# Databases

Lecture 13

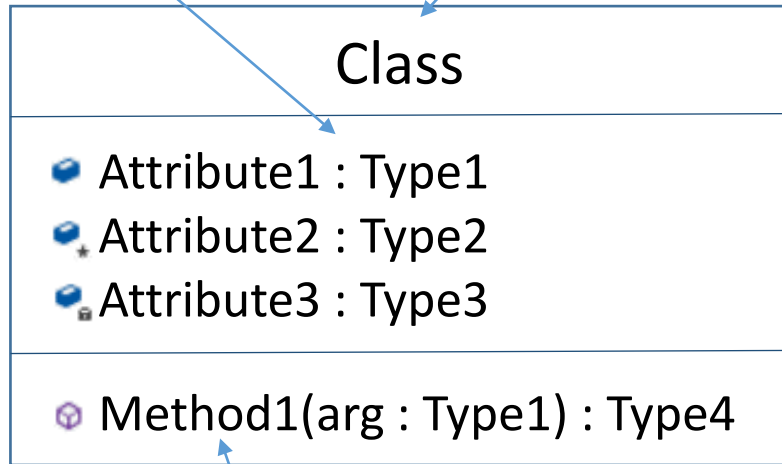Conceptual Modeling

Sabina S. CS

- database design - stages
  - requirements analysis
    - what data will the DB store?
    - what are the main operations to be supported?
    - what apps will be powered by the DB?
  - conceptual DB design
    - high level description of data and integrity constraints
  - logical DB design
    - translate the conceptual DB design to a DB schema in terms of the model supported by the DBMS (e.g., relational)
  - schema refinement
    - normalization
    - eliminate redundancy and associated problems

- database design - stages
  - physical DB design
    - create indexes
    - redesign parts of the schema
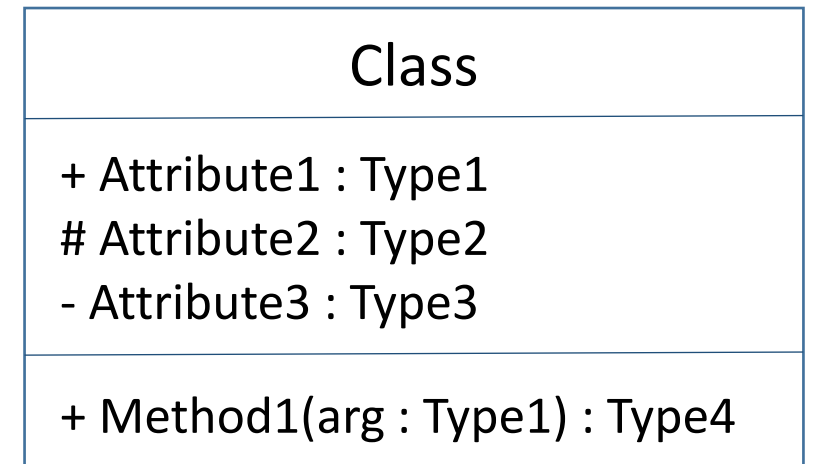
- UML class diagram
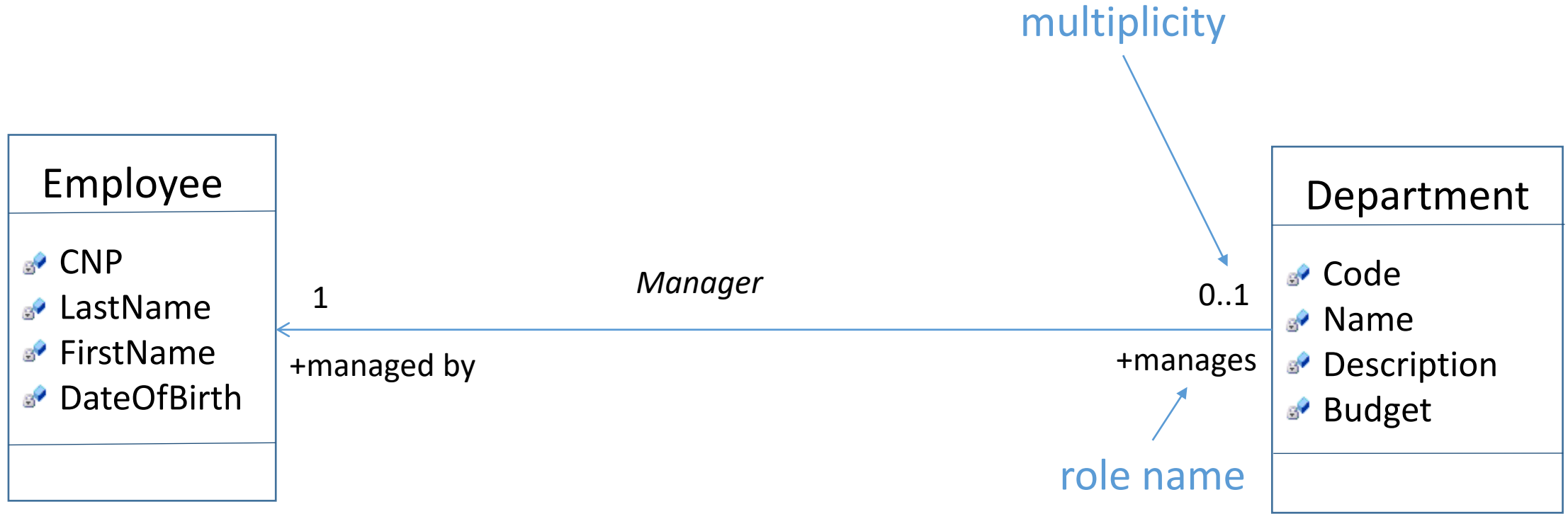  - classes

attributes                    name

| Class |
| --- |
| Attribute1 : Type1<br>Attribute2 : Type2<br>Attribute3 : Type3 |
| Method1(arg : Type1) : Type4 |

public
protected
private

| Class |
| --- |
| + Attribute1 : Type1<br># Attribute2 : Type2<br>- Attribute3 : Type3 |
| + Method1(arg : Type1) : Type4 |

methods

- UML class diagram
  - associations

multiplicity

| Employee | | Department |
|---|---|---|
| CNP LastName FirstName DateOfBirth | | Code Name Description Budget |

Manager

1          0..1

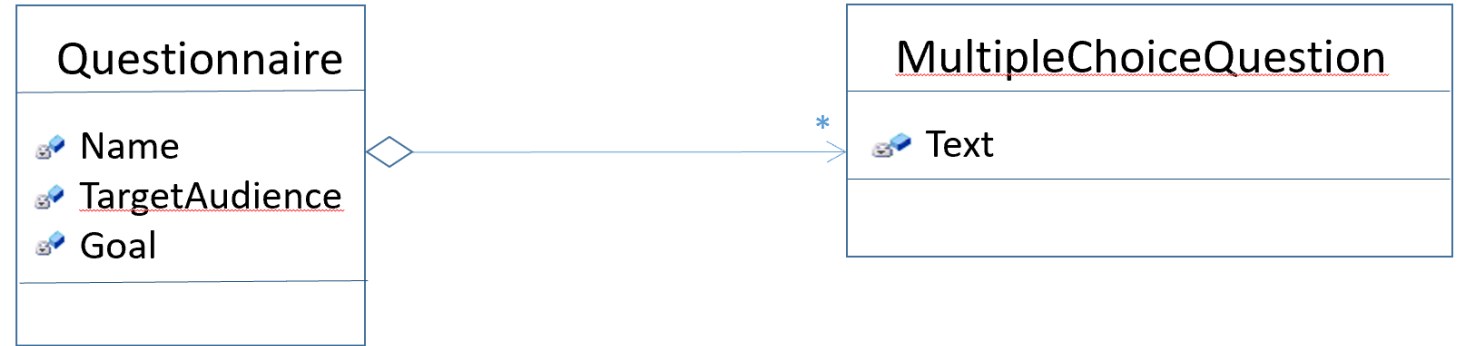+managed by          +manages

role name

  - navigability – unidirectional, bidirectional
  - multiplicity – examples
    - 0..1          • 7..10
    - 5
    - 0..*

