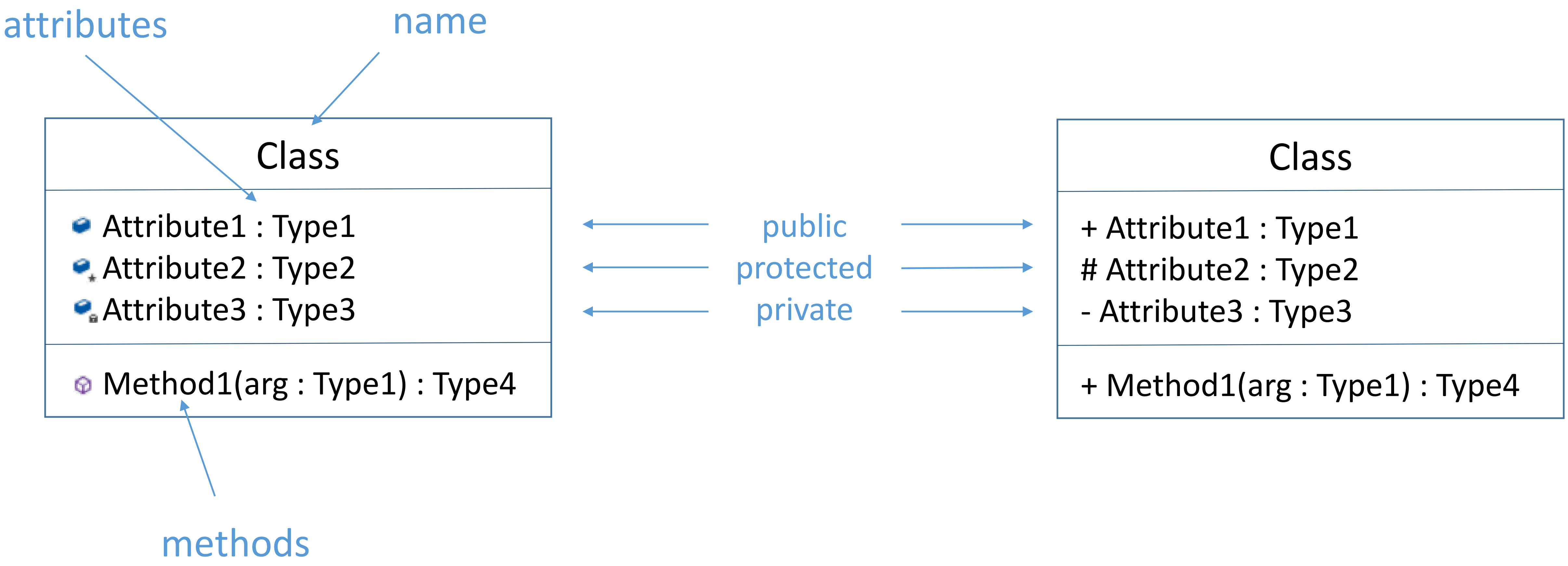
Databases

Lecture 13

Conceptual Modeling

- database design - stages
  - conceptual design - identify entities and relationships
  - translate the conceptual model into a model supported by the DBMS (e.g., relational)
  - schema refinement (normalization)
    - eliminate redundancy and associated problems
  - physical database design
    - create indexes
    - redesign parts of the schema

