Chapter 4
High-Level Database Models

Université Grenoble Alpes

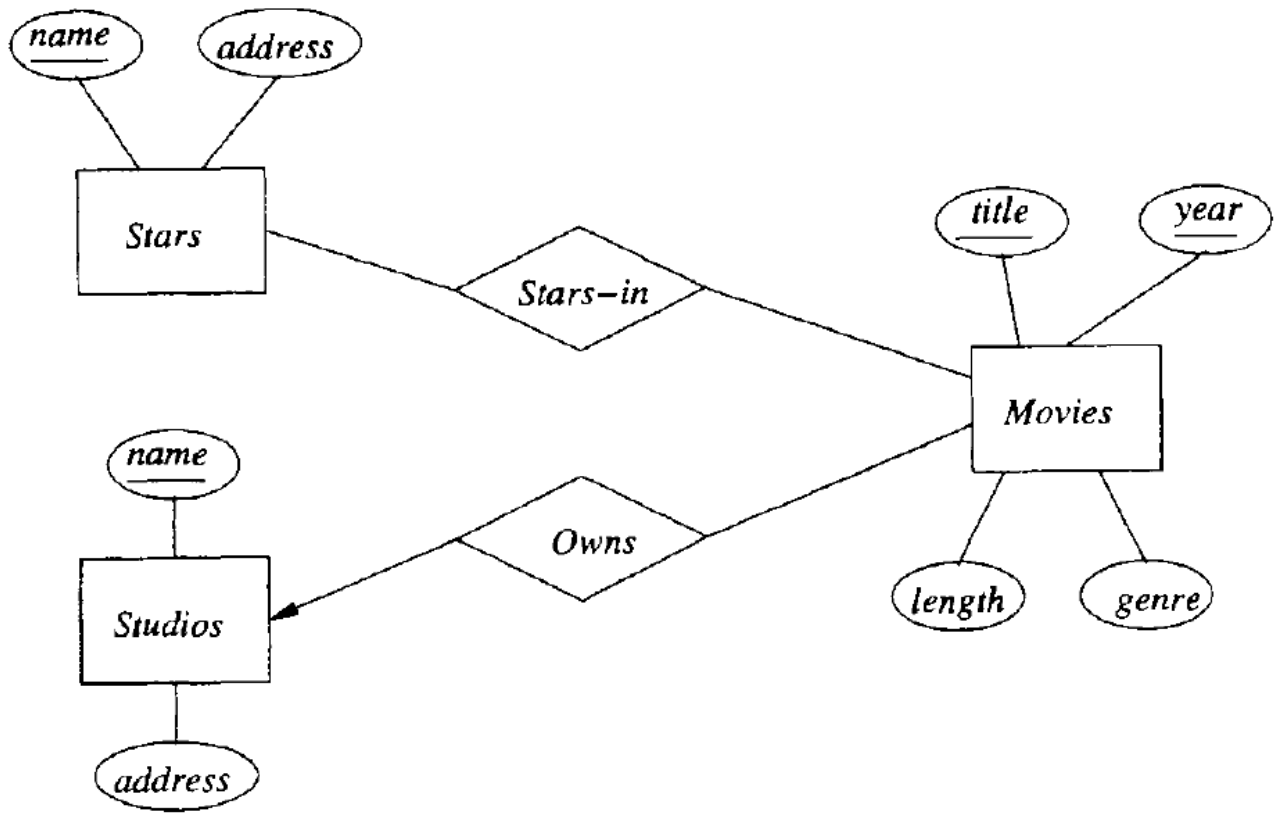
21/02/2023

Bahareh Afshinpour

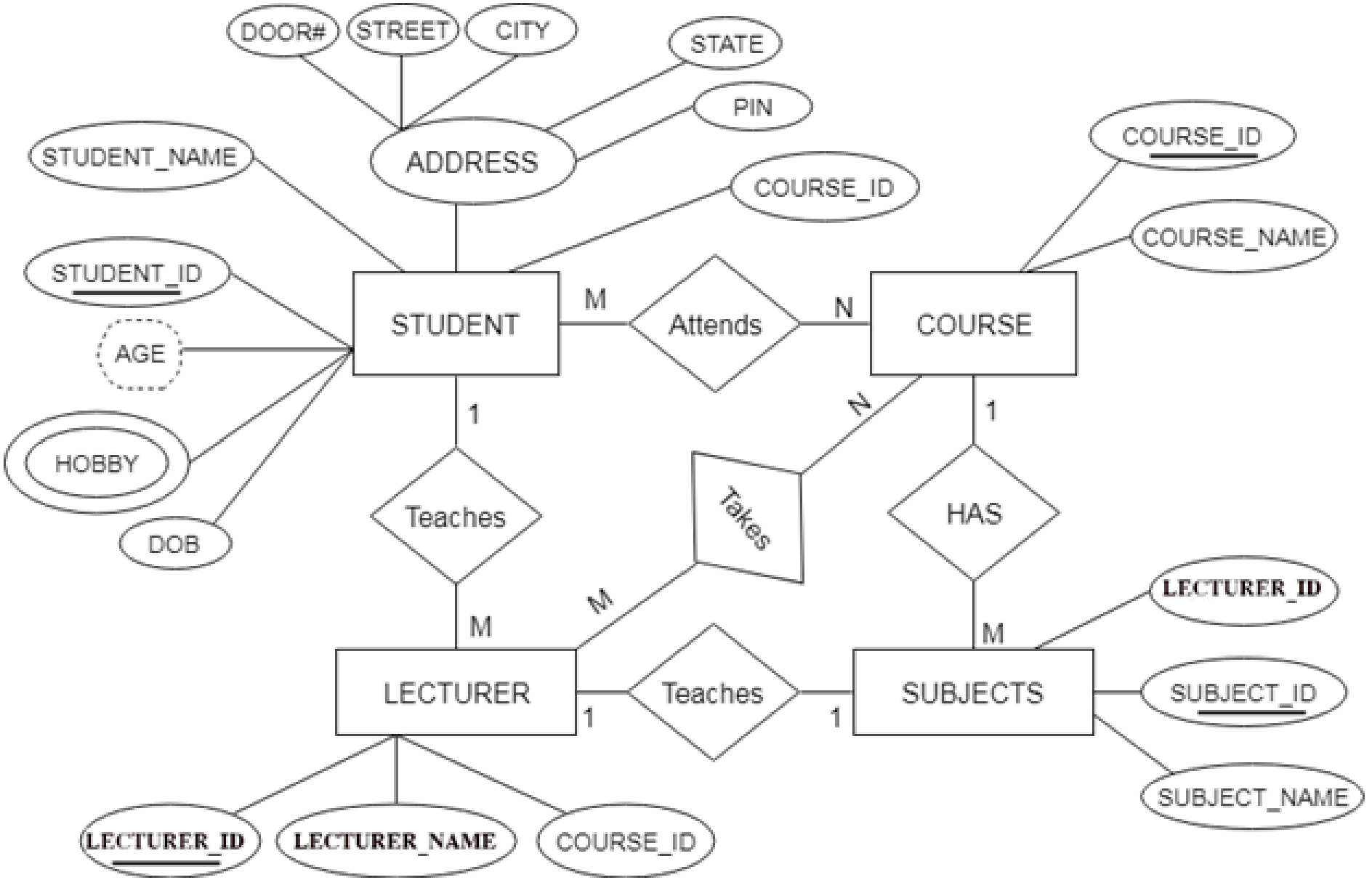
bahareh.afshinpour@univ-grenoble-alpes.fr

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

Representing Keys in the E/R Model



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

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.

Design Principles

- **Faithfulness**

➢ **Entity sets and their attributes should reflect reality.**

Example 4.12: If we define a relationship *Stars-in* between *Stars* and *Movies*, it should be a many-many relationship. The reason is that an observation of the real world tells us that stars can appear in more than one movie, and movies can have more than one star. It is incorrect to declare the relationship *Stars-in* to be many-one in either direction or to be one-one. □

Example 4.13: On the other hand, sometimes it is less obvious what the real world requires us to do in our E/R design. Consider, for instance, entity sets *Courses* and *Instructors*, with a relationship *Teaches* between them. Is *Teaches* many-one from *Courses* to *Instructors*? The answer lies in the policy and intentions of the organization creating the database. It is possible that the school has a policy that there can be only one instructor for any course. Even if several instructors may "team-teach" a course, the school may require that exactly one of them be listed in the database as the instructor responsible for the course. In either of these cases, we would make *Teaches* a many-one relationship from *Courses* to *Instructors*.

