Design and Analysis of Information Systems (MAS)

Mariusz Trzaska, Ph. D.
mtrzaska@pjwstk.edu.pl

Lecture 09

Utilization of the Relational Model in Object-oriented Programming Languages

Outline

- The relational model
- Popularity of relational DBs.
- Mapping of particular object-oriented constructs
  - Classes,
  - Associations,
  - Inheritance.
- Relational DBs in object-oriented programming languages (e.g. JDBC).

These slides make use of the following sources:

The „Relacyjne Bazy Danych" lecture prepared by L. Banachowski, J. Wierzbicki.

The Relational Model

- First formulated and proposed by Edgar Codd (1970)
- Terms
  - Tables
  - Relations
  - Keys

The Relational Model (2)

- Tables
  - The number of columns is fixed.
  - Each column has a name and a domain describing a set of possible values stored in the column.
  - A row describes an entity (a piece of information), e.g. a person.
  - The order of rows and columns is not fixed. Hence it is not a good idea to relay on it.

The Relational Model (3)

- Tables – *cont.*

IdLecturer	FirstName	LastName	Title
237	Jan	Kowalski	Doktor
3245	Maciej	Jankowski	Docent
8976	Artur	Malinowski	Profesor

LectureName	Code	IdLecturer
Bazy danych	BDA	1237
Projektowanie systemów informacyjnych	PSI	3245
Technologie internetowe	TIN	3245
Programowanie obiektowe	POB	8976
Systemy decyzyjne	SDE	1237

The Relational Model (4)

- A meaning of the IdLecturer column
  - The value does not describe a lecture's property.
  - It represents a relationship between a lecturer and its lecture. The lecture information is stored in a different table.
  - Hence it is extremely important that the identifier uniquely points to a particular lecturer – this is the only way of identification of a row.

The Relational Model (5)

- A key
  - Primary. One or more columns which values identify a whole row.
  - Secondary
    - One or more columns which values are interpreted as pointers to rows in a other table.
    - Example: in the Lectures table the secondary key is IdLecturer, which values come from the IdLecturer in the Lecturers table.

The Relational Model (6)

- The `Null` value
  - Domains (possible values) of the columns are extended with a special flag `Null` – meaning no value.
  - This lack of value could be
    - temporary,
    - business oriented.
  - The `Null` value is different than `0` or „”
- An index
  - Allows for fast retrieving values in indexed columns.

The Relational Model Popularity

- The popularity of
  - The relational model,
  - Or database systems?
- The role of legacy systems.
- The „database” term is associated mainly with a relational DB.
- Probably it will change in the future.
  - Persistent libraries, e.g. Hibernate;
  - Object-oriented DBs;
  - NoSQL.

Database Management Systems

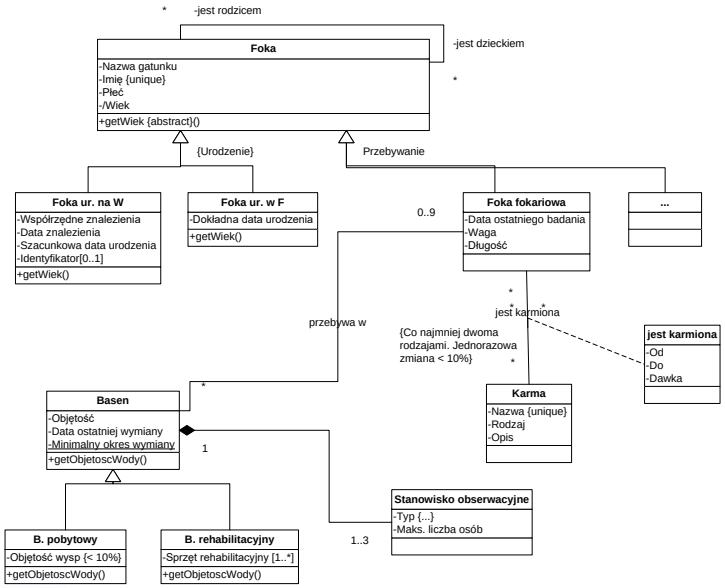
- Main reason for utilization of DBMS:
  - A query language,
  - Iterative way of storing data,
  - Performance,
  - Security,
  - Safety,
  - Ability for storing a big amount of data.
- Disadvantages of the relational DBMS
  - The relational model itself ☺
  - Other…

