Chapter 4
High-Level Database Models

Université Grenoble Alpes

23/02/2023

Bahareh Afshinpour

bahareh.afshinpour@univ-grenoble-alpes.fr

Main reference:
A First Course in Database Systems (and associated material) by J. Ullman and J. Widom, Prentice-Hall

Design approach

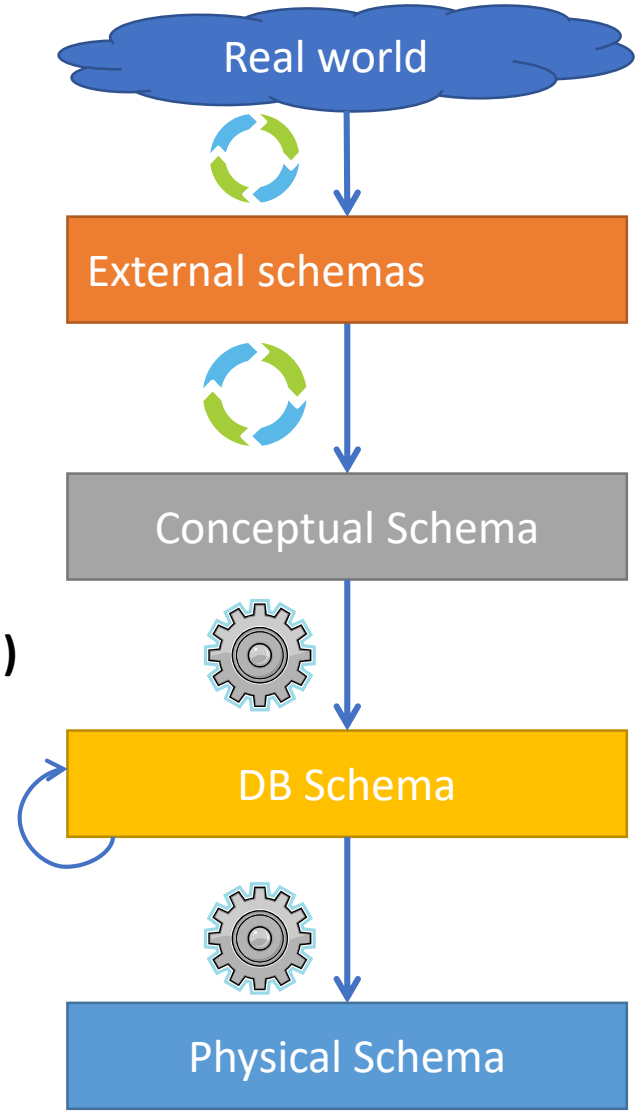
1. Perceiving the real world

2. Drawing up the conceptual schema

3. Designing the DB schema (logical)

4. Refining the DB schema (logical)

5. Drawing-up the physical schema



Graphical models &
Natual Language
(UML, FlowCharts,...)

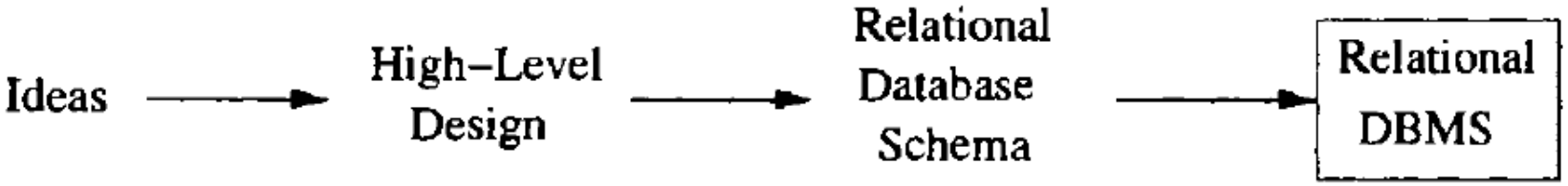
Graphical models
(**UML**,
Entity−Association)

Text Models
(Object, **Relational**)

Platform Models
(Java, C++, **SQL**)

Recall

- In practice, it is often easier to start with a higher-level model and then convert the design to the relational model.



The database modeling and implementation process

- There are several options for the notation in which the design is expressed.
  - Entity-relationship diagram
  - UML (class diagram)
  - ODL(object description language)

ER Diagrams, Naming Conventions, and Design Issues

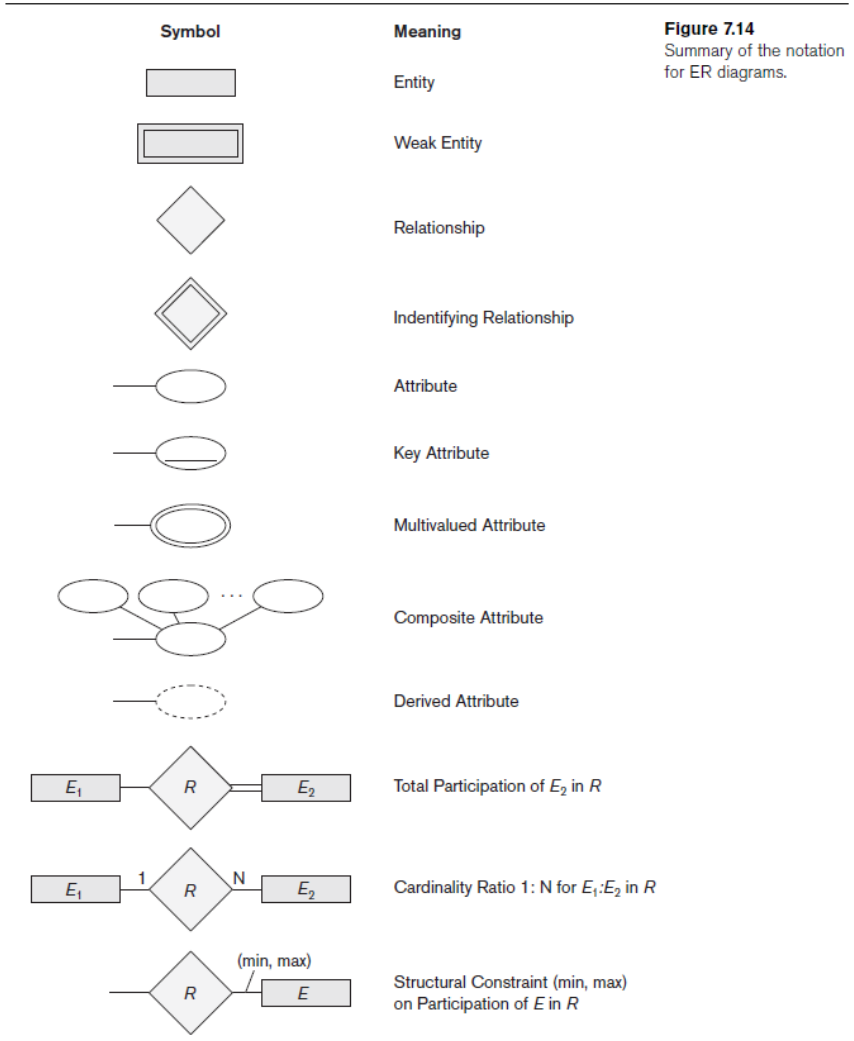
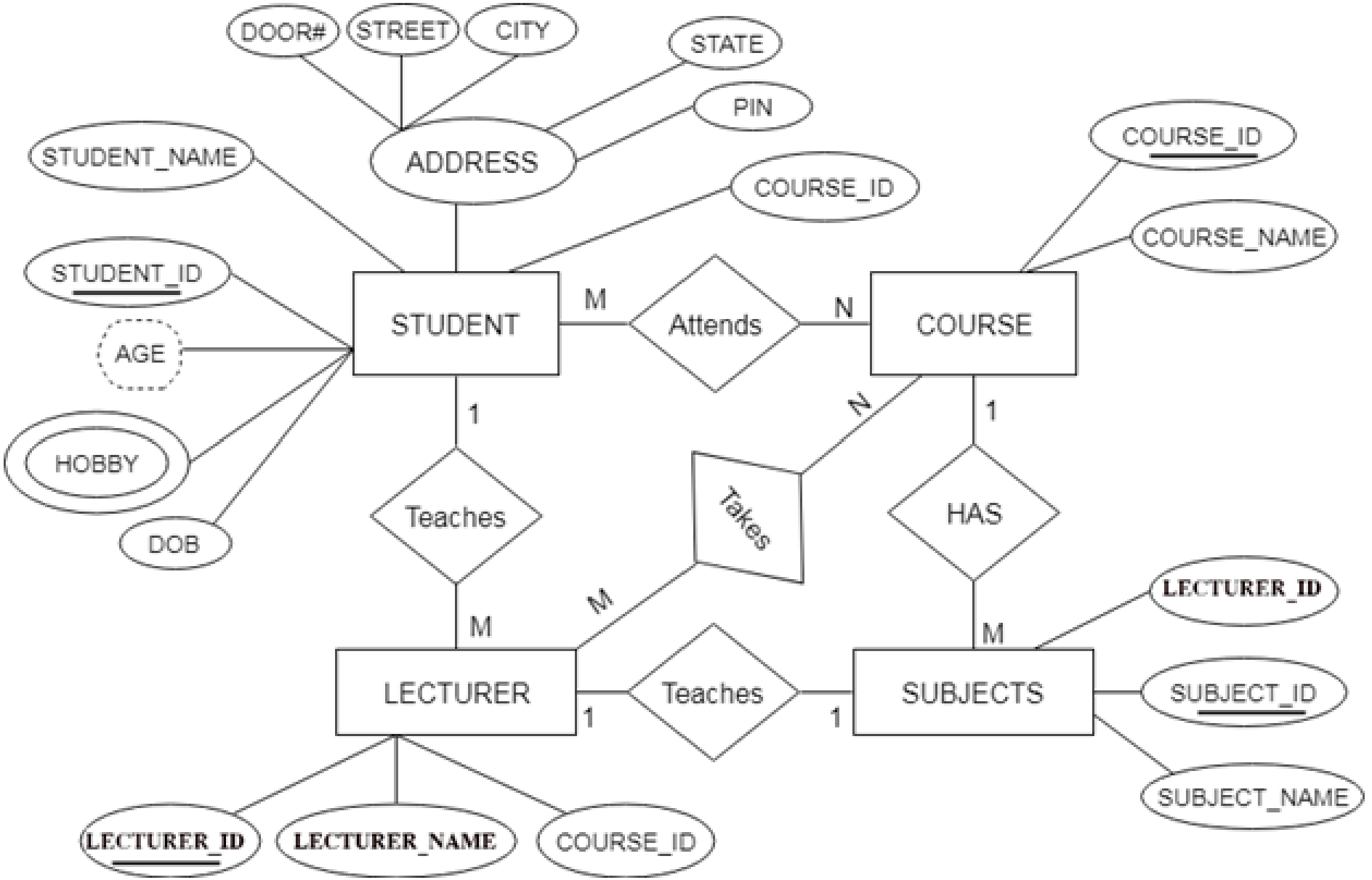
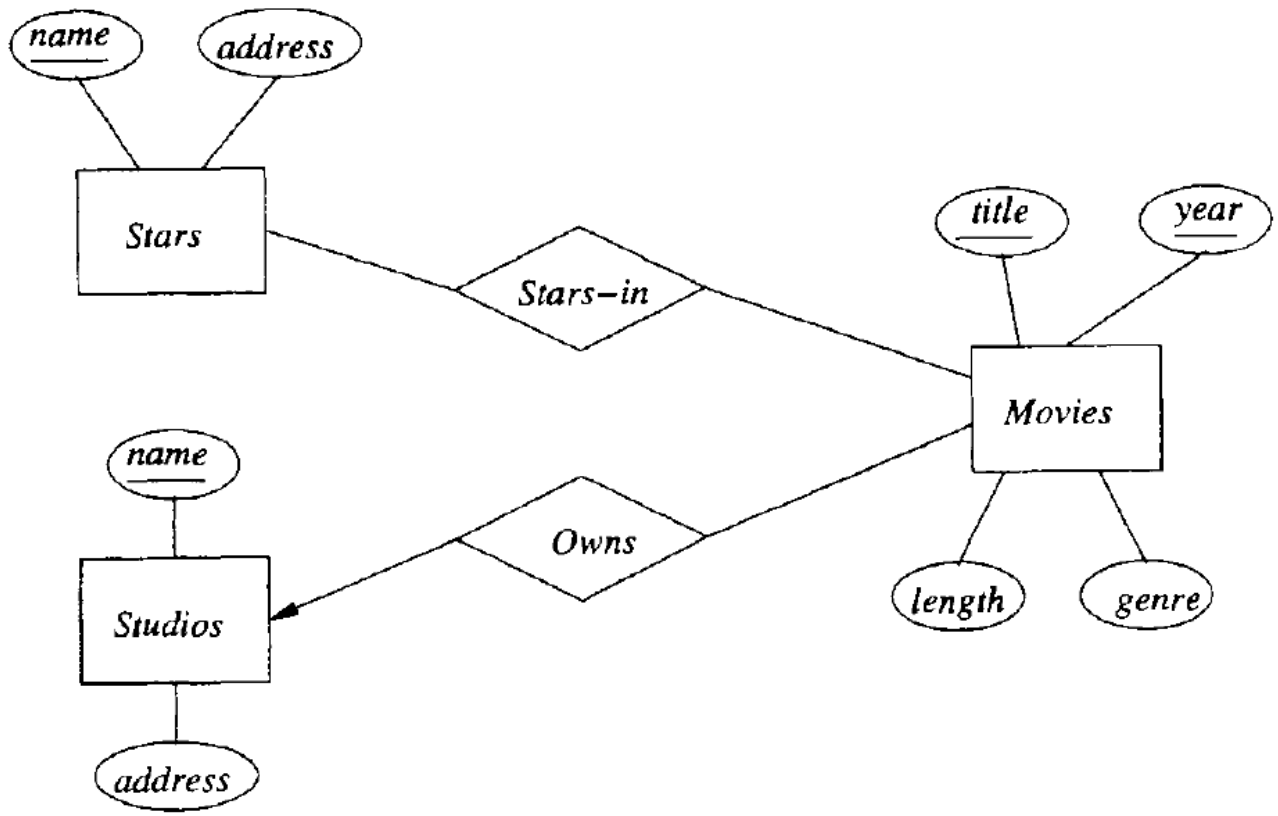