- UML class diagram
  - classes



attributes

name

Class

Attribute1 : Type1
Attribute2 : Type2
Attribute3 : Type3

Method1(arg : Type1) : Type4

methods

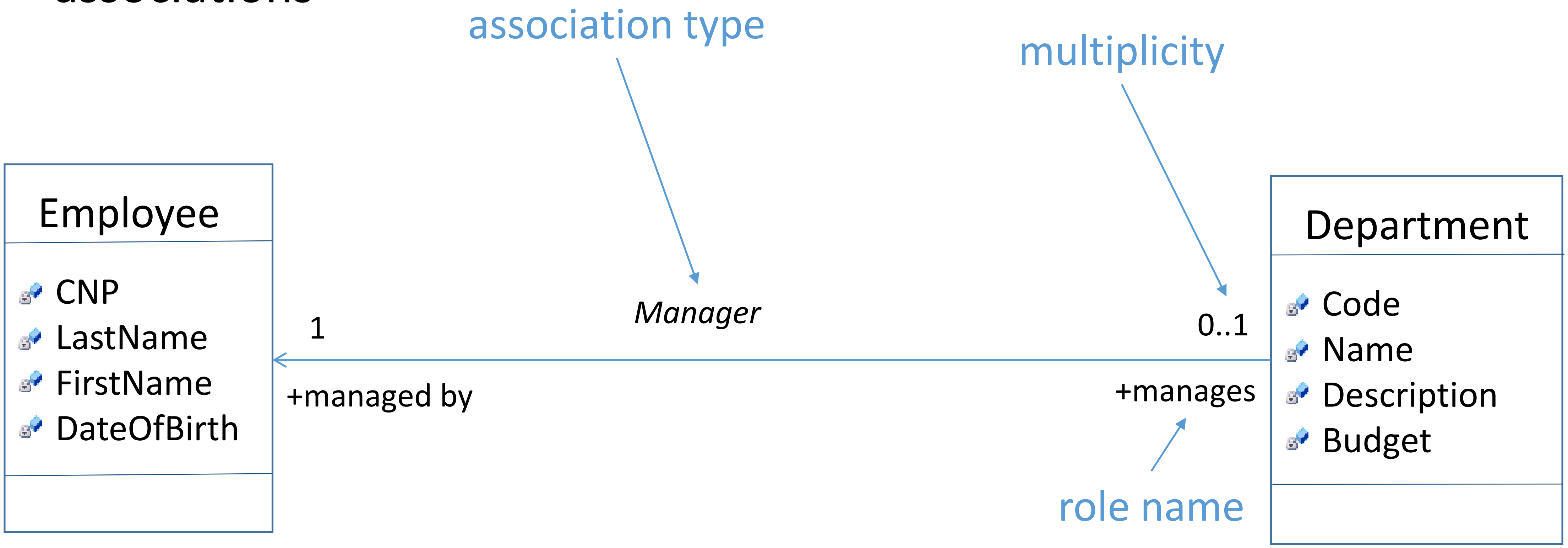
public
protected
private

Class

+ Attribute1 : Type1
# Attribute2 : Type2
- Attribute3 : Type3

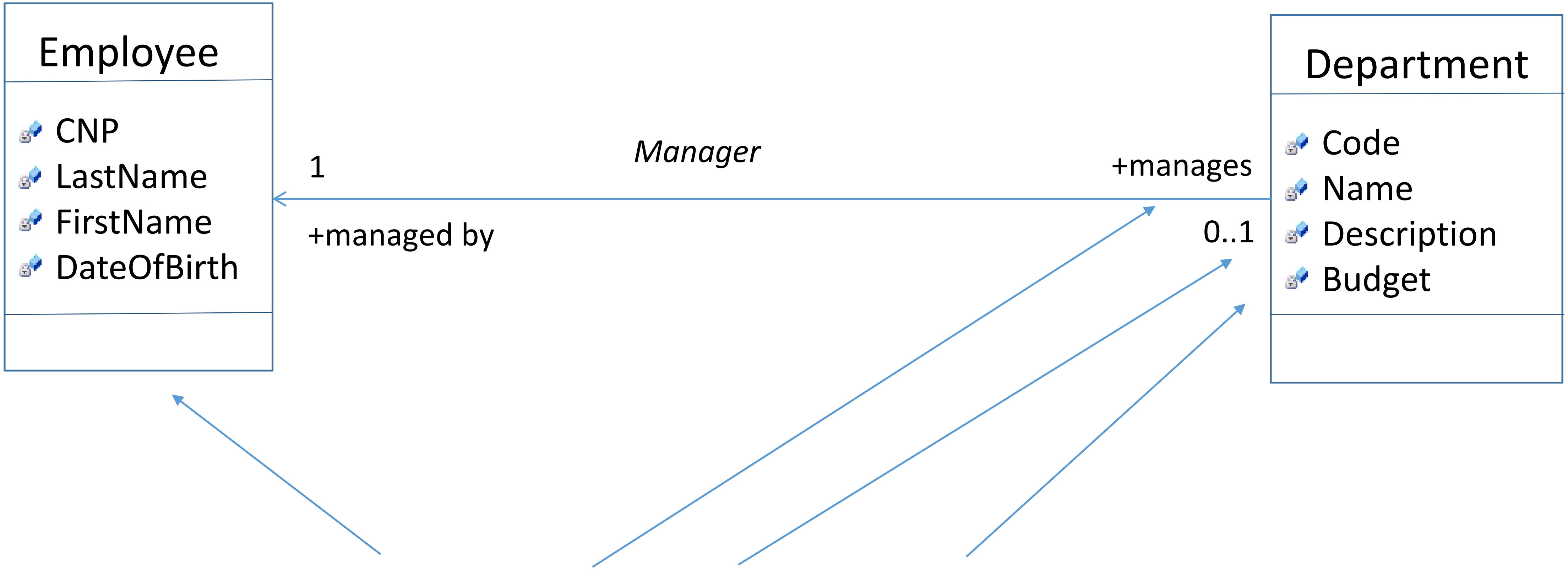
+ Method1(arg : Type1) : Type4

- UML class diagram
  - associations



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

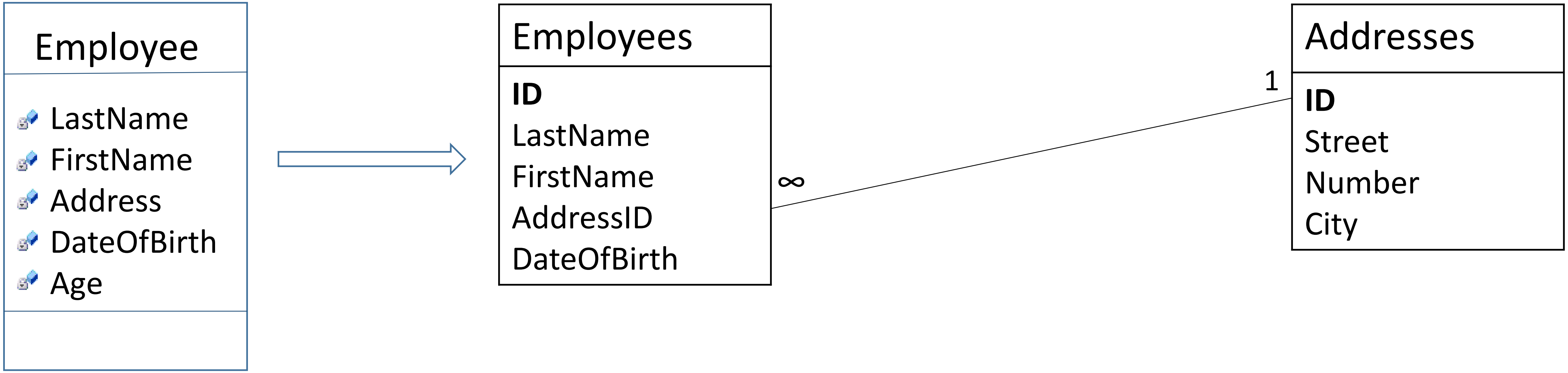
- UML class diagram
  - associations



An employee manages 0 or 1 departments.

- 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 (link table)
    - inheritance is improperly handled

- class -> table



Employee

LastName
FirstName
Address
DateOfBirth
Age

Employees

ID
LastName
FirstName
AddressID
DateOfBirth

Addresses

ID
Street
Number
City

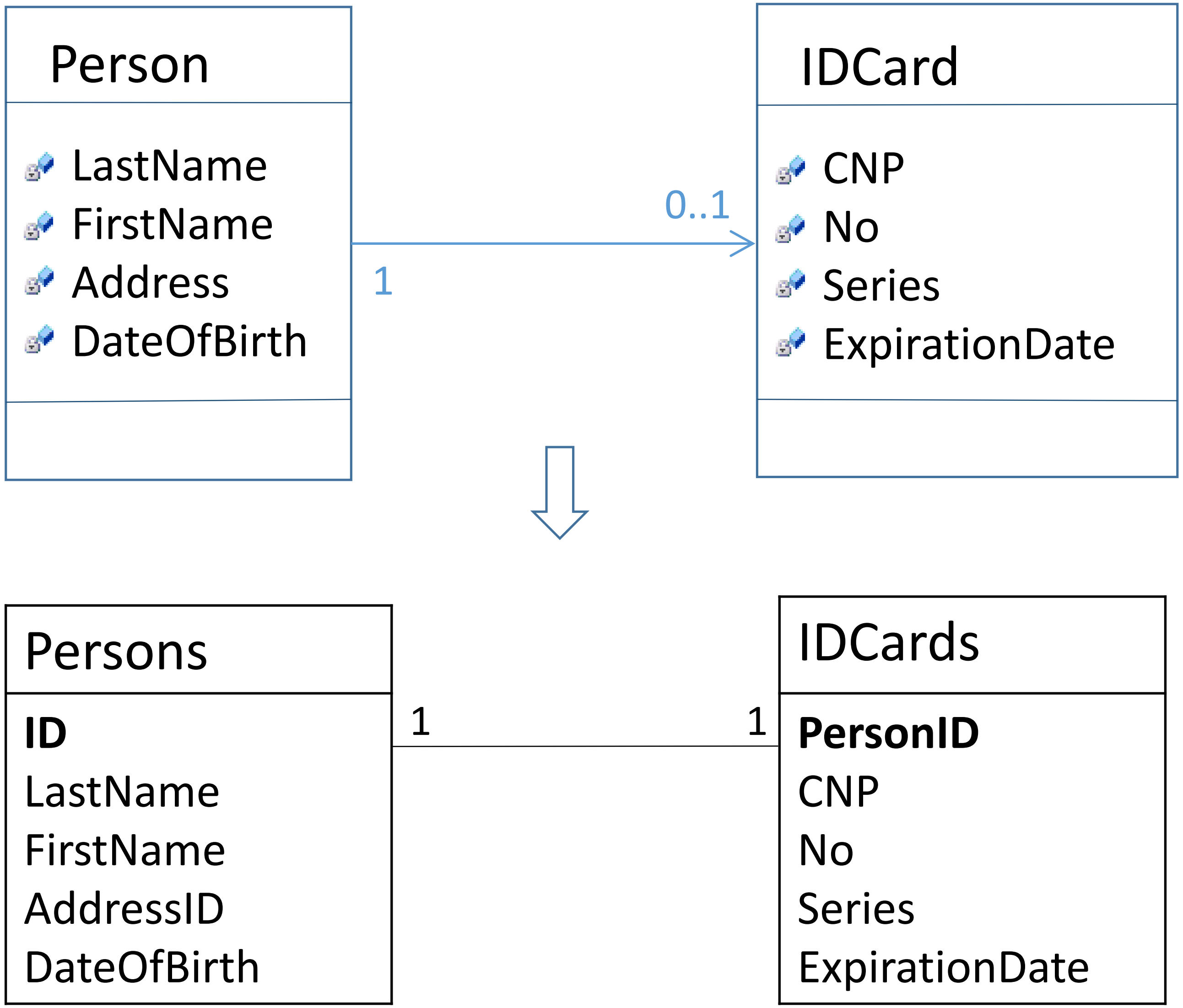
- the plural of the class name becomes the name of the table
- simple class attributes become table fields
- composite attributes become tables
- derived attributes are not mapped to table fields
- surrogate keys are added

- 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
- multiplicity 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
  - only one key is automatically generated – usually, the one corresponding to the *1* table