- **Avoiding Redundancy**

➢We should be careful to say everything once only.

  - Need extra space
  - Update-anomaly may happen(ex: change relationship but not attribute)

- **Choosing the Right Relationships**

➢Adding to our design every possible relationship is not often a good idea.

➢It can lead to redundancy, update anomalies, and deletion anomalies, where the connected pairs or sets of entities for one relationship can be deduced from one or more other relationships.

In summary, we cannot tell you whether a given relationship will be redundant. You must find out from those who wish the database implemented what to expect.

- **Simplicity counts**

➢Avoid introducing more elements into your design than is absolutely necessary.

- **Picking the Right kind of element**
  - ➢In general, an attribute is simpler to implement than either an entity set or a relationship.
  - ➢However, making everything an attribute will usually get us into trouble.

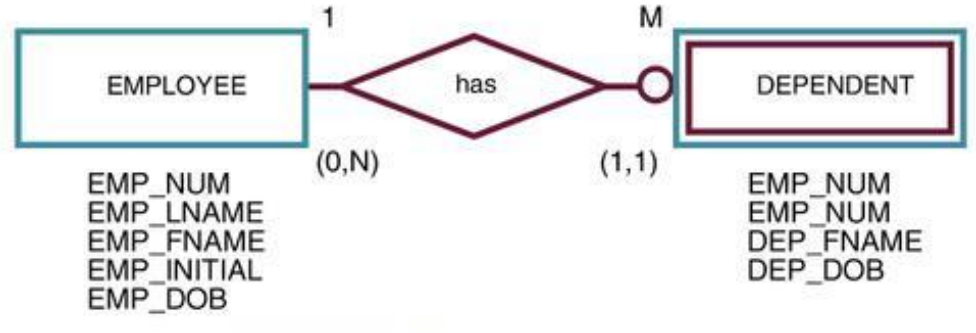
Example

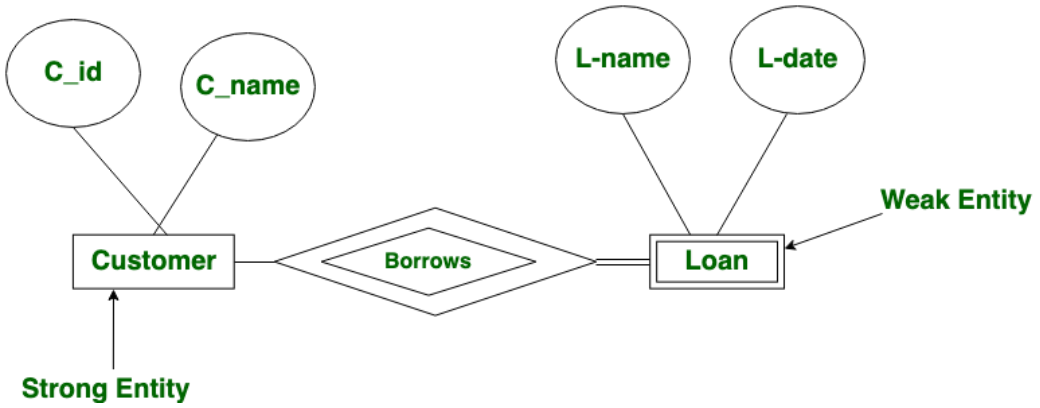
Example 4.17: Let us consider a specific problem. In Fig. 4.2, were we wise to make studios an entity set? Should we instead have made the name and address of the studio be attributes of movies and eliminated the *Studio* entity set? One problem with doing so is that we repeat the address of the studio for each movie. We can also have an update anomaly if we change the address for one movie but not another with the same studio, and we can have a deletion anomaly if we delete the last movie owned by a given studio.