Figure 7.14
Summary of the notation for ER diagrams.

Exercise 4.1.9: Design a database suitable for a university registrar. This database should include information about students, departments, professors, courses, which students are enrolled in which courses, which professors are teaching which courses, student grades, TA's for a course (TA's are students), which courses a department offers, and any other information you deem appropriate. Note that this question is more free-form than the questions above, and you need to make some decisions about multiplicities of relationships, appropriate types, and even what information needs to be represented.


DOOR#
STREET
CITY
STATE
PIN
ADDRESS
STUDENT_NAME
COURSE_ID
COURSE_ID
COURSE_NAME
STUDENT_ID
AGE
HOBBY
DOB
STUDENT
M
Attends
N
COURSE
1
N
1
Teaches
Takes
HAS
M
M
M
LECTURER_ID
LECTURER
Teaches
SUBJECTS
SUBJECT_ID
1
1
SUBJECT_NAME
LECTURER_ID
LECTURER_NAME
COURSE_ID

Representing Keys in the E/R Model



4.1.12 Exercises for Section 4.1

Exercise 4.1.1: Design a database for a bank, including information about customers and their accounts. Information about a customer includes their name, address, phone, and Social Security number. Accounts have numbers, types (e.g., savings, checking) and balances. Also record the customer(s) who own an account. Draw the E/R diagram for this database. Be sure to include arrows where appropriate, to indicate the multiplicity of a relationship.

Exercise 4.1.2: Modify your solution to Exercise 4.1.1 as follows:

Exercise 4.1.3: Give an E/R diagram for a database recording information about teams, players, and their fans, including:

1. For each team, its name, its players, its team captain (one of its players), and the colors of its uniform.

2. For each player, his/her name.

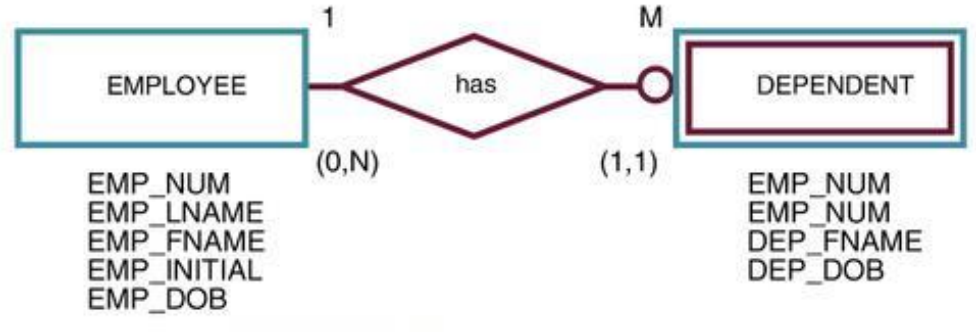
3. For each fan, his/her name, favorite teams, favorite players, and favorite color.

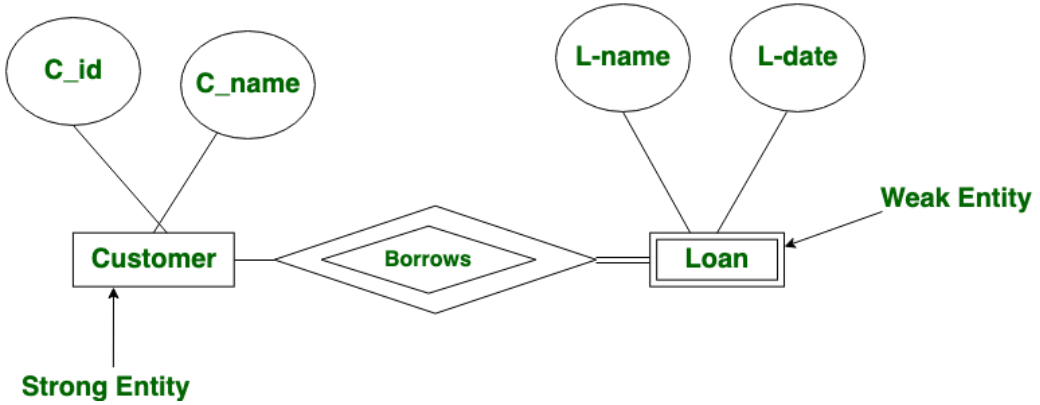
Remember that a set of colors is not a suitable attribute type for teams. How can you get around this restriction?

Exercise 4.1.5: Modify Exercise 4.1.3 to record for each player the history of teams on which they have played, including the start date and ending date (if they were traded) for each such team.

Weak entity

- Weak Entities
  - **A weak entity is an entity that**
    - **Is existence-dependent and**
    - **Has a primary key that is partially or totally derived from the parent entity in the relationship.**
  - **The existence of a weak entity is indicated by a double rectangle.**
  - **The weak entity inherits all or part of its primary key from its strong counterpart.**





Example 4.20: A movie studio might have several film crews. The crews might be designated by a given studio as crew 1, crew 2, and so on. However, other studios might use the same designations for crews, so the attribute *number* is not a key for crews. Rather, to name a crew uniquely, we need to give both the name of the studio to which it belongs and the number of the crew. The situation is suggested by Fig. 4.20. The double-rectangle indicates a weak entity set, and the double-diamond indicates a many-one relationship that helps provide the key for the weak entity set. The notation will be explained further in Section 4.4.3. The key for weak entity set *Crews* is its own *number* attribute and the *name* attribute of the unique studio to which the crew is related by the many-one *Unit-of* relationship. □

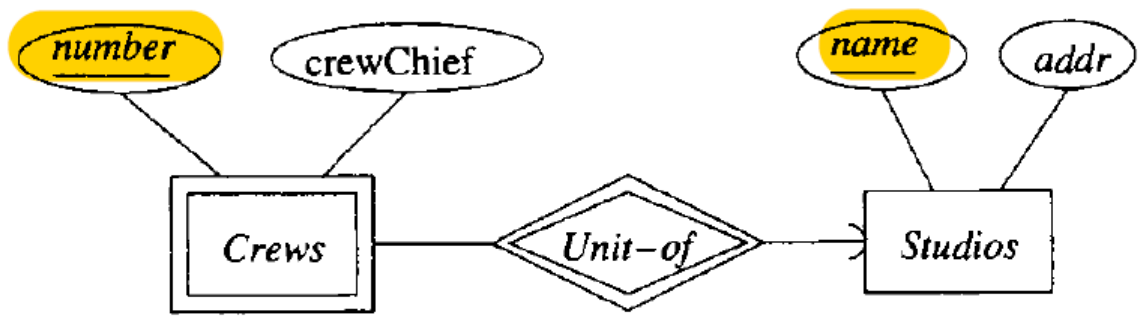


Figure 4.20: A weak entity set for crews, and its connections

4.4.3 Weak Entity Set Notation

We shall adopt the following conventions to indicate that an entity set is weak and to declare its key attributes.