->

- mapping simple associations
- multiplicity 1 : 0..1



Person

LastName
FirstName
Address
DateOfBirth

1

0..1

IDCard

CNP
No
Series
ExpirationDate

Persons

ID
LastName
FirstName
AddressID
DateOfBirth

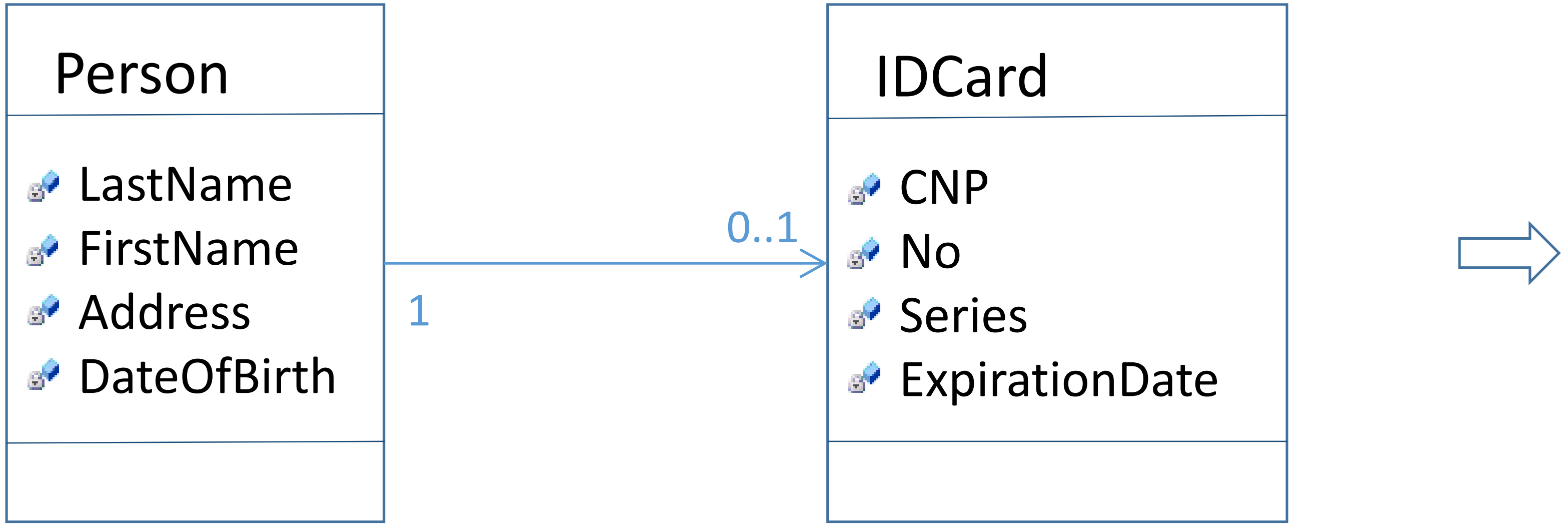
1

1

IDCards

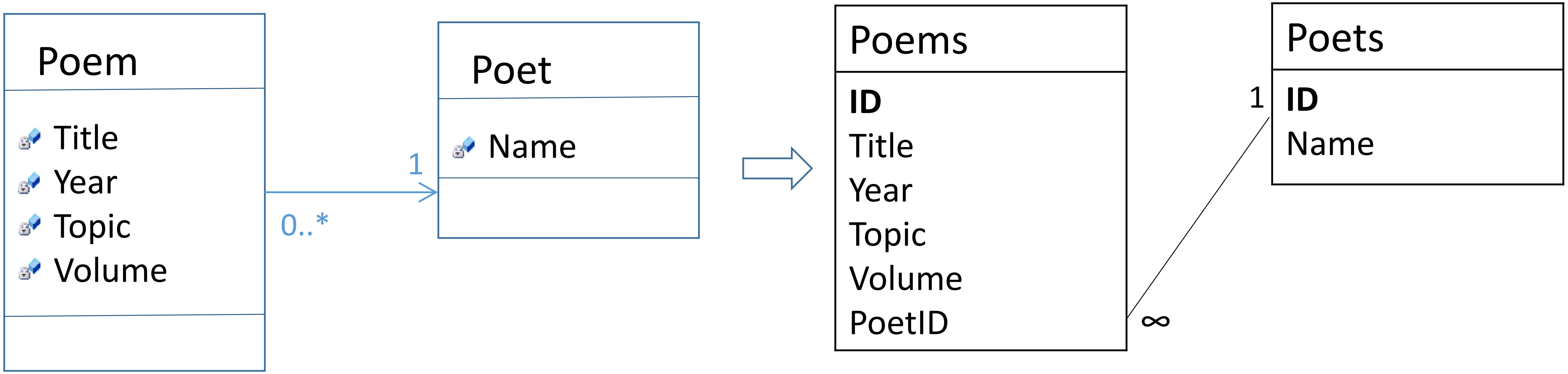
PersonID
CNP
No
Series
ExpirationDate

- mapping simple associations
- multiplicity 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 1st class are not associated with objects in the 2nd class)



Persons
ID
LastName
FirstName
AddressID
DateOfBirth
CNP
IDCardNo
IDCardSeries
IDCardExpirationDate

- mapping simple associations
- multiplicity 1 : n
  - 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

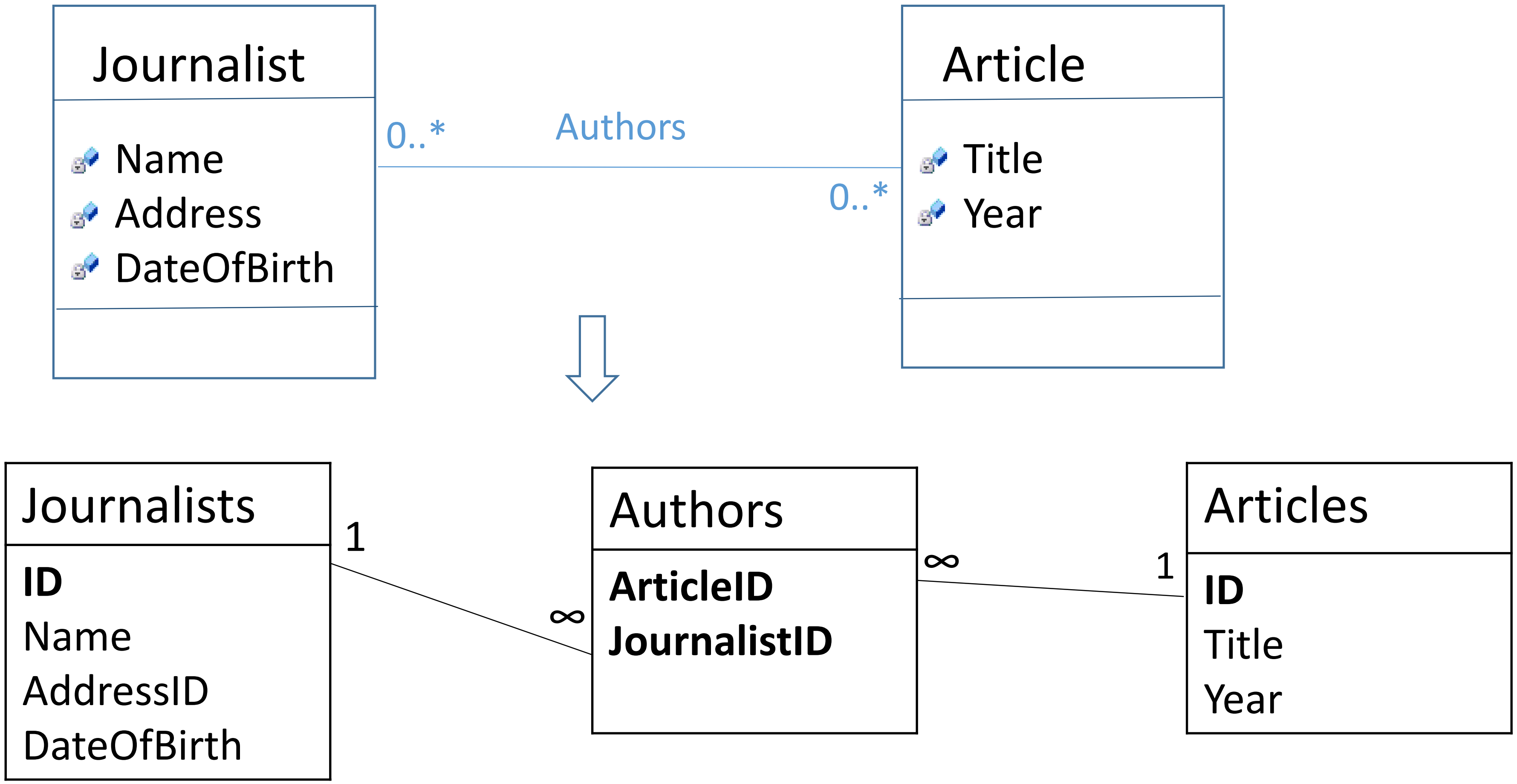
ID

Name

- mapping simple associations
- multiplicity m : n
    - create one table / class
    - create an additional table, i.e., the *link table*
    - the primary keys of the 2 initial tables become foreign keys in the link table
    - the primary key of the link table:
        - composite, containing the 2 foreign keys
        - surrogate key
    - the name of the link 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 link table

