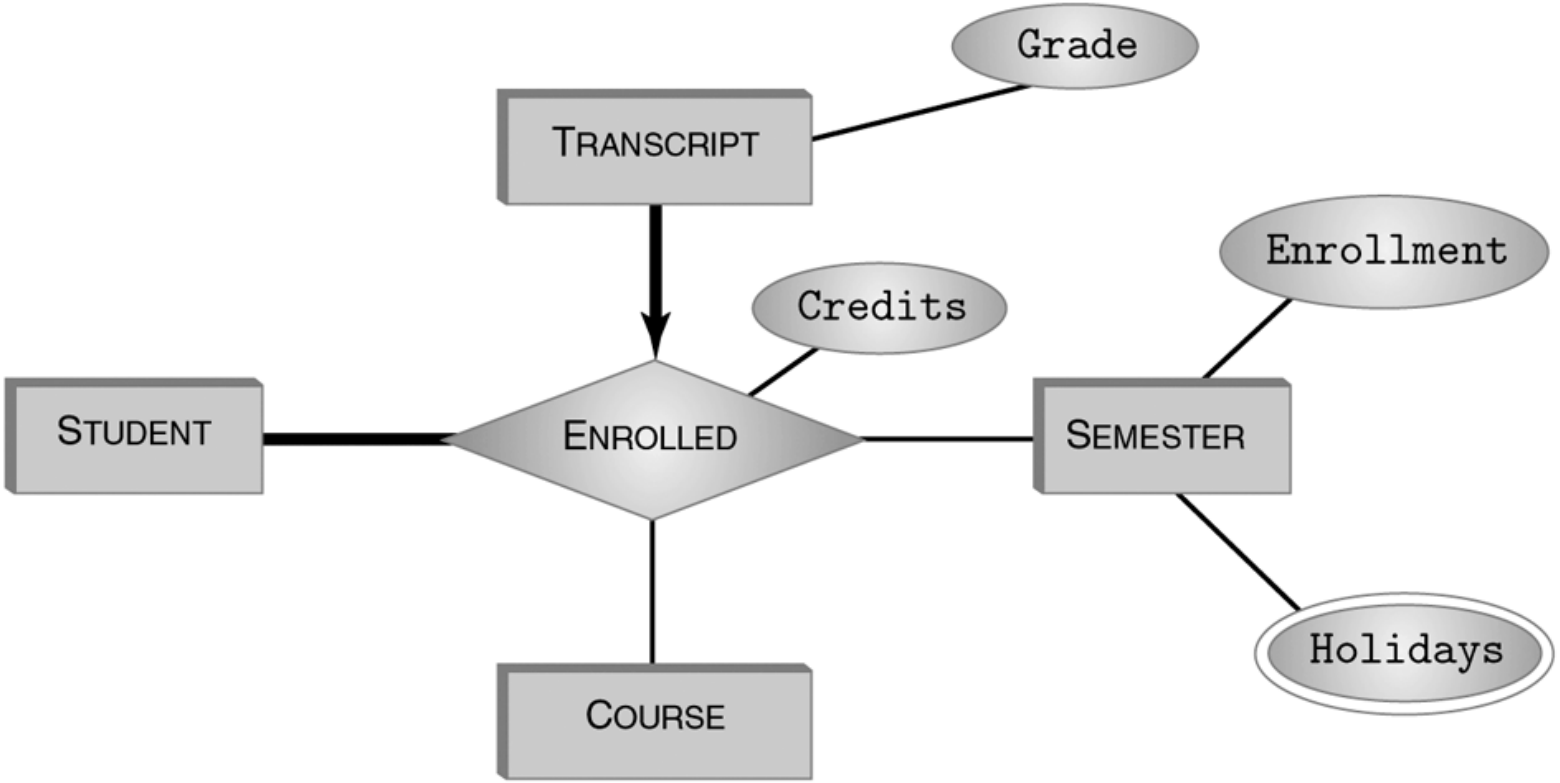
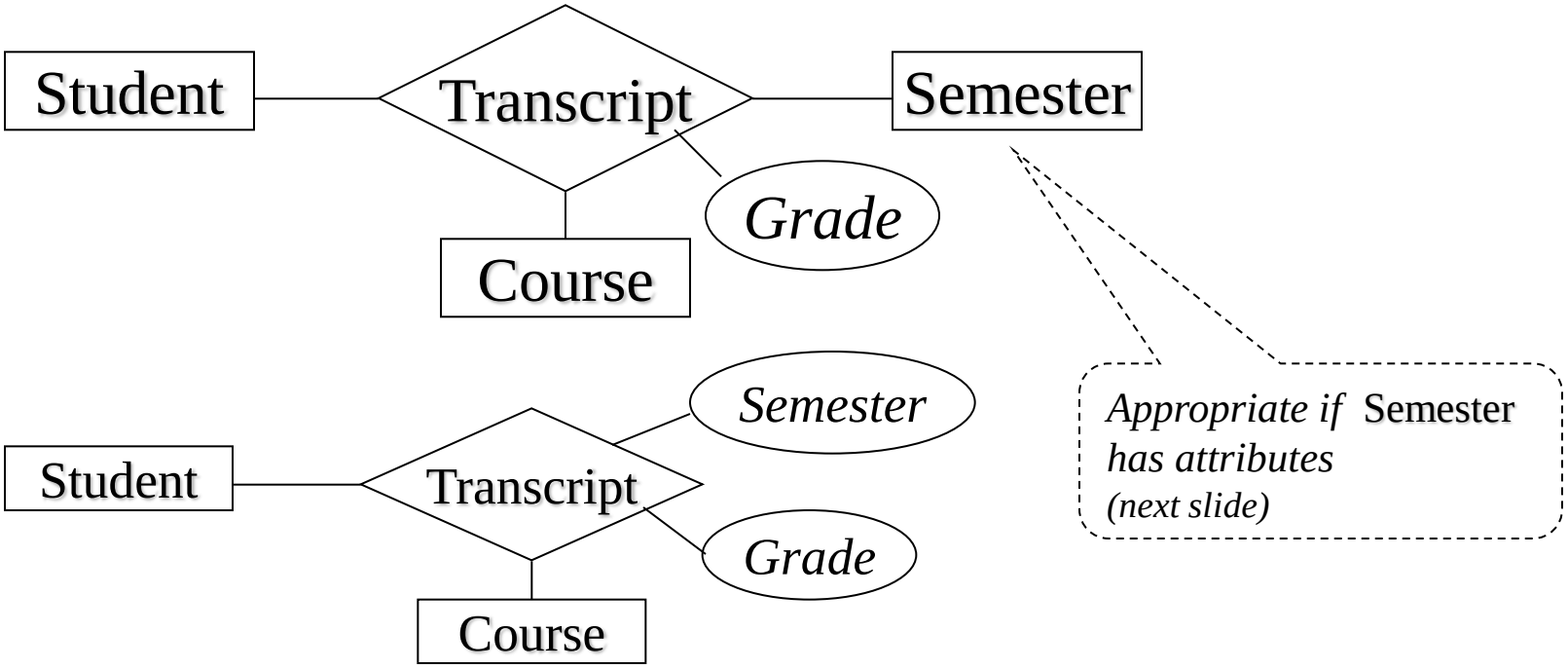