1. If an entity set is weak, it will be shown as a rectangle with a double border. Examples of this convention are *Crews* in Fig. 4.20 and *Contracts* in Fig. 4.22.

2. Its supporting many-one relationships will be shown as diamonds with a double border. Examples of this convention are *Unit-of* in Fig. 4.20 and all three relationships in Fig. 4.22.

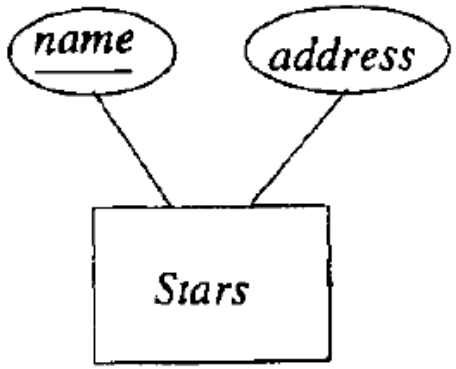
3. If an entity set supplies any attributes for its own key, then those attributes will be underlined. An example is in Fig. 4.20, where the number of a crew participates in its own key, although it is not the complete key for *Crews*.

We can summarize these conventions with the following rule:

- Whenever we use an entity set E with a double border, it is weak. The key for E is whatever attributes of E are underlined plus the key attributes of those entity sets to which E is connected by many-one relationships with a double border.

From E/R diagrams to Relational designs

1. **Turn each entity set into a relation with the same set of attributes**



name	*address*
Carrie Fisher	123 Maple St., Hollywood
Mark Hamill	456 Oak Rd., Brentwood
Harrison Ford	789 Palm Dr., Beverly Hills

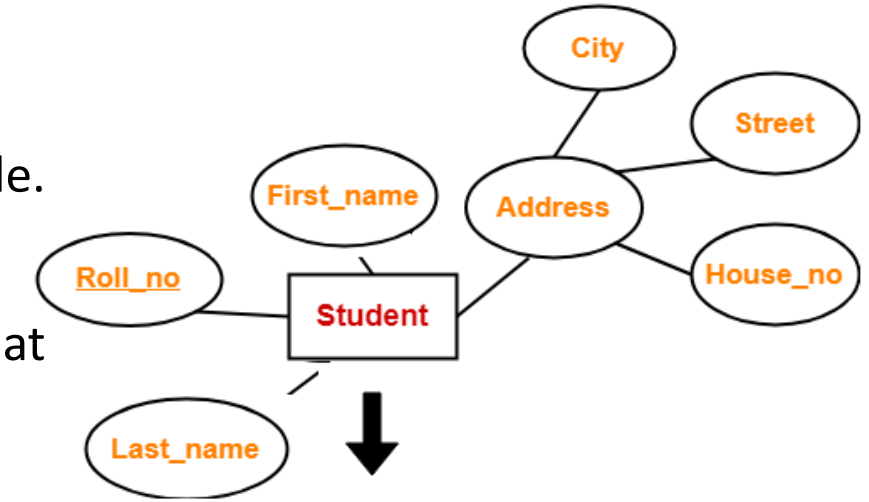
From E/R diagrams to Relational designs

3. Strong entity set with composite attributes

- In the relational model, a strong entity set with

any number of composite attributes will require only one table.

- During conversion, only the simple attributes of composite at tributes are considered, not the composite attribute itself



Roll_no	First_name	Last_name	House_no	Street	City

Schema : Student (Roll_no , First_name , Last_name , House_no , Street , City)

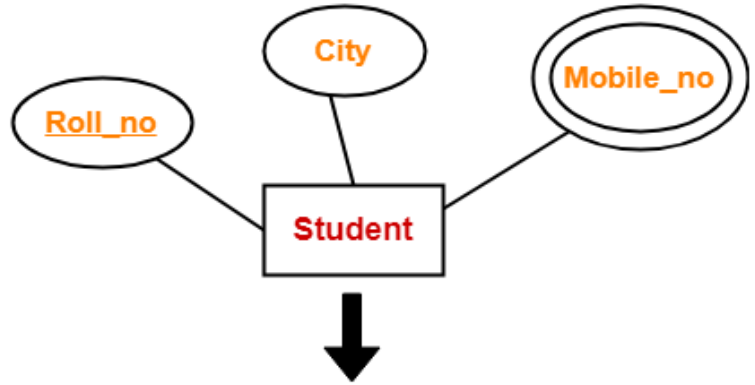
https://www.gatevidyalay.com/er-diagrams-to-tables/

From E/R diagrams to Relational designs

- **4. For Strong Entity Set With Multi-Valued Attributes**

In relational model, a strong entity set with any number of multivalued attributes will require **two** tables.

- All simple attributes will be stored in a single table with a primary key.
- Another table will contain the **primary key** and all attributes with multiple values.



Roll_no	City

Roll_no	Mobile_no

Combining relations : one to many

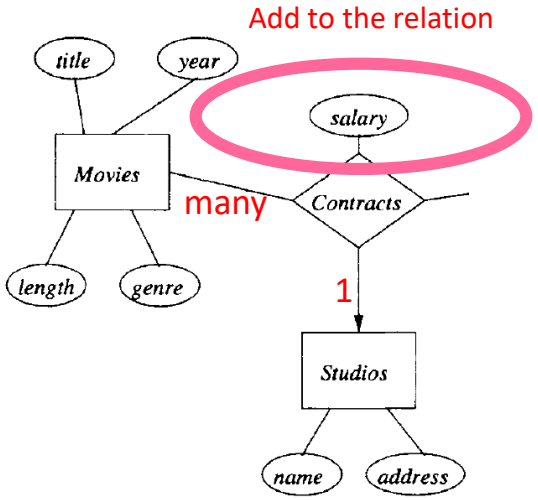
Because R is many-one, all these attributes are functionally determined by the key for E, and we can combine them into one relation with a schema consisting of:

1. All attributes of E.
2. The key attributes of F.
3. Any attributes belonging to relationship R.

-No new table for relation
- We modify many side(1 to many) table
- We add
  - Attribute from relation(contracts)
  - Primary key of 1 side

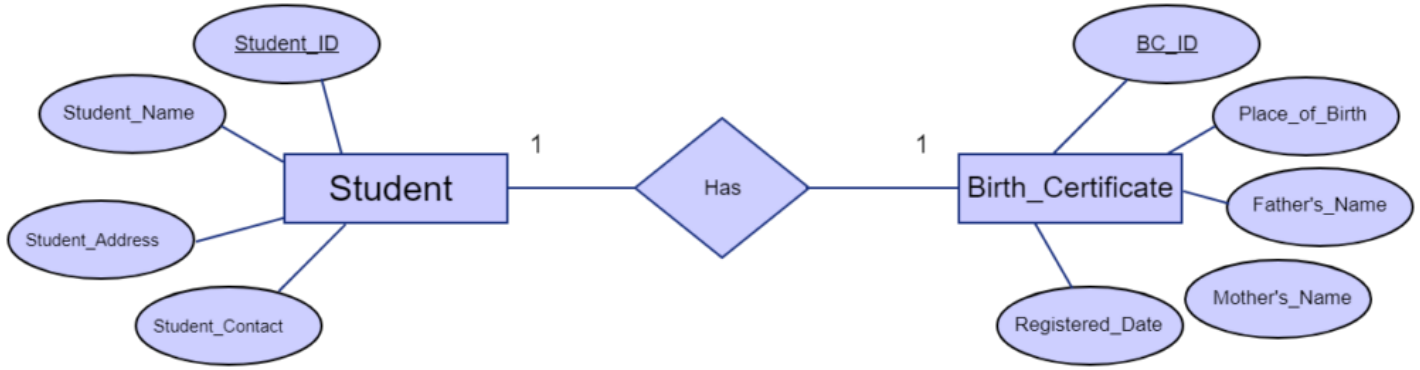
Movies:

title	*year*	*length*	*genre*	*studioName*	salary
Star Wars	1977	124	sciFi	Fox	
Gone With the Wind	1939	239	drama	MGM	
Wayne's World	1992	95	comedy	Paramount	



Here, two tables will be required : -studios - Movies

Combining relations : one to one



Here, two tables will be required. Either combine 'R' with 'A' or 'B'

Way-01:

1.student (a1 , a2 ,a3, …, **b1**)
2.Birth (b1 , b2, b3,….)

Way-02:

1.student (a1 , a2, a3, ….)
2.Birth (**a1** , b1 , b2, b3, ….)