On the other hand, if we did not record addresses of studios, then there is no harm in making the studio name an attribute of movies. We have no anomalies in this case. Saying the name of a studio for each movie is not true redundancy, since we must represent the owner of each movie somehow, and saying the name of the studio is a reasonable way to do so. □

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. □

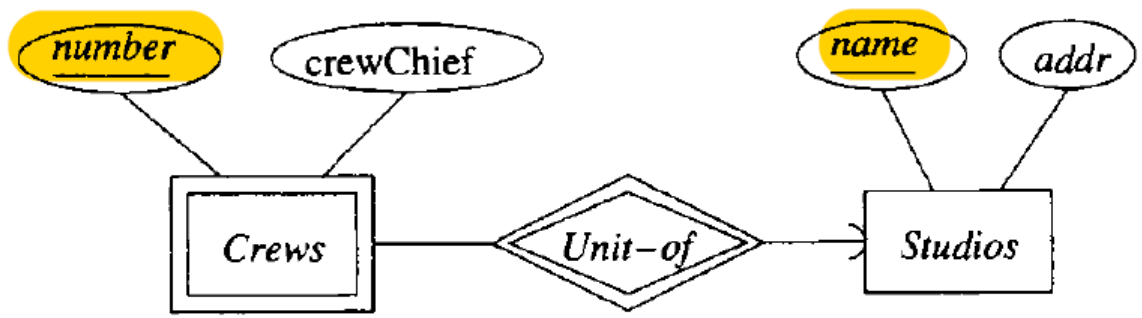


Figure 4.20: A weak entity set for crews, and its connections

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.

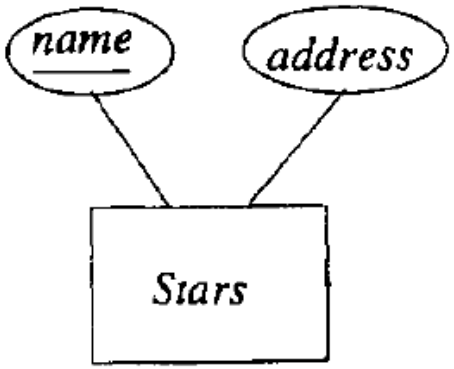
3. If an entity set supplies any attributes for its own key, then those attributes will be underlined. An example is in Fig. 4.20, where the number of a crew participates in its own key, although it is not the complete key for *Crews*.

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

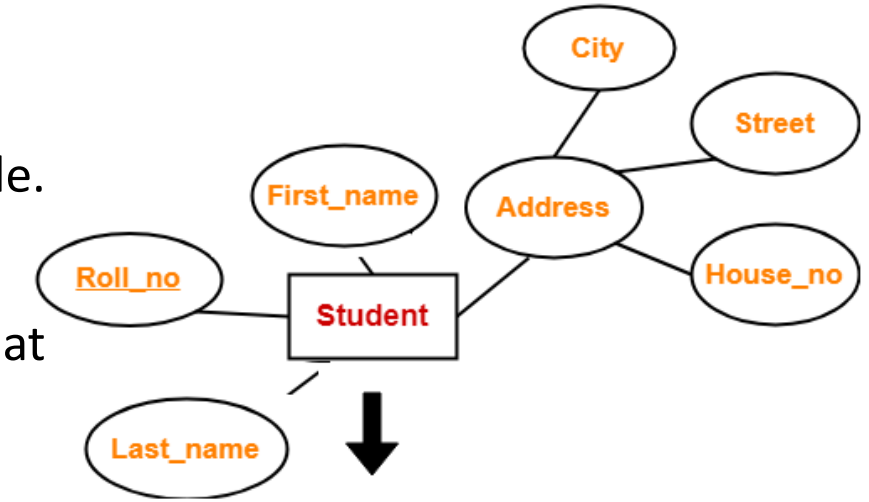
From E/R diagrams to Relational designs

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)

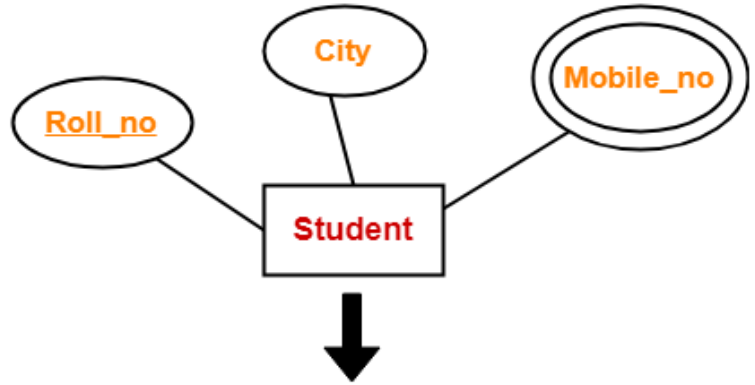
https://www.gatevidyalay.com/er-diagrams-to-tables/