- UML class diagram
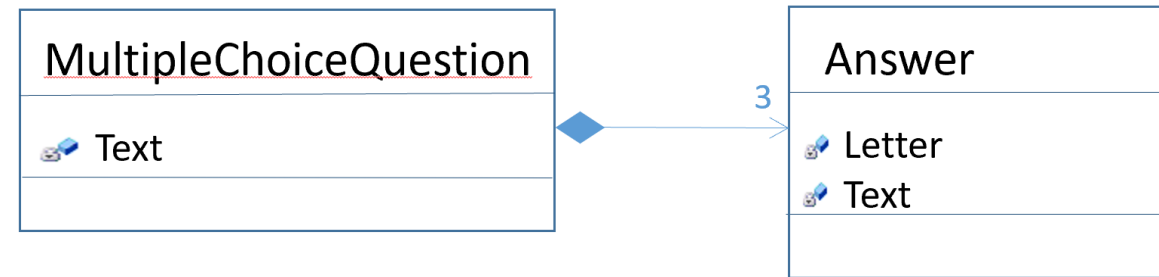  - associations



**Employee**

- CNP
- LastName
- FirstName
- DateOfBirth

**Department**

- Code
- Name
- Description
- Budget

1   *Manager*   +manages

+managed by   0..1

An employee manages 0 or 1 departments.

Sabina S. CS

- UML class diagram
  - aggregation
  - composition
  - reflexive association

**Questionnaire**

- Name
- TargetAudience
- Goal

**MultipleChoiceQuestion**

- Text

◇————————→ *

**MultipleChoiceQuestion**

- Text

**Answer**

- Letter
- Text

◆————→ 3

**Employee**

- Name
- Address
- Profession

+supervisor  0..1

0..*
+subordinate

Sabina S. CS
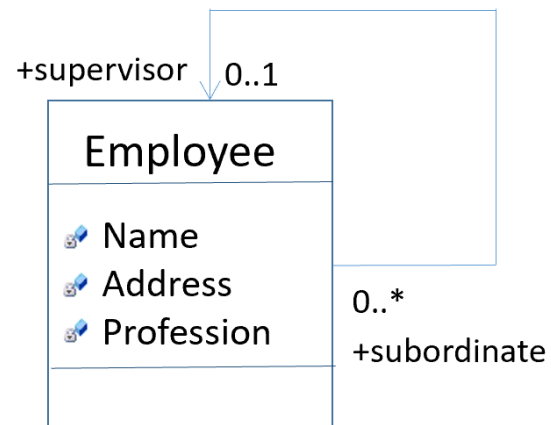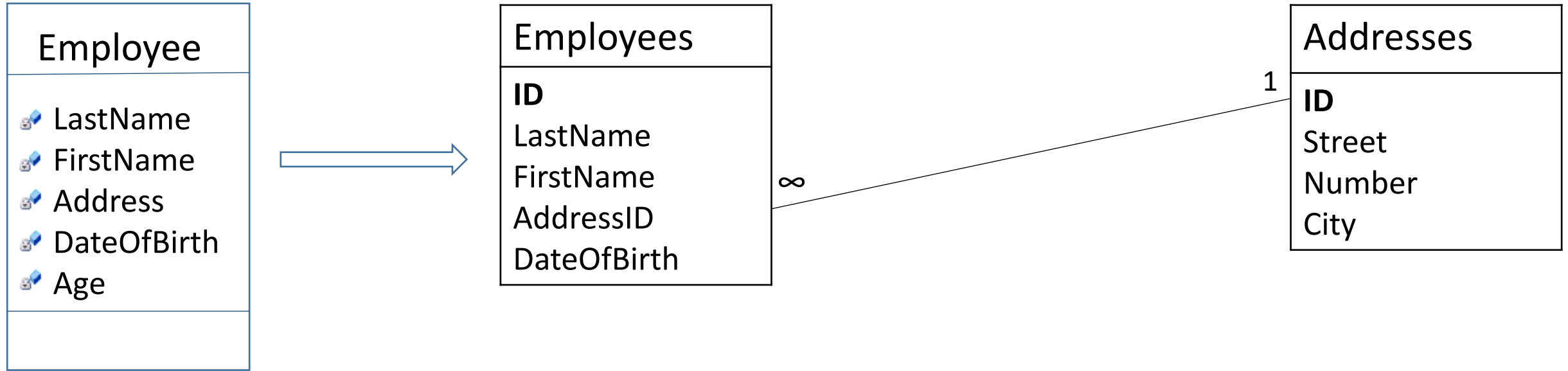
- conceptual model => relational database
- 1:1 mapping, i.e., classes become tables
- drawbacks
  - one could create too many tables
    - too many tables => too many join operations
  - necessary tables could be omitted; m:n associations require a third table (join table)
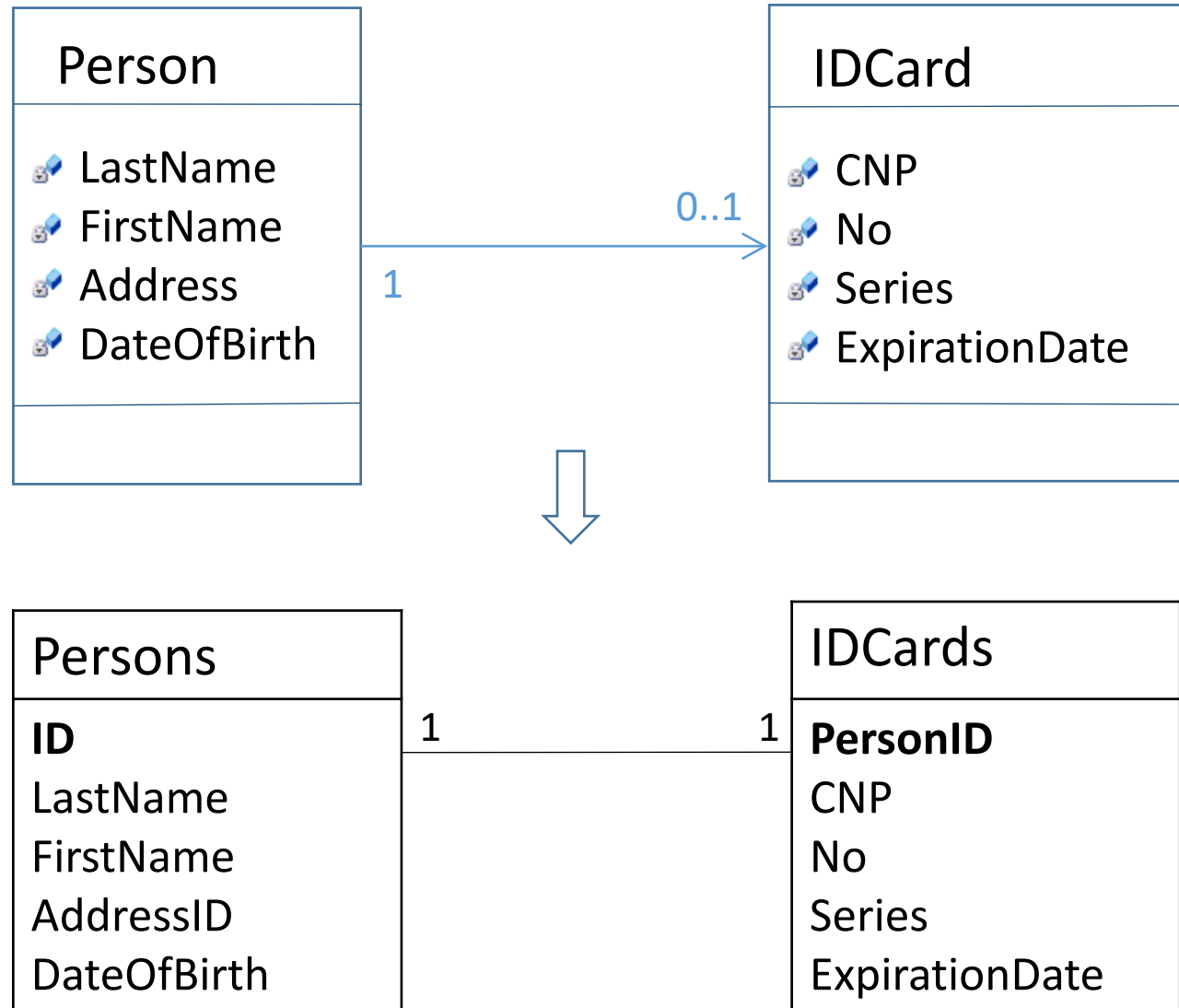  - inheritance is not properly handled

