Course 5. Conceptual Database Design

One common problem in application design phase is how could be translated persistent classes in tables of a relational database.
Easiest / simplest way: one-to-one mapping between classes (entity types) to relations (tables).This approach has a lot of drawbacks:
- Too many tables – more tables result than are actually necessary
- Too many joins – too many tables -> too many JOIN operations
- Missed tables – many-to-many associations require a third relational table
- Inappropriate handling of generalization (inheritance)
- Denormalization of data - the same data is duplicated in multiple tables

Mapping a class to a table

Rules:
- Table name is plural variant of class name
- One-to-one mapping of all simple attributes to table fields
- Add surrogate key in corresponding table
- Composed attributes become tables
- Derived attributes are not mapped to fields (exception for efficiency reasons)

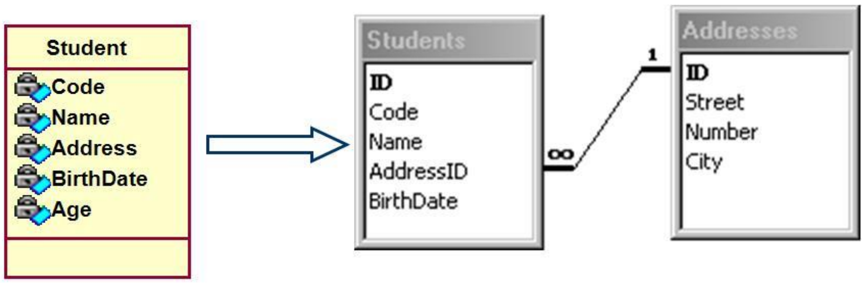


Figure 5.1

Mapping simple associations

1 : 0,1 multiplicity
- create a table for each class involved in association
- the key of “1” table is a foreign key in the related table.
- only one key (usually the key of the table corresponding to multiplicity 1) will be automatically generated

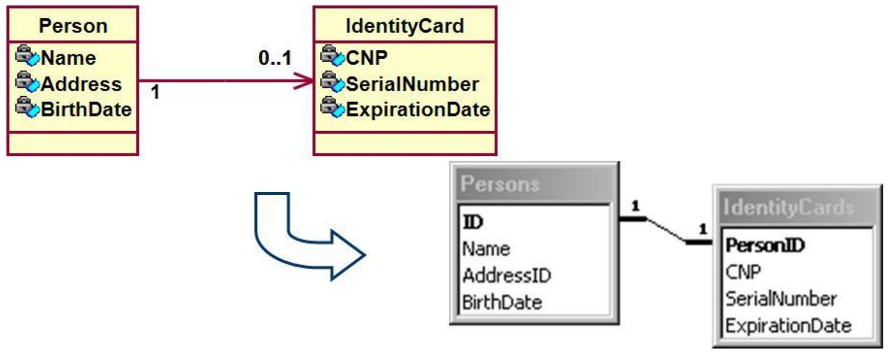


Figure 5.2.

1 : 1 multiplicity
- create only one table which contains attributes of both classes
- also applicable for 1 : 0,1 associations, if there are a low number of cases when objects of the first class are not linked with objects from the second class

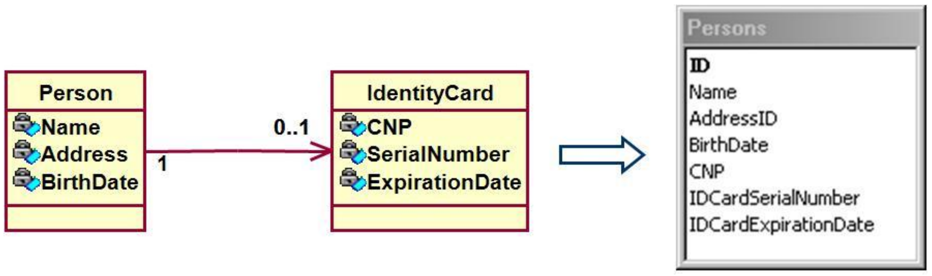


Figure 5.3.