->

- mapping simple associations
- multiplicity m : n



Journalist
Name
Address
DateOfBirth

0..*
Authors
0..*

Article
Title
Year

Journalists
ID
Name
AddressID
DateOfBirth

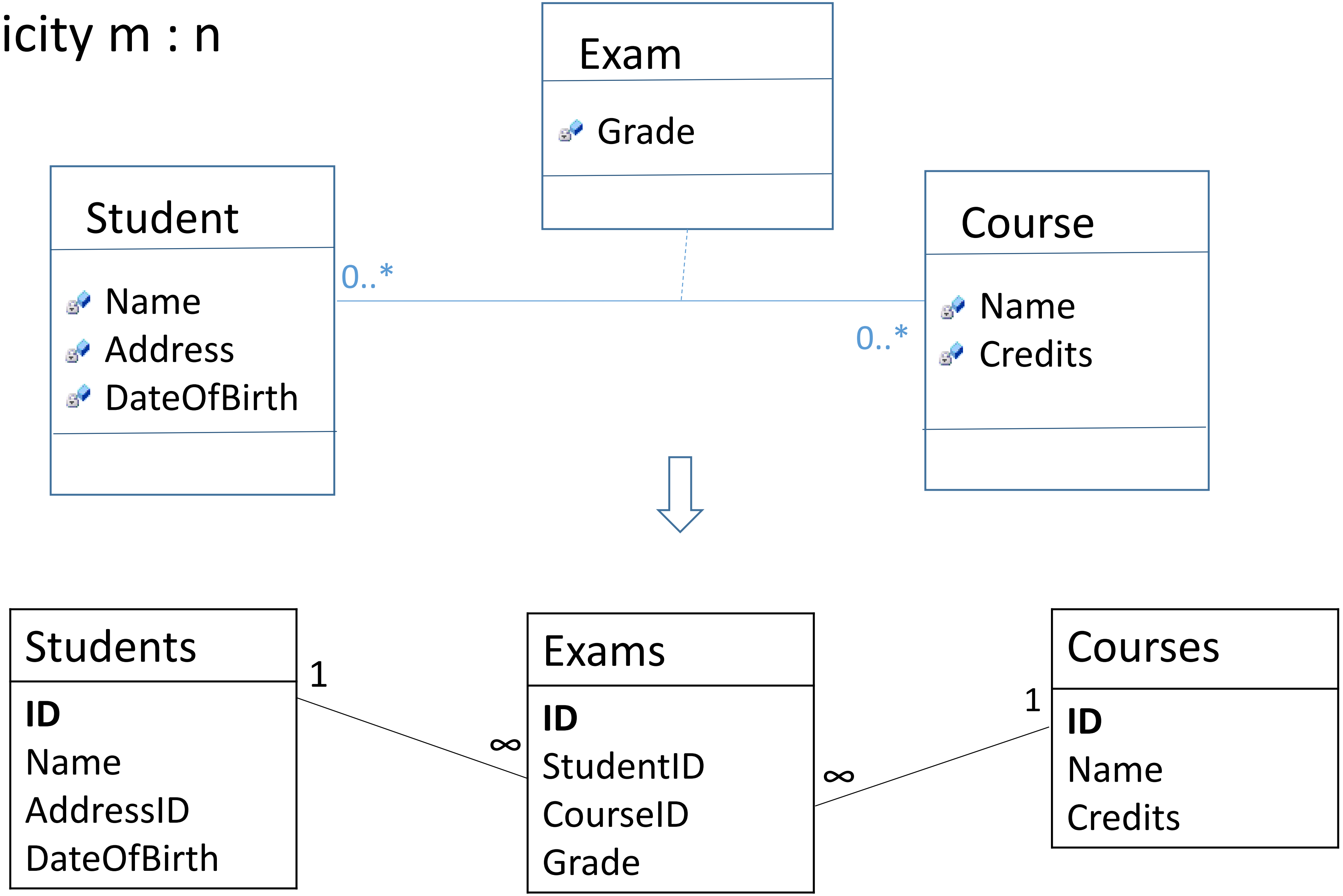
1
∞

Authors
ArticleID
JournalistID

∞
1

Articles
ID
Title
Year

- mapping simple associations
- multiplicity m : n



Exam
Grade

Student
Name
Address
DateOfBirth

0..*

0..*

Course
Name
Credits

Students
ID
Name
AddressID
DateOfBirth

1

∞

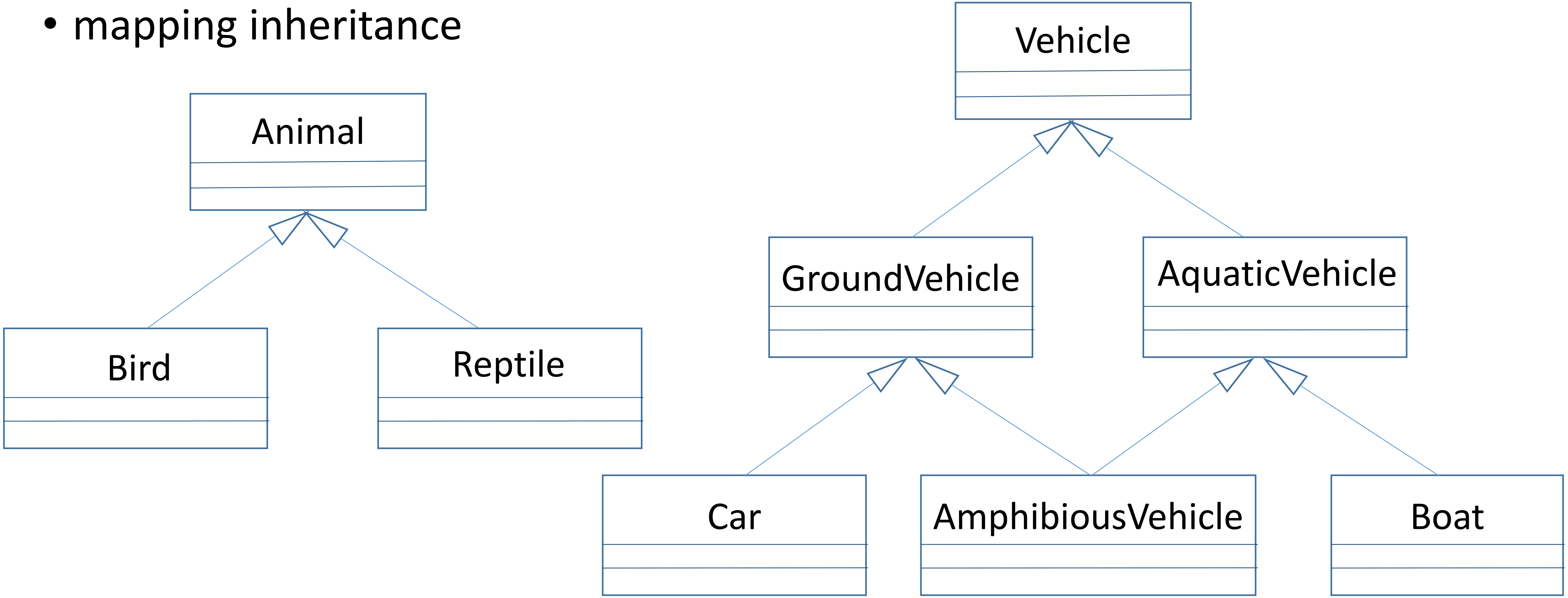
Exams
ID
StudentID
CourseID
Grade

∞

1

Courses
ID
Name
Credits

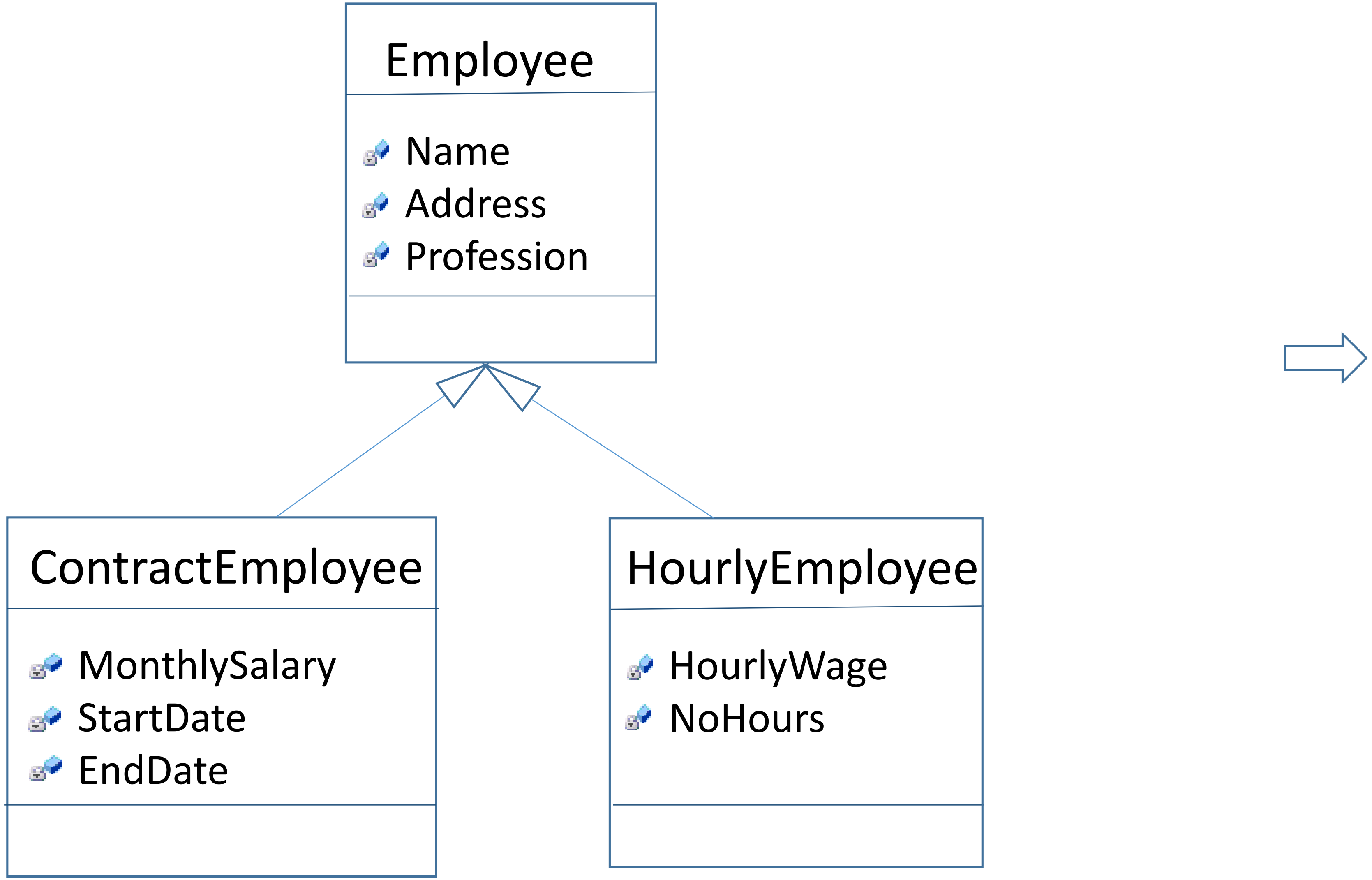
- mapping inheritance



- 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
  - used when the number of records is relatively small (so performance is not a concern)