- class -> table



- the plural of the class name becomes the name of the table
- simple class attributes become table fields
- composite attributes become tables
- derived attributes are not mapped to table fields
- surrogate keys are added

Sabina S. CS

- class -> table
  - surrogate key
    - key that isn't obtained from the domain of the modeled problem
  - when possible, use integer keys that are automatically generated by the DBMS
    - easy to maintain - the responsibility of the system
    - efficient approach (fast queries)
    - simplified definition of foreign keys
  - possible approach
    - surrogate key name: *ID*
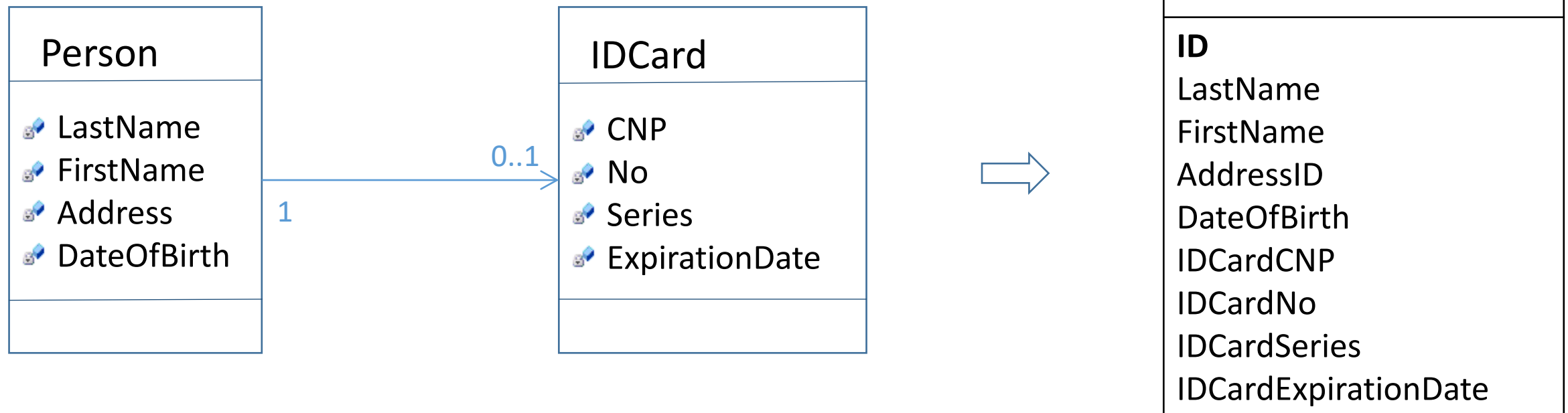    - foreign key name: *<SingularTableName>ID*

- mapping simple associations
- 1 : 0..1
  - create 1 table per class
  - the key of the *1* table (i.e., table at the *1* end of the association) becomes a foreign key in the 2nd table
  - usually, only one key is automatically generated (the one corresponding to the *1* table)
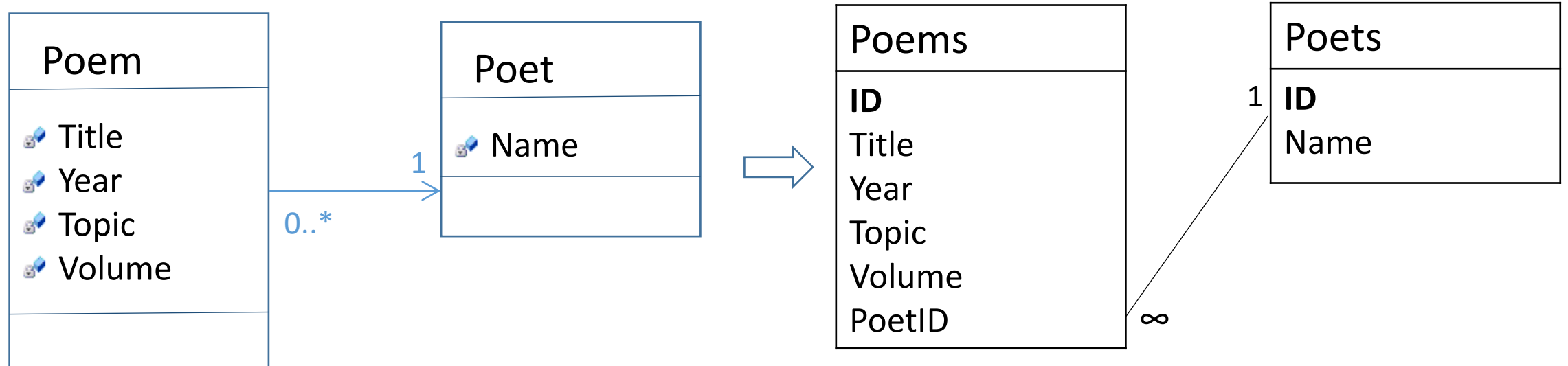
->