1 : n multiplicity
- create a table for each class involved in association
- the key of "1" table is a foreign key in the "1..*" table

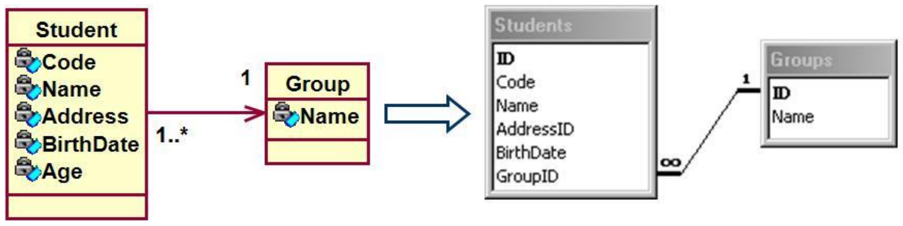


Figure 5.4.

m : n multiplicity
- create a table for each class involved in association
- create an additional intersection (cross) table
- the primary keys of each initial table are defined as foreign keys in the intersection table
- the primary key of the intersection table is, usually, the composite of the two foreign keys from other tables. It can be, also, a surrogate key
- if the association is linked with an association class, all attributes of this class will be inserted in intersection table
- usually, the name of intersection table is composed between the names of the initial classes, but not necessary

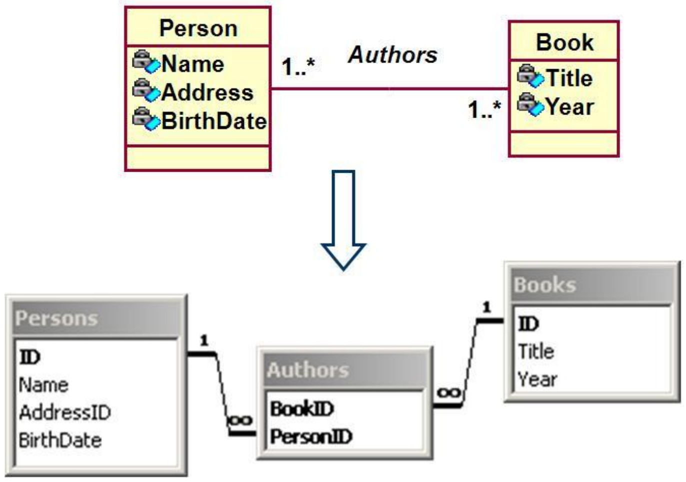


Figure 5.5. **m:n** translation - first approach

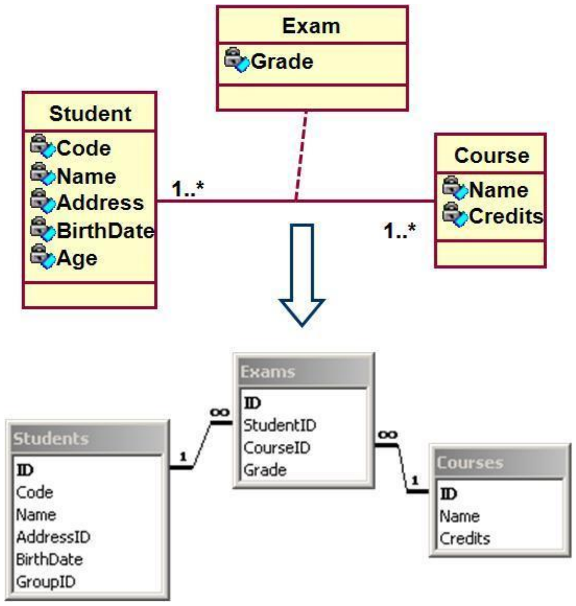
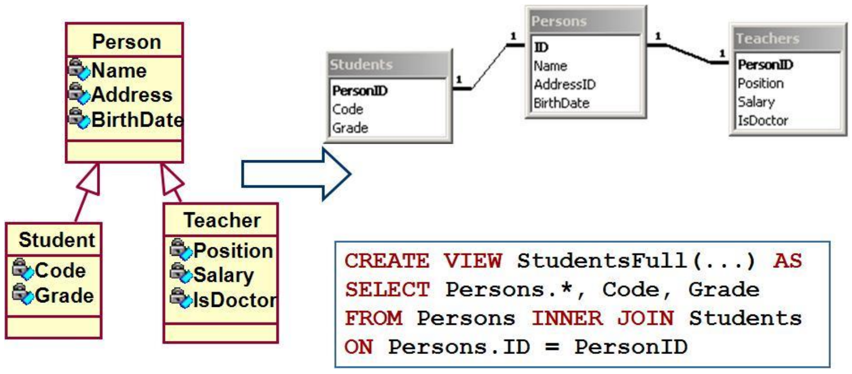