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

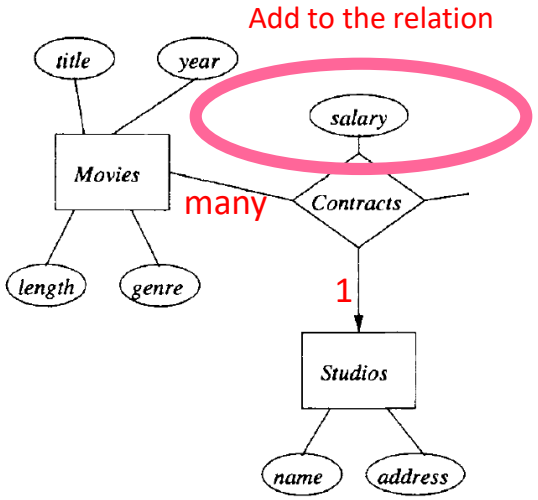
Because R is many-one, all these attributes are functionally determined by the key for E, and we can combine them into one relation with a schema consisting of:

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

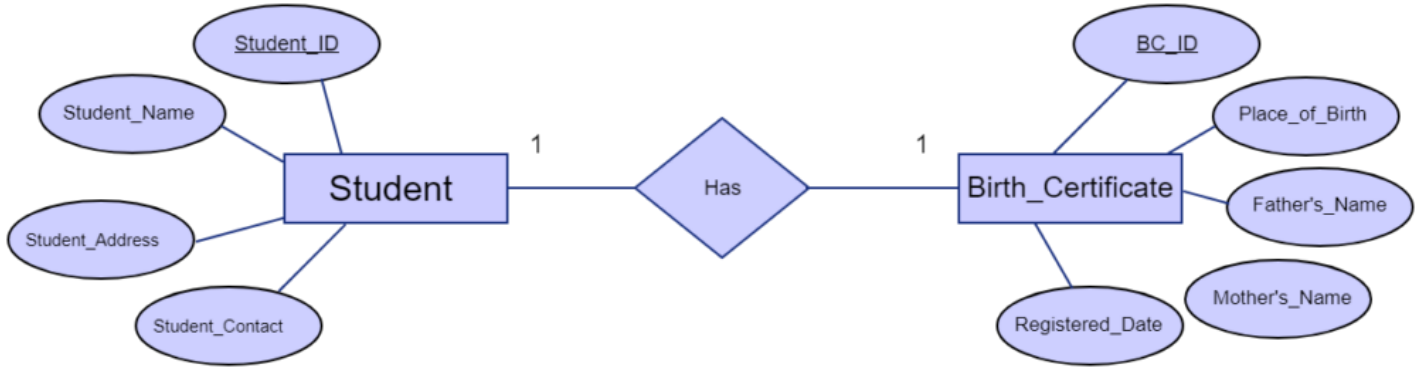
Movies:

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, ….)

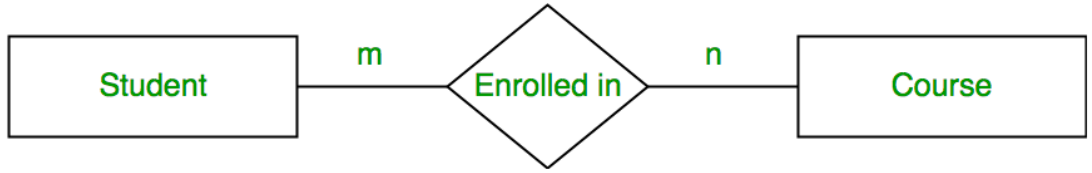
There's no need for a new table. Only the primary key of one entity should be added to another

Combining relations : many to many

- Here, three tables will be required:

Create a new table for the relation

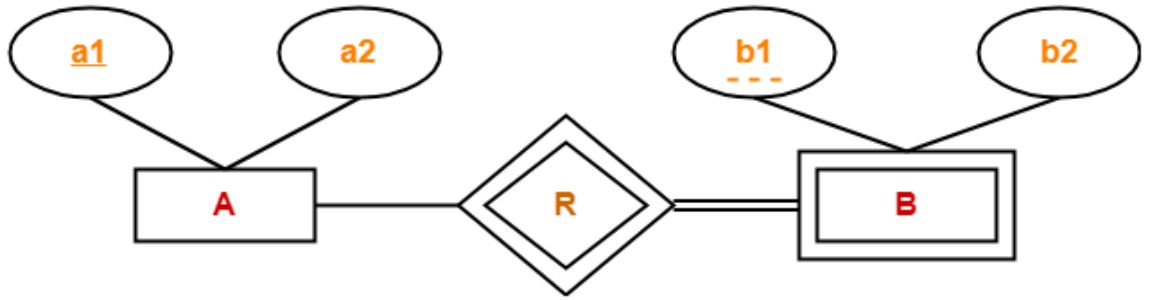
- ✓ **Student(a1,a2,a3,….)**
- ✓ **Course(c1,c2,c3,…)**
- ✓ **Enrolled(a1,c1,…)**



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

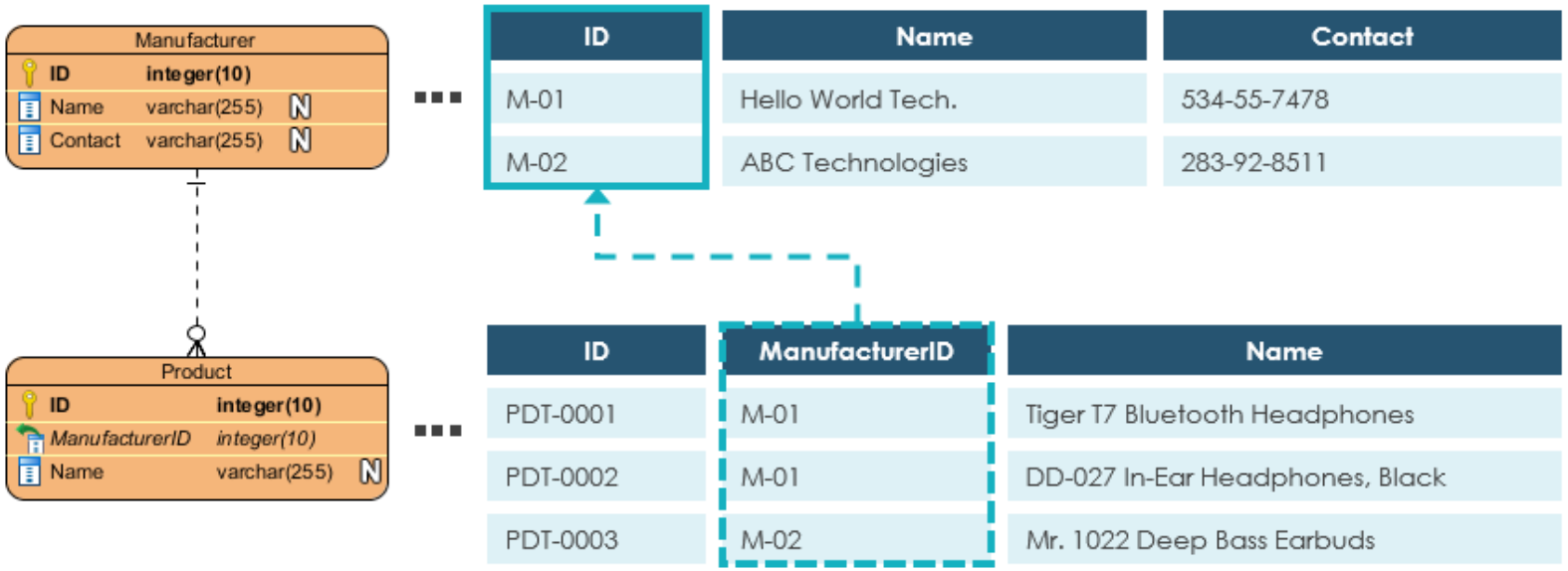


Here, two tables will be required-

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.

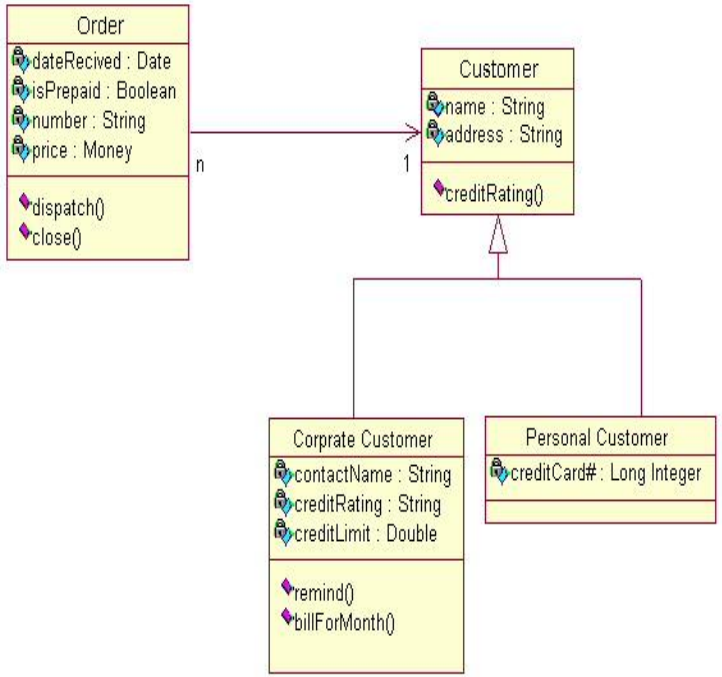
UML	E/R Model
Class	Entity set
Association	Binary relationship
Association Class	Attributes on a relationship
Subclass	Isa hierarchy
Aggregation	Many-one relationship
Composition	Many-one relationship with referential integrity

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.

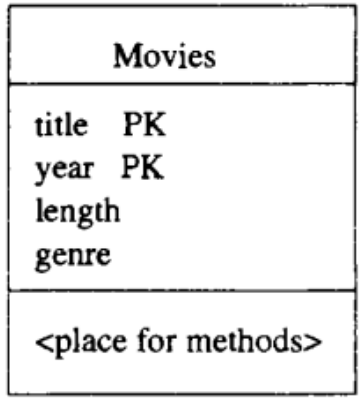


Figure 4.35: The *Movies* class in UML

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.

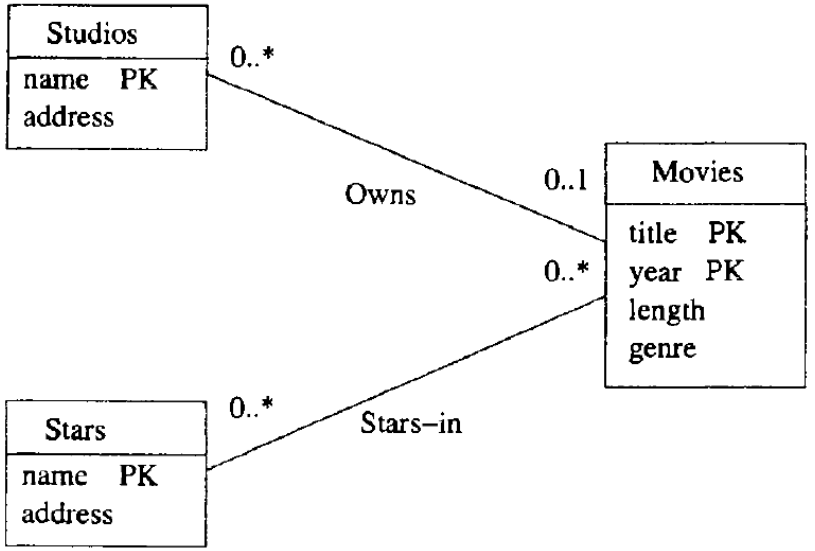
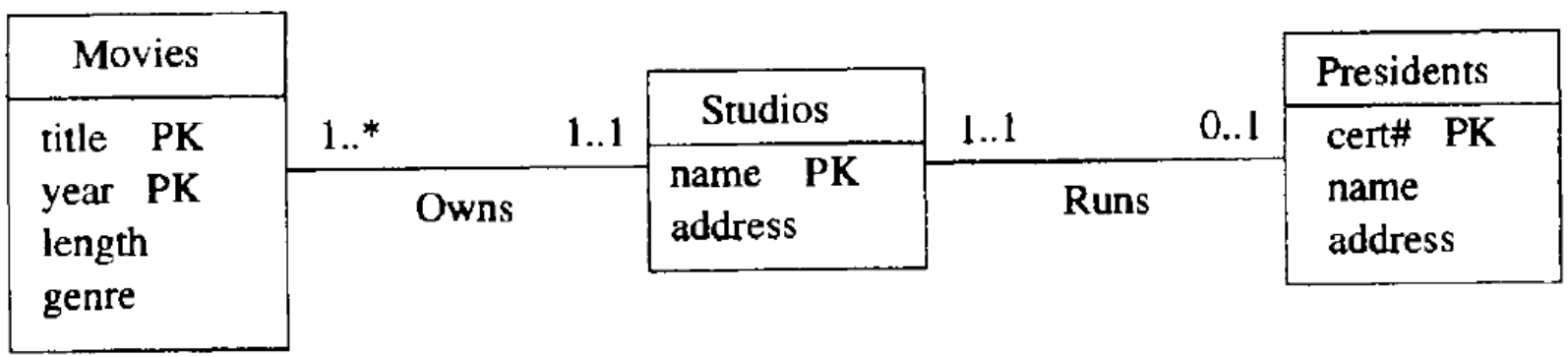


Figure 4.36: Movies, stars, and studios in UML

- 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."

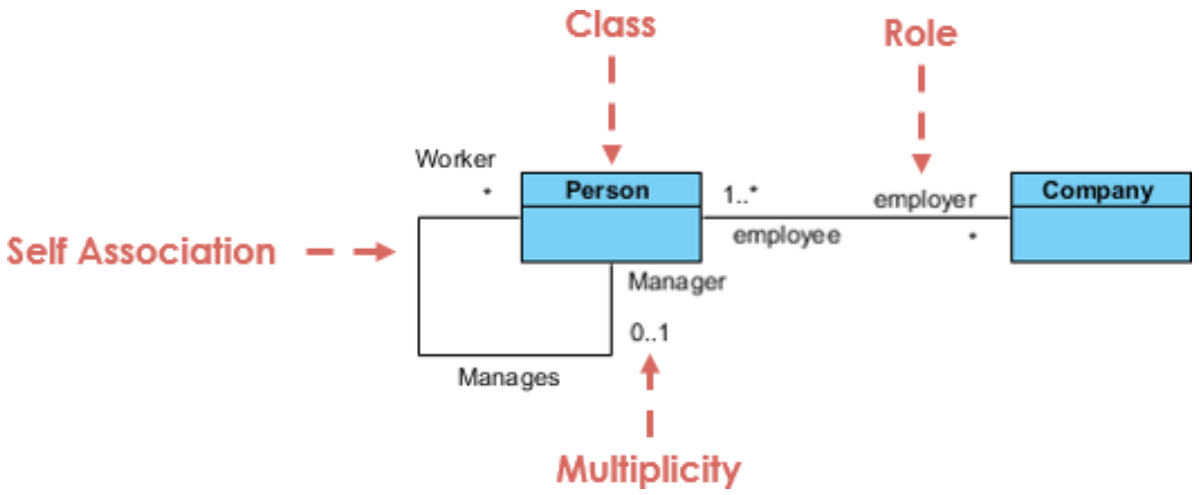
Example

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. □



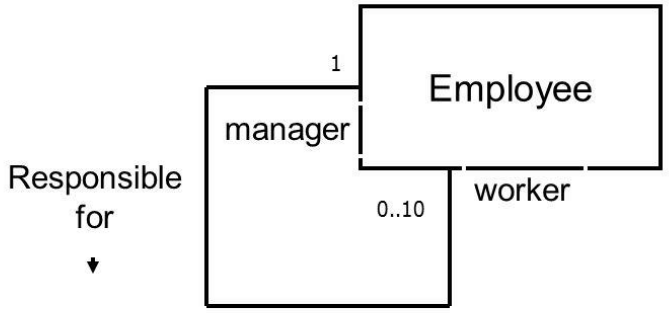
Self- Associations

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

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.