Conceptual Modeling of Databases with Entity-Relationship Diagrams and Unified Modeling Language (UML) Class Diagrams

Paul Fodor

CSE316: Fundamentals of Software Development

Stony Brook University

http://www.cs.stonybrook.edu/~cse316

Database Design

- 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:
    - Entity-relationship (E-R) diagrams
    - 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 (SQL) 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 database design
  - We 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 in the requirements.
      - Examples: a product, 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 product, 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 Smith, 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

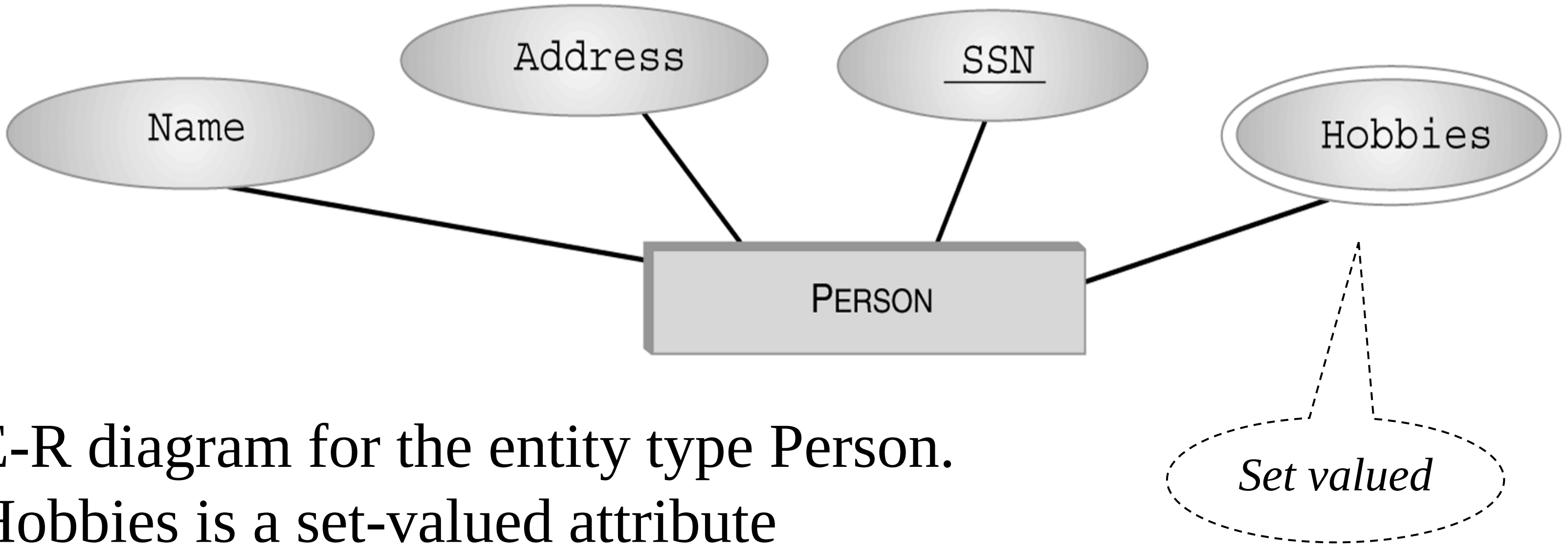
- An Entity type is described by set of attributes
  - E.g.: Person: Id, Name, Address, Hobbies
- ***Domain***: possible values of an attribute
- *An attribute value can be a set* (in contrast to relational model)
  - (111111, John, 123 Main St, {stamps, coins})
- ***Key***: a minimum set of attributes 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
  - *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 **<u>or more entities</u>**
  - John ***majors*** in Computer Science
- ***Relationship Type***: set of similar relationships
  - Student (entity type) related to Department (entity type) by **MajorsIn (relationship type).**

Relationships and Relationship Types

- Distinction between the DB relational model and the ER database design model:
  - relation (in the relational model) - set of tuples
  - relationship (in the E-R database design 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)

Roles

- A *role* of a relationship type names one of the related entities
  - e.g., John is value of <u>Student role</u>, CS value of <u>Department role</u> of MajorsIn relationship type
  - (John, CS; 2016) describes a relationship

Attributes

- Relationships can have attributes in the ER DB design model:
  - 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 - <u>value of SINCE attribute of MajorsIn relationship type</u>

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 relationship can relate elements of same entity type.

Roles

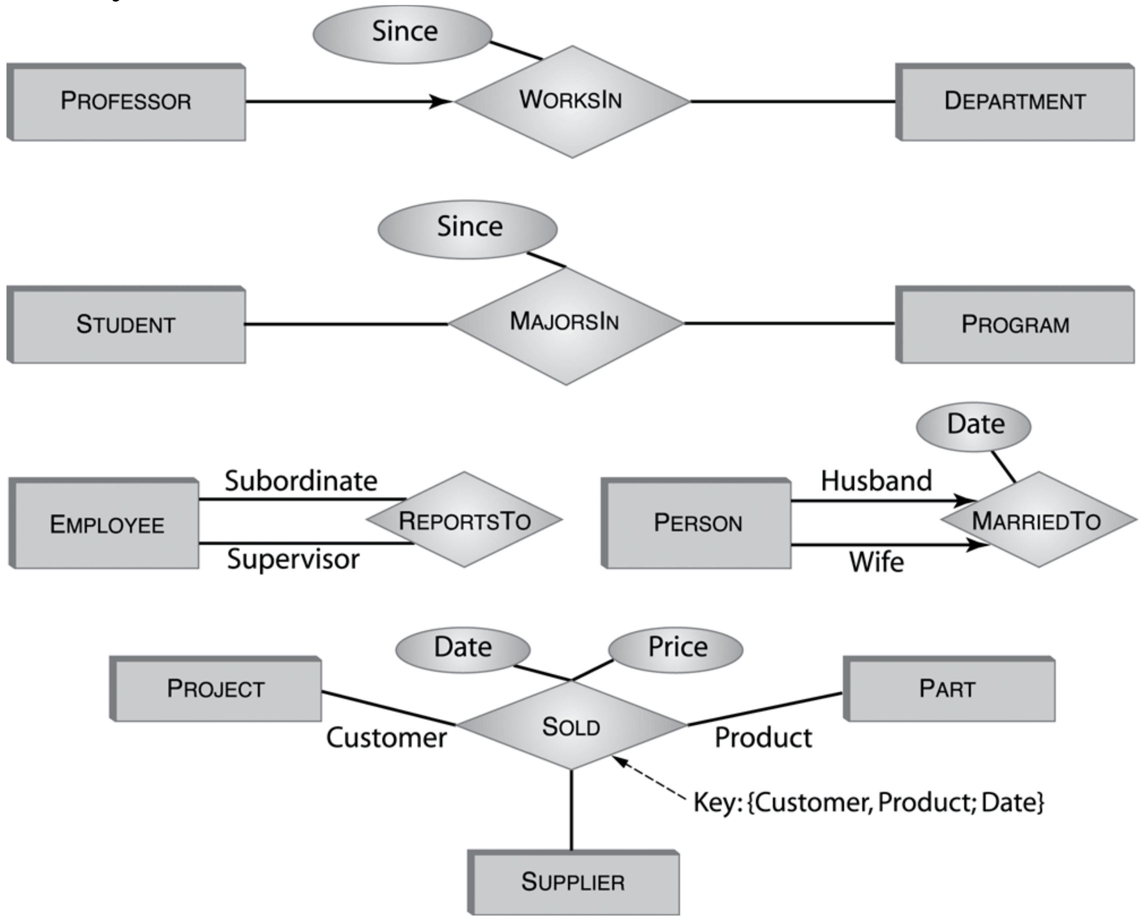
- Problem: relationship can relate elements of same entity type
  - e.g., ReportsTo relationship type relates two elements of Employee entity type
    - e.g., Bob reports to Mary since 2016
  - If we do not have distinct names for the roles, then it is not clear who reports to whom

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 the attribute Since
    - Values of Subordinate and Supervisor both drawn from entity type Employee

Graphical Representation

- Roles are edges labeled with role names (omitted if the role name = name of entity set).



Most attributes have been omitted.

Schema of a Relationship Type

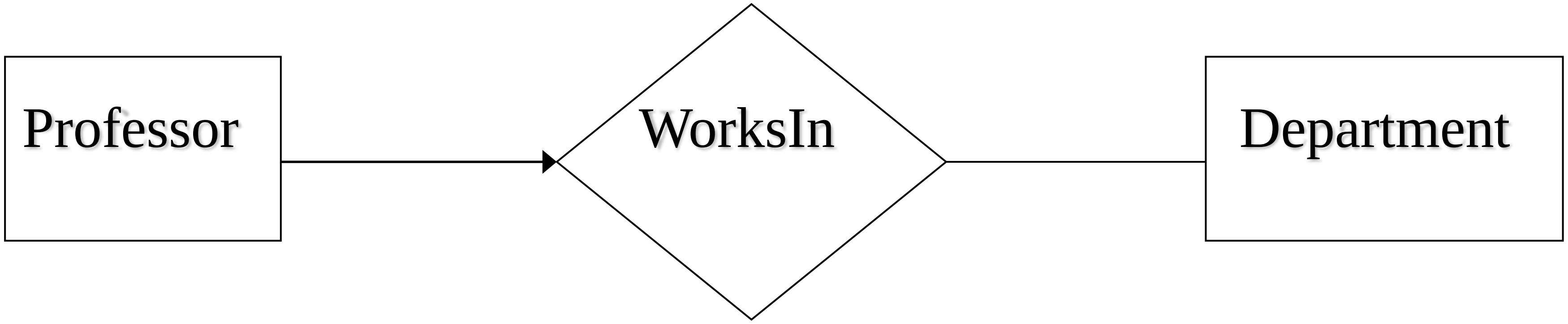
- The *schema of a relationship type* includes:
  - A list of attributes along with their corresponding domains (adverbs in natural language).
    - An attribute can be single-valued or set-valued.
  - A list of roles along with their corresponding entity types.
    - Unlike attributes, roles are always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

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**

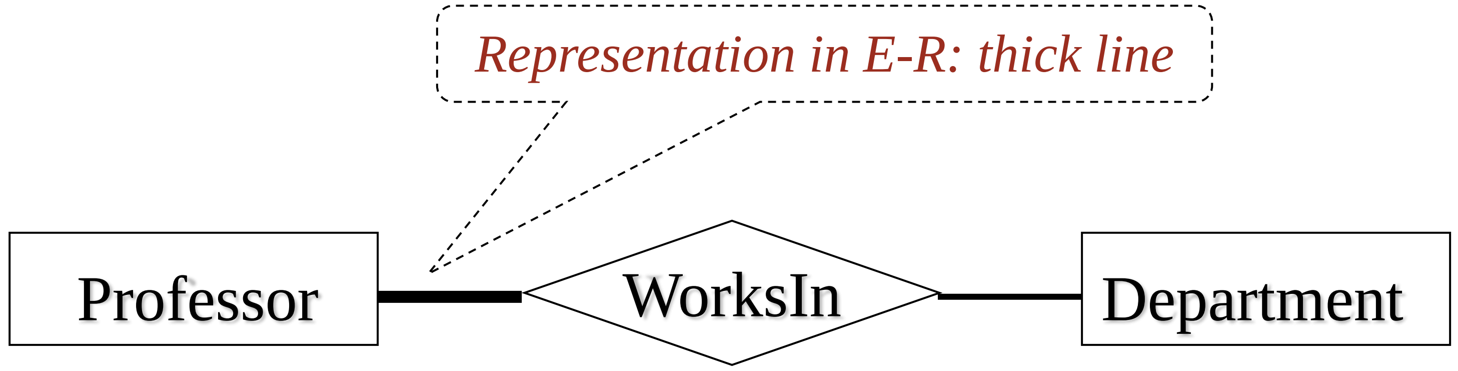


Graphical Representation

- There are many notation styles: Chen style, Bachman Style, Martin Style, crow foot ☺, EERD
  - we will use the original Chen style

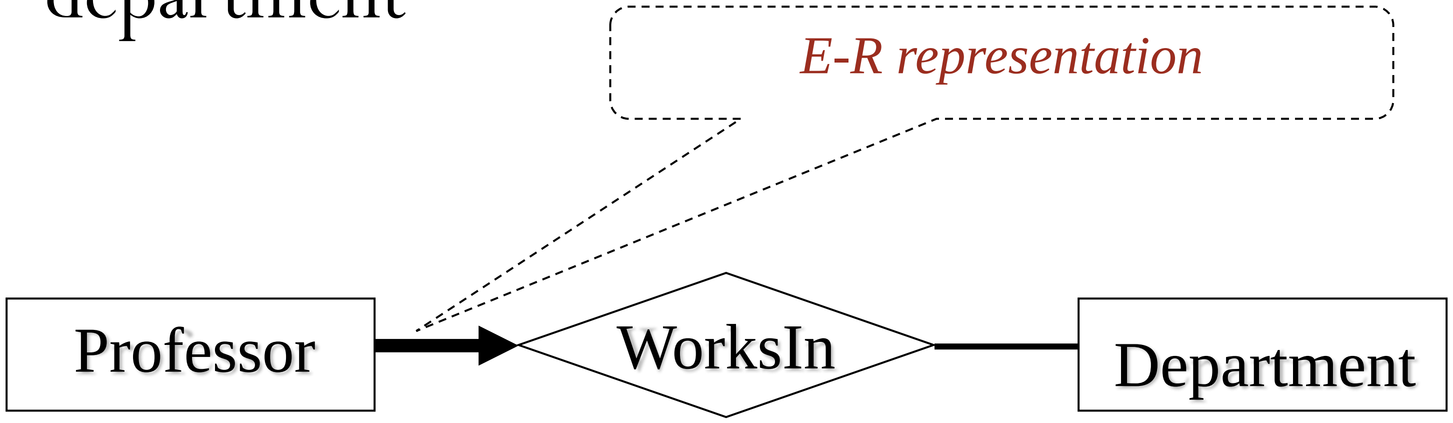
Participation Constraints

- If every entity participates in at least one relationship, a participation constraint holds:
  - e.g., every professor works in at least one department



Participation and Key Constraints

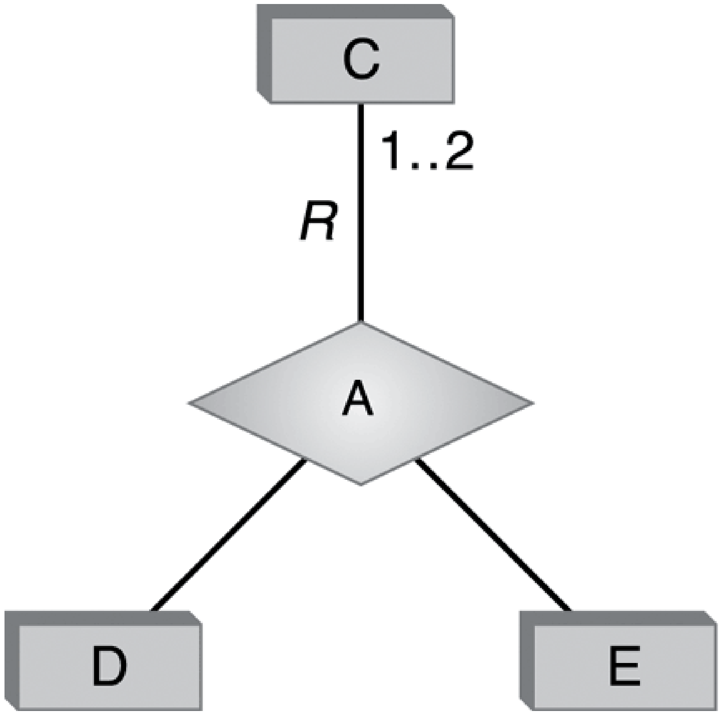
- If every entity participates in exactly one relationship, both a participation and a key constraint hold:
  - e.g., every professor works in exactly one department

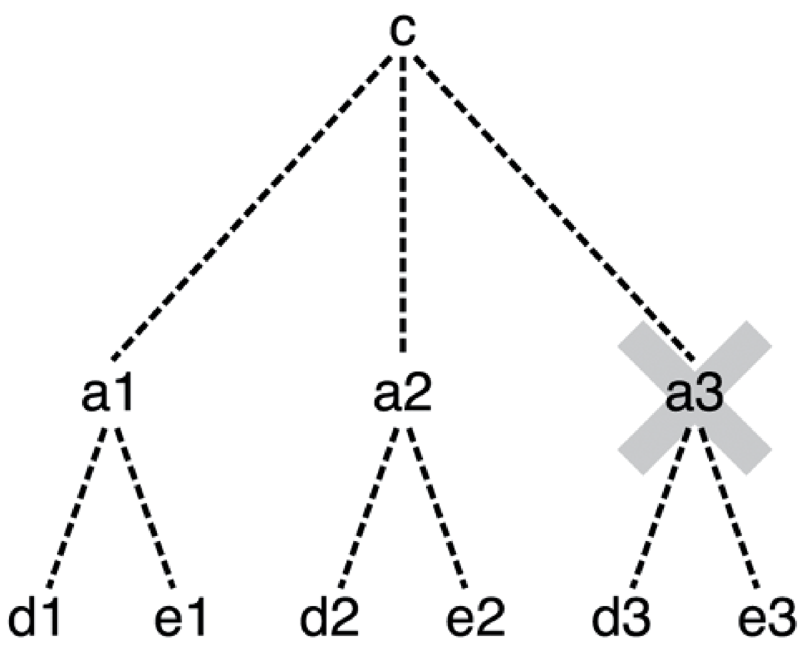


Cardinality constraints

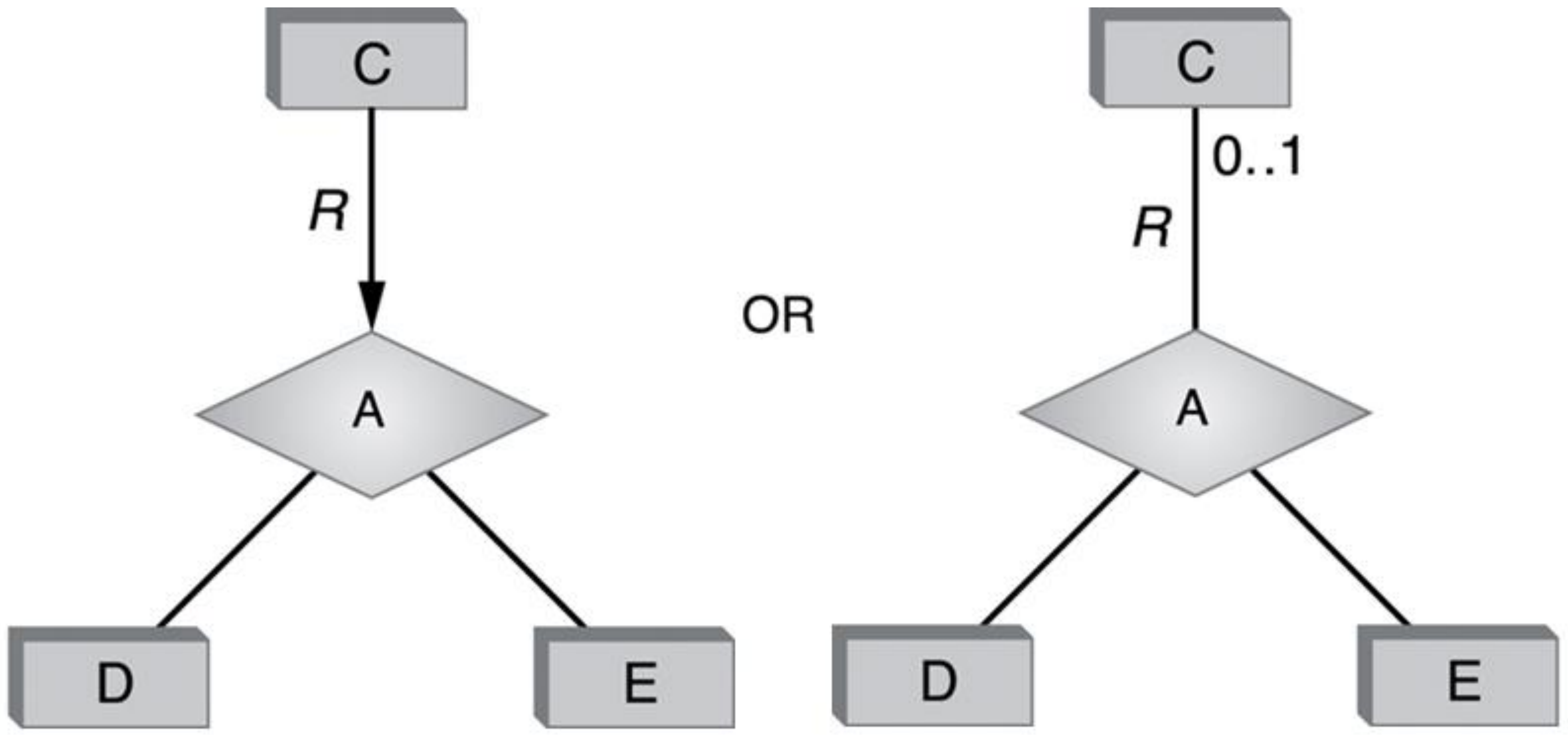
- Let C be an entity type and A be a relationship type that is connected to C via a role R.
  - A cardinality constraint on the role R is a statement of the form min..max attached to R and it restricts the number of relationship instances of type A in which a single entity of type C can participate in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

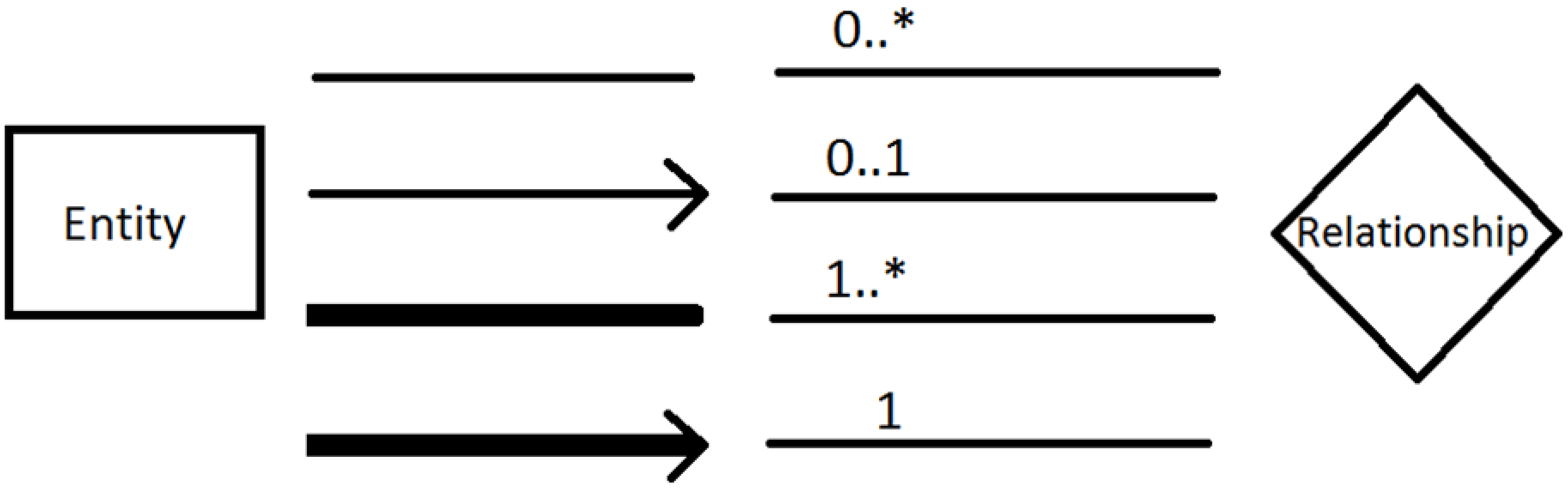


Cardinality constraints

- max can be the * symbol, which represents infinity

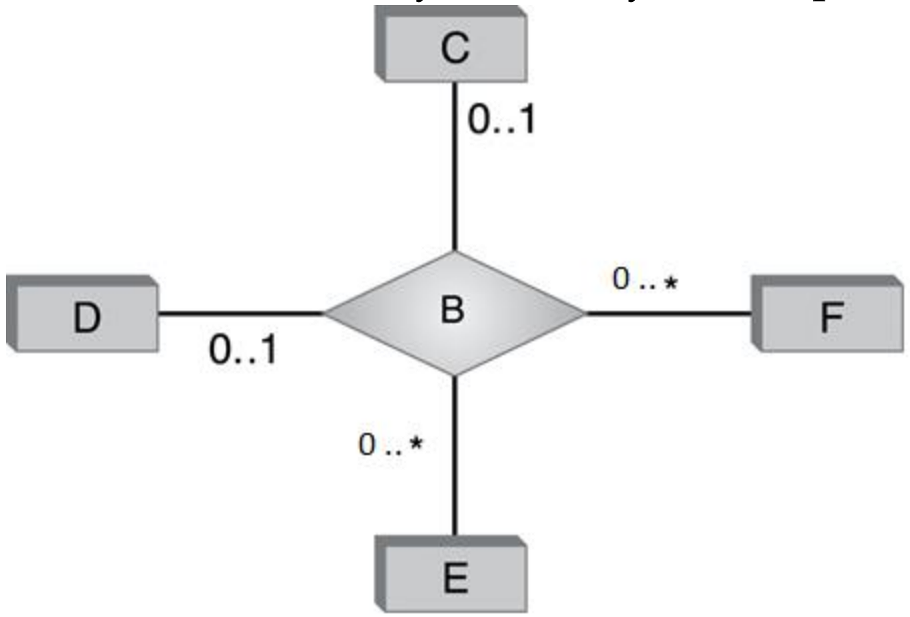
Indicator	**Meaning**
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only *n* (where *n* > 1)
0..n	Zero to *n* (where *n* > 1)
1..n	One to *n* (where *n* > 1)

Two ways to represent key constraints



Cardinality constraints

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

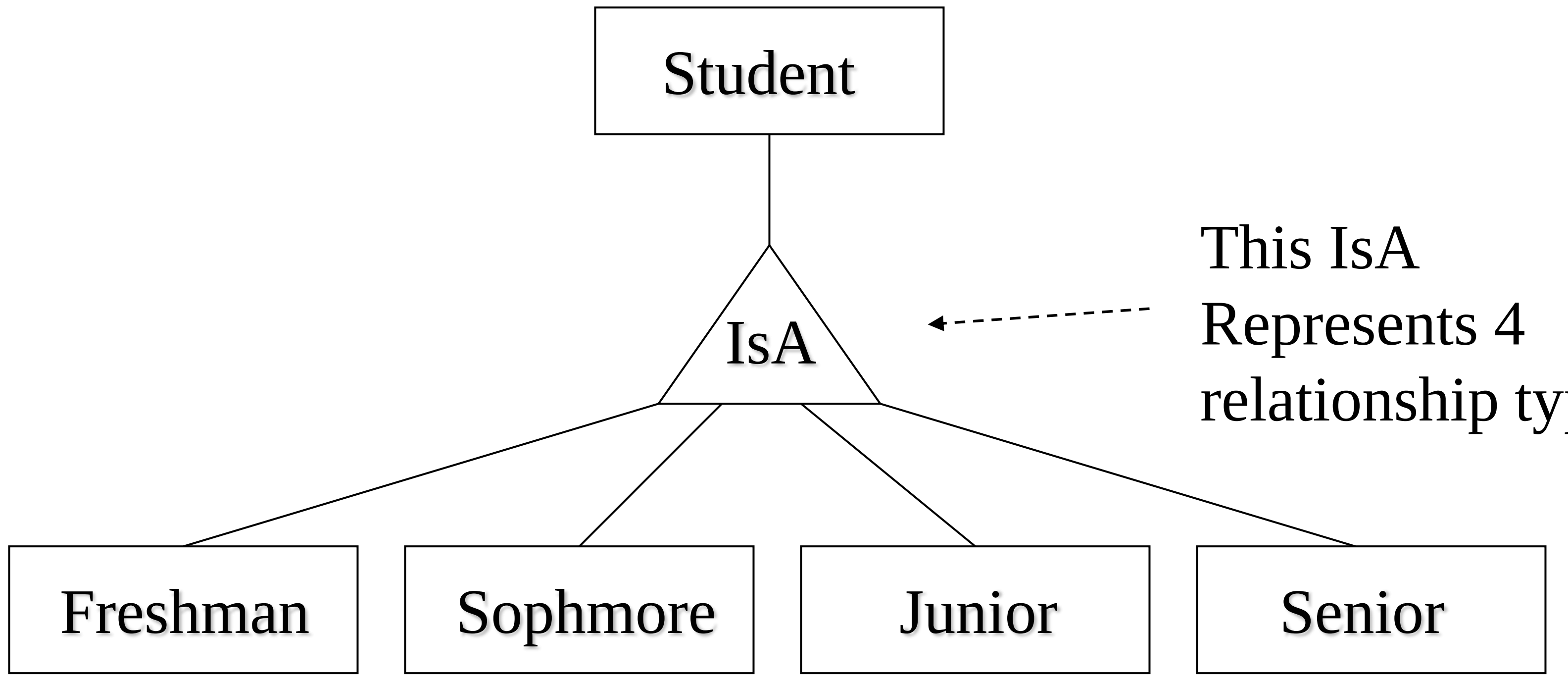


  - A *one-to-one* correspondence between the entity types C and D: an entity of type C can be associated with at most one entity of type D, and vice versa
  - A *one-to-many* correspondences of E to C and D: an entity of type E can be associated with any number (including zero) of entities of types C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

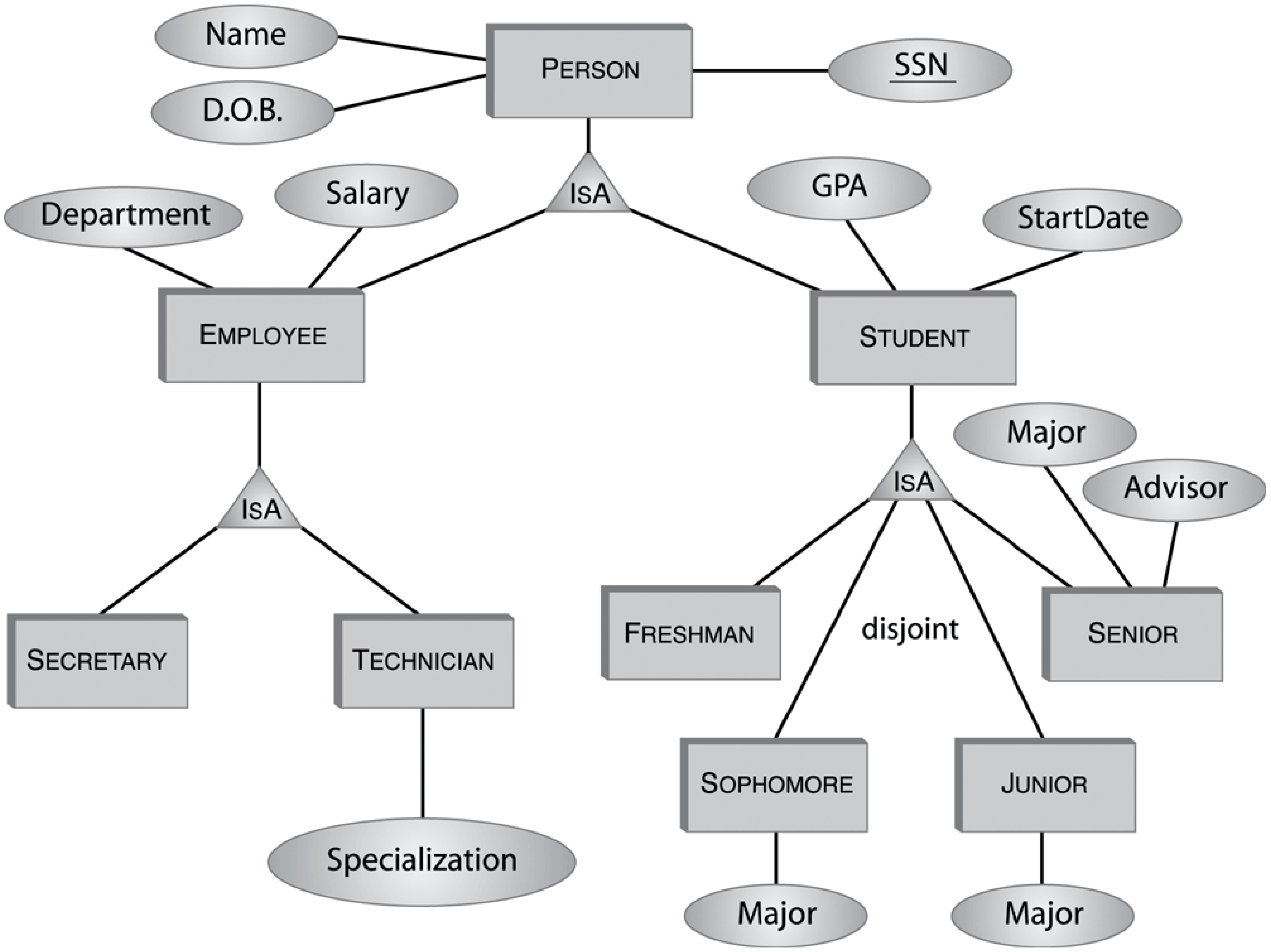


This IsA Represents 4 relationship types

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

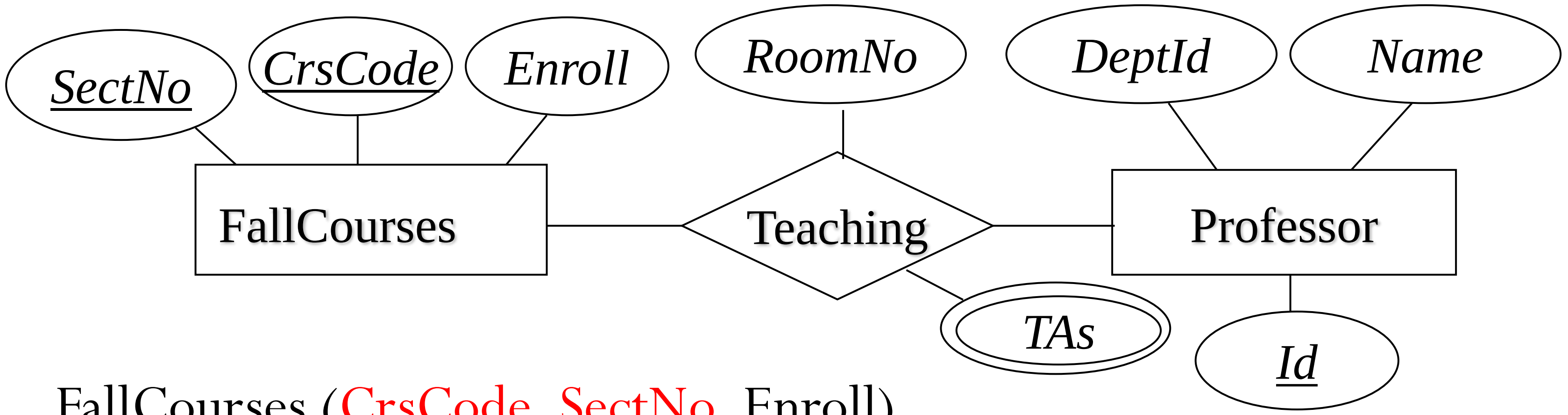
From E-R Diagrams to Relational Database Schemas

- 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

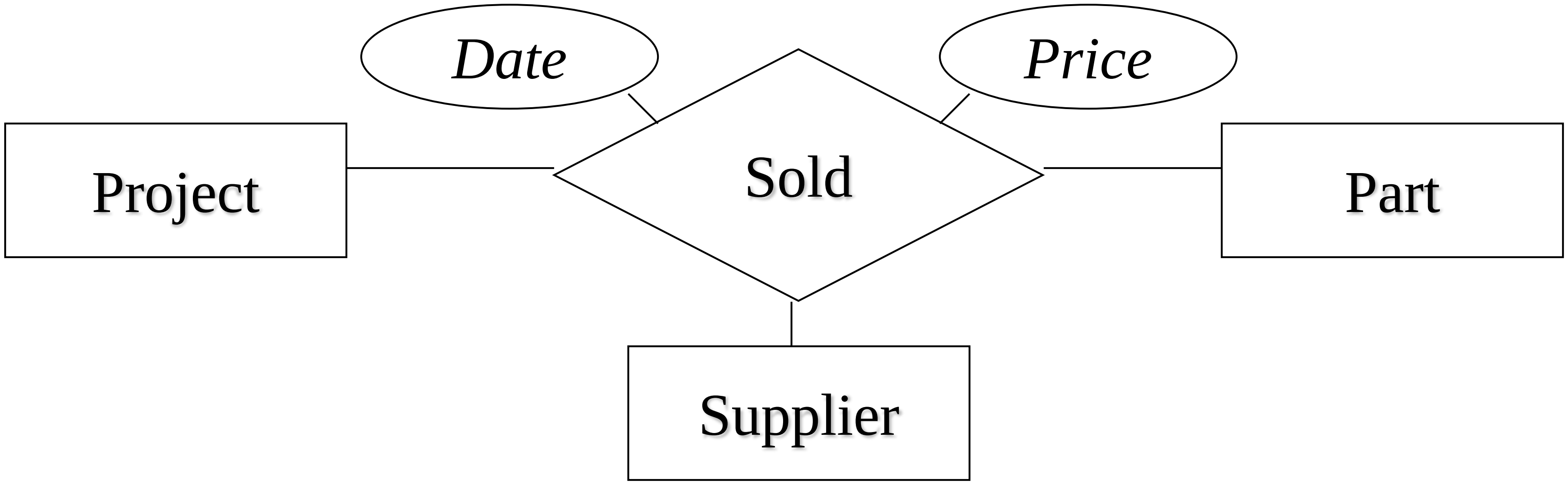
- Candidate key of corresponding table = candidate key of the relationship
  - Except when there are set valued attributes
    - Example: Teaching (CrsCode, SectNo, Id, RoomNo, TAs)
      - Key of relationship type = (CrsCode, SectNo)
      - Key of relation = (CrsCode, SectNo, TAs)

Set valued

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

Representation in SQL

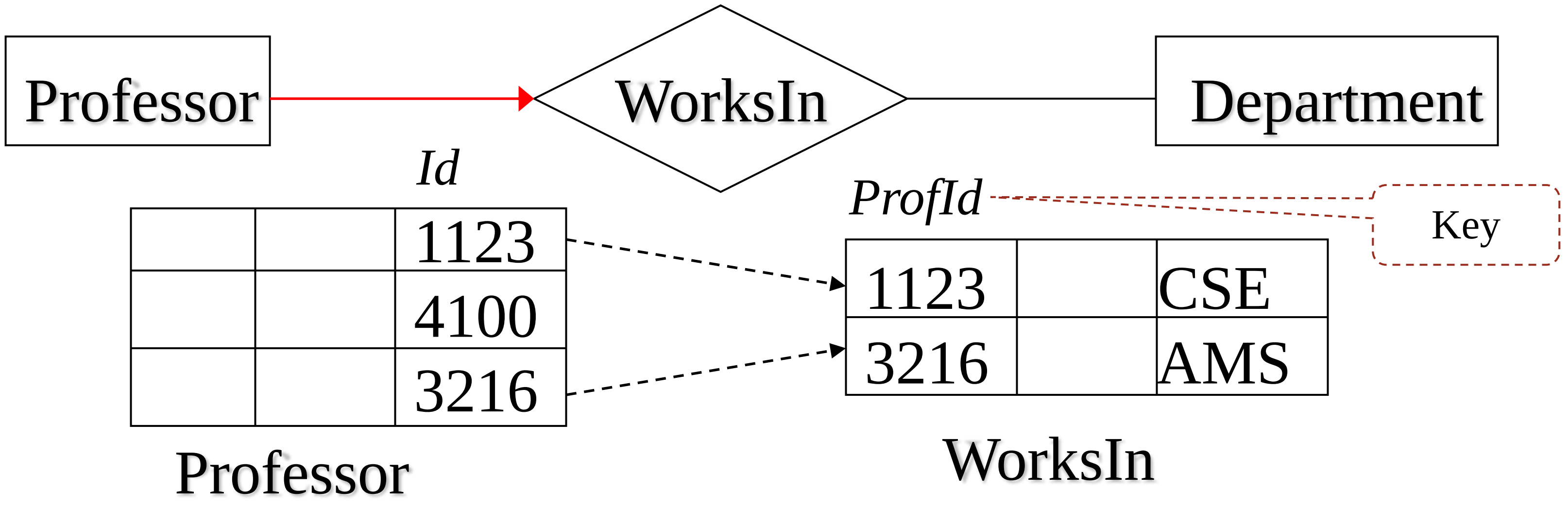
- Each role of relationship type produces a foreign key in corresponding relation
  - Foreign key references table corresponding to entity type from which role values are drawn



```
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)  )
```

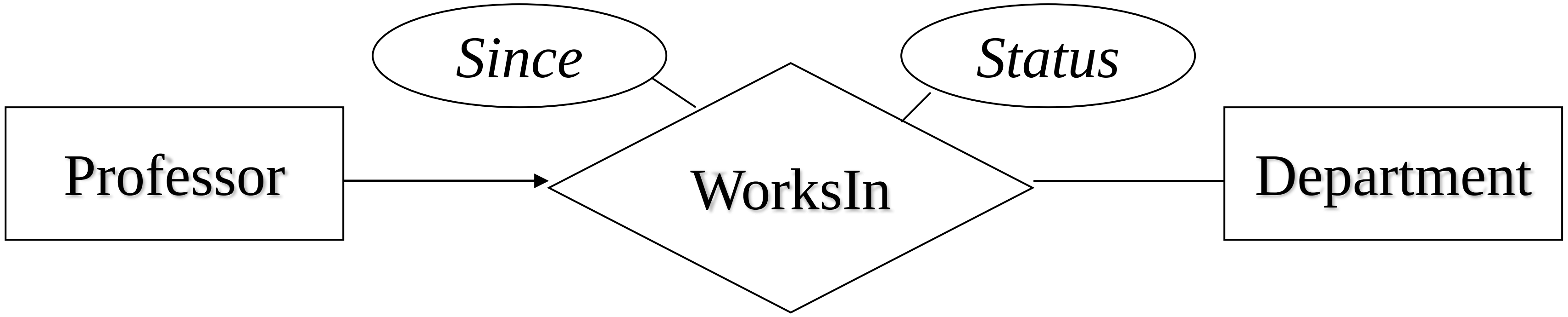
Representation of Single Role Key Constraints in the Relational Model

- 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.



		Id
		1123
		4100
		3216

ProfId		
1123		CSE
3216		AMS



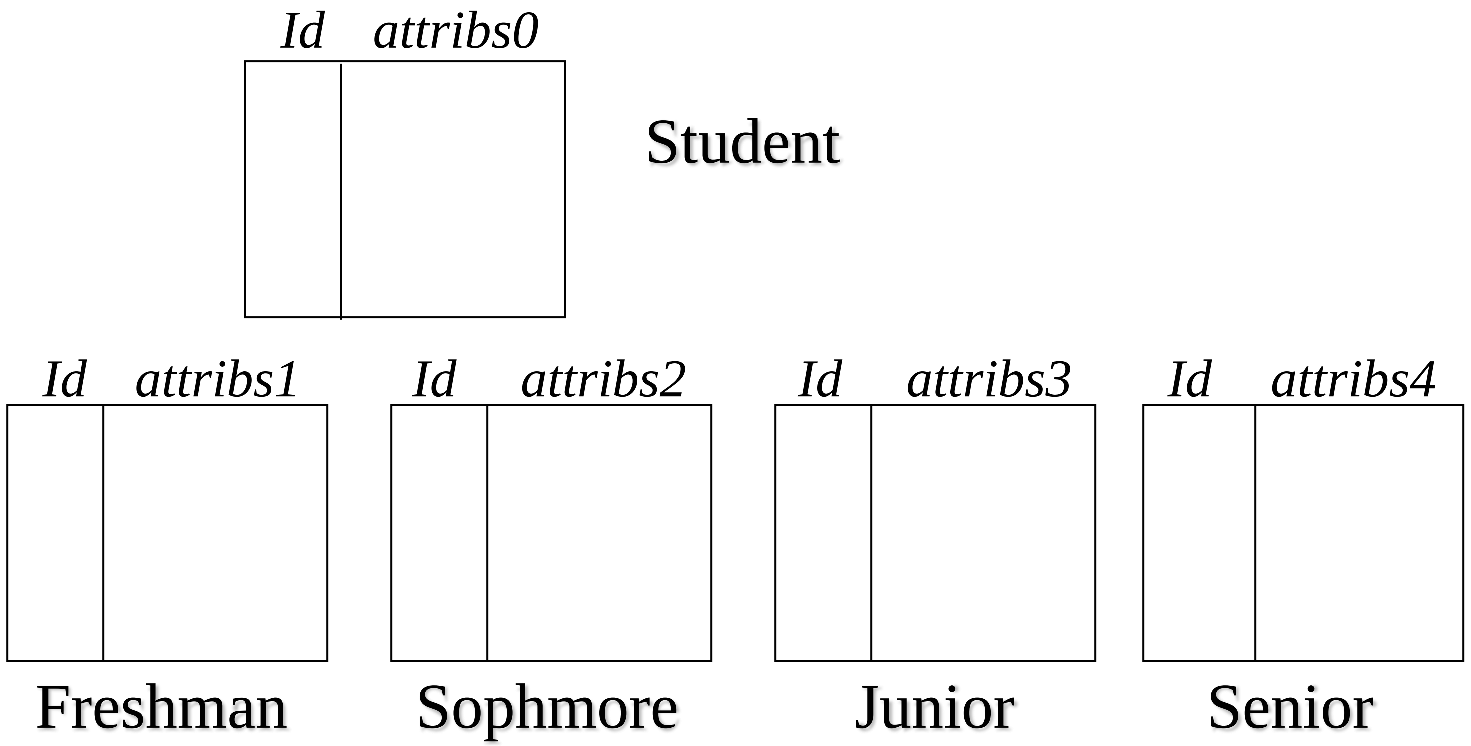
```
CREATE TABLE WorksIn (
    Since DATE,               -- attribute
    Status CHAR (10),         -- attribute
    ProfId INTEGER,           -- role (key of Professor)
    DeptId CHAR (4),          -- role (key of Department)
    PRIMARY KEY (ProfId),  -- since a professor works in at most one department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

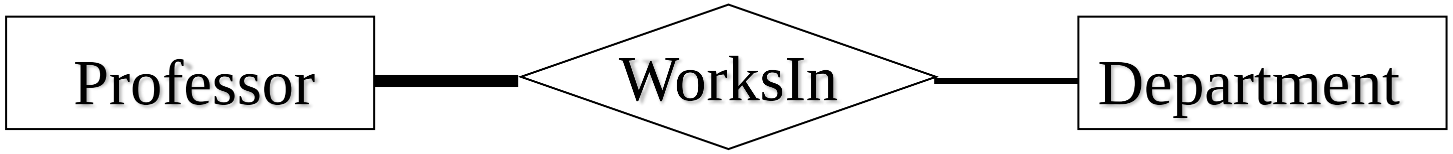
SSN	*Department*	*Salary*
1234	Accounting	35000

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:

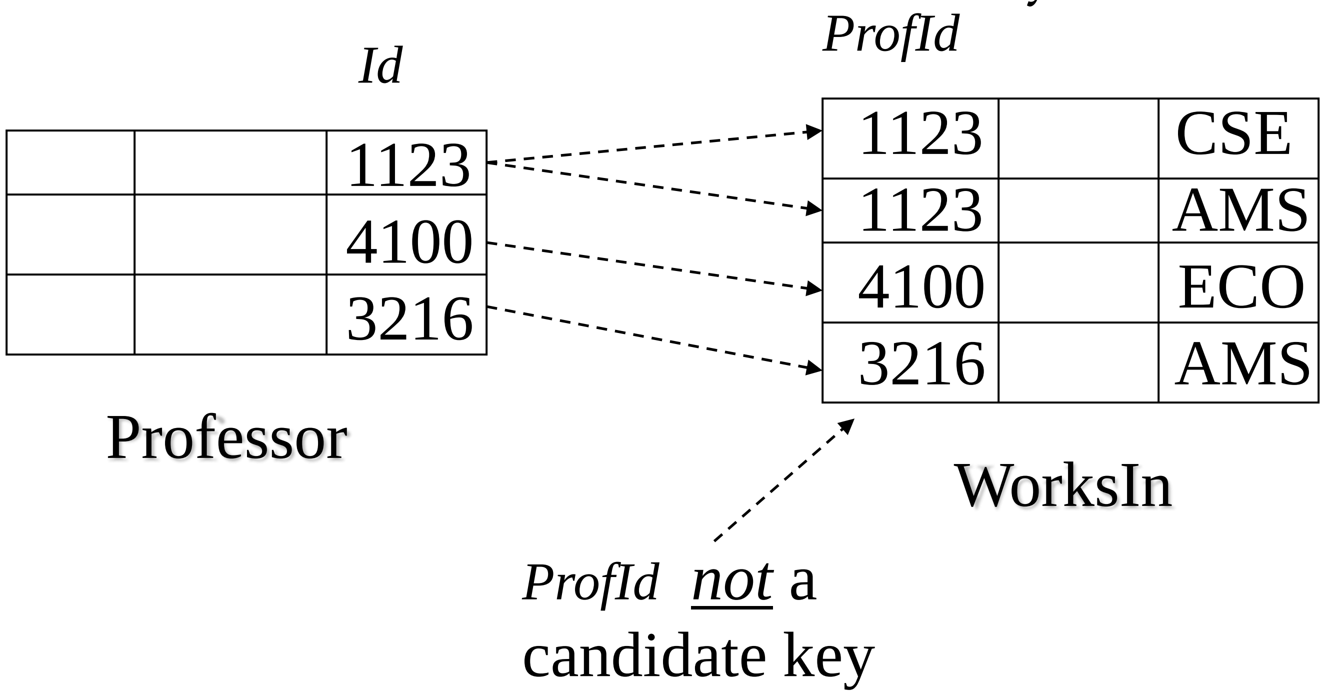
```
FOREIGN KEY  Id  REFERENCES  WorksIn (ProfId)
```

    - 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 ) ) )
```

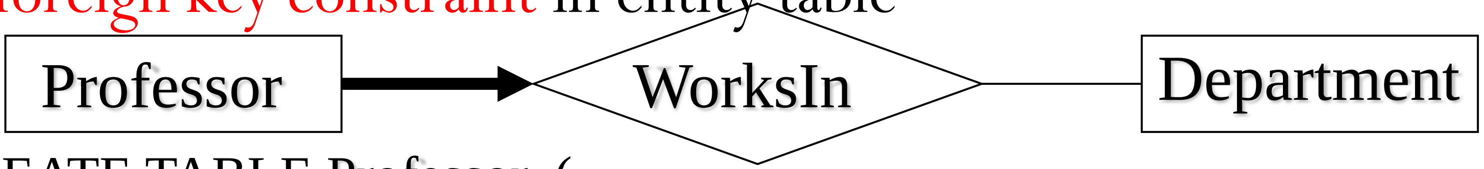
Representing Participation Constraints in the Relational Model

- General case Example cont. :can't use foreign key in Professor if ProfId is not a candidate key in WorksIn

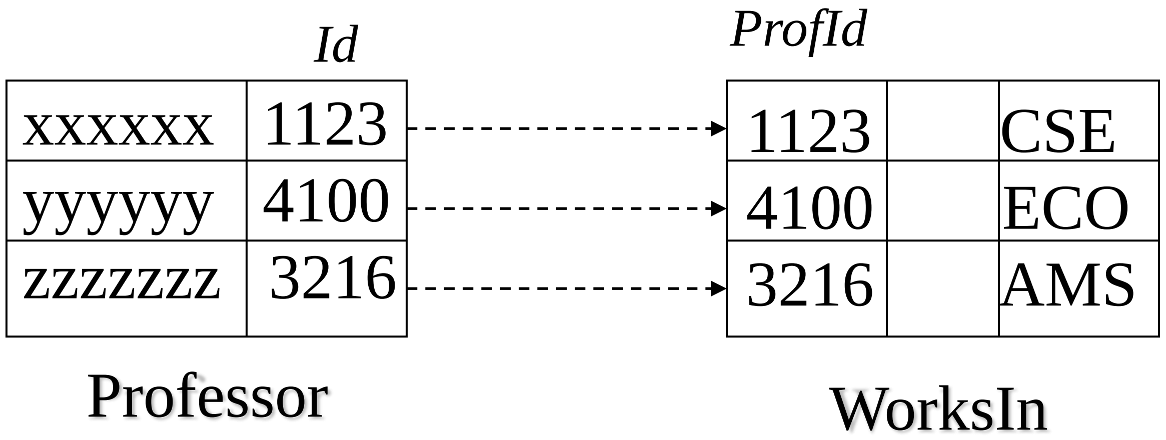


Representing Participation and Key Constraint in SQL

- 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

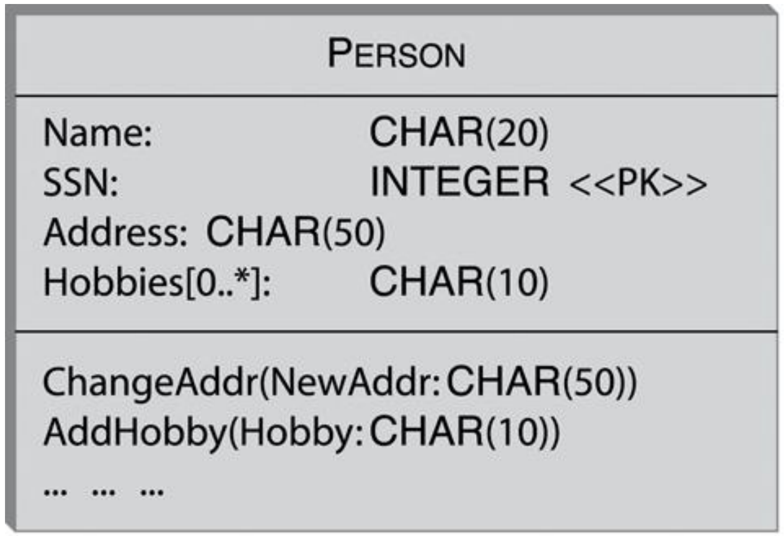
Prof_WorksIn

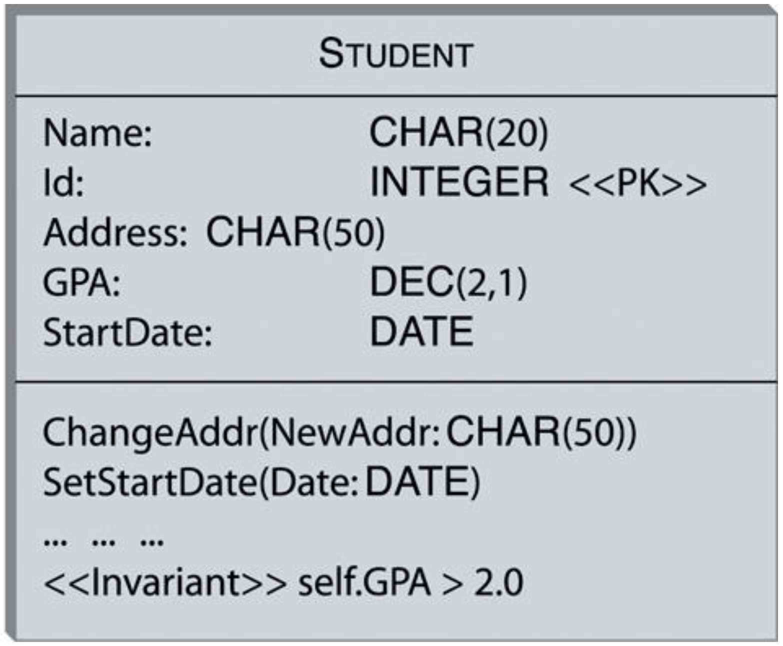
Unified Modeling Language (UML)

- 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)

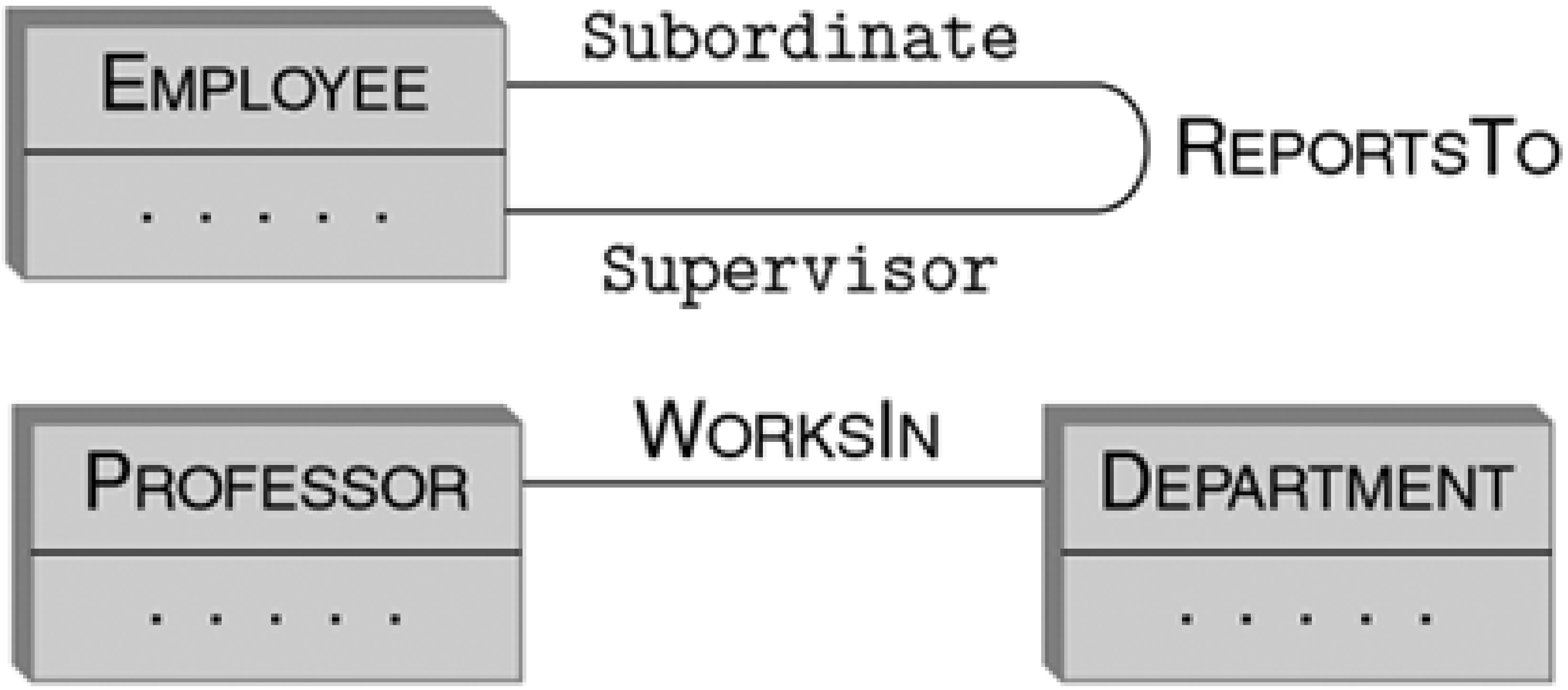
- Representing Entities in UML
  - Attributes are listed under the class.





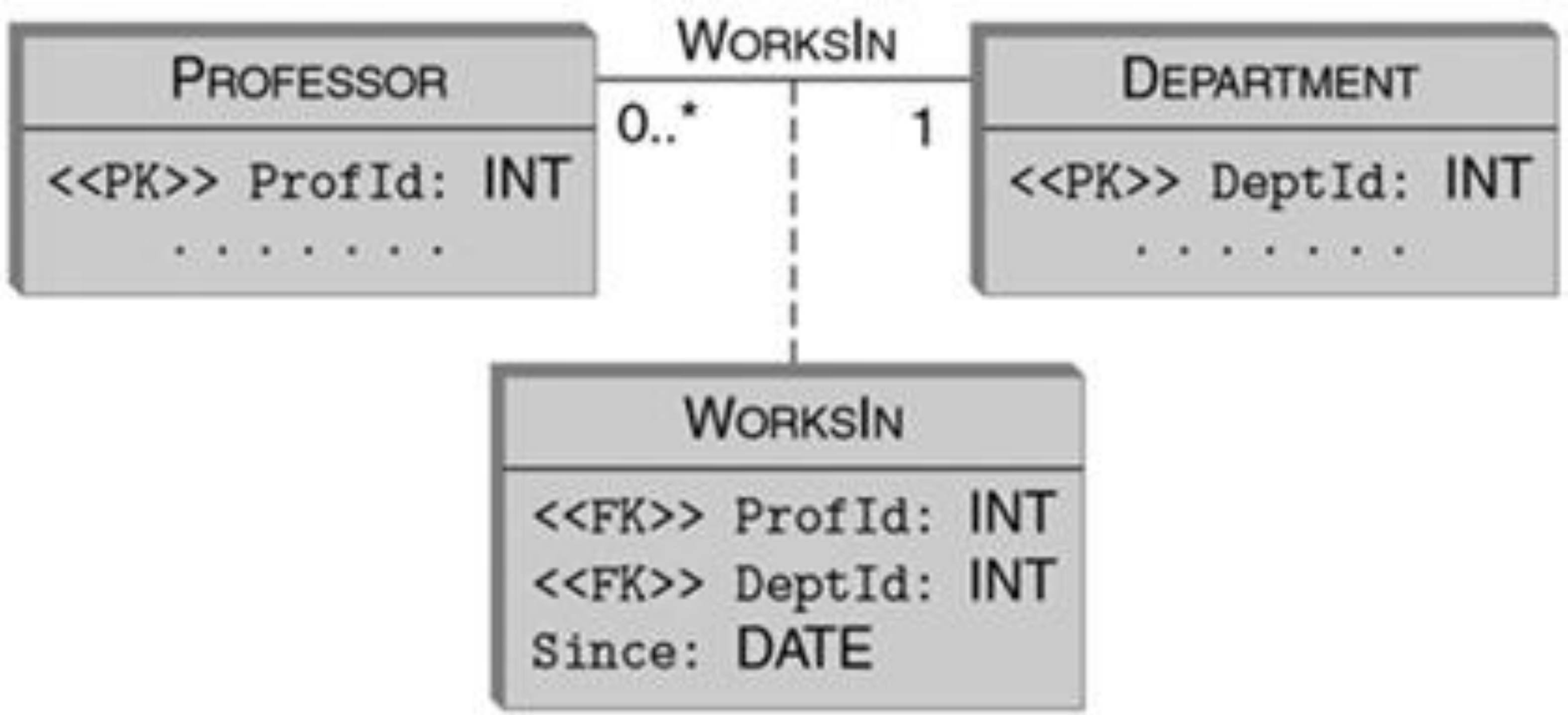
Unified Modeling Language (UML)

- Representation of Relationships



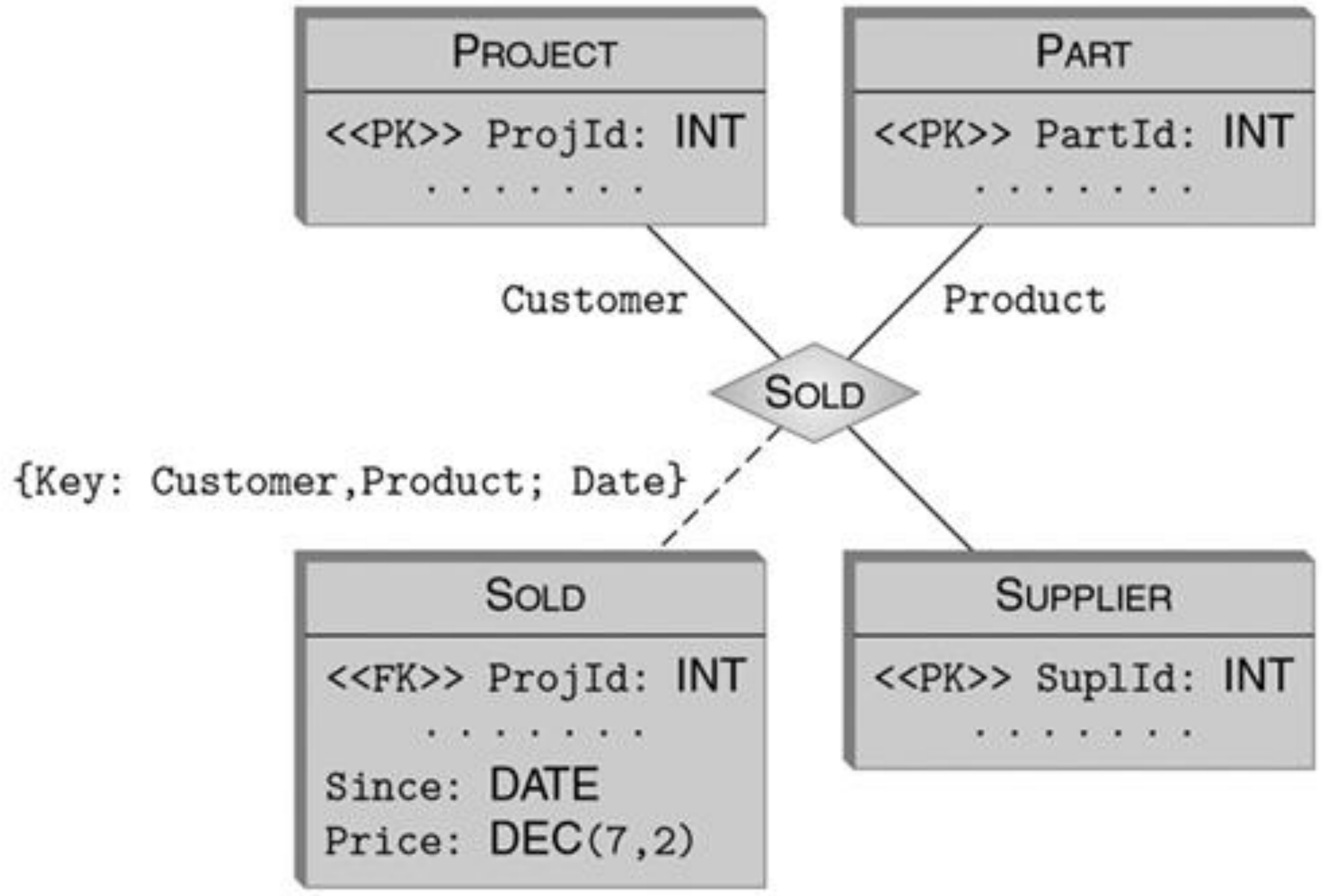
Association classes

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



Association classes

- Representation of Relationships

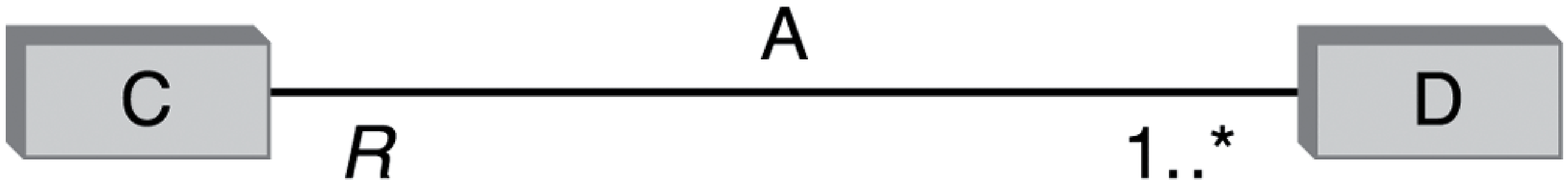


Multiplicity constraints

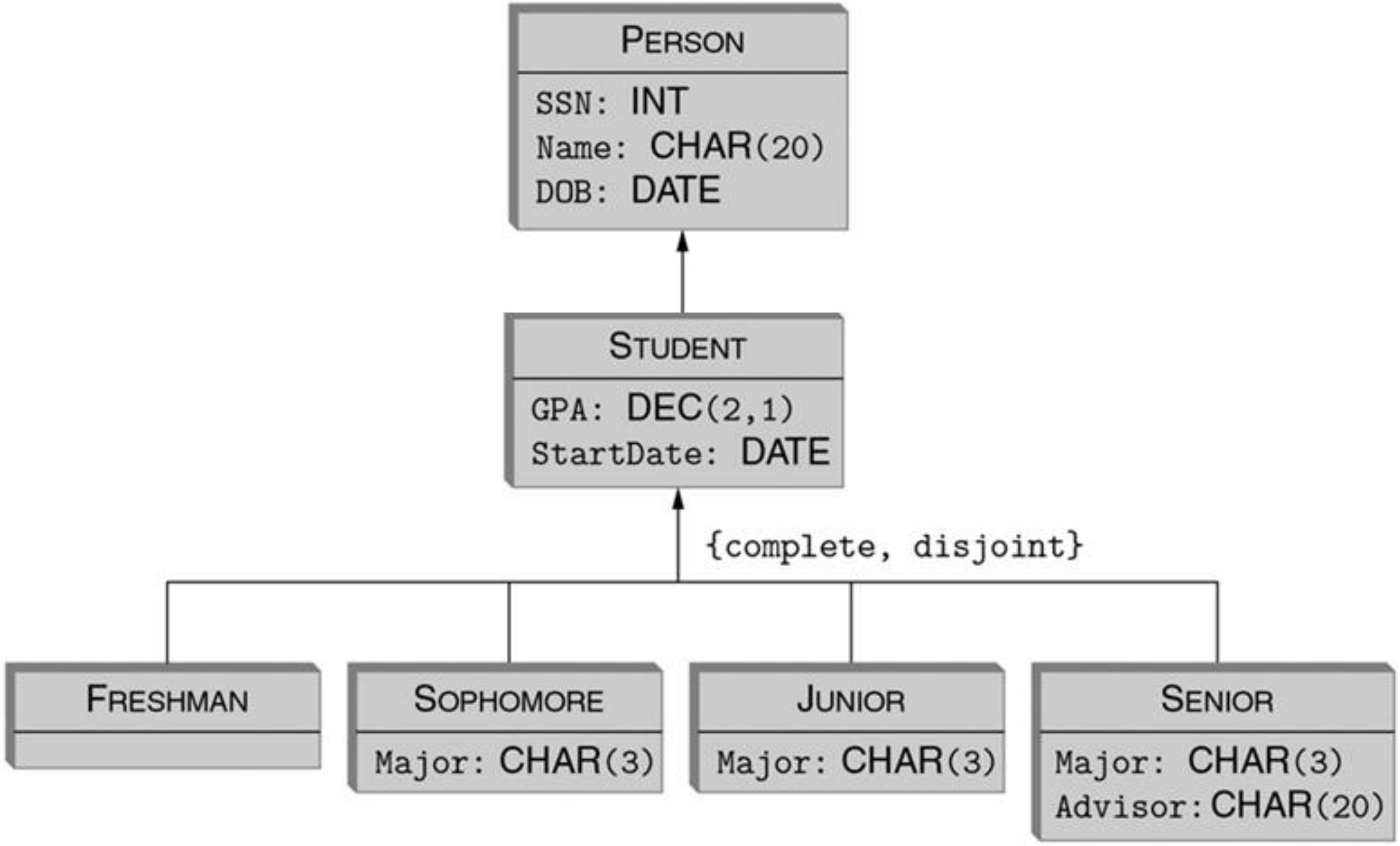
- A multiplicity constraint on a role, R, that connects an association type, A, with a class, C, is a range specification of the form n..m attached to R, where $n \geq 0$ is a nonnegative integer and m is either the * symbol or an integer $\geq n$.
  - The range gives the lower and upper bounds on the 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).
- 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.

Participation constraints

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

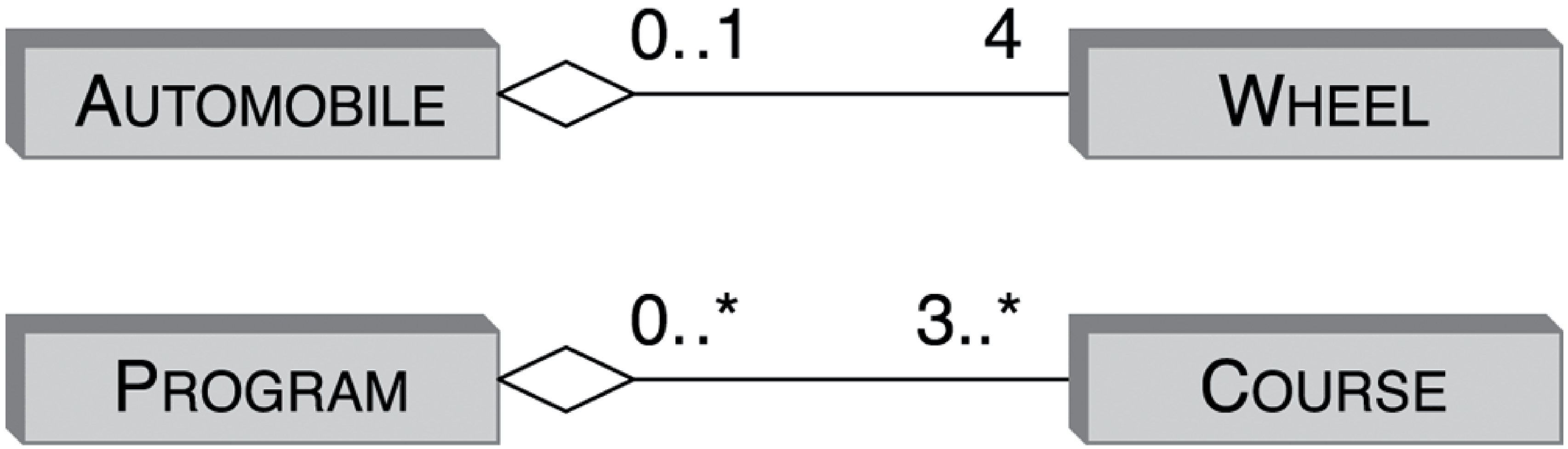


Representing IsA Hierarchies in UML



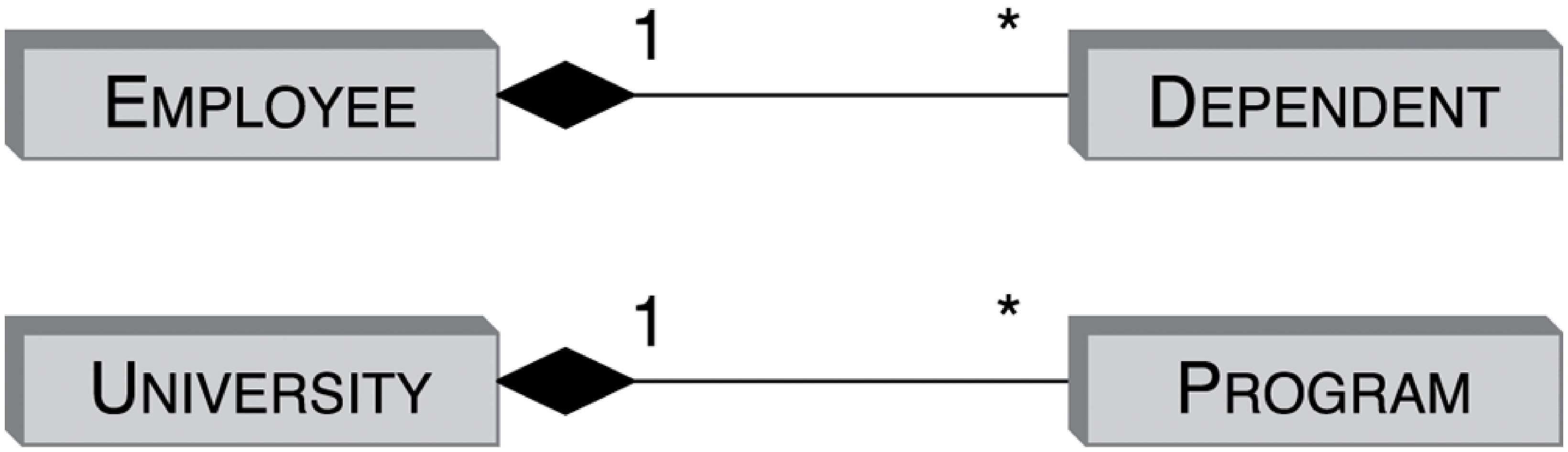
Part-of relationships

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



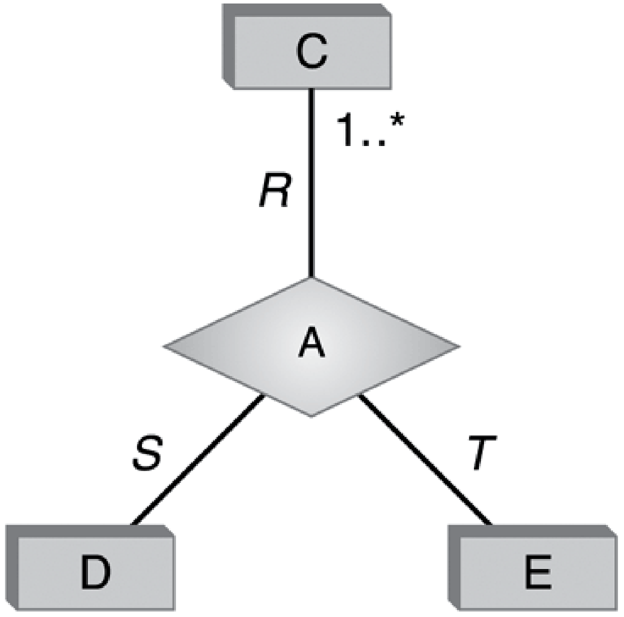
Part-of relationships

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

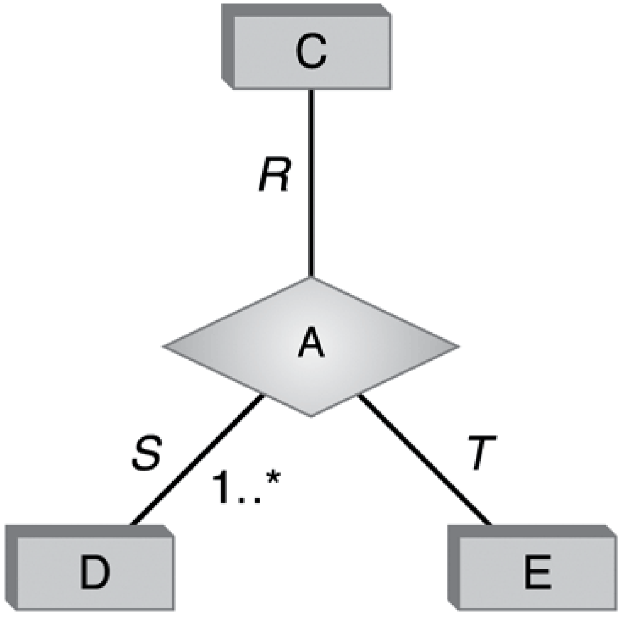


Participation constraints

- For ternary relations: not always possible to represent them in UML



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



(b) Partial simulation of the same in UML

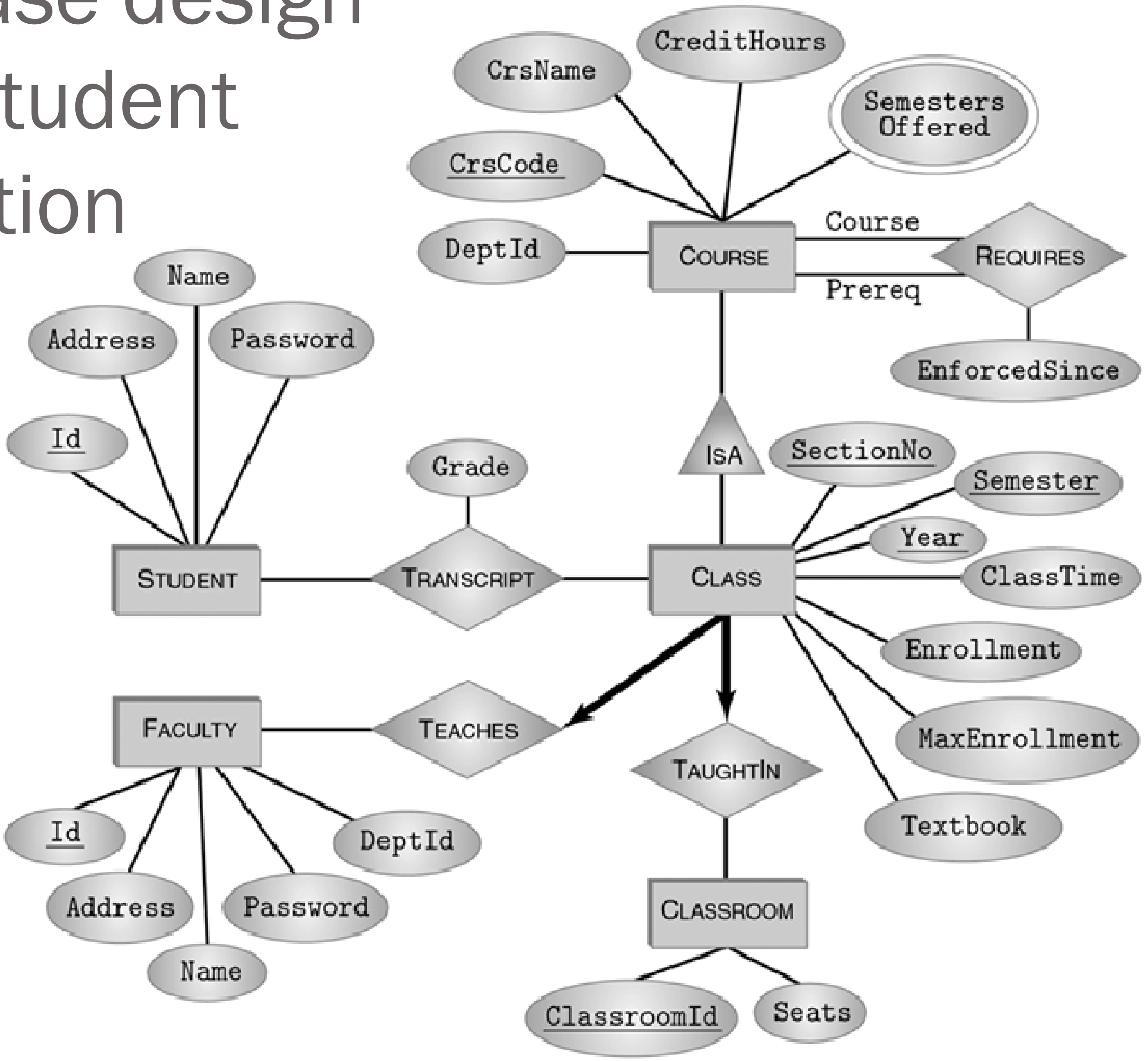
For every entity c in C there are entities d ∈ D and e ∈ E that participate in a relationship a ∈ A with c.

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.

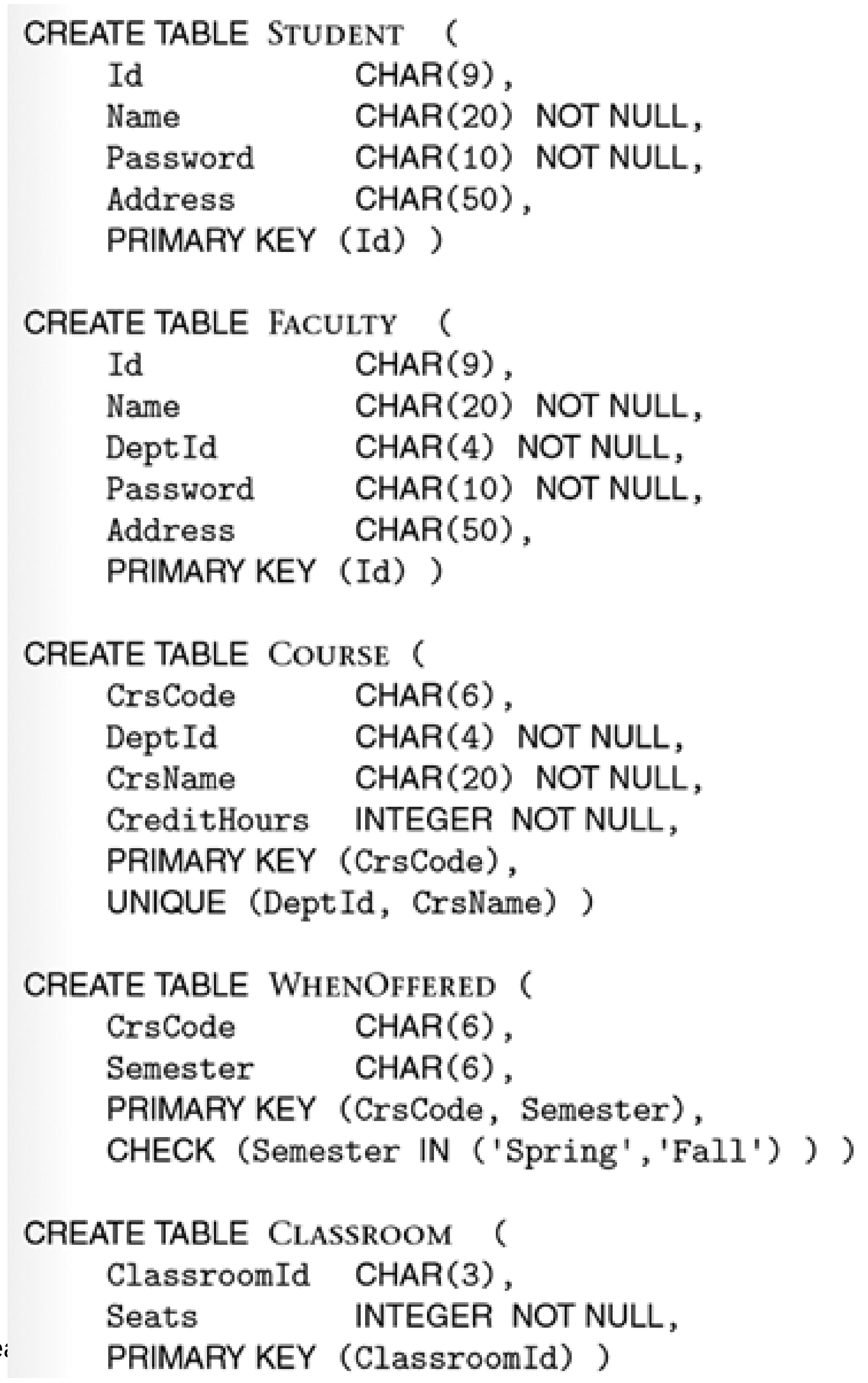
Participation constraints

- 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

A database design for the Student Registration System



SQL for the Student Registration System



```
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) )
```

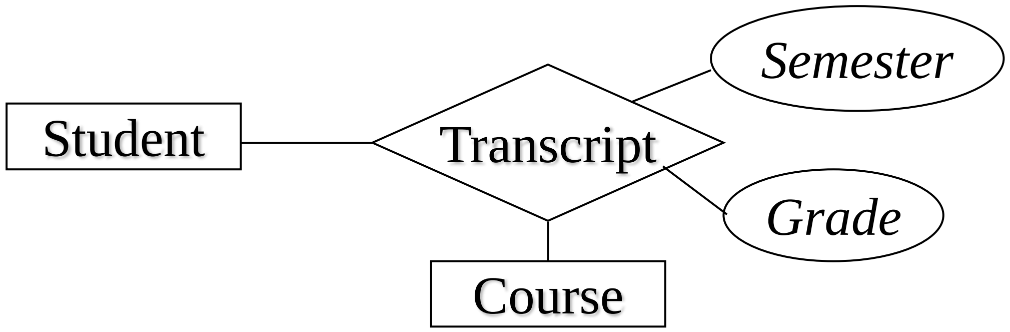
```
CREATE TABLE  REQUIRES   (
    CrsCode        CHAR(6),
    PrereqCrsCode CHAR(6),
    EnforcedSince DATE     NOT NULL,
    PRIMARY KEY (CrsCode, PrereqCrsCode),
    FOREIGN KEY (CrsCode) REFERENCES COURSE(CrsCode),
    FOREIGN KEY (PrereqCrsCode) REFERENCES COURSE(CrsCode)  )

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)  )
```

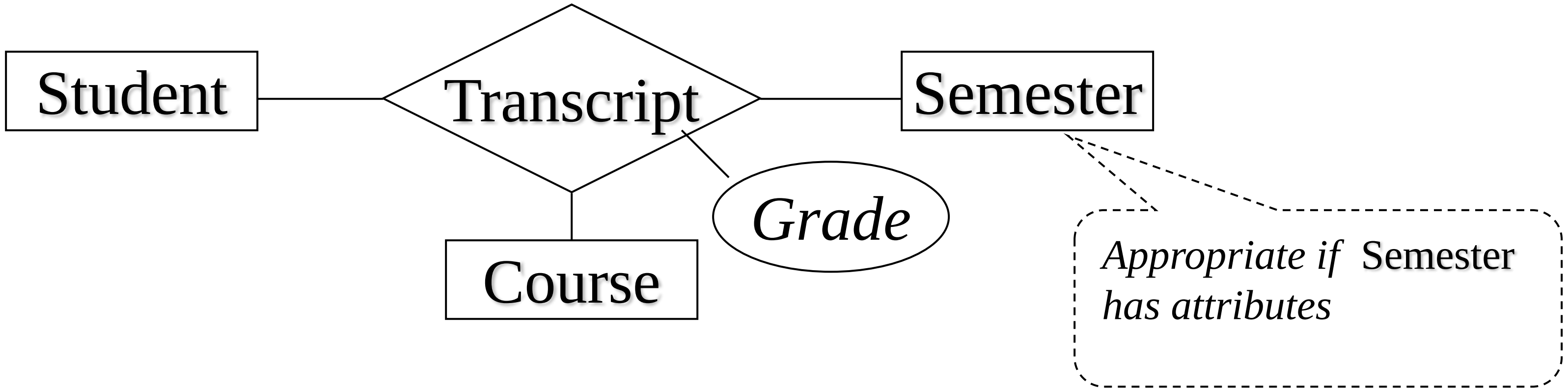
```
CREATE TABLE  TRANSCRIPT   (
    StudId         CHAR(9),
    CrsCode        CHAR(6),
    SectionNo      INTEGER,
    Semester       CHAR(6),
    Year           INTEGER,
    Grade          CHAR(1),
    PRIMARY KEY (StudId,CrsCode,SectionNo,Semester,Year),
    FOREIGN KEY (StudId) REFERENCES STUDENT(Id),
    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') )   )
```

Limitations of the Data Modelling Methodologies

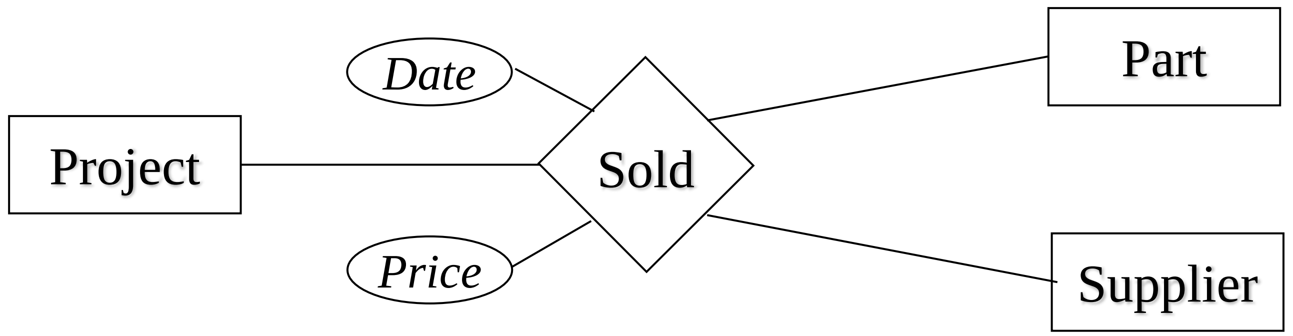
- Entity or Attribute?
  - Sometimes information can be represented as either an entity or an attribute.

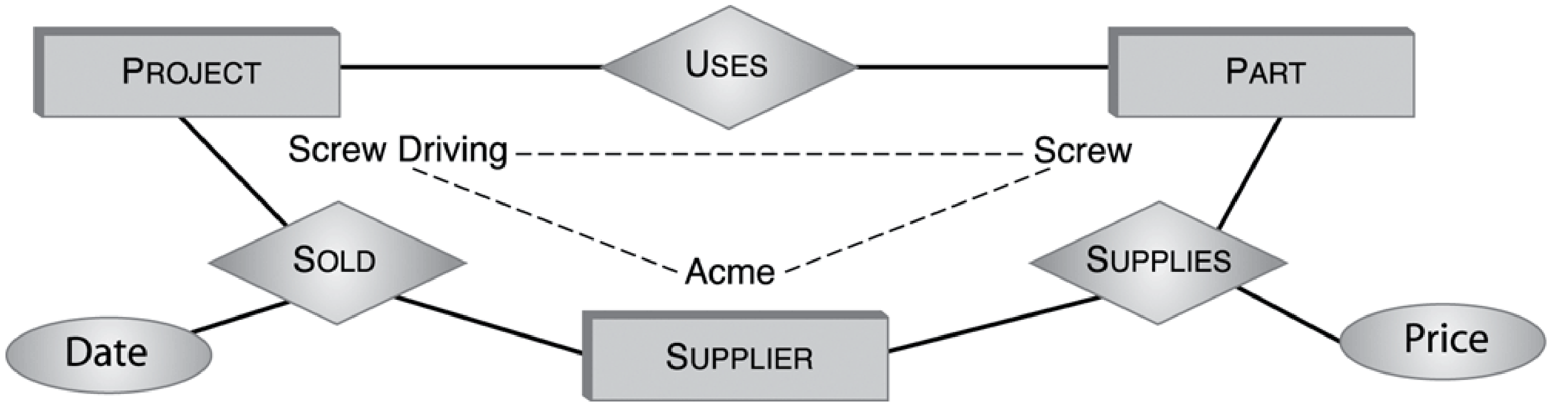


could be:



Transformations between binary and ternary relationships


Date
Price
Project
Sold
Part
Supplier


PROJECT
USES
PART
Screw Driving
Screw
Acme
SOLD
SUPPLIES
Date
SUPPLIER
Price