Figure 5.6. **m:n** translation - second approach

Mapping inheritance

Alternative 1: Create a table for each class and an SQL view for each superclass/subclass pair
- Flexibility - allow future subclasses to be added without impact on other classes or views
- It results in the most DBMS objects (tables and views)
- It affects the system performance: each access will always require an SQL join through the view

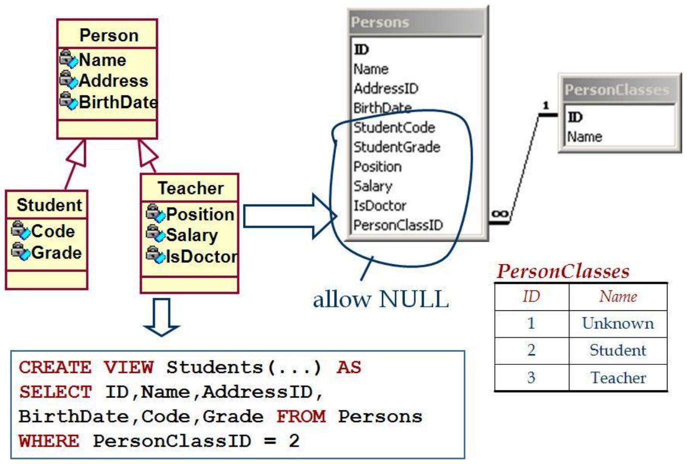


```
CREATE VIEW StudentsFull(...) AS
SELECT Persons.*, Code, Grade
FROM Persons INNER JOIN Students
ON Persons.ID = PersonID
```

Figure 5.7

Alternative 2: Create one table (for superclass) and de-normalize all attributes from subclasses into the one superclass table.
- It results in the least DBMS objects (only one table) -optional, views and a subclass table could be defined
- It typically results in the best overall performance

- Future subclassing requires structure modification
- Requires “dead space” => increase record length => could impact performance



```
CREATE VIEW Students(...) AS
SELECT ID,Name,AddressID,
BirthDate,Code,Grade FROM Persons
WHERE PersonClassID = 2
```

PersonClasses

ID	*Name*
1	Unknown
2	Student
3	Teacher

Figure 5.8.

Alternative 3: Create a table for each subclass and de-normalize all superclass attributes into each subclass table
- This alternative results in adequate performance
- Future subclassing will not affect the existing tables
- Superclass structure changes affect all defined tables!