There's no need for a new table. Only the primary key of one entity should be added to another

Combining relations : many to many

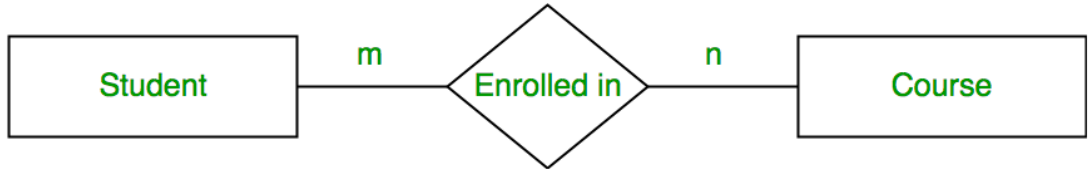
- Here, three tables will be required:

Create a new table for the relation

✓ **Student(a1,a2,a3,….)**

✓ **Course(c1,c2,c3,…)**

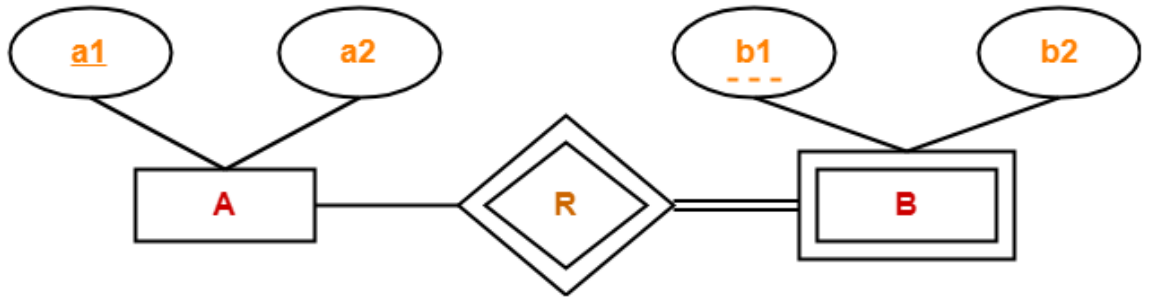
✓ **Enrolled(a1,c1,…)**



Weak entity

Weak entity set always appears in association with identifying relationship with total participation constraint.

- Create a new table
- Put the owner's primary key in this table
- Combination of the owner and weak entity 's primary key is new primary key in this table

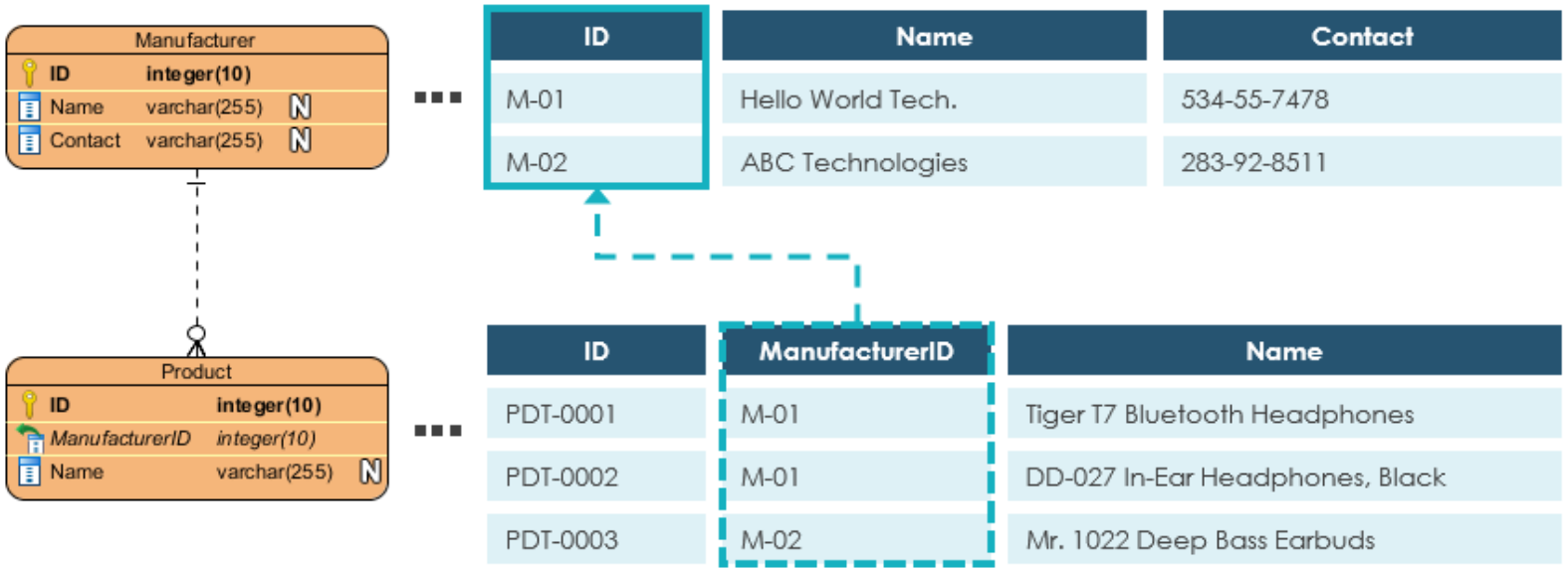


Here, two tables will be required-

1. A (a1 , a2)
2. BR (a1 , b1 , b2)

Foreign key (Also known as FK)

This concept is used in relational databases for an attribute that is the primary key of another table and is used to create a link between that table and the table in which it also appears as an attribute.



A foreign key is a **reference to a primary key in a table**.

Note that foreign keys need not be unique. Multiple records can share the same values.

https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/

Unified modeling diagram

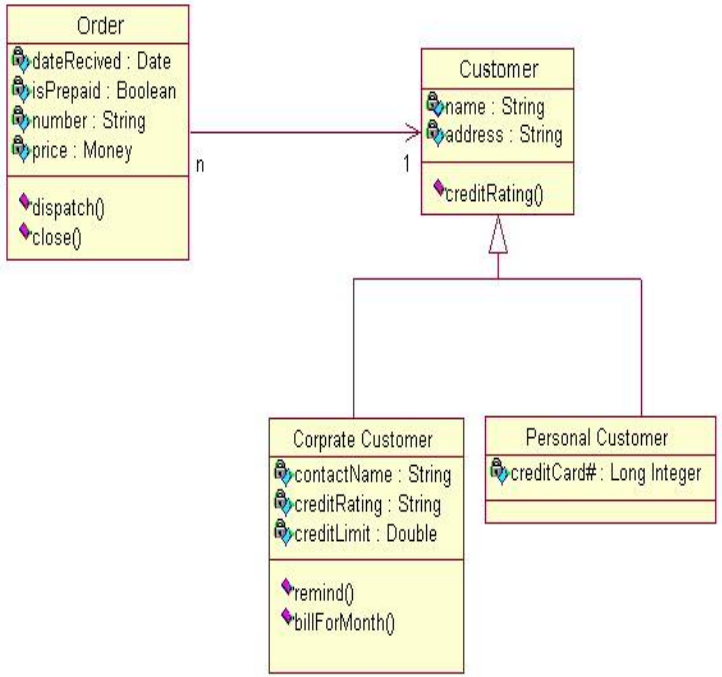
- Modeling: Describing a software system at a high level of abstraction
- UML offers much the same capabilities as the E/R model, with the exception of multiway relationships.
- Here you can see different terminology that is used by E/R and UML.

UML	E/R Model
Class	Entity set
Association	Binary relationship
Association Class	Attributes on a relationship
Subclass	Isa hierarchy
Aggregation	Many-one relationship
Composition	Many-one relationship with referential integrity

Figure 4.34: Comparison between UML and E/R terminology

Class diagram

- A class diagram depicts classes and their relationships
- Provide a conceptual model of the system in terms of entities and their relationships



UML Classes

- Sets of objects, with attributes (state) and methods (behavior).
- Each class is represented by a rectangle subdivided into three compartments
  - Name
  - Attributes
  - Operations
- Attributes have types.
- PK indicates an attribute in the object's primary key (optional).
- Methods have declarations: arguments (if any) and return type.

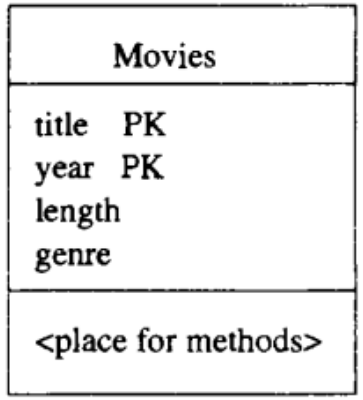


Figure 4.35: The *Movies* class in UML

Example 4.34: We might have added an instance method *lengthInHours*(). The UML specification doesn't tell anything more about a method than the types of any arguments and the type of its return-value. Perhaps this method returns *length*/60.0, but we cannot know from the design. □

Associations

- A binary relationship between classes is called an association.
- No multiway relationship (it is broken into binary relationships)
- The association is a set of pairs of objects, one from each of the classes it connects.

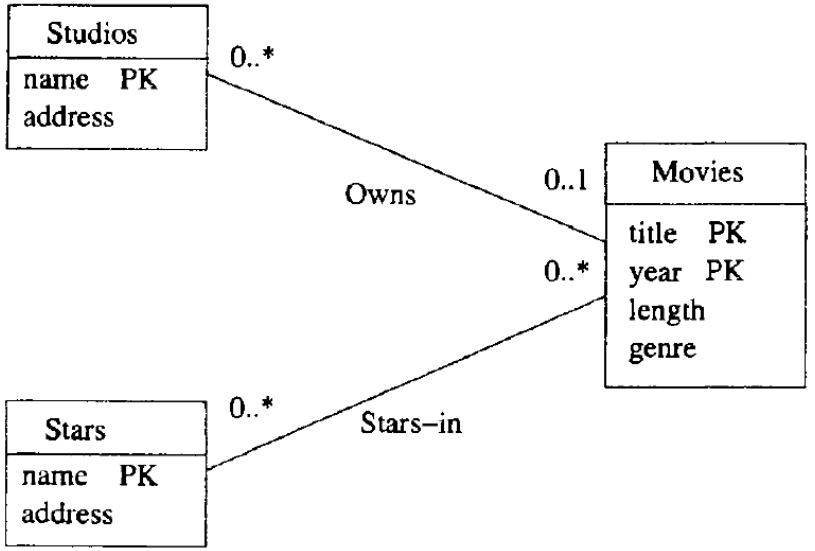
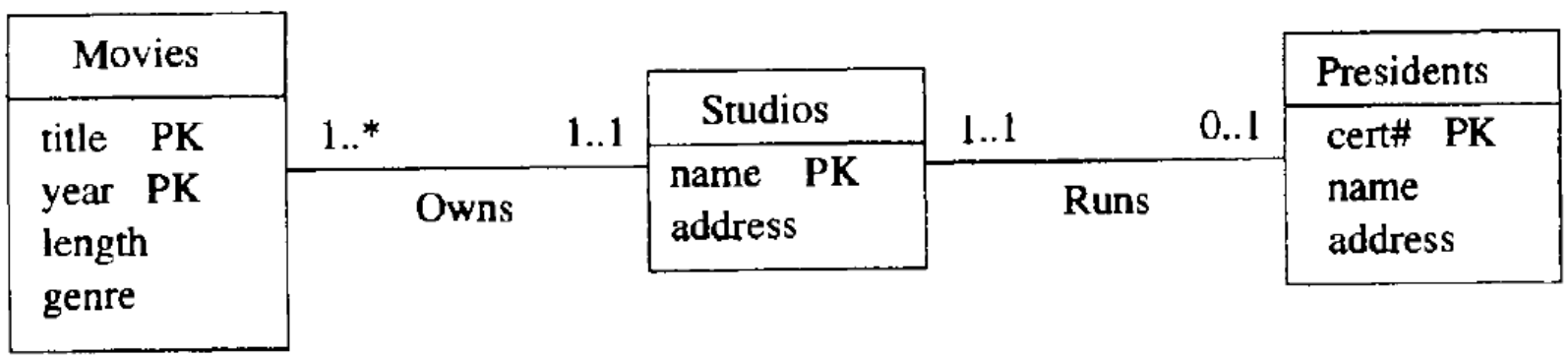


Figure 4.36: Movies, stars, and studios in UML

- If there is no label at all at an end of an association edge, then the label is taken to be 1..1, i.e., "exactly one."