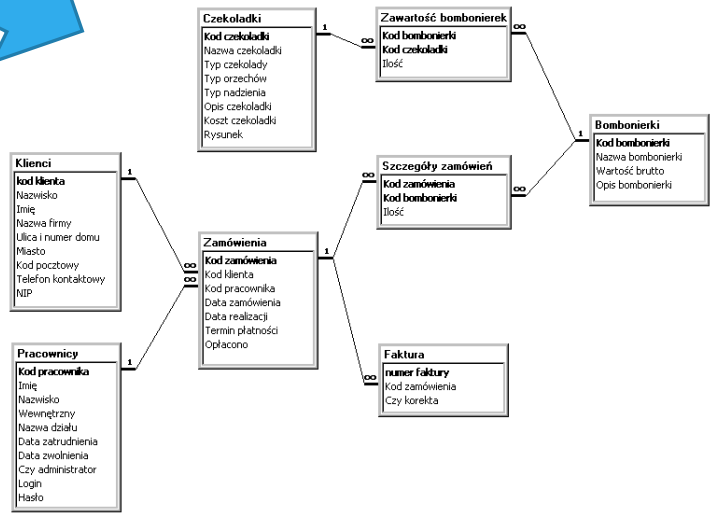
Utilization of the DBMS

- The bad consequence of the relational DBMS utilization from object-oriented programming languages is ***the impedance mismatch***.
- It means necessity of tailoring two „different worlds”:
  - A relational one,
  - An object-oriented one.

Utilization of the DBMS (2)

- The impedance mismatch





The transformation: OO model <-> relational model

- Classes
  - Attributes:
    - Simple and complex
    - Mandatory and optional
    - Single and multi-valued
    - Of an object or a class
    - Derived
  - Methods:
    - Of a class
    - Of an object
    - Overloading and overriding

The transformation: OO model <-> relational model (2)

- Associations
  - binary,
  - attribute association,
  - qualified,
  - n-ary,
  - an aggregation,
  - a composition.

The transformation: OO model <-> relational model (3)

- The inheritance
  - overlapping,
  - complete, incomplete,
  - multi-inheritance,
  - multi-aspect,
  - dynamic.

The Class Mapping

- Each class is replaced by a table.
- Each attribute is stored in a column.
- For each table add a special attribute – the *primary key*. Do not use business-oriented attributes.
- Special kinds of attributes
  - Simple and complex,
  - Mandatory and optional,
  - Single and multi-valued,
  - Of an object or a class,
  - Derived.

The Class Mapping (2)

- Special kinds of attributes
  - Simple

    A column in a table, i.e. *LastName*.

  - Complex
    - In the business table, as many attributes (columns) created by „unwrapping" the complex attribute;
    - As a single flat attribute, i.e. an address: *"Marszałkowska street 12, 03-333 Warsaw, Poland"*

The Class Mapping (3)

- Special kinds of attributes
  - Complex – cont.
    - As a single flat attribute but with a dedicated syntax, i.e.

```
<Address>
          <Street>Marszałkowska</Street>
          <No>12</No>
          <Zip>03-333</Zip>
          <Country>Polska</Country>
</Address>
```

    - Using a separate table (i.e. an *Address*) and relationship to the „main" one

IdAddress	Street	No	Zip	Country
328	Marszałkowska	12	03-333	Polska

  - Optional

    The column has to allow for the `null` value. Be carefull with, e.g. queries.

The Class Mapping (4)

- Special kinds of attributes
  - Multi-valued
    - As a single column with a special syntax. Each value is separated with a special character or using e.g. XML – the same way like in the complex attribute.
    - Creation of the dedicated table and linking with the „main” one.
  - The class attribute
    - Special table storing values of the class attributes for all classes (with only one row?).
    - Storing outside the DB, i.e. in the source code of the business logic program.

The Class Mapping (5)

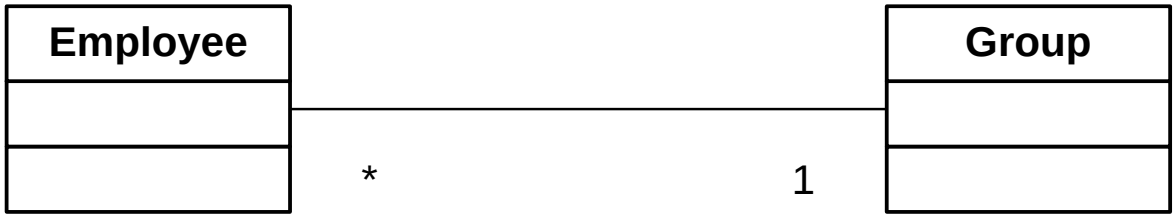
- Special kinds of attributes
  - Derived

    Creation of the:

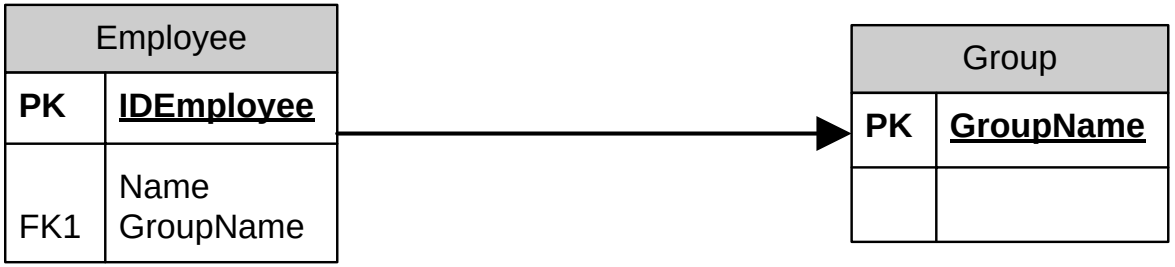
    - Database View (if it is allowed by the DBMS),
    - Special method in the DB programming language (e.g. PL-SQL),
    - Method outside the DBMS.

The Associations Mapping

- Binary associations
  - Cardinality 1 – 1, 1 - *



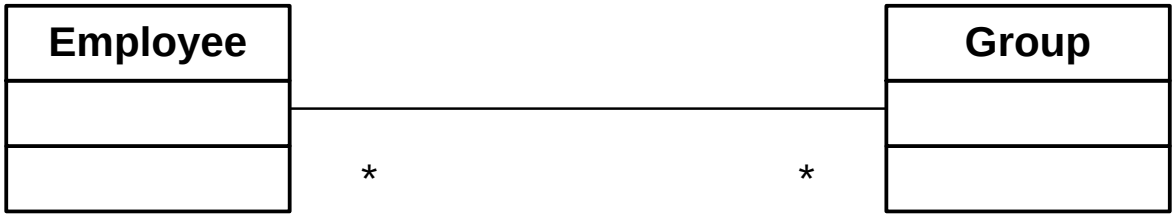
  - Replaced by the relationships.
  - Added a foreign key to the right class.



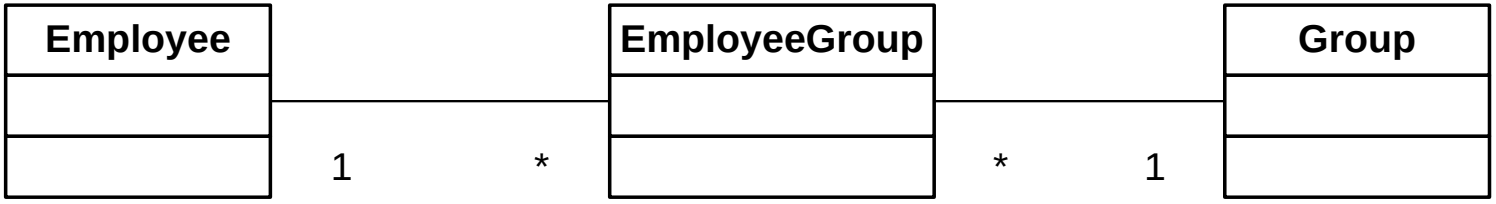
  - To find all employees belonging to the particular group we need to check them all.

The Associations Mapping (2)

- Binary associations
  - Cardinality * - *

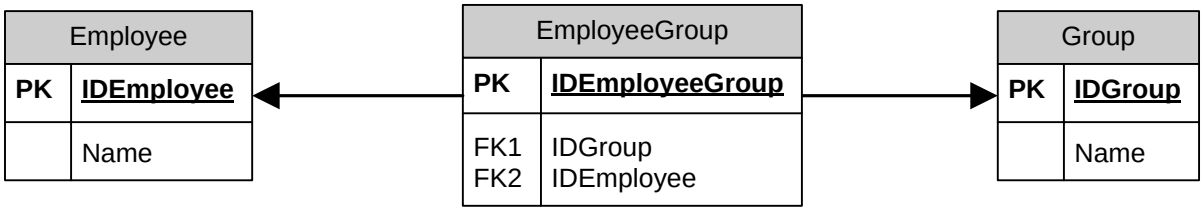


    - We need to introduce a middle-class (new role names?).
    - As the result we have two associations „1 - *" rather then one „* - *".



The Associations Mapping (3)

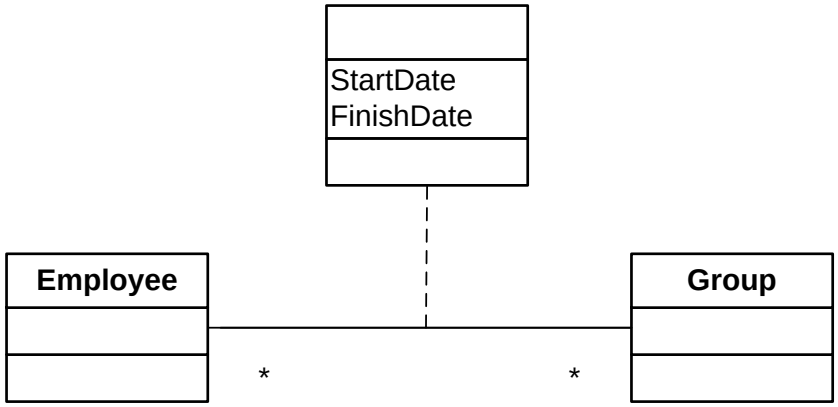
- Binary associations
  - Cardinality * - * - cont.
    - Three classes are mapped on three tables.
    - There will be foreign keys in the middle table.