->

- mapping inheritance
- a1



Employee

- Name
- Address
- Profession

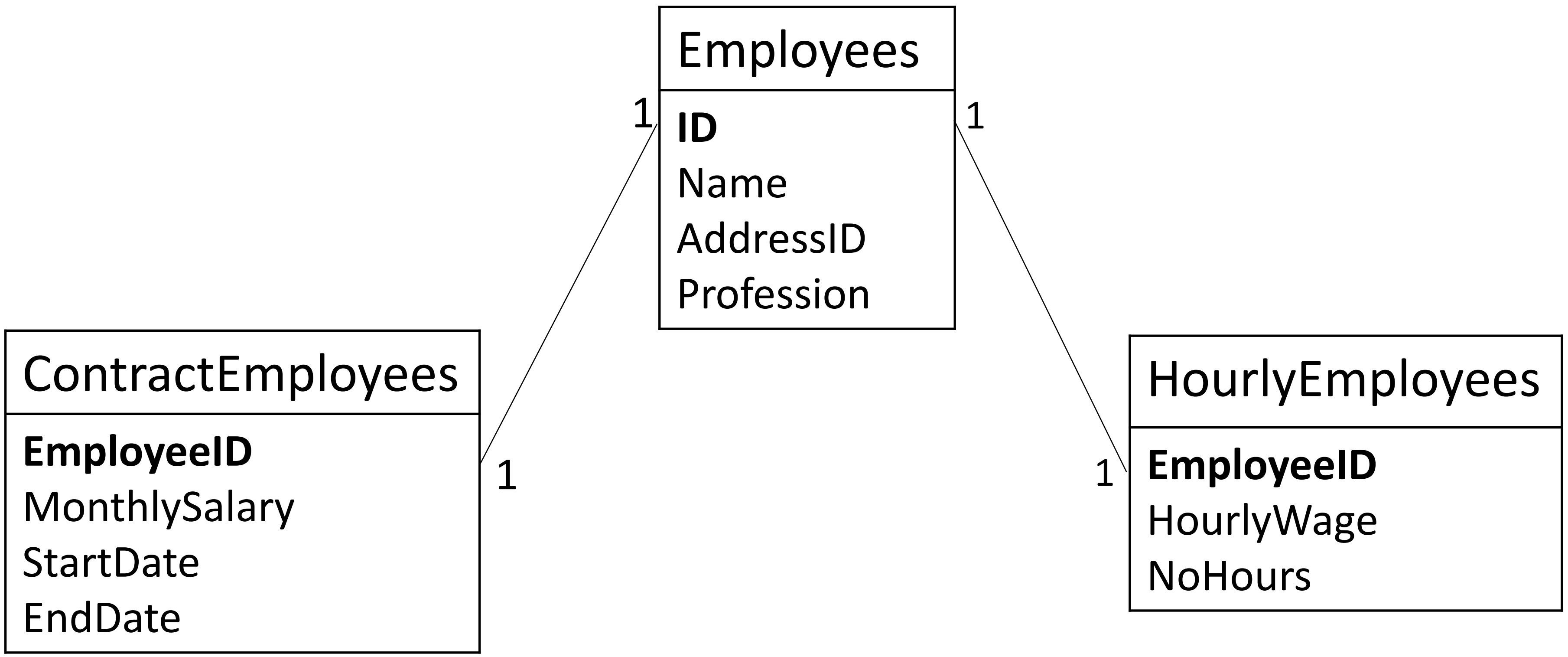
ContractEmployee

- MonthlySalary
- StartDate
- EndDate

HourlyEmployee

- HourlyWage
- NoHours

- mapping inheritance
- a1



Employees

ID
Name
AddressID
Profession

ContractEmployees

EmployeeID
MonthlySalary
StartDate
EndDate

HourlyEmployees

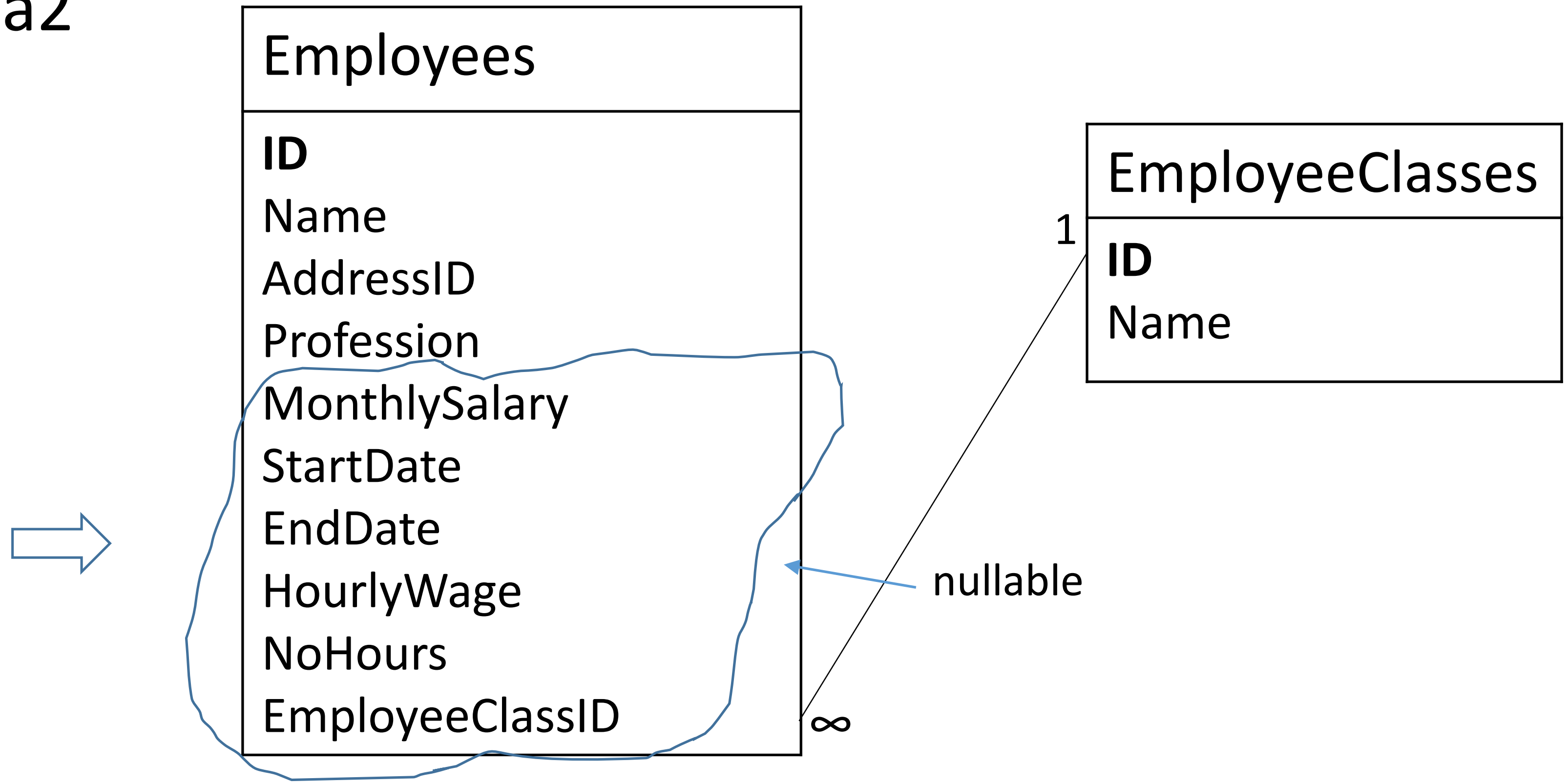
EmployeeID
HourlyWage
NoHours

```
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

nullable

1

∞

EmployeeClasses

ID
Name

EmployeeClasses

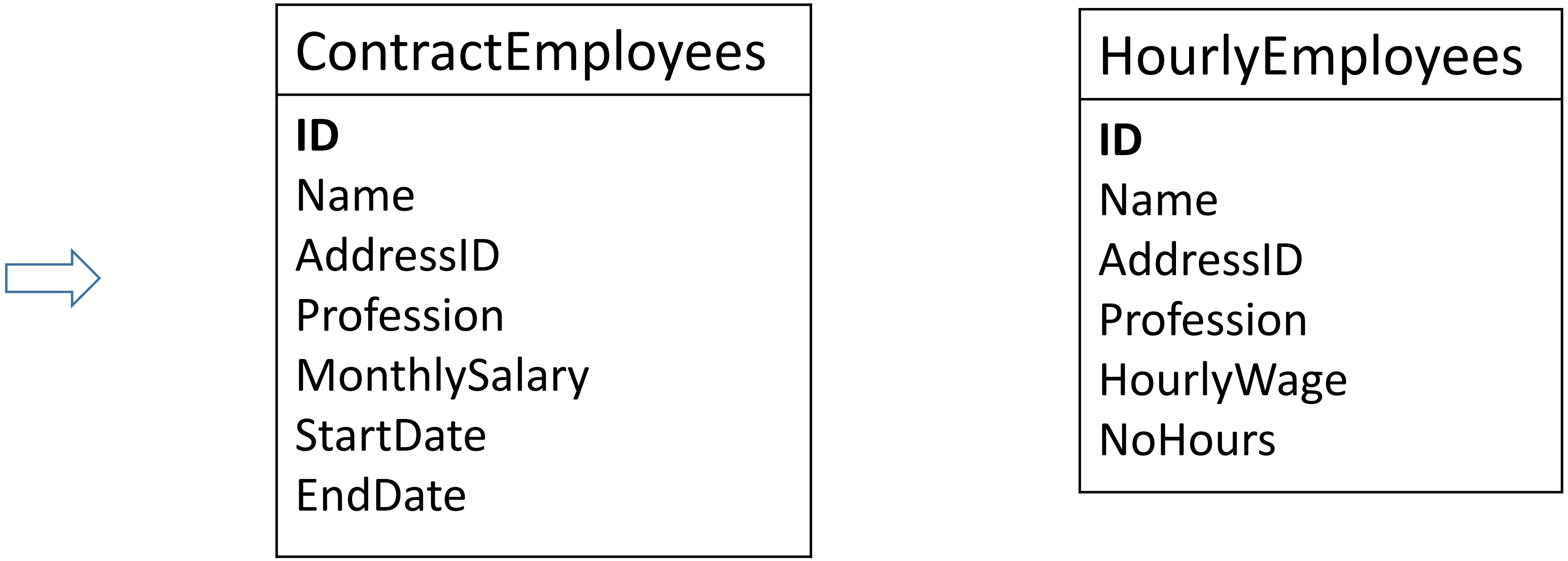