- mapping simple associations
- 1 : 0..1

**Person**

- LastName
- FirstName
- Address
- DateOfBirth

1

0..1

**IDCard**

- CNP
- No
- Series
- ExpirationDate

**Persons**

**ID**
LastName
FirstName
AddressID
DateOfBirth

1

1

**IDCards**

**PersonID**
CNP
No
Series
ExpirationDate

- mapping simple associations
- 1 : 1
  - create 1 table containing the attributes of both classes
  - this approach can also be used for 1 : 0..1 associations (when only a few objects in the 1<sup>st</sup> class are not associated with objects in the 2<sup>nd</sup> class)
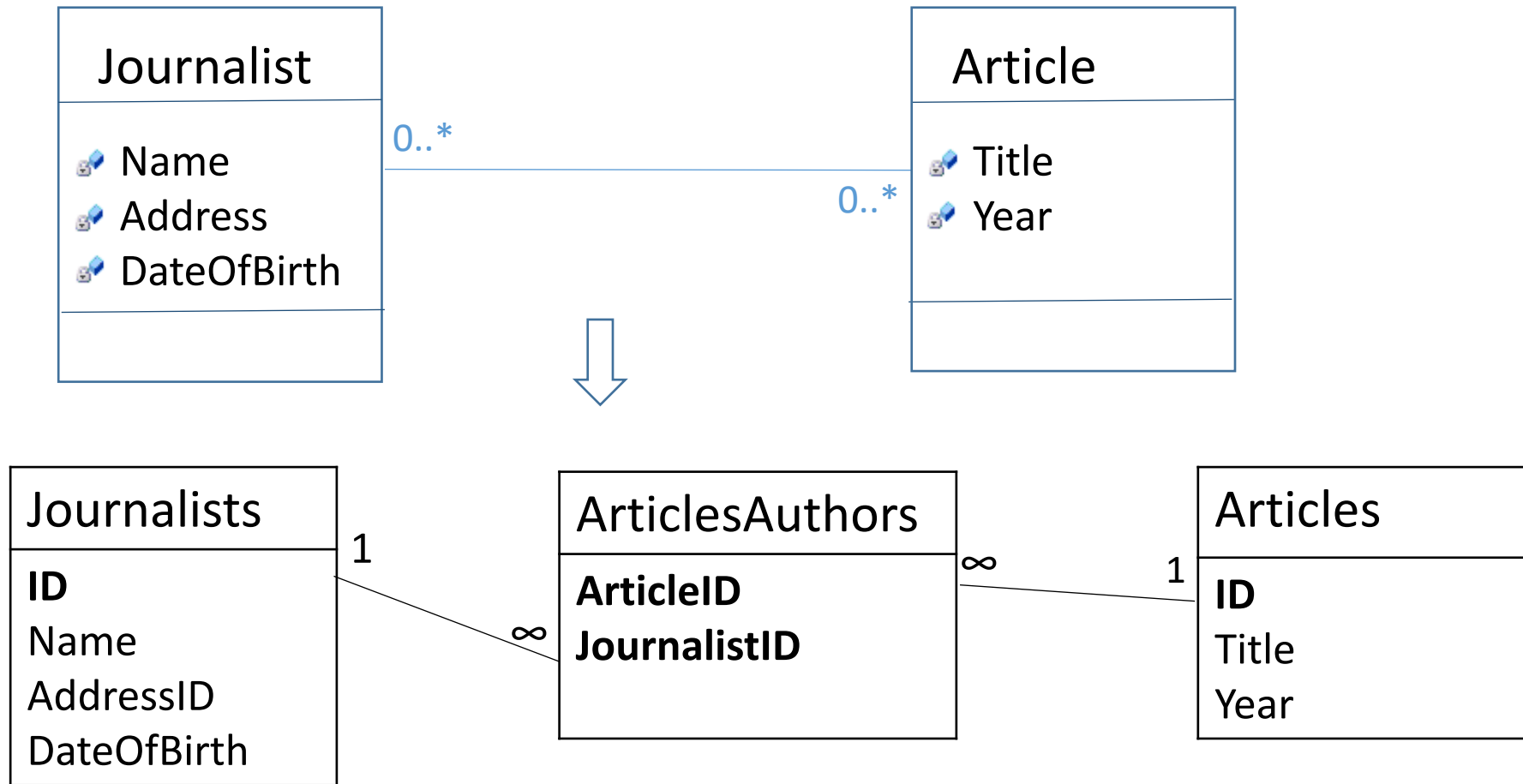
**Person**

- LastName
- FirstName
- Address
- DateOfBirth

1 — 0..1 →

**IDCard**

- CNP
- No
- Series
- ExpirationDate

⇨

**Persons**

**ID**
LastName
FirstName
AddressID
DateOfBirth
IDCardCNP
IDCardNo
IDCardSeries
IDCardExpirationDate

- mapping simple associations
- one-to-many
  - create 1 table / class
  - the key of the *1* table becomes a foreign key in the 2nd table



**Poem**

- Title
- Year
- Topic
- Volume

**Poet**

- Name

**Poems**

**ID**
Title
Year
Topic
Volume
PoetID

**Poets**
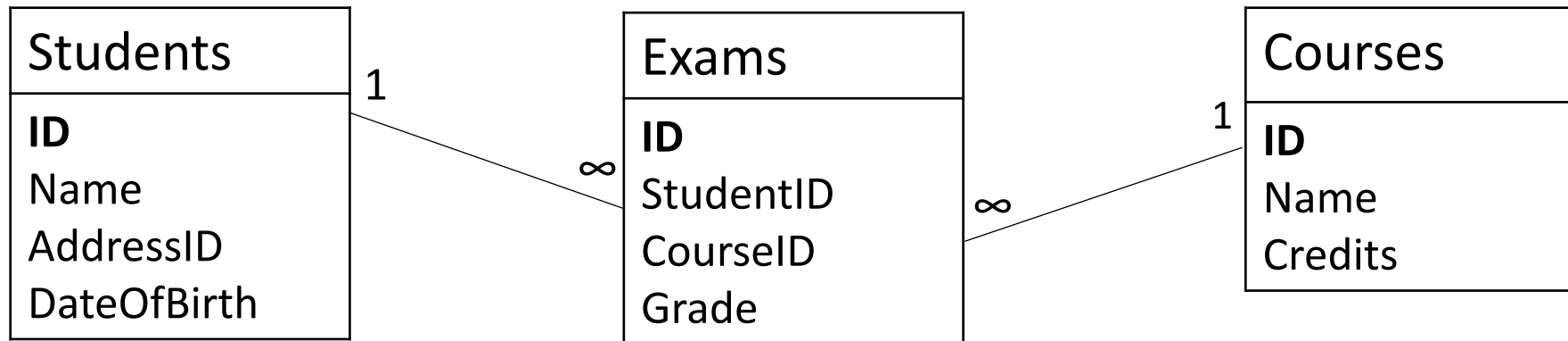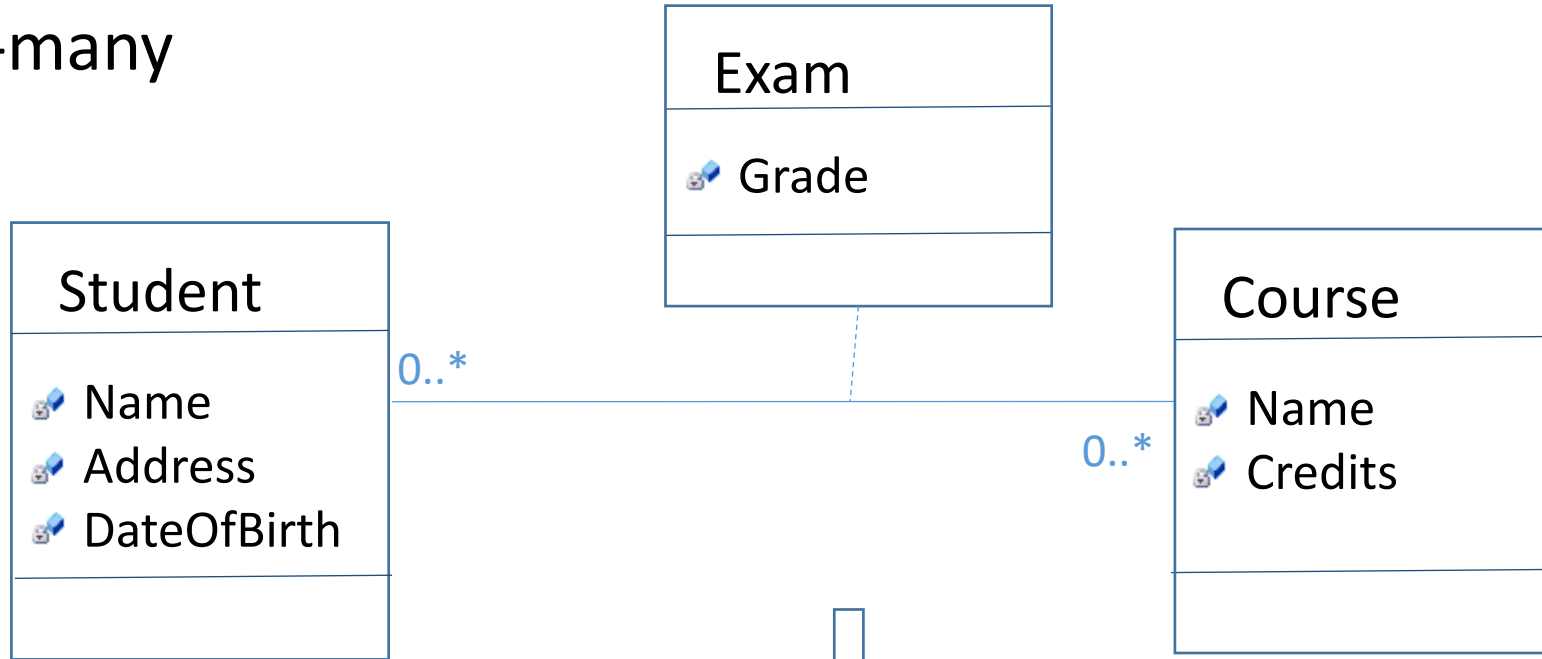
**ID**
Name

1

∞

Sabina S. CS

- mapping simple associations
- many-to-many
  - create one table / class
  - create an additional table, i.e., the *join table*
  - the primary keys of the 2 initial tables become foreign keys in the join table
  - the primary key of the join table:
    - composite, containing the 2 foreign keys
    - surrogate key
  - the name of the join table is usually a combination of the names of the 2 initial tables (not mandatory)
  - if an association class exists, its attributes become fields in the join table
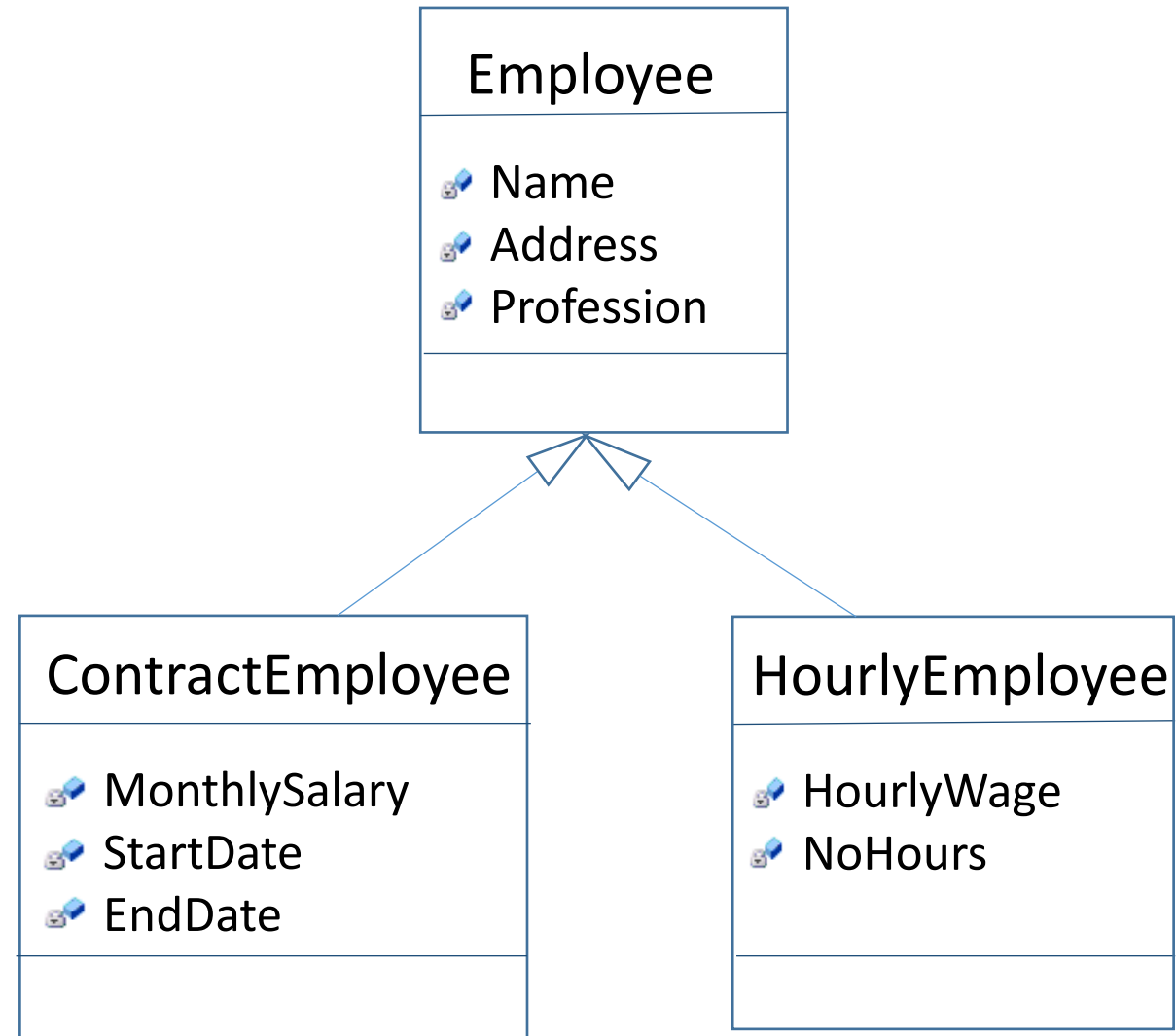
->

- mapping simple associations
- many-to-many



Journalist

- Name
- Address
- DateOfBirth

0..*                      0..*

Article

- Title
- Year

Journalists

**ID**
Name
AddressID
DateOfBirth

1          ∞

ArticlesAuthors

**ArticleID**
**JournalistID**

∞          1

Articles

**ID**
Title
Year

Sabina S. CS

- mapping simple associations
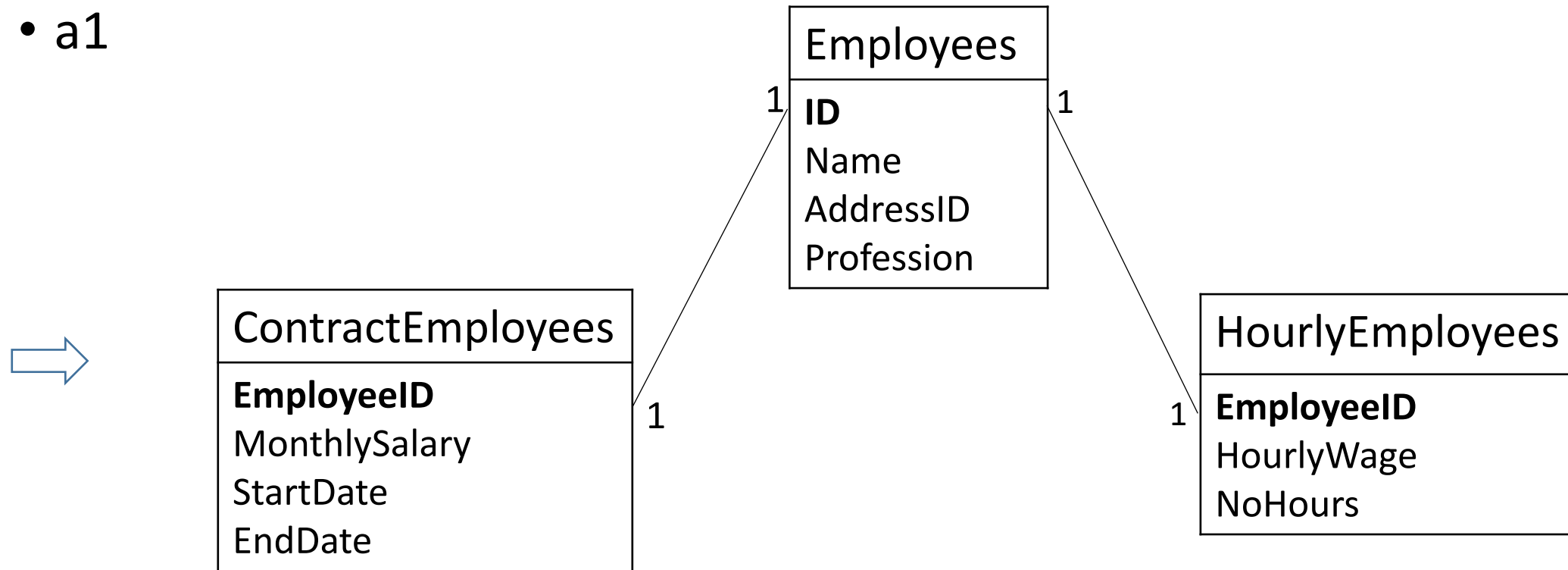- many-to-many



Sabina S. CS

- mapping inheritance
- a1
  - create one table / class
  - create one view / superclass-subclass pair
  - it generates the largest number of objects (tables, views)
  - flexibility - no impact on existing tables / views when adding other subclasses
  - possible performance problems – every access requires a join through the view
  - can be used when the number of records is relatively small (so performance is not a concern)

->

- mapping inheritance
- a1



Employee

- Name
- Address
- Profession

ContractEmployee

- MonthlySalary
- StartDate
- EndDate

HourlyEmployee

- HourlyWage
- NoHours

Sabina S. CS

- mapping inheritance
- a1

**Employees**

1 | 1
**ID**
Name
AddressID
Profession

**ContractEmployees**

**EmployeeID**
MonthlySalary
StartDate
EndDate

1

**HourlyEmployees**

1 | **EmployeeID**
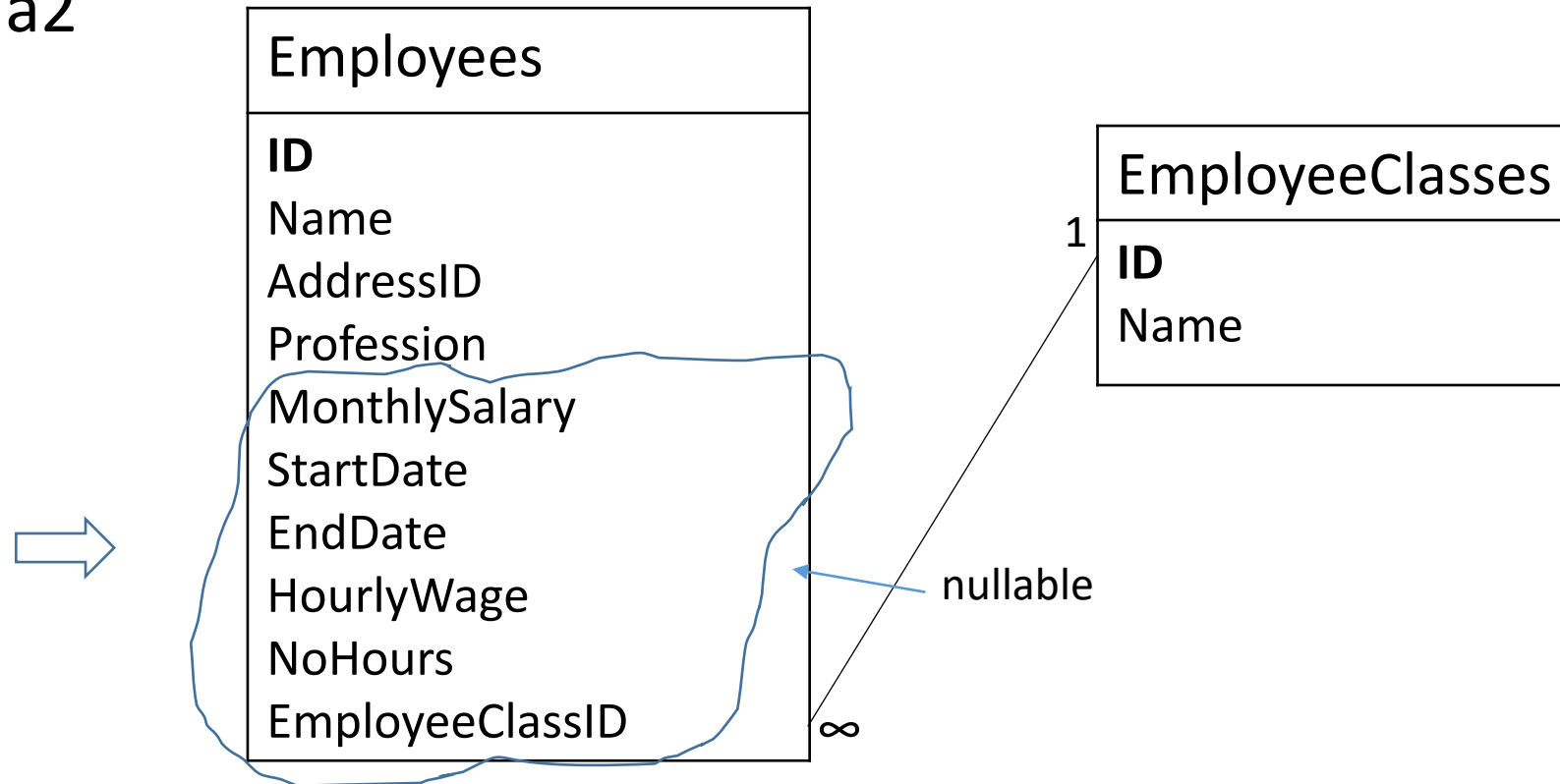HourlyWage
NoHours

```
CREATE VIEW ContractEmployeesComplete(...)
AS
  SELECT Employees.*, MonthlySalary, StartDate, EndDate
  FROM Employees INNER JOIN ContractEmployees
    ON Employees.ID = EmployeeID
```

- mapping inheritance
- a2
  - create one table for the superclass
  - the attributes of the subclasses become fields in the table
  - it generates the smallest number of objects
  - optionally, a subclasses table and a view / subclass can be added
  - usually – best performance
  - when adding a subclass, the existing structure has to be changed
  - "artificial" increase of used space

                                                                    ->

- mapping inheritance
- a2

**Employees**

**ID**
Name
AddressID
Profession
MonthlySalary
StartDate
EndDate
HourlyWage
NoHours
EmployeeClassID

**EmployeeClasses**

**ID**
Name

1

∞

nullable

*EmployeeClasses*

| ID | Name |
|----|------|
| 1 | Unknown |
| 2 | ContractEmployee |
| 3 | HourlyEmployee |

```
CREATE VIEW ContractEmployees(...)
AS
  SELECT ID, Name, AddressID, Profession, MonthlySalary, StartDate,
    EndDate
  FROM Employees
  WHERE EmployeeClassID = 2
```

Sabina S. CS

- mapping inheritance
- a3
  - create one table / subclass
  - the attributes of the superclass become fields in each of the created tables
  - satisfactory performance
  - subclasses can be subsequently added without affecting existing tables
  - changing the structure of the superclass impacts all existing tables
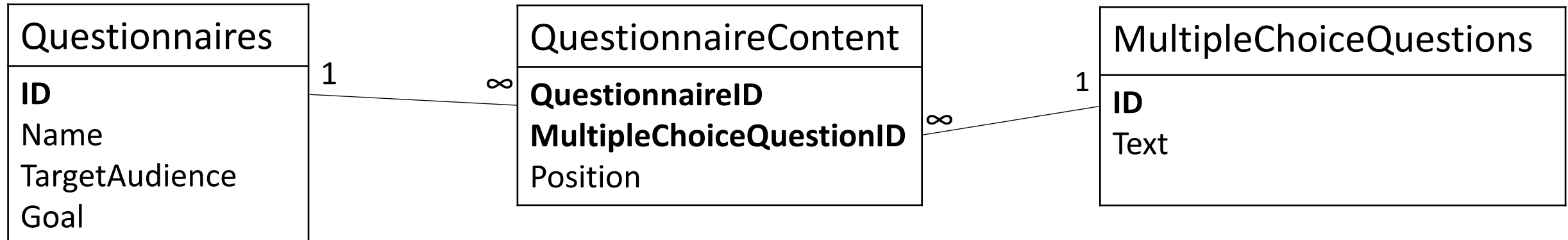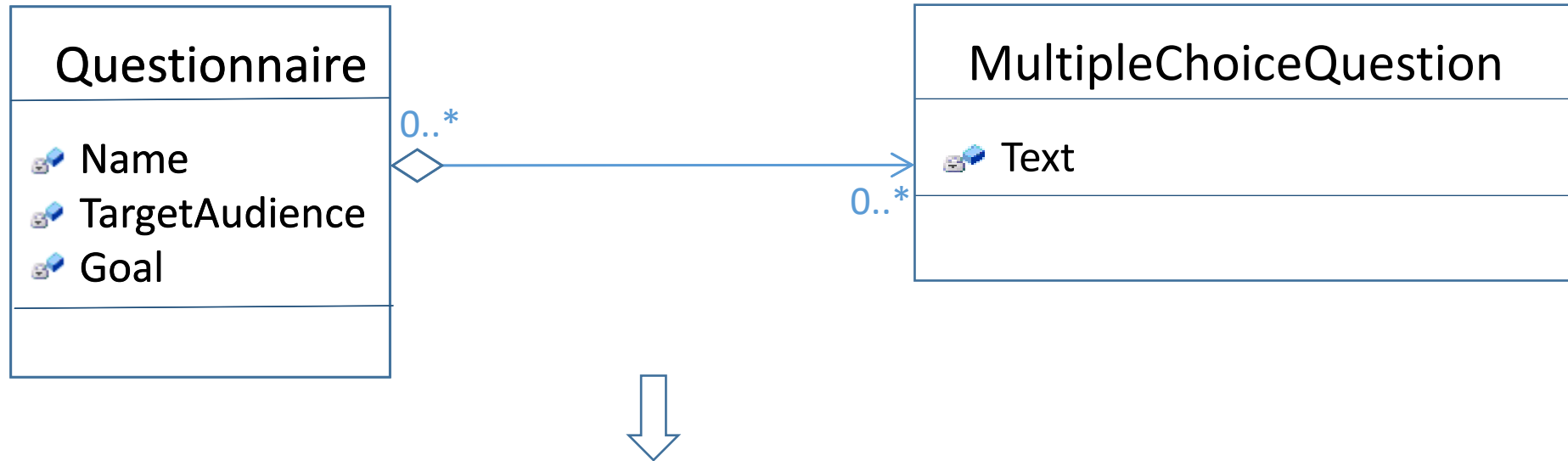
->

- mapping inheritance
- a3

| ContractEmployees |
| --- |
| **ID** |
| Name |
| AddressID |
| Profession |
| MonthlySalary |
| StartDate |
| EndDate |

| HourlyEmployees |
| --- |
| **ID** |
| Name |
| AddressID |
| Profession |
| HourlyWage |
| NoHours |

- mapping aggregation / composition
  - similar to mapping simple associations
  - fixed number of *parts* in a *whole* => can declare the same number of foreign keys in the *whole* table
  - composition - ON DELETE CASCADE option (not required for aggregation)

  ->

- mapping aggregation / composition

| Questionnaire | | MultipleChoiceQuestion |
|---|---|---|

0..*

Name
TargetAudience
Goal

Text

0..*

| Questionnaires | QuestionnaireContent | MultipleChoiceQuestions |
|---|---|---|
| **ID** | **QuestionnaireID** | **ID** |
| Name | **MultipleChoiceQuestionID** | Text |
| TargetAudience | Position | |
| Goal | | |

1 ∞          ∞ 1

- obs. a questionnaire can also have open answer questions, etc.
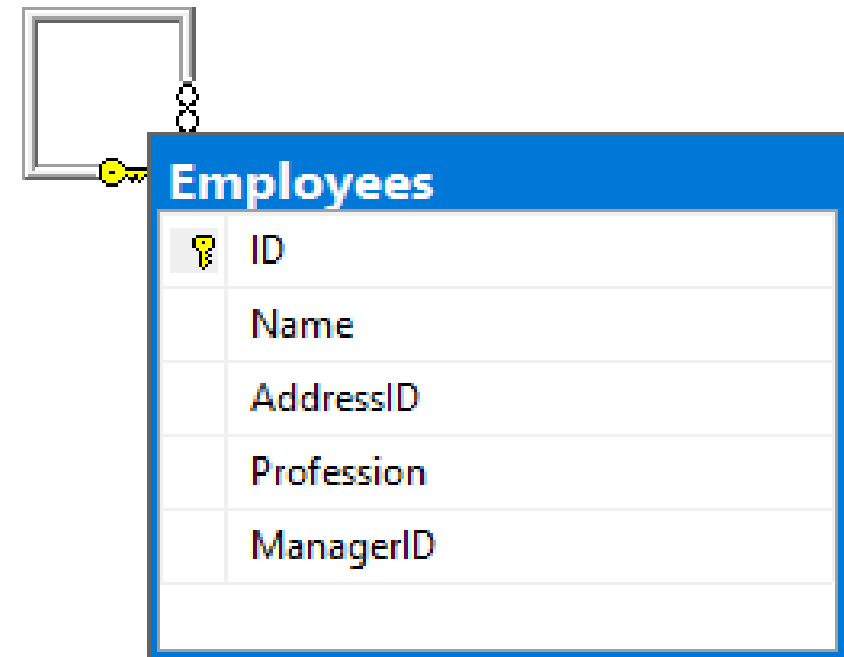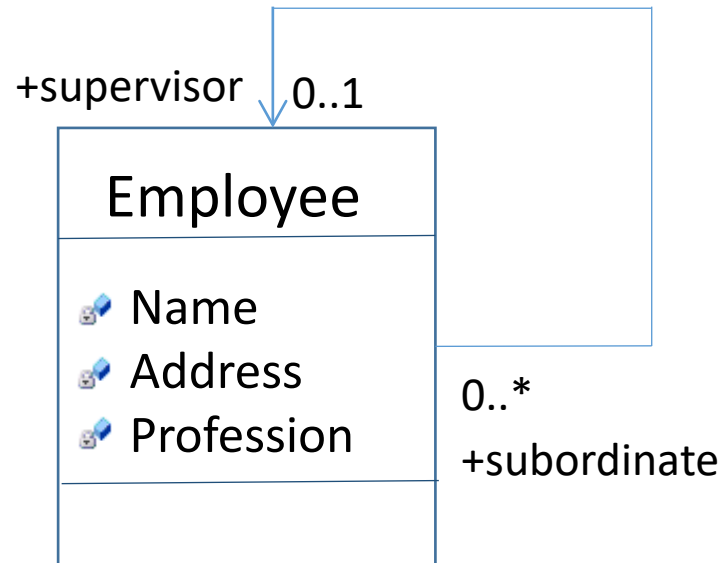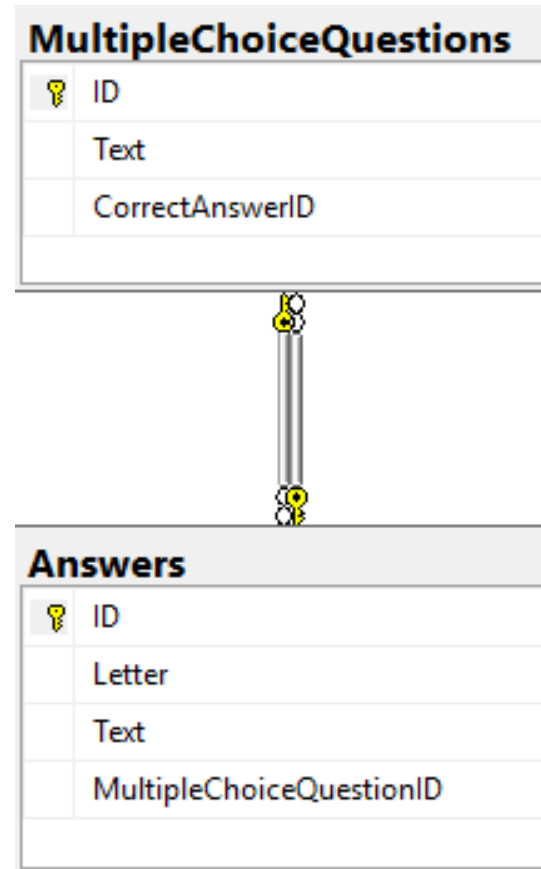
Sabina S. CS

- mapping aggregation / composition
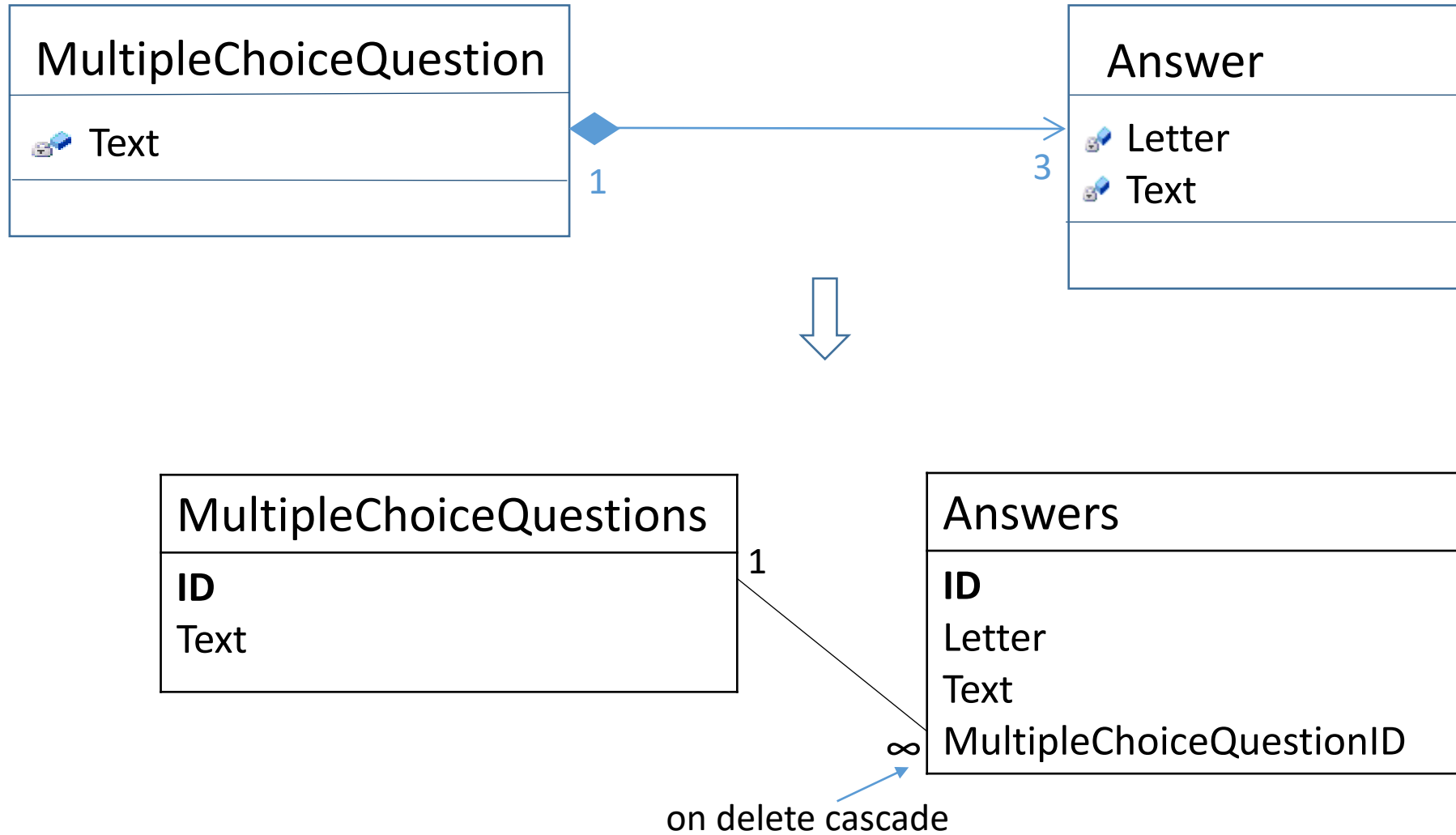
- mapping aggregation / composition
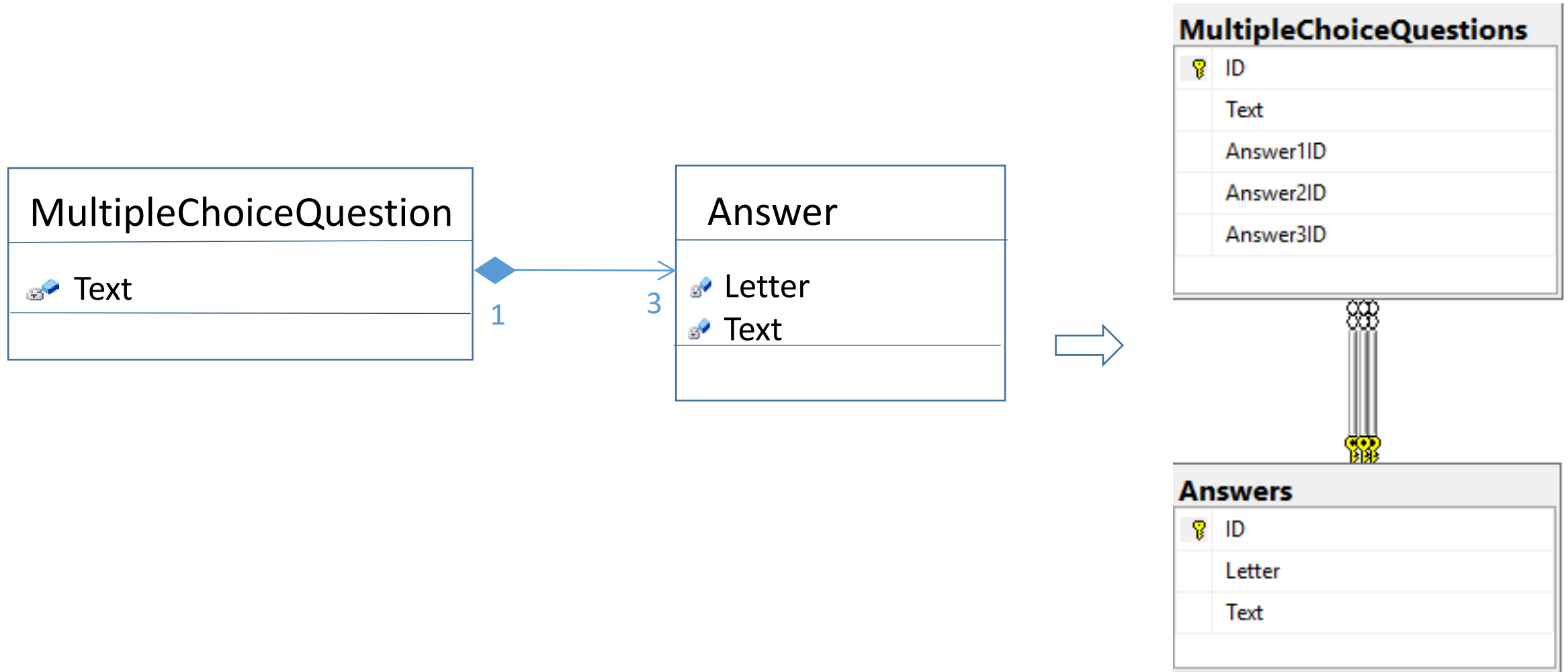
- mapping reflexive associations
- add a new field, referencing the same table (recursive relationship)
- ON DELETE CASCADE - error

# Obs. 2 different tables, each with a foreign key referencing the other one, ON DELETE CASCADE - error

References

[Ta13] ȚÂMBULEA, L., Curs Baze de date, Facultatea de Matematică și Informatică, UBB, 2013-2014

[Da03] DATE, C.J., An Introduction to Database Systems (8th Edition), Addison-Wesley, 2003

[Si11] SILBERSCHATZ, A., KORTH, H., SUDARSHAN, S., Database System Concepts (6th Edition), McGraw-Hill, 2011

[Ga09] GARCIA-MOLINA, H., ULLMAN, J., WIDOM, J., Database Systems: The Complete Book (2nd Edition), Pearson Education, 2009