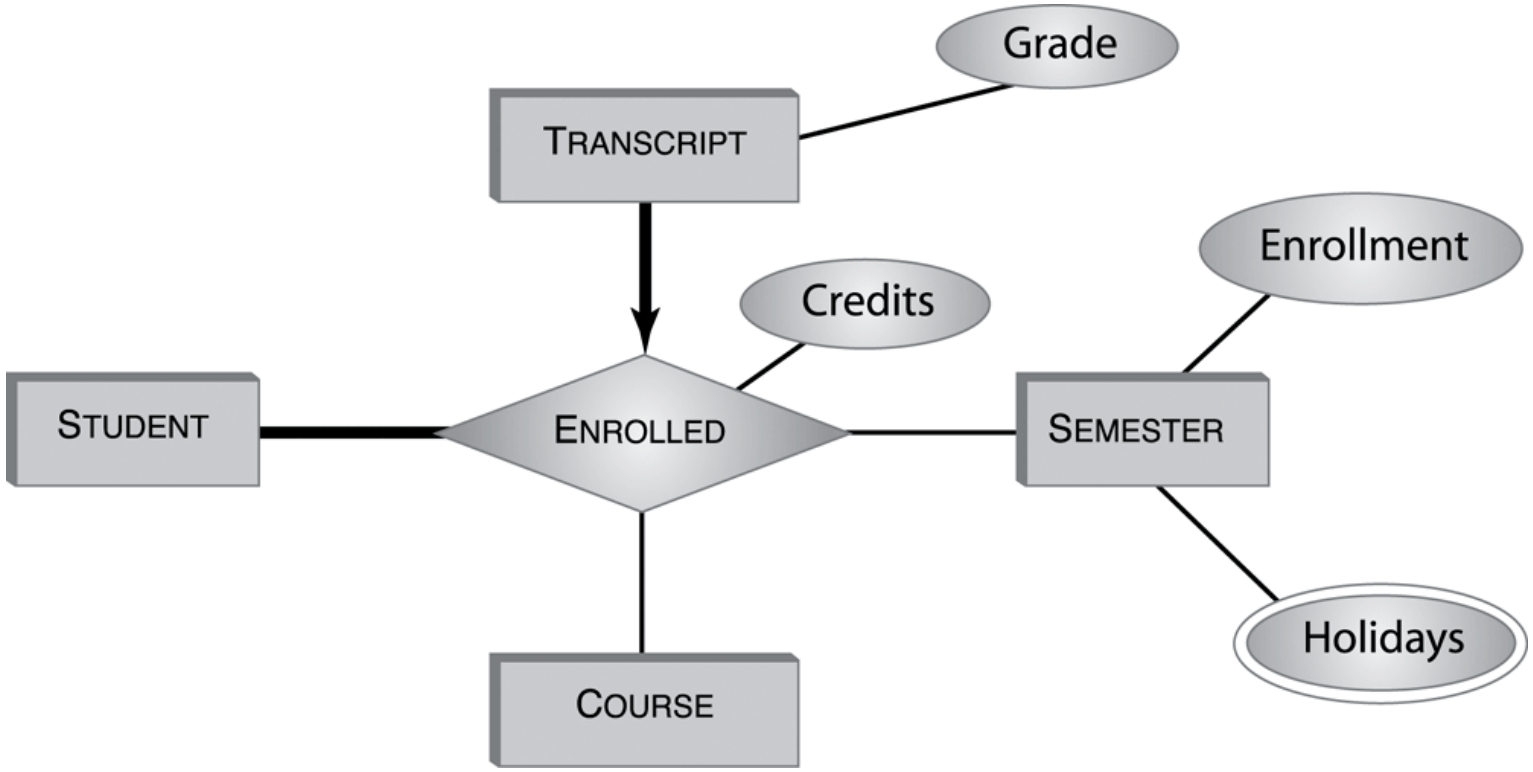
FIGURE 4.36 An alternative representation of the transcript information.