ID	Name
1	Unknown
2	ContractEmployee
3	HourlyEmployee

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

- 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

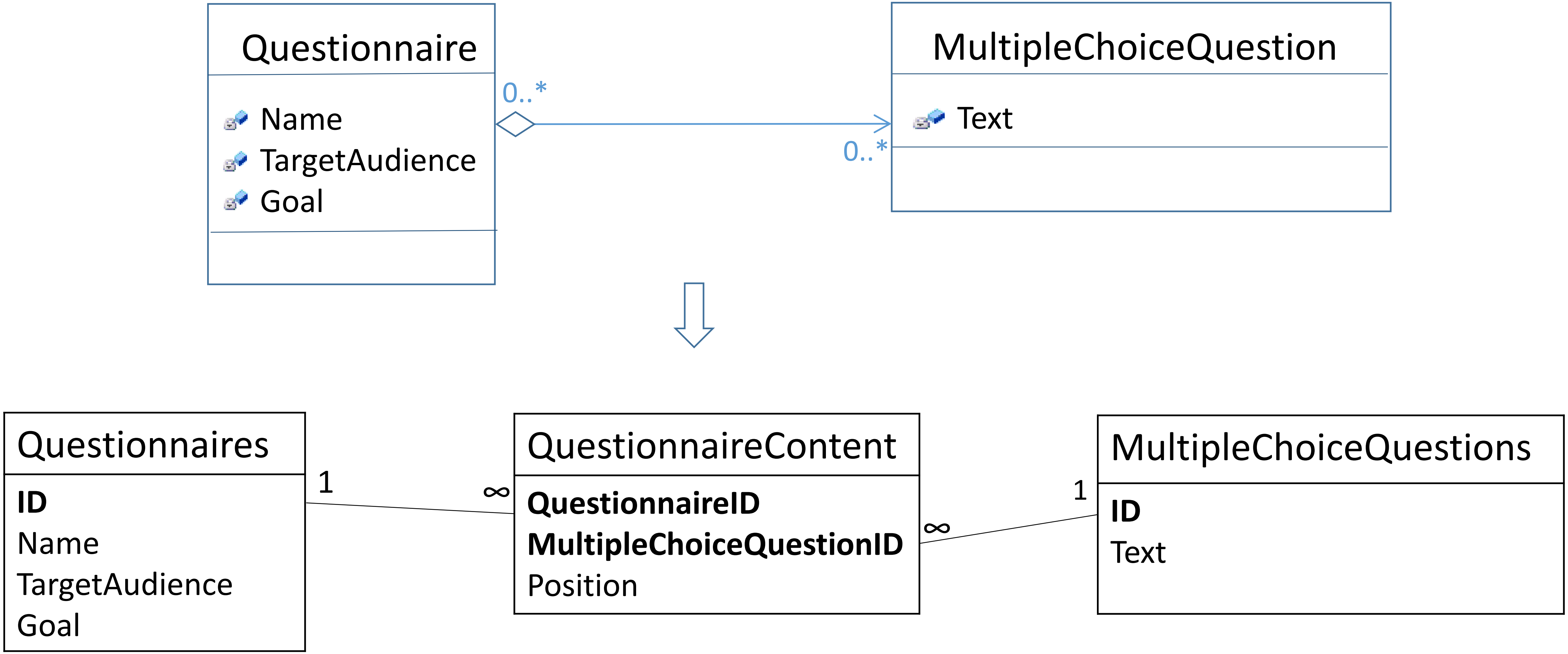
HourlyEmployees

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

- Name
- TargetAudience
- Goal

0..*

0..*

MultipleChoiceQuestion

- Text

Questionnaires

ID
Name
TargetAudience
Goal

1

∞

QuestionnaireContent

QuestionnaireID
MultipleChoiceQuestionID
Position

∞

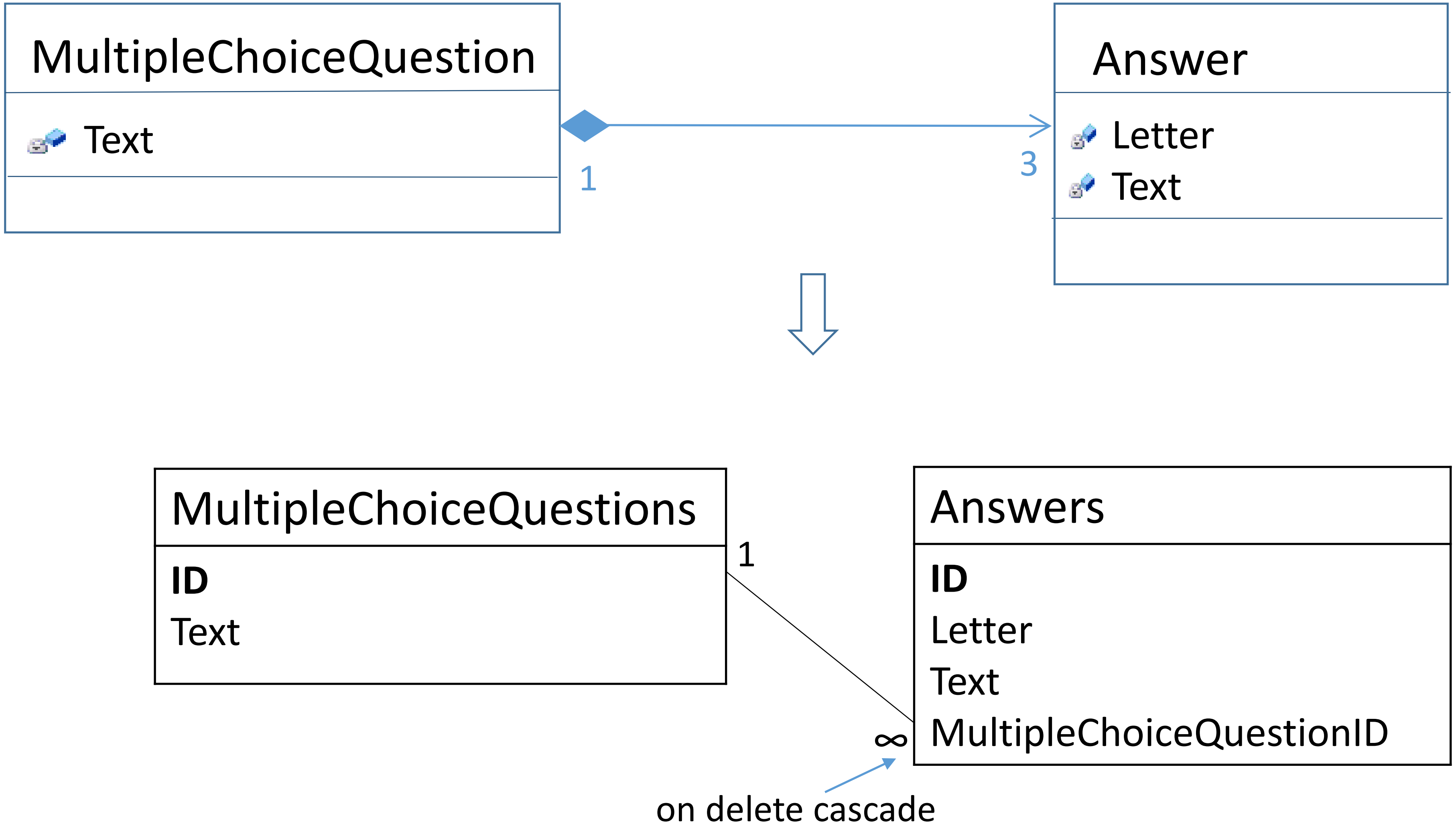
1

MultipleChoiceQuestions

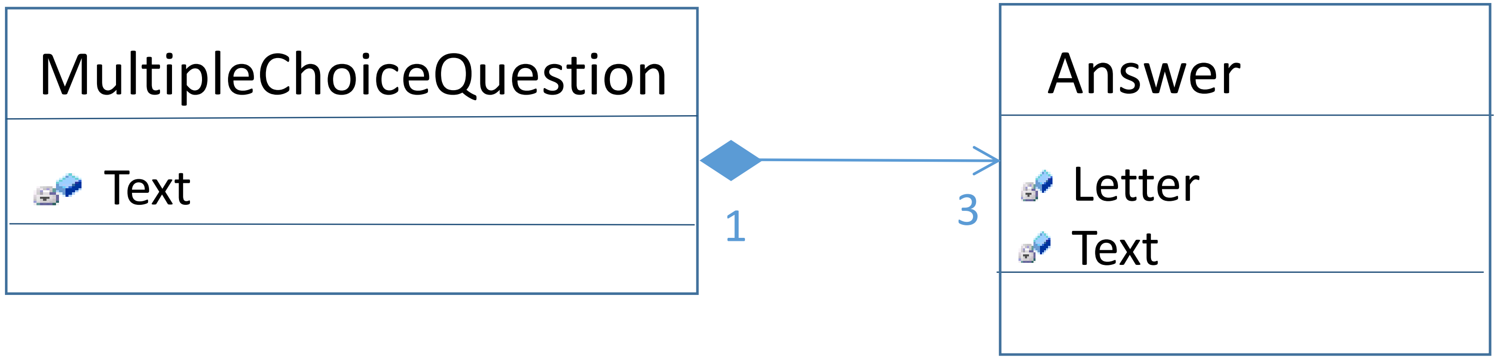
ID
Text

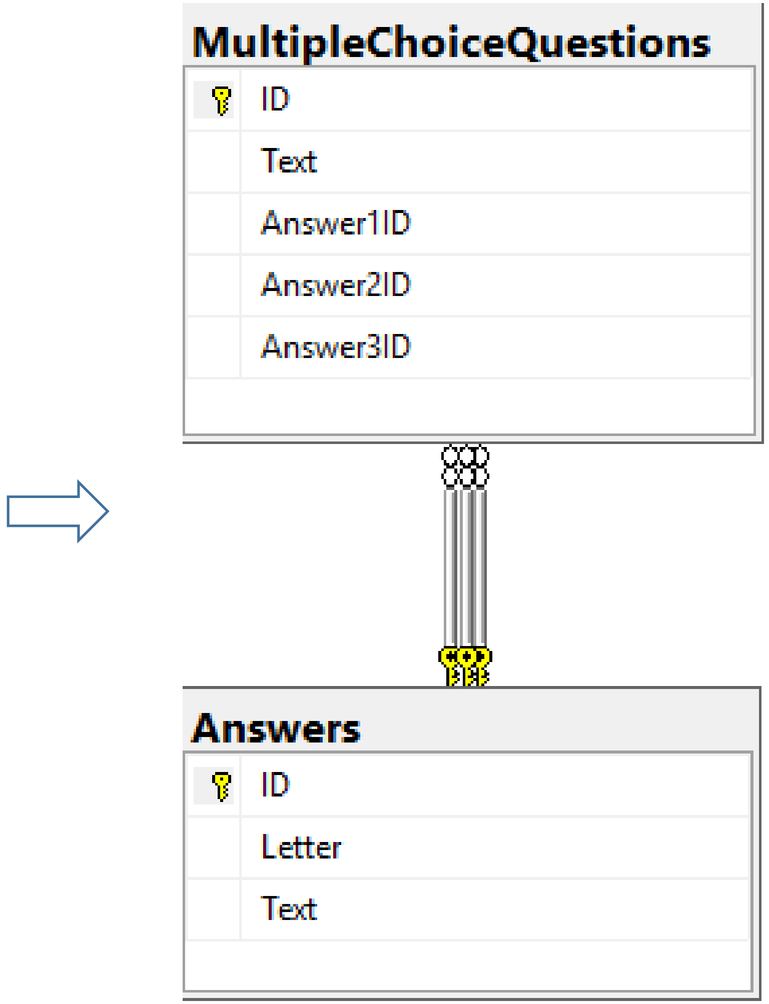
- obs. a questionnaire can also have open answer questions, etc.

- mapping aggregation / composition

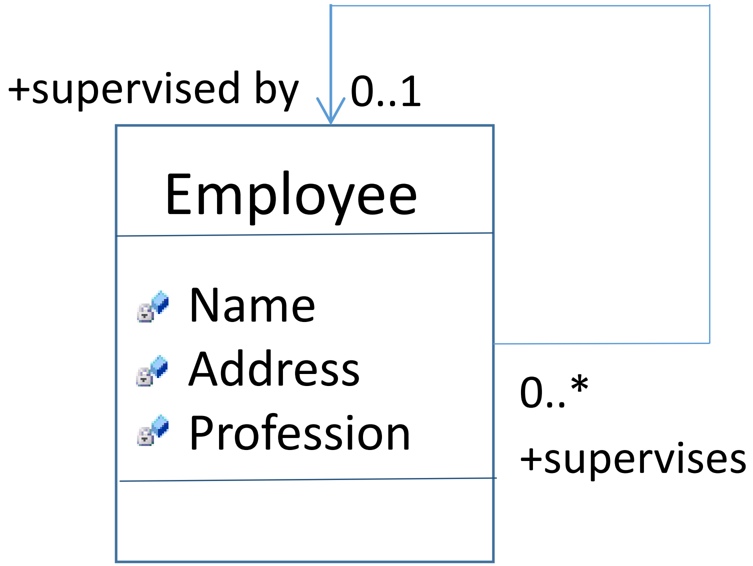


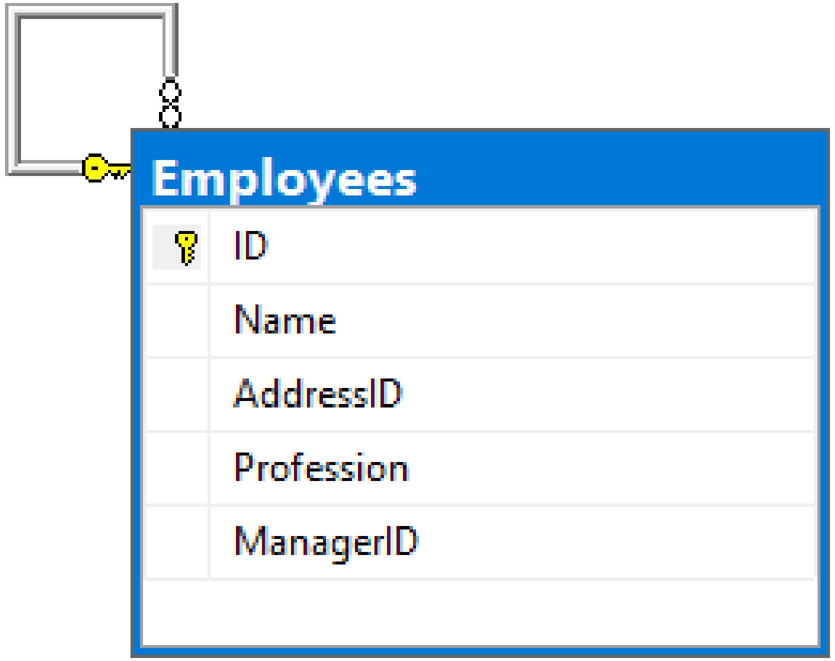
- mapping aggregation / composition





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

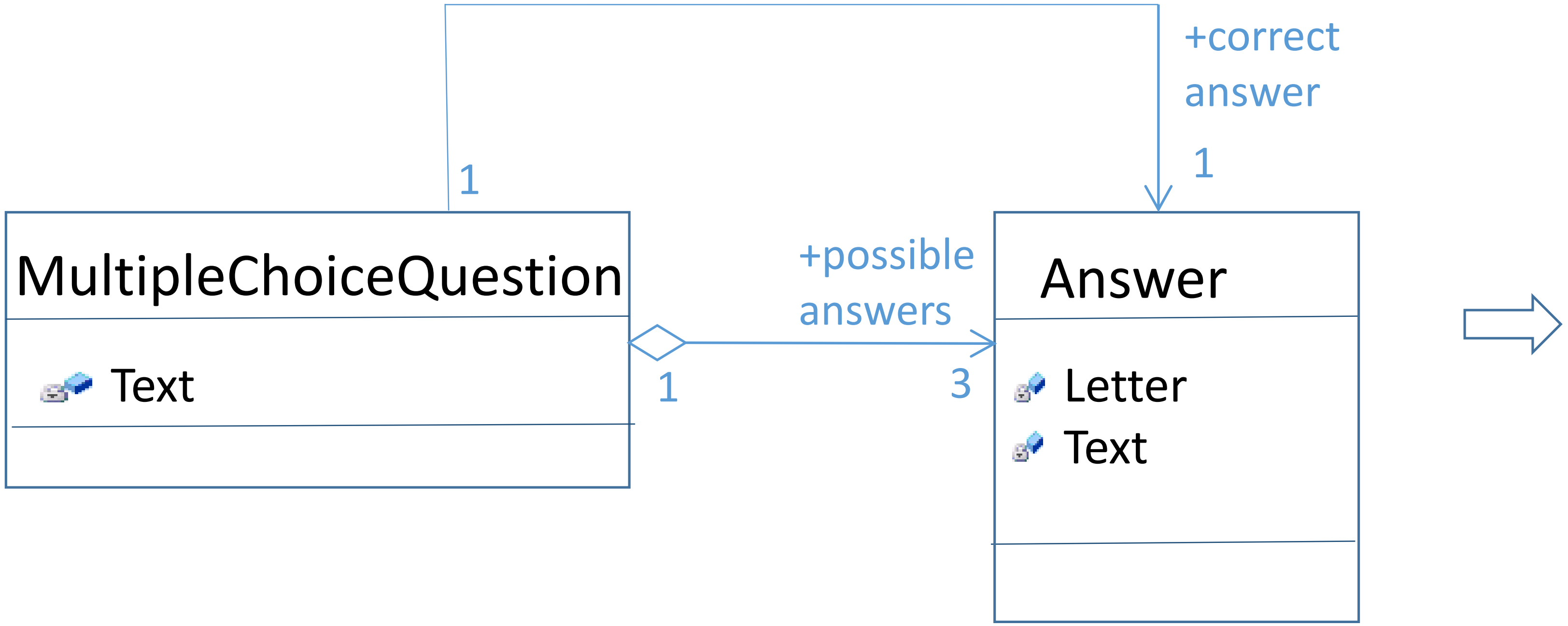


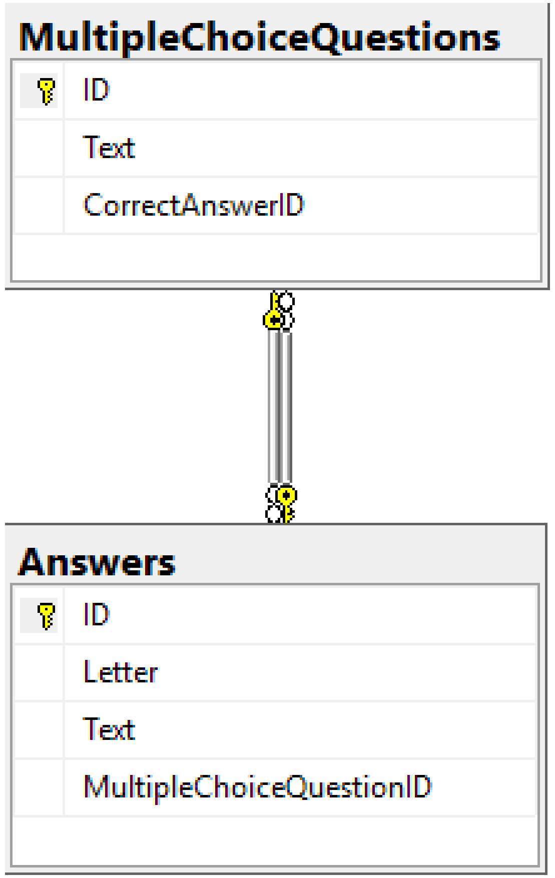


Employees

ID
Name
AddressID
Profession
ManagerID

- mapping reflexive associations
- ON DELETE CASCADE, similar problem: 2 different tables, each with a foreign key referencing the other one





MultipleChoiceQuestions

ID
Text
CorrectAnswerID

Answers

ID
Letter
Text
MultipleChoiceQuestionID

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

[Si10] SILBERSCHATZ, A., KORTH, H., SUDARSHAN, S., Database System Concepts, McGraw-Hill, 2010, http://codex.cs.yale.edu/avi/db-book/

[Kn76] KNUTH, D.E., Tratat de programare a calculatoarelor. Sortare și căutare. Ed. Tehnică, București, 1976

[Ga08] GARCIA-MOLINA, H., ULLMAN, J., WIDOM, J., Database Systems: The Complete Book, Prentice Hall Press, 2008