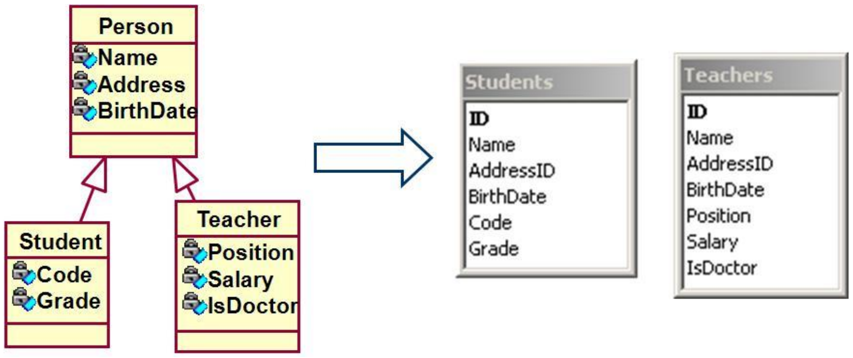


Figure 5.9.

Hint on mapping inheritance:
- If the number of records is somewhat limited (so, the performance is not a concern) then the most flexible alternative should be selected - Alternative 1
- If the number of attributes in superclass is small (compared to its subclasses) a good choice is the Alternative 3.
- If the amount of data in the subclasses is sparse then Alternative 2 might be best.

Mapping Aggregation/Composition

- Aggregation and composition are modeled as standard relations
- Composition relationships often are implemented as just one relational table (because they involve many one-to-one relationships)
- Fixed number of "parts" for a "whole" => introduce the same number of foreign key in "whole" table
- If composition is implemented in separate tables => cascading of deletes must be implemented (for aggregation it is not necessary)

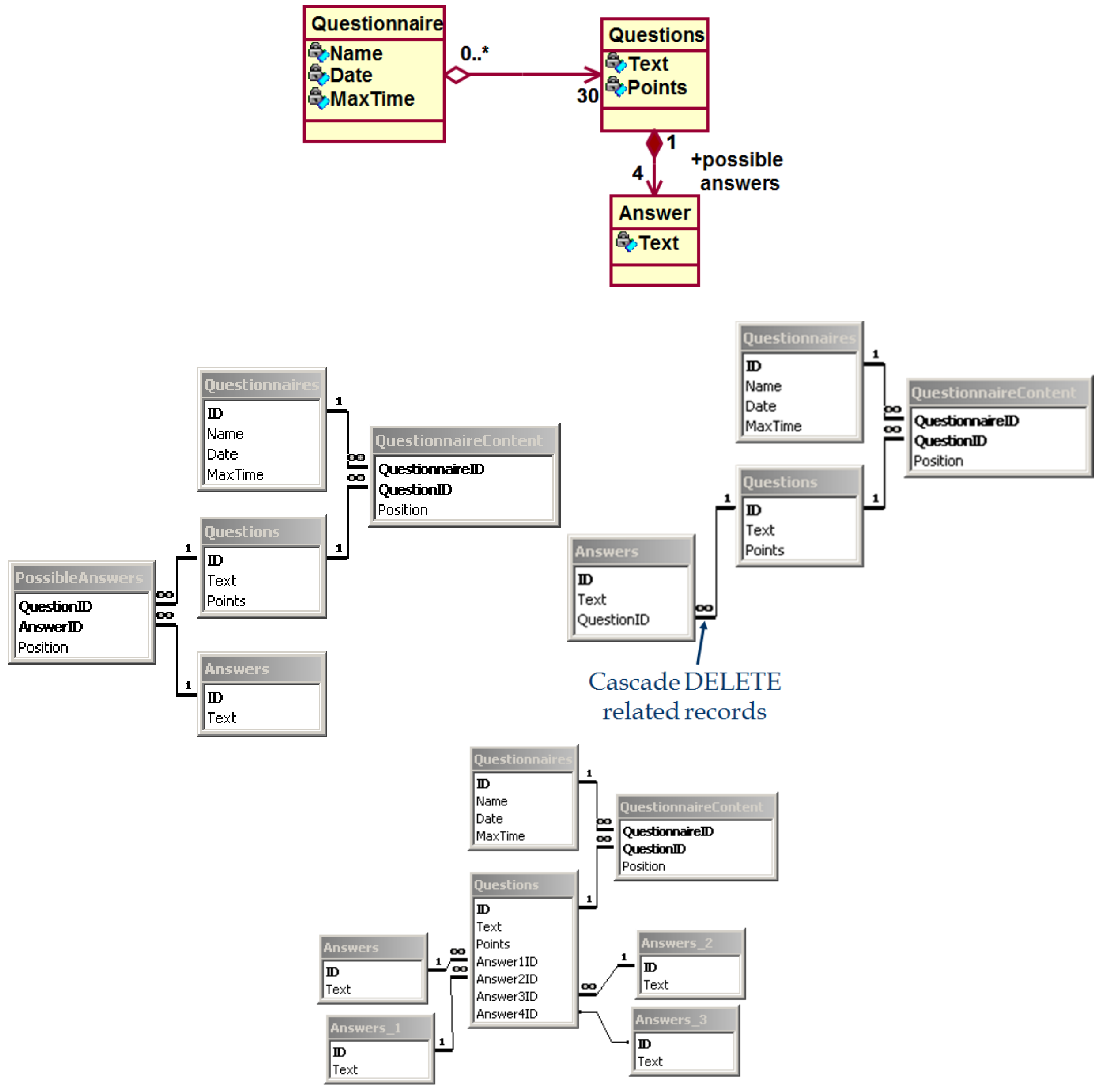


Figure 5.10

Mapping Reflexive Associations

- A new field is added which is itself foreign key (called recursive relationship)
- If cascade delete is set and there are 2 records that point one to each other an error occurs

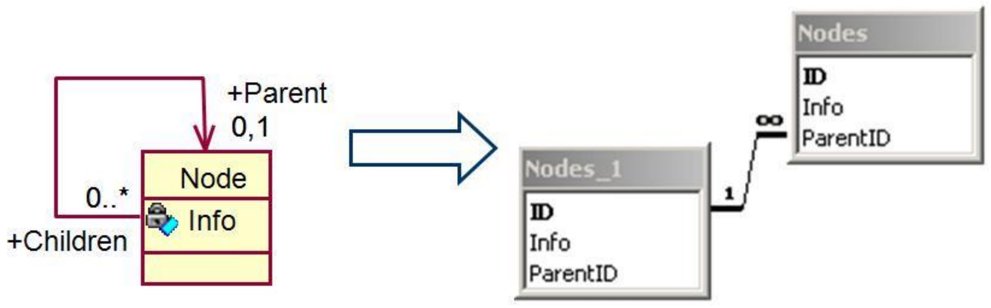


Figure 5.11.

It is a very similar situation for 2 distinct tables which have defined foreign keys one to another

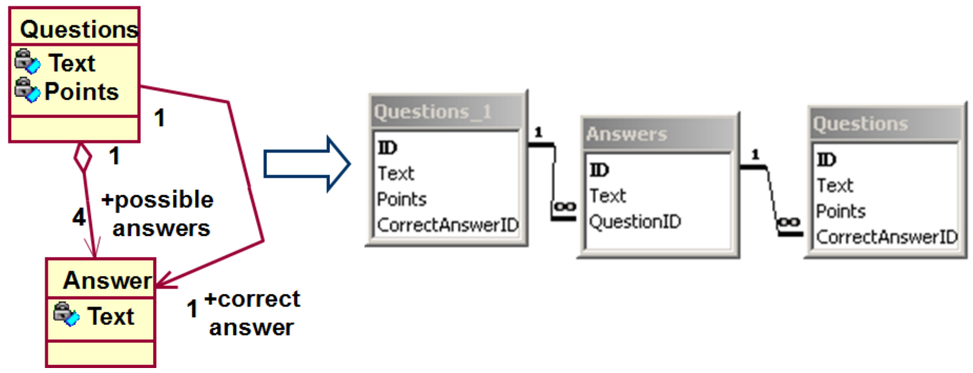


Figure 5.12.