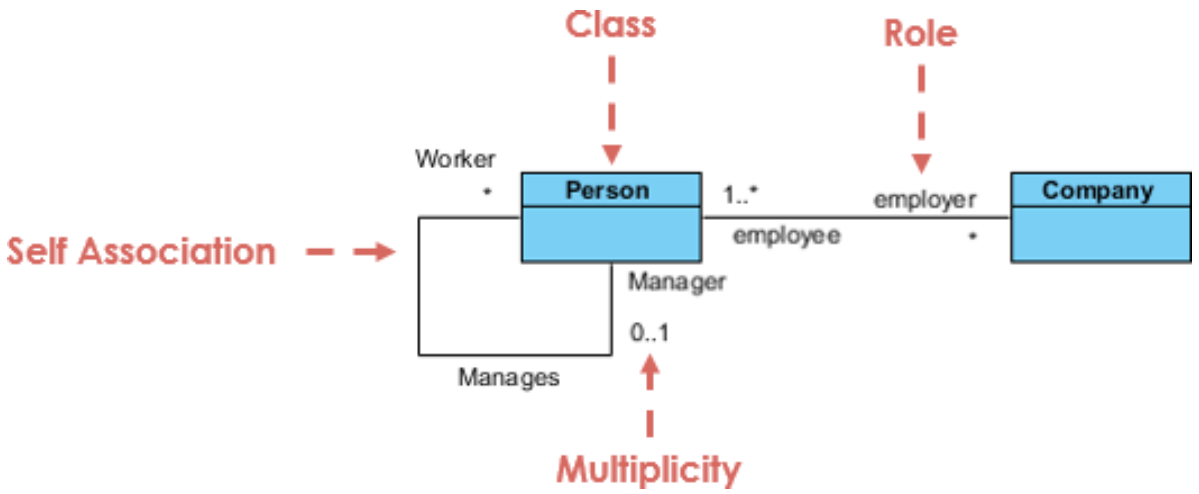
Example

Example 4.36 : In Fig. 4.36 we see 0..∗ at the *Movies* end of both associations. That says that a star appears in zero or more movies, and a studio owns zero or more movies; i.e., there is no constraint for either. There is also a 0..∗ at the *Stars* end of association *Stars-in*, telling us that a movie has any number of stars. However, the label on the *Studios* end of association *Owns* is 0..1, which means either 0 or 1 studio. That is, a given movie can either be owned by one studio, or not be owned by any studio in the database. Notice that this constraint is exactly what is said by the pointed arrow entering *Studios* in the E/R diagram of Fig. 4.17. □



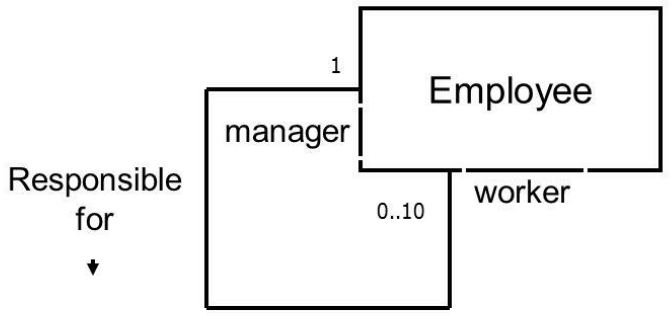
Self- Associations

An association can have both ends at the same class



Association - Self

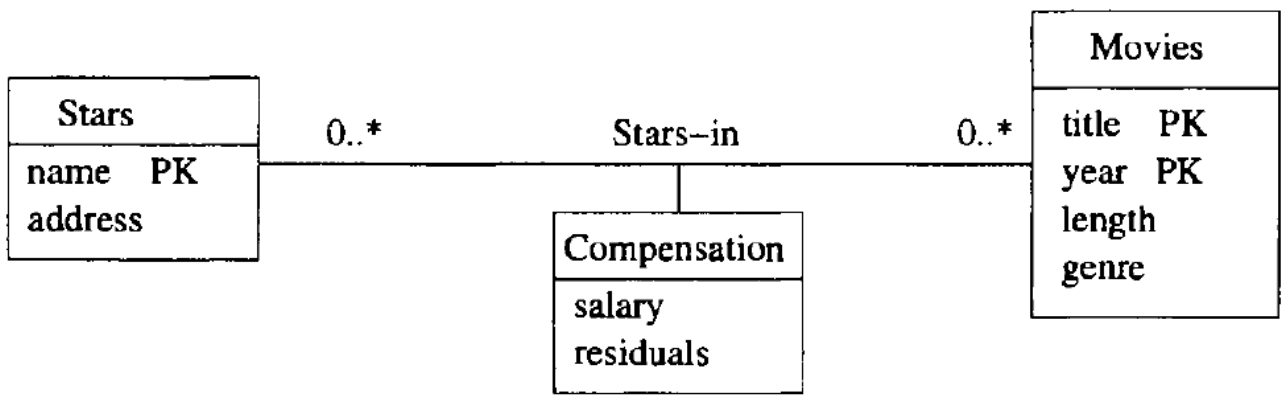
- A Company has Employees.
- A single manager is responsible for up to 10 workers.



7

4.7.5 Association Classes

We can attach attributes to an association in much the way we did in the E/R model, in Section 4.1.9.[5] In UML, we create a new class, called an *association class*, and attach it to the middle of the association. The association class has its own name, but its attributes may be thought of as attributes of the association to which it attaches.



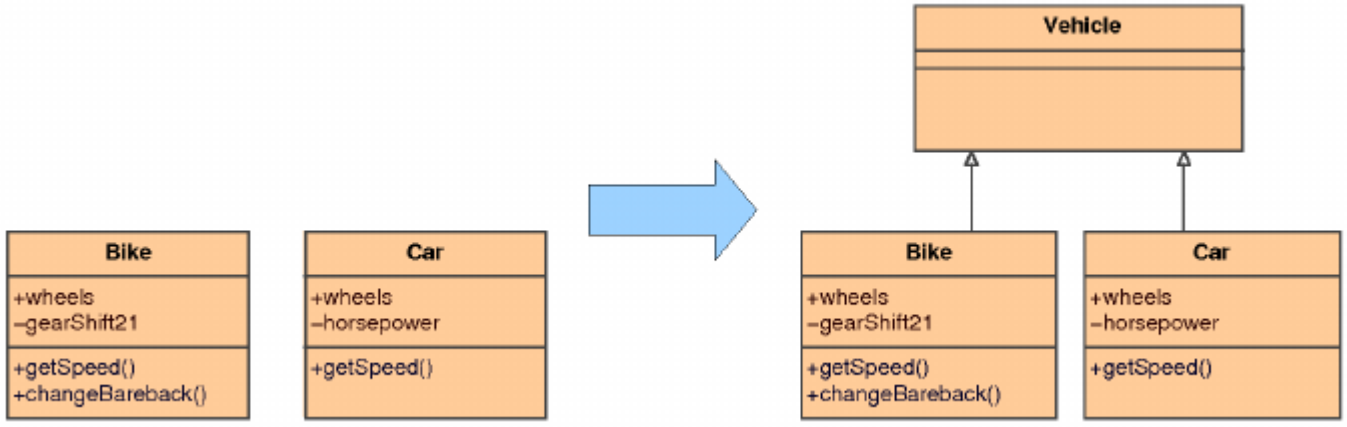
Subclasses in UML

- There are two kinds of Relationships
  - Generalization (parent-child relationship)
  - Association (student enrolls in the course)

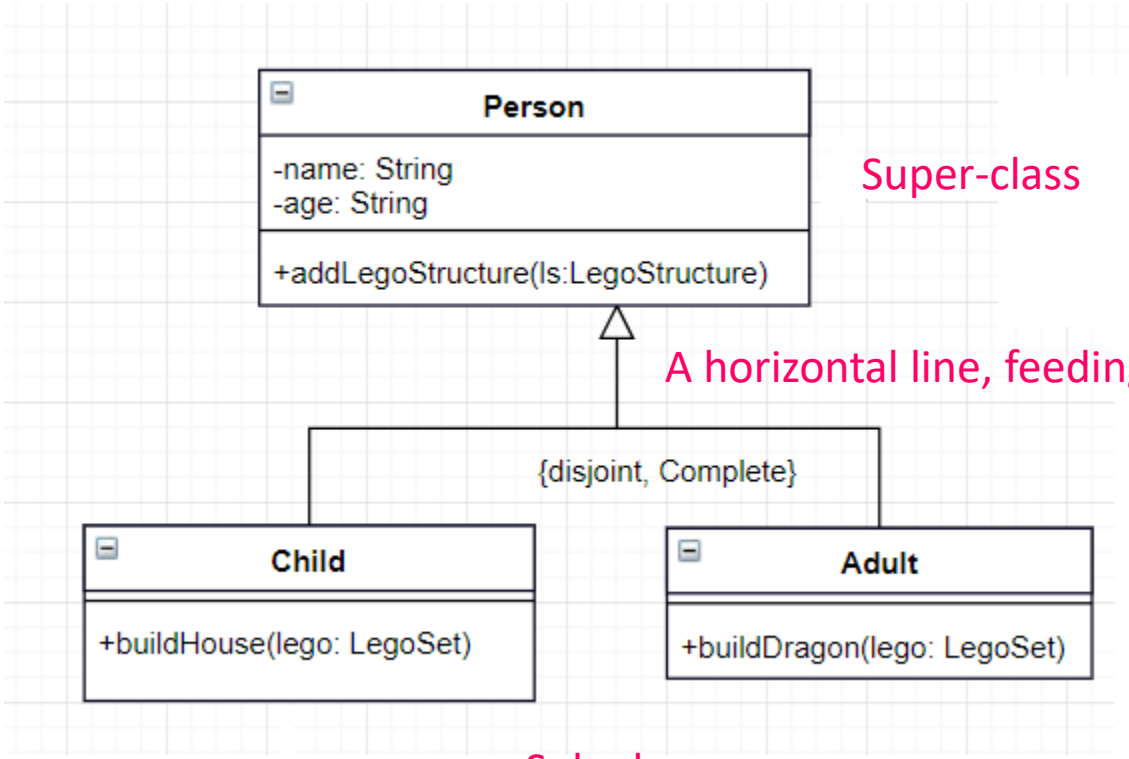
- Associations can be further classified as
  - Aggregation
  - Composition

Subclasses in UML

- Subclasses are presented by rectangles, like any class.
- We assume a sub-class inherits the properties(attributes and associations) from its superclass.



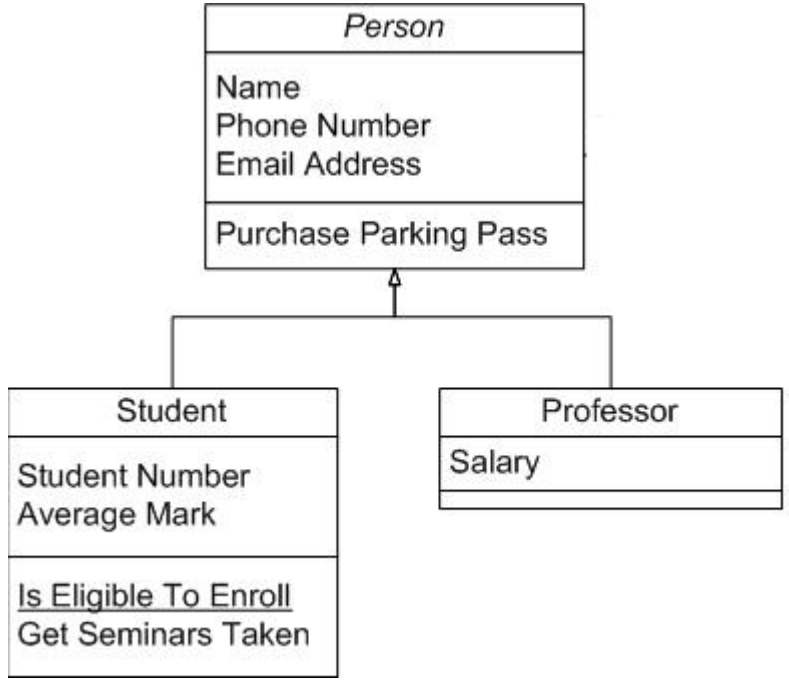
Subclasses in UML



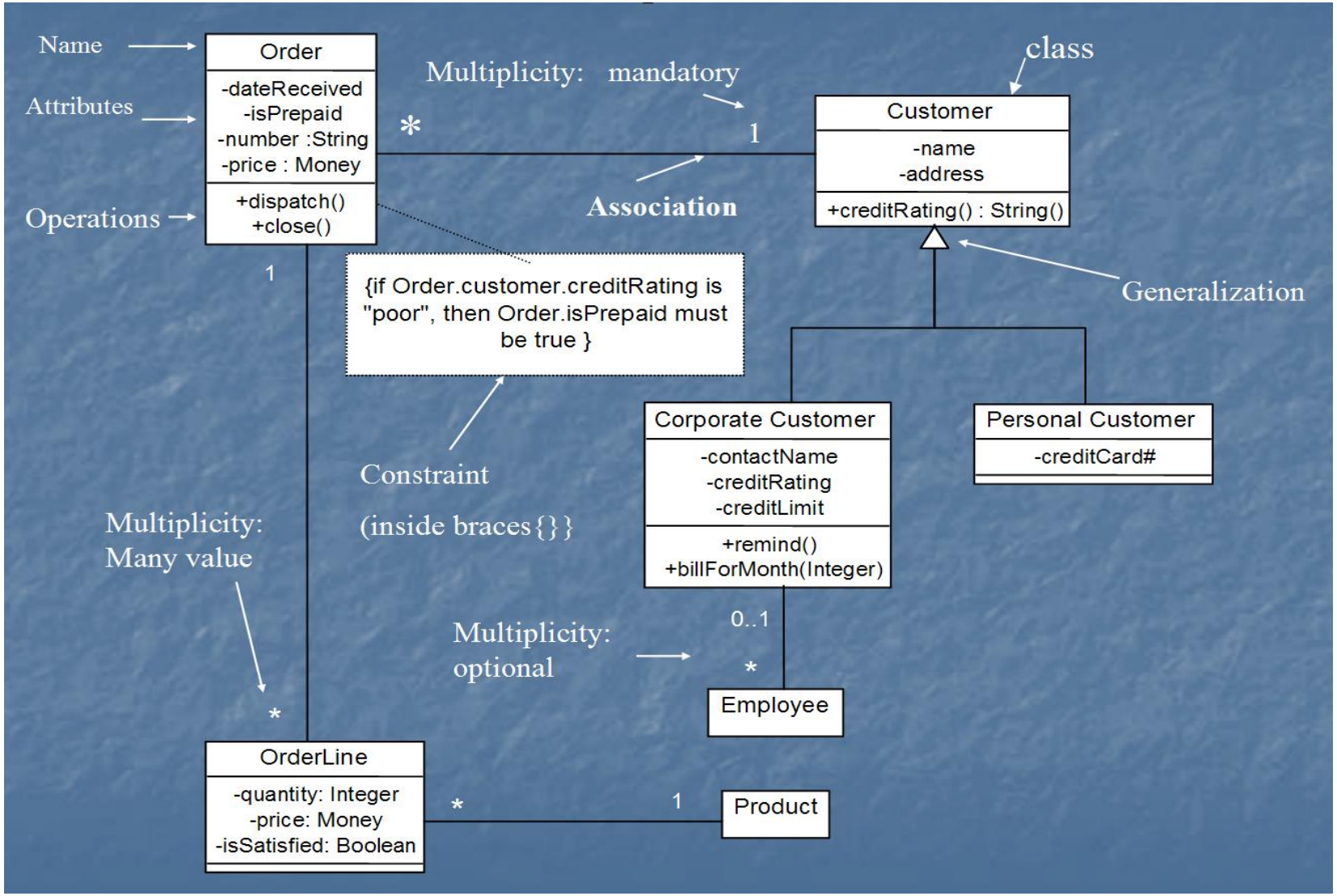
Super-class

A horizontal line, feeding into the arrow

Sub-class

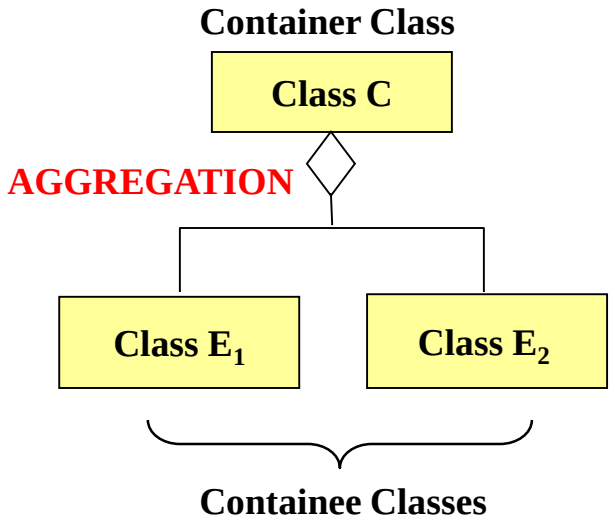


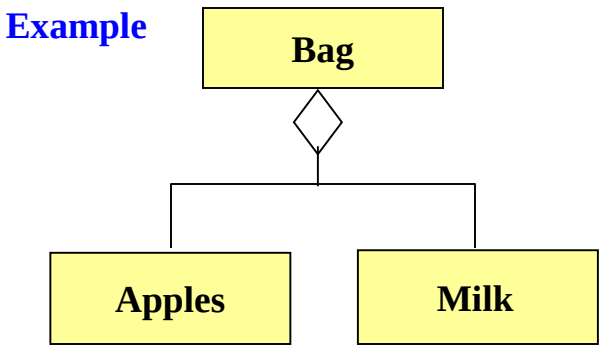
Sub-class may have its own attributes and additional, association


Name
Order
-dateReceived
-isPrepaid
-number :String
-price : Money
+dispatch()
+close()
Attributes
Operations
Multiplicity: mandatory
*
1
Association
class
Customer
-name
-address
+creditRating() : String()
Generalization
1
{if Order.customer.creditRating is "poor", then Order.isPrepaid must be true }
Constraint
(inside braces{})
Corporate Customer
-contactName
-creditRating
-creditLimit
+remind()
+billForMonth(Integer)
Personal Customer
-creditCard#
Multiplicity: Many value
*
Multiplicity: optional
0..1
*
Employee
OrderLine
-quantity: Integer
-price: Money
-isSatisfied: Boolean
*
1
Product

Aggregations

- expresses a relationship among instances of related classes. It is a specific kind of Container-Containee relationship.

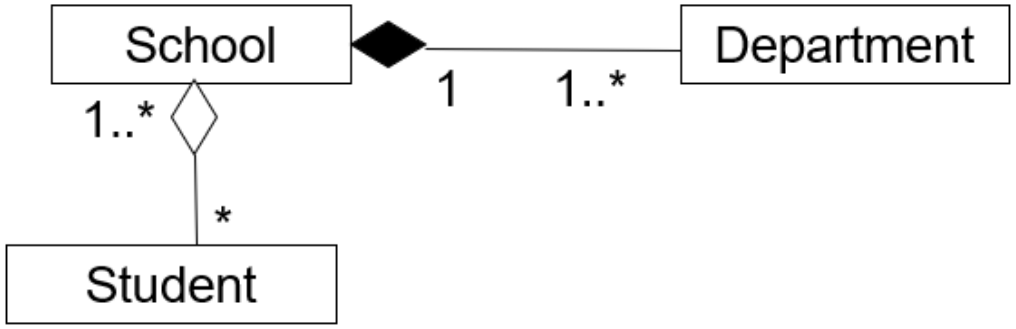




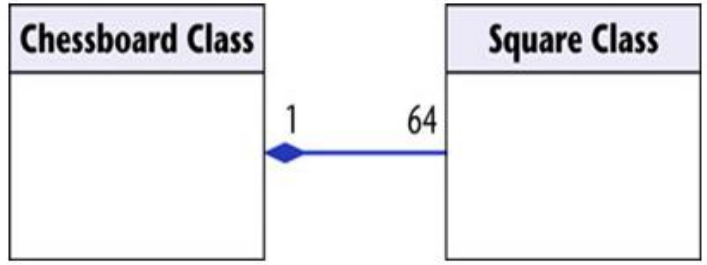
[From Dr.David A. Workman]

Composition

- Stronger relationship
  - ✓ One can not exist without the other
  - ✓ If the school folds, students live on
    - ♦ but the departments go away with the school



However, it is also possible in UML, to use **composition** as we used supporting relationships for **weak entity sets** in the E/R model.



The McGraw-Hill Companies, 2005 Figure 16.7

- ✓ Model aggregation or composition? When in doubt, use association (just a simple line)

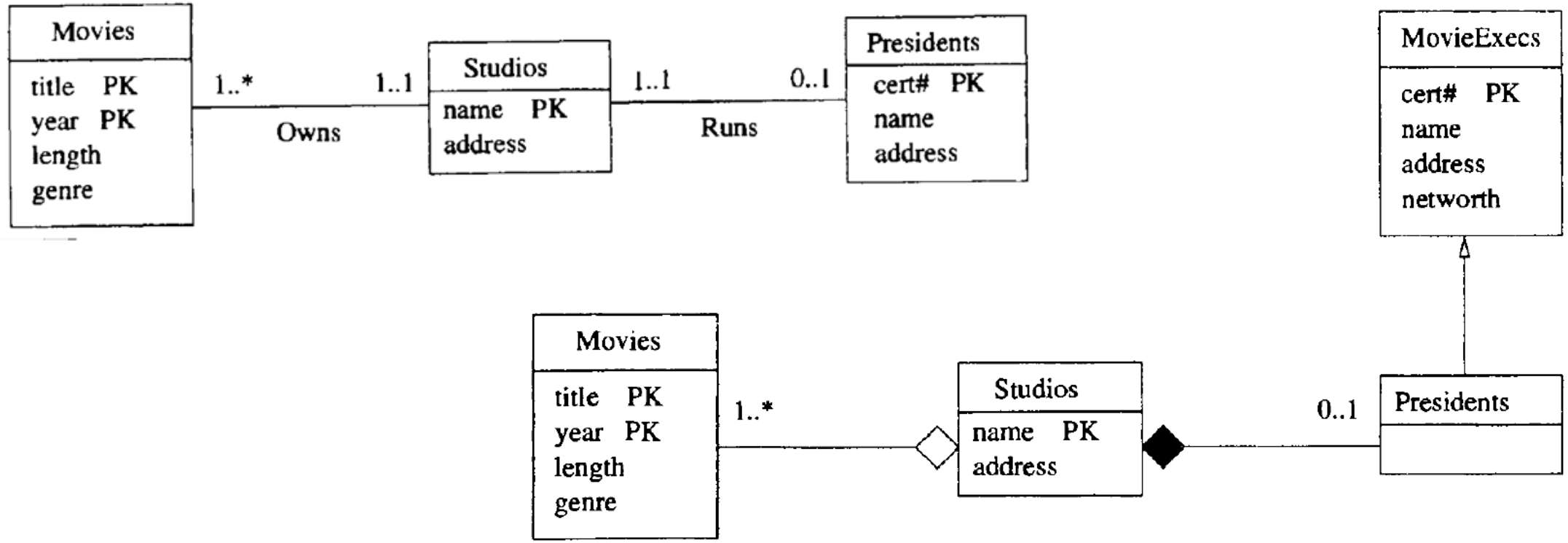


Figure 4.41: An aggregation from *Movies* to *Studios* and a composition from *Presidents* to *Studios*

A *composition* is similar to an association, but the label at the diamond end must be 1..1. That is, every object at the opposite end from the diamond must be connected to exactly one object at the diamond end. Compositions are distinguished by making the diamond be solid black.

From UML Diagram to Relations

- **Class to relations**
  - For each class, create a relation whose name is the name of the class
  - And whose attributes are the attributes of the class.
- **Associations to Relations**
  - For each association, create a relation with the name of that association
  - The attributes of the relation are the key attributes of the two connected classes (Rename if necessary).
  - If there is an association class attached to the association, include the attributes of the association class among the attributes of the relation.

Example 4.42: Consider the UML diagram of Fig. 4.36. For the three classes we create relations:

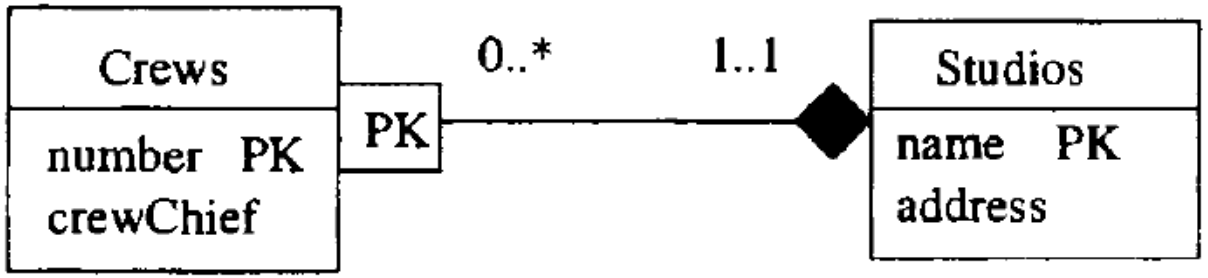
```
Movies(title, year, length genre)
Stars(name, address)
Studios(name, address)
```

For the two associations, we create relations

```
Stars-In(movieTitle, movieYear, starName)
Owns(movieTitle, movieYear, studioName)
```

```
Stars-In(movieTitle, movieYear, starName, salary, residuals)
```

Examples

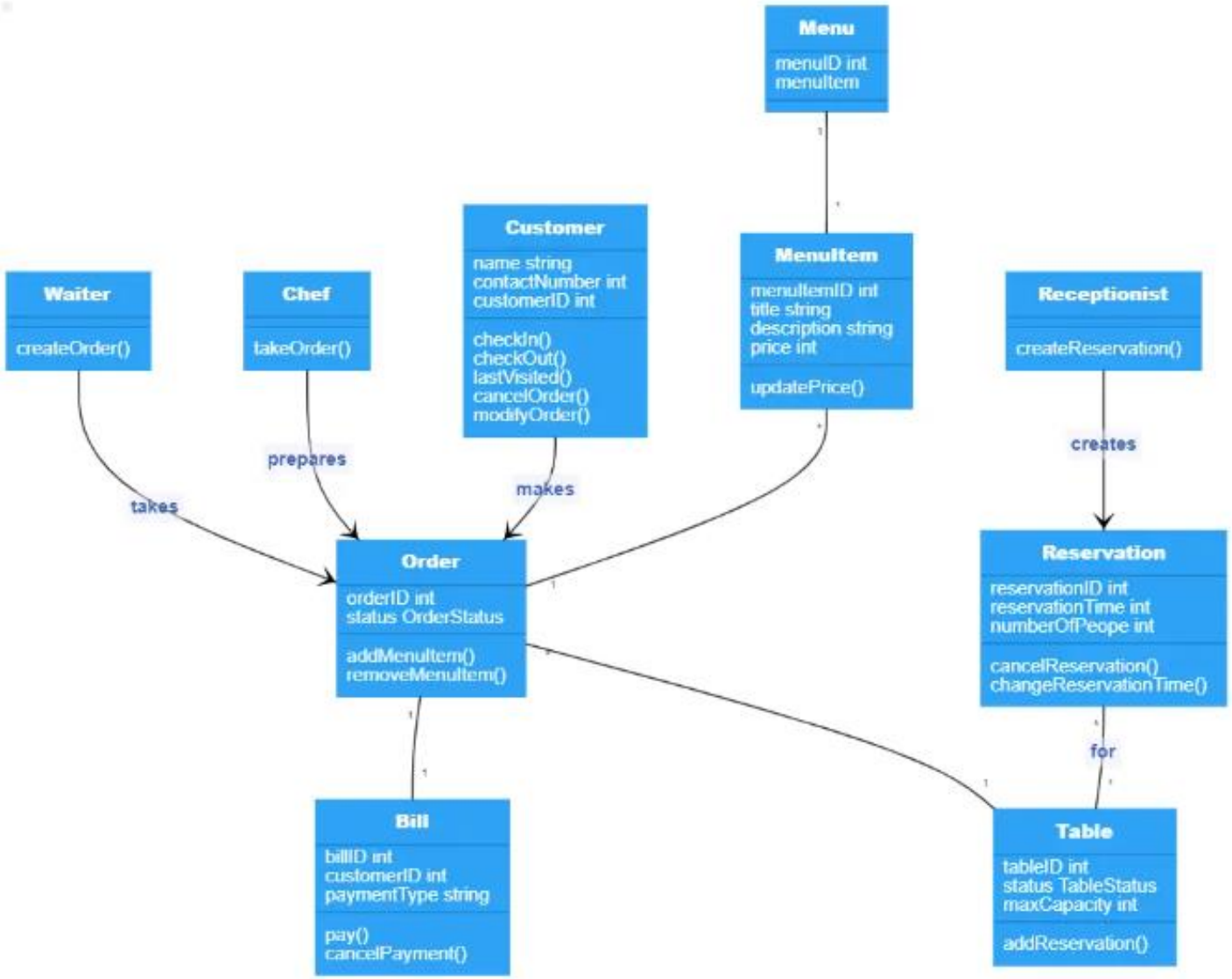


- Box labeled “PK” indicates that this composition provides part of the key for crews.
- The relation for class crews includes not only its own attribute **number**, but the key for class at the end of the composition, which is studios(name).

The relations for Example 4.44 are thus:

```
Studios(name, address)
Crews(number, crewChief, studioName)
```

As before, we renamed the attribute *name* of *Studios* in the `Crews` relation, for clarity.


Menu
menuID int
menuItem
Customer
name string
contactNumber int
customerID int
checkIn()
checkOut()
lastVisited()
cancelOrder()
modifyOrder()
Waiter
createOrder()
Chef
takeOrder()
MenuItem
menuItemID int
title string
description string
price int
updatePrice()
Receptionist
createReservation()
takes
prepares
makes
creates
Order
orderID int
status OrderStatus
addMenuItem()
removeMenuItem()
Reservation
reservationID int
reservationTime int
numberOfPeope int
cancelReservation()
changeReservationTime()
for
Bill
billID int
customerID int
paymentType string
pay()
cancelPayment()
Table
tableID int
status TableStatus
maxCapacity int
addReservation()

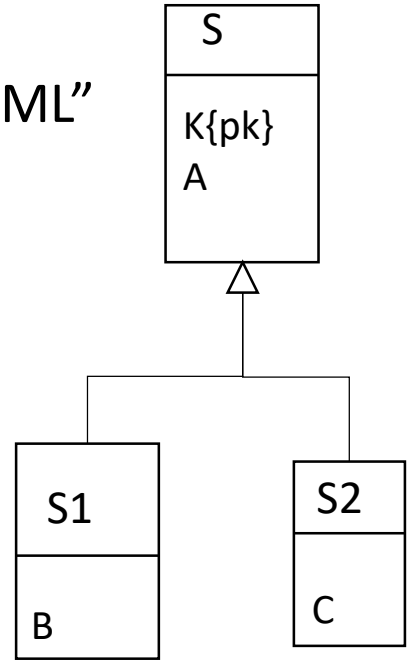
Converting sub-classes

- Three approaches to convert entity sub-classes in relations
  - Subclass relations contain superclass key + specialized attrs. ("UML" style)

  EX. S(<u>K</u>,A) S1(<u>K</u>,B), S2(<u>K</u>,C)
  - Subclass relations contain all attributes ("OO" Style)

  EX. S(<u>K</u>,A) S1(<u>K</u>,A, B), S2(<u>K</u>,A,C)
  - One relation containing all superclass + subclass attrs.

  EX. S(<u>K</u>,A, B, C)



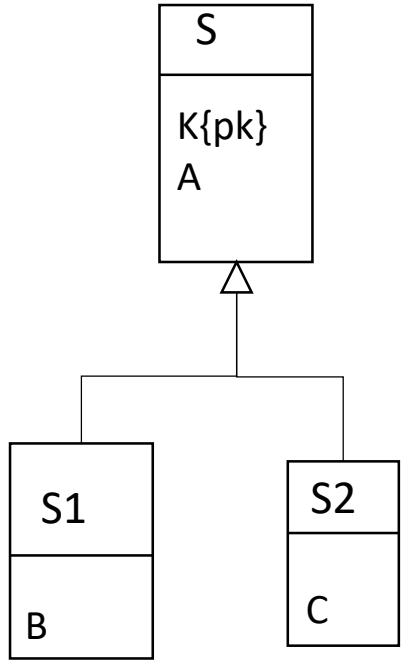
- Pros/cons depend on:
  - the frequent queries…
  - data characteristics
  - sub-classes type (complete/partial; disjoint/overlapping)

Converting entity sub-classes: “UML” style

- Create a relation for the “root” class (as usual)
  - It’s key k is the identifier of the class

- For each sub-class create a relation with the key attributes (k) + **its own specific attributes**

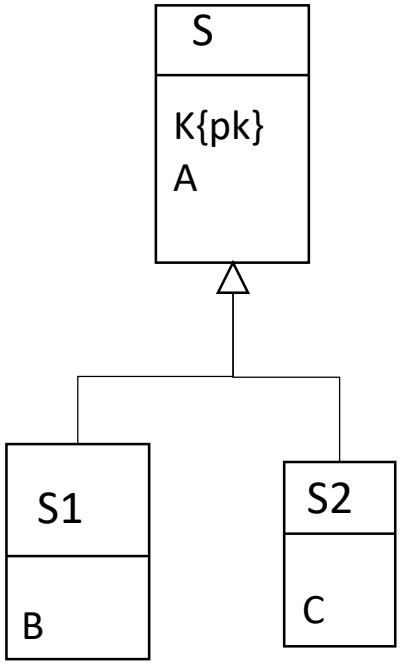
  S(K,A) S1(K,B) S2(K,C)



Converting entity sub-types: “OO” style

- Create a relation for each class and for each sub-class with all its attributes (**own+inherited**)
  - The key is based on the identifier of the “root” entity

    S(K,A) S1(K,A, B) S2(K,A,C)



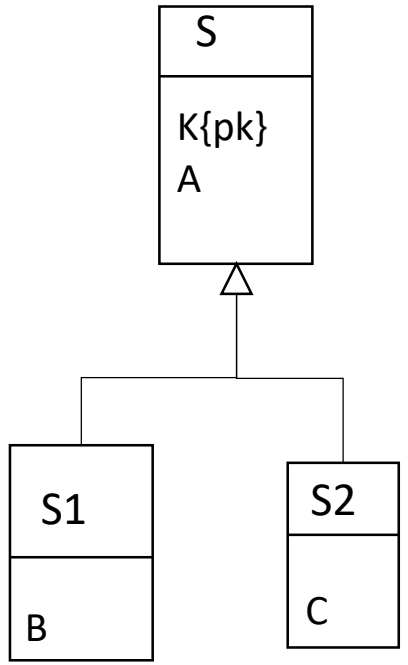
Converting sub-classes: attributes and null values

- Create one single relation with all the attributes of the class hierarchy

  S(K,A, B, C)

- Instances have null in attributes that don't belong to them

- Specific attributes can be used to reflect sub-classes



Object Definition Language

- ODL is as a text-based language for specifying the structure of databases in object-oriented terms.
- Like UML, the class is the central concept in ODL.

A declaration of a class in ODL, in its simplest form, is:

```
class <name> {
    <list of properties>
};
```

Attributes in ODL

- In ODL, attributes need not be of simple types such as integers
- An attribute is represented in the declaration for its class by the keyword attribute, the type, and the name of attribute.

```
class Movie {
   attribute string title;
   attribute integer year;
   attribute integer length;
   attribute enum Genres
          {drama, comedy, sciFi, teen} genre;
};
```

Here genres is enumerated type (list of symbolic constants).
The four values that genre is allowed to take are drama, comedy, …..

Attributes in ODL

- Attribute Address has a type that is a record structure
- The name of this structure is Addr. It consists of two fields: street and city

```
class Star {
    attribute string name;
    attribute Struct Addr
        {string street, string city} address;
};
```

Relationships in ODL

- An ODL relationship is declared inside a class declaration by the keyword relationship, a type, and the name of the relationship.

- For example, the best way to represent the connection between the Movie and Star classes is with a relationship.
- We add this line in the declaration of class Movie.

```
relationship Set<Star> stars;
```

Multiplicity of relationships

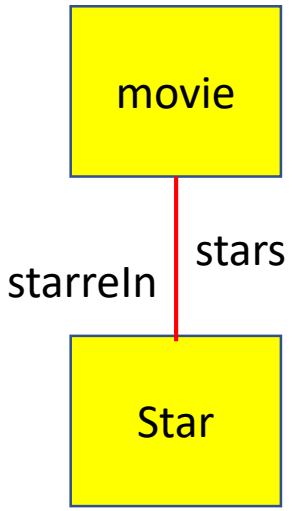
- If we have **many-many** relationships between classes C and D
  - Set<D>, Set <C>
- If the relationship is **many-one** from C to D,
  - The type of the relationship in C is just D
  - while the type of the relationship in D is set<C>.
- If the relationship is **one-one**,
  - the type of the relationship in C is just D
  - and in D it is just C.

```
1)  class Movie {
2)     attribute string title;
3)     attribute integer year;
4)     attribute integer length;
5)     attribute enum Genres
              {drama, comedy, sciFi, teen} genre;
6)     relationship Set<Star> stars
                  inverse Star::starredIn;
7)     relationship Studio ownedBy
                  inverse Studio::owns;
    };

8)  class Star {
9)     attribute string name;
10)    attribute Struct Addr
           {string street, string city} address;
11)    relationship Set<Movie> starredIn
                  inverse Movie::stars;
    };

12) class Studio {
13)    attribute string name;
14)    attribute Star::Addr address;
15)    relationship Set<Movie> owns
                  inverse Movie::ownedBy;
    };
```

Many-many relationships between Star and movie



Since the type of OwnedBy is Studio, while the type of owns is Set<Movie>, we see that this pair od inverse relationship is many-one from Movie to studio.

Declaring Keys in ODL

- The declaration of a key or keys for a class is optional.
  - ODL assumes that all objects have an object-identity

```
class Movie (key (title, year)) {
```

Subclasses in ODL

- Class C to be a subclass of another class D
  - Follow the name C in its declaration with the keyword **extends** and the name D
  - Then class C inherits all the properties of D and may have additional properties of its own.

```
class MurderMystery extends Movie {
   attribute string weapon;
};
```

From ODL Design to relational Design

- Page 193-196