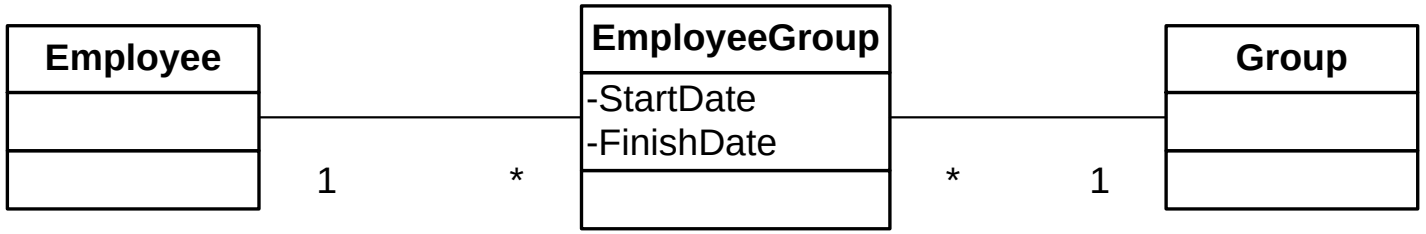
    - Unfortunately to find
      - all employees belonging to the particular group
      - All groups connected with a particular employee
    - we need to check all entities from the EmployeeGroup table.

The Associations Mapping (4)

- An association with an attribute (a class)



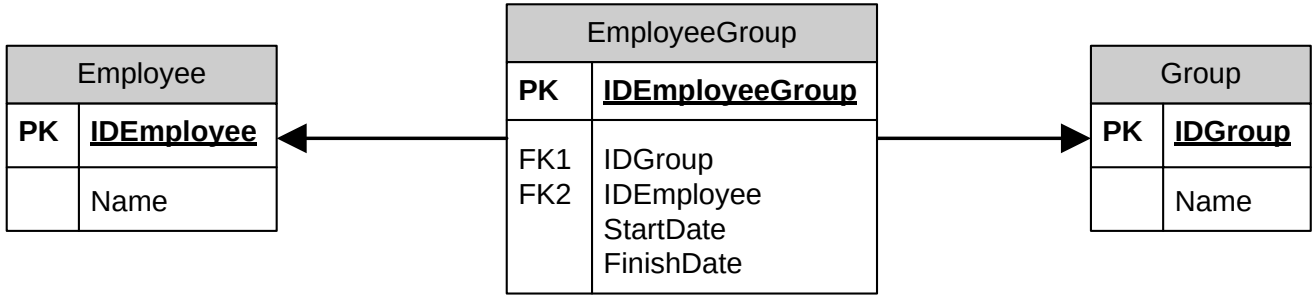
- Similar approach like in case „* - *" association
  - The middle class with attributes of the association.



The Associations Mapping (5)

- An association with an attribute (a class) – cont.
  - Three classes are mapped on three tables.
  - There will be:
    - foreign keys,
    - association attributes,

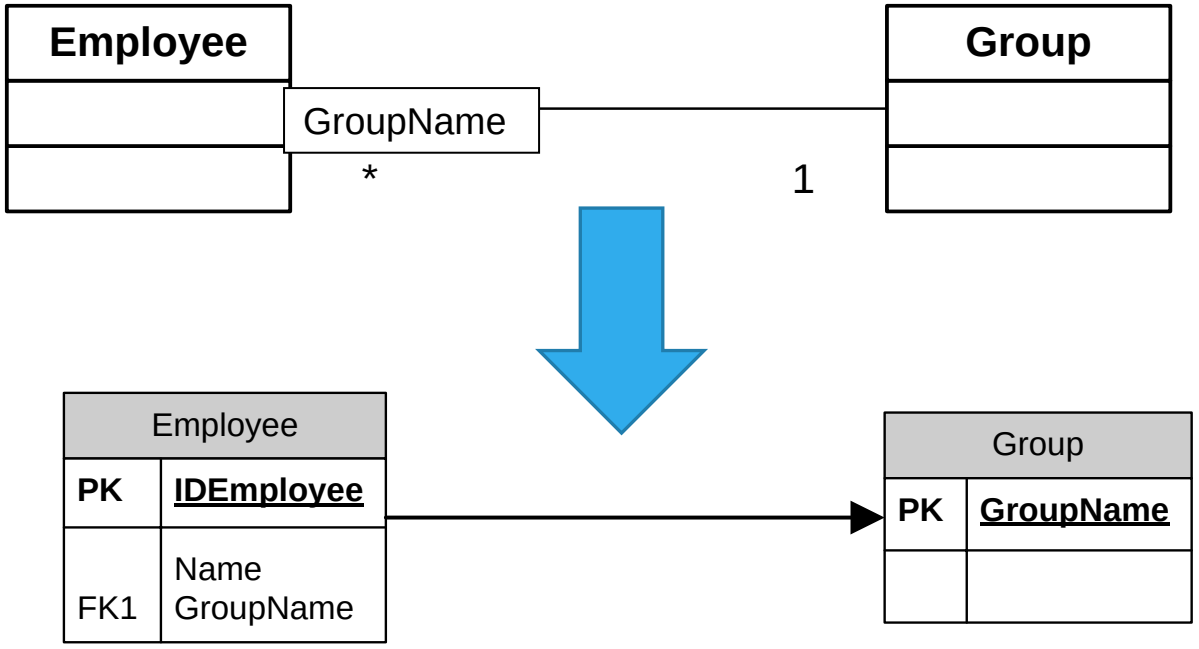
    in the middle table.



- Similar disadvantages like in case „* - *” association.

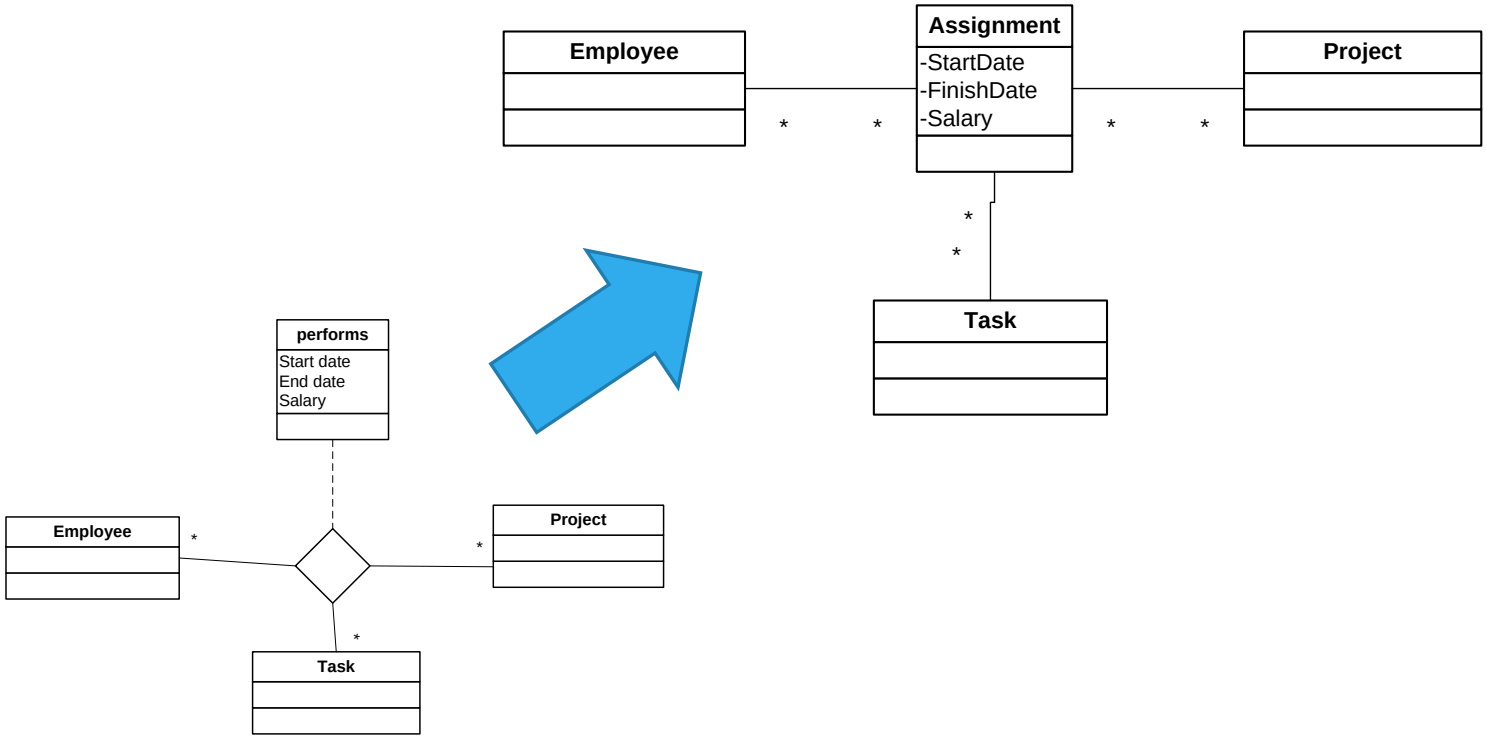
The Associations Mapping (6)

- Qualified Association
  - No counterpart in the relational model.
  - Could be implemented using a special foreign key (the qualifier).



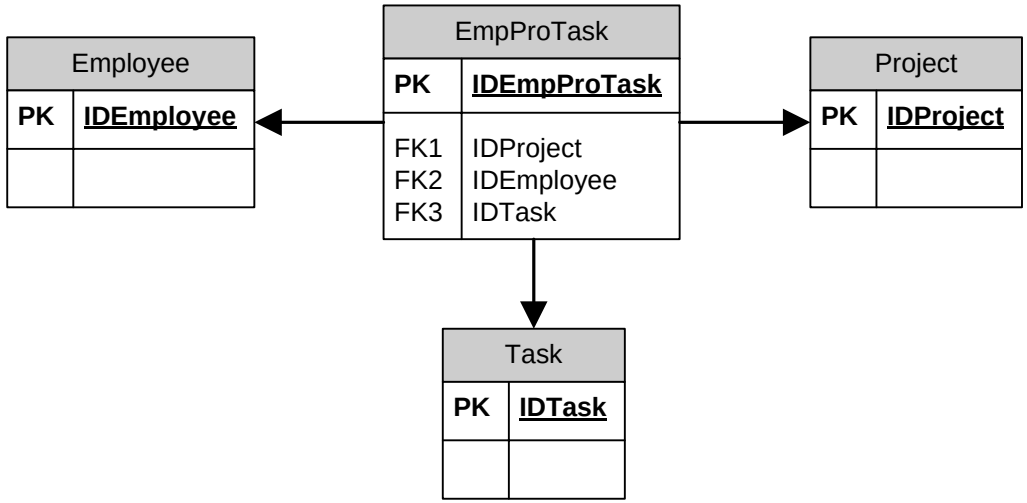
The Associations Mapping (7)

- N-ary Association
  - Introducing an additional middle-class



The Associations Mapping (8)

- N-ary Association – cont.
  - Four classes (n = 3) are mapped on four tables.
  - The „middle" table contains foreign keys and middle-class attributes. The diagram does not present middle tables for „* - *" relationships.



  - Similar disadvantages like in case „* - *" association.

The Associations Mapping (9)

- An aggregation
  - Similarly to the move from conceptual model to the implementation model, the aggregations are implemented as associations.
- A composition
  - If it is allowed by a DBMS we can:
    - Create a dedicated *database view*,
    - Use database *integrity constrains* and/or *triggers*.

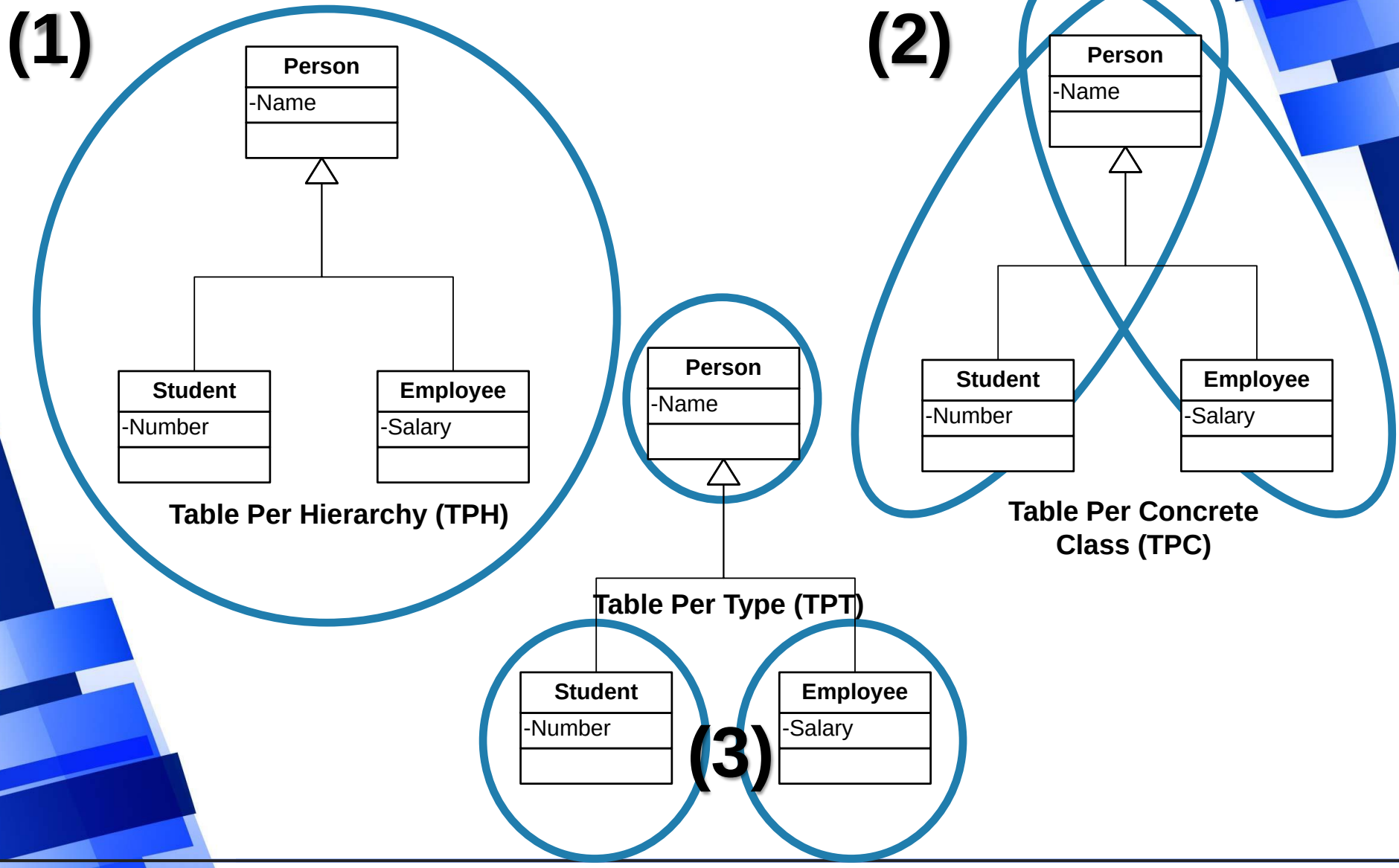
The Inheritance Mapping

- There is no inheritance in pure relational DBMS.
- In some modern systems there is a simple (disjoint) inheritance.
- Of course there are some way for „faking" the inheritance.

The Inheritance Mapping (2)

- The approaches to bypassing the inheritance are similar to the ones utilized during conceptual model transformations.
  - Utilization of the relationships (associations) among tables (classes),
  - Flattening the hierarchy.

The Inheritance Mapping (3)

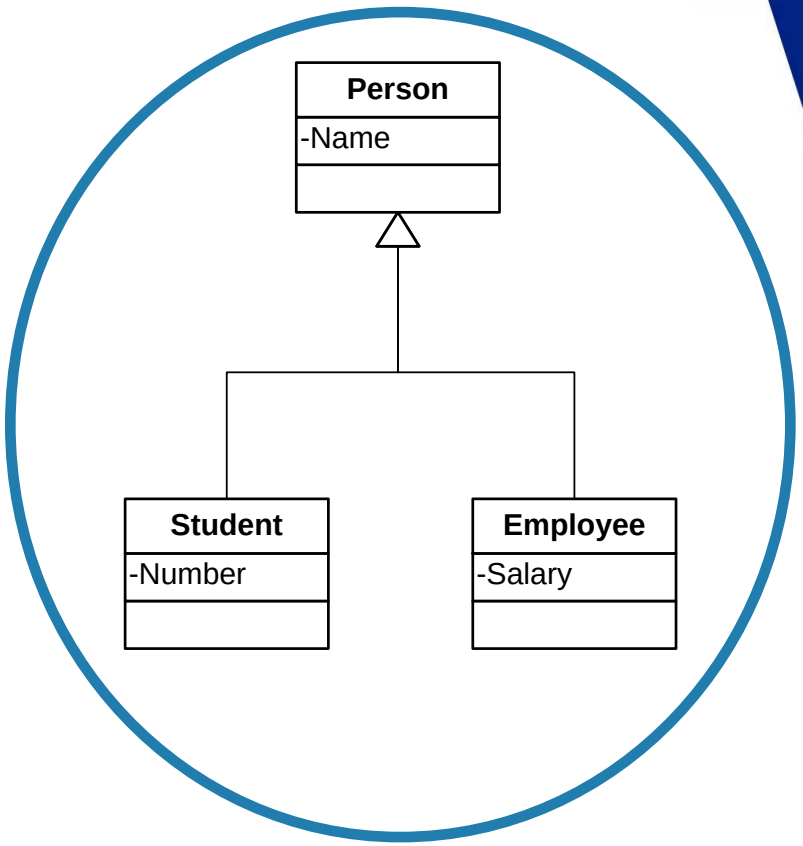


The Inheritance Mapping (4)

- Approach no 1 (TPH - *Table Per Hierarchy*)
  - Entire hierarchy (all classes) is located in a single table.
  - Adding a column

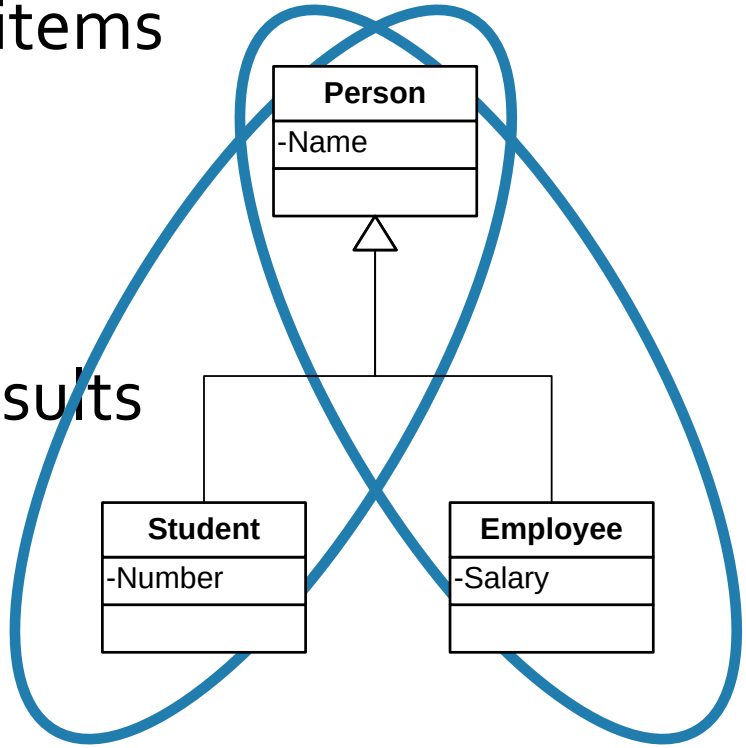
    determining the type

    (kind of) of the entity.
  - Pros and cons.



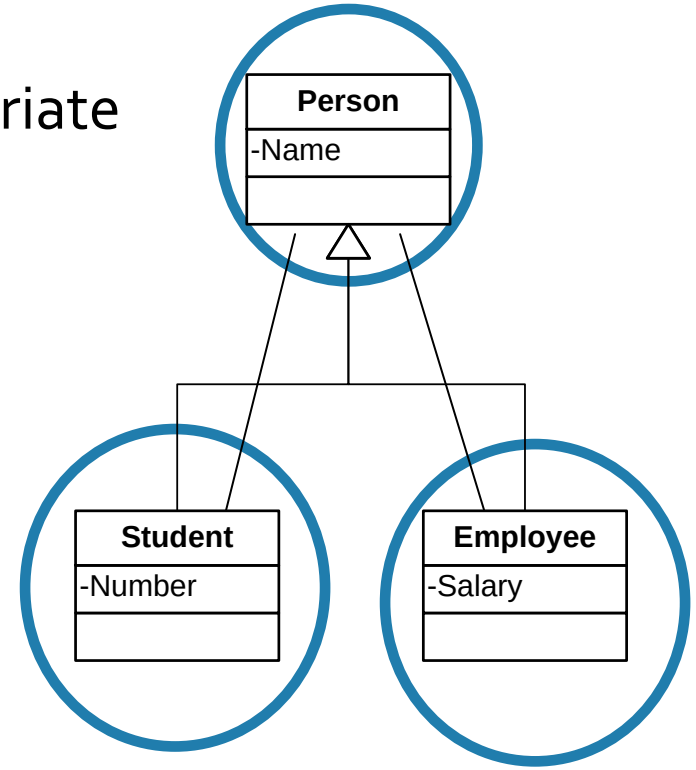
The Inheritance Mapping (5)

- Approach no 2 (TPC - *Table Per Concrete Class*)
  - Single table for each subclass.
  - There will be super class items in each „sub-table”.
  - Pros and cons.
  - In order to process the results one can use the `Union` operator (SQL).



The Inheritance Mapping (6)

- Approach no 3 (TPT - *Table Per Type*)
  - Each class has its own table.
  - Inheritance is replaced with aggregations which means relationships.
  - We also need to add appropriate keys (primary and foreign/secondary).
  - Pros and cons.



Relational DBMS in object-oriented programming languages

- Every popular programming language has a dedicated library to work with a RDB.
- Java
  - JDBC
  - Dedicated solutions for particular DBs,
- Microsoft C#
  - ADO,
- C++
  - ADO (MS),
  - ODBC.

RDB in the Java

- The most popular way is utilizing the functionality provided by the JDBC.

```
// try to load the driver'a
try
{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
}
catch (ClassNotFoundException cnfe)
{
        // Error
        System.err.println ("Error: " + cnfe);
        System.exit(0);
}