Limitations of the Data Modelling Methodologies

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

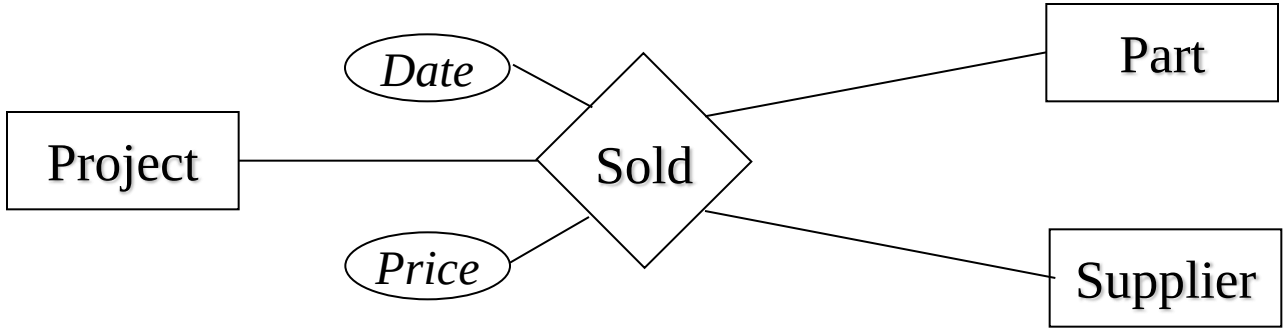


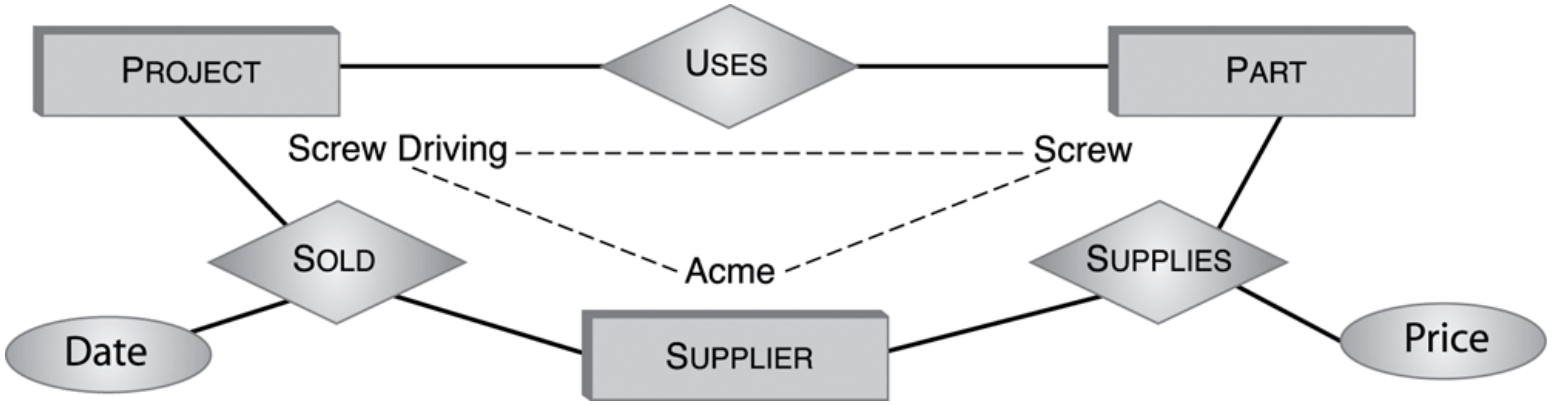
Entity or Relationship?



(Non-) Equivalence of Diagrams

- Transformations between binary and ternary relationships.





Universal Modeling Language

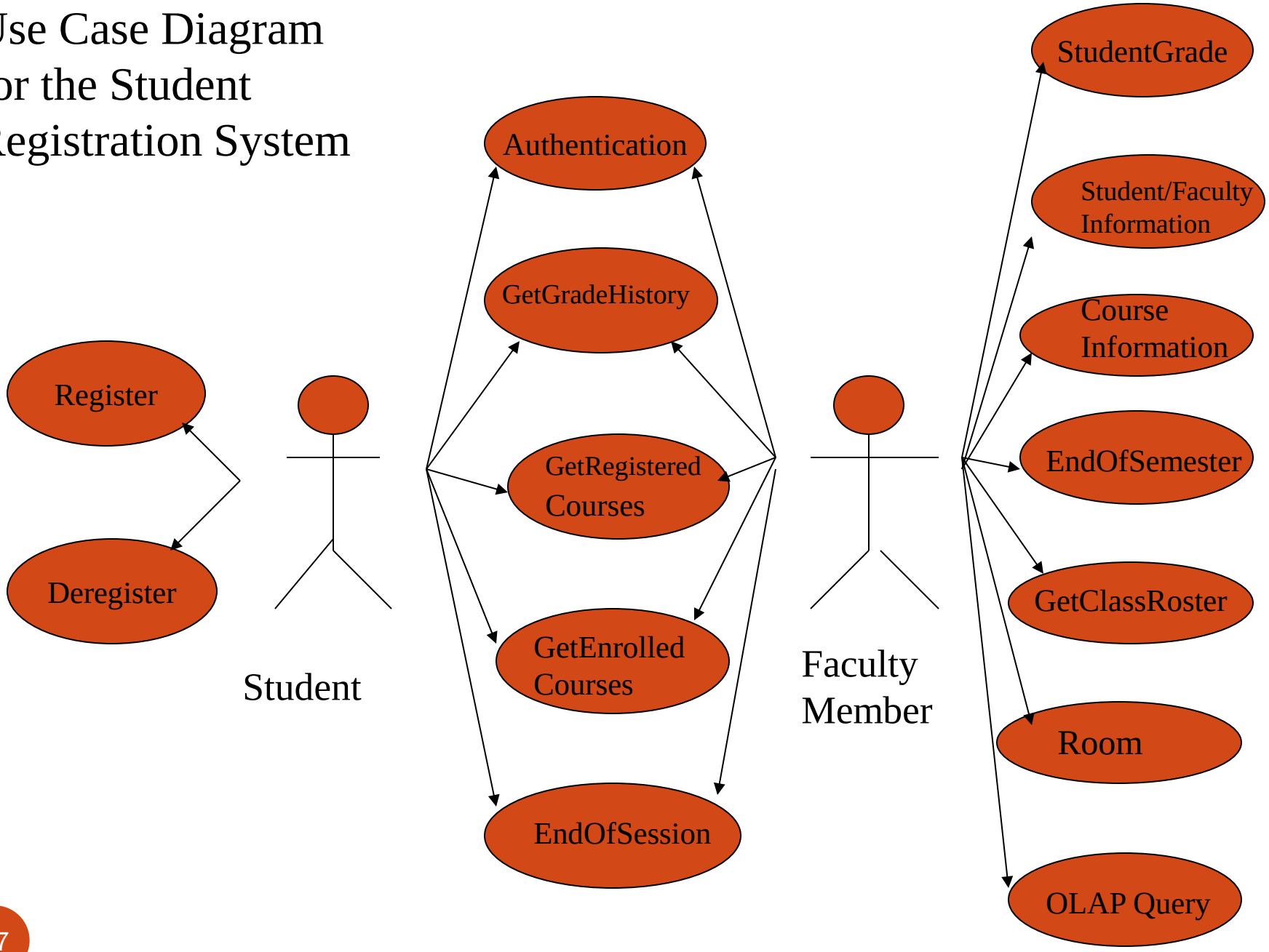
Supplemental material

- Use cases are part of the UML:
  - UML is also a graphic language for modeling 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

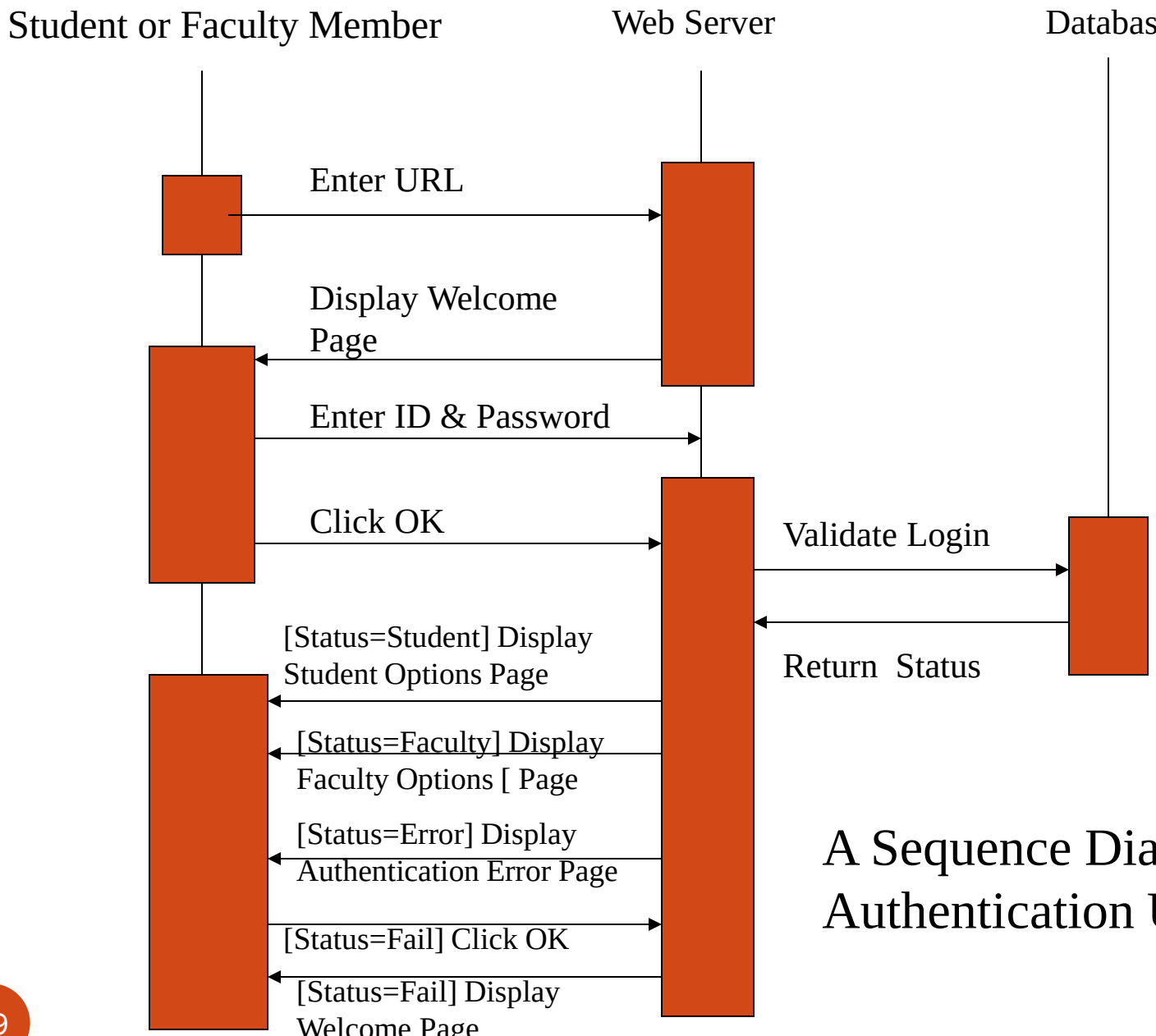
- UML provides a graphic way to display all the use cases in an application
- These diagrams can be used to communicate with the
  - Customer to determine if the current set of use cases is adequate
  - Developers to determine what the system is supposed to do from the customer's viewpoint

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

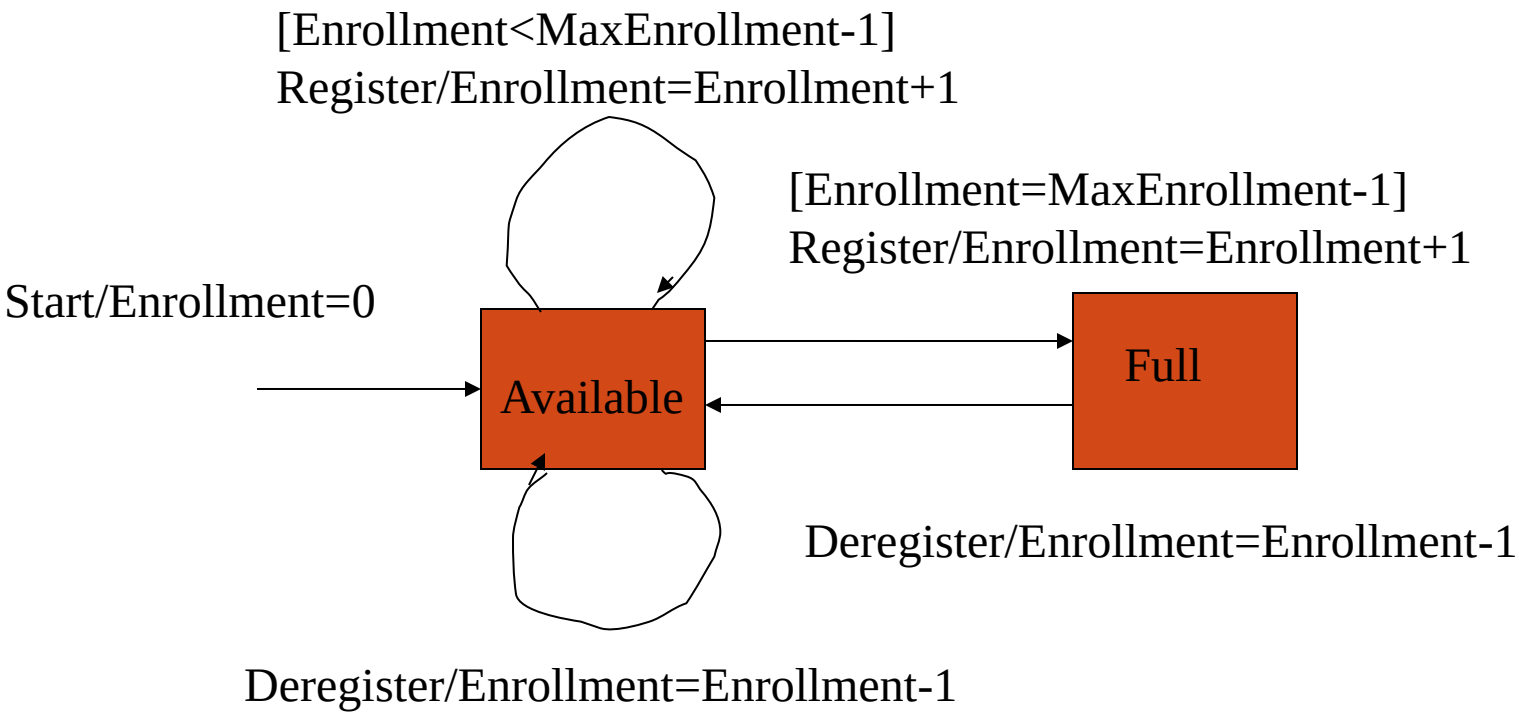
UML State Diagrams

- 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

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.



A UML State Diagram for the Class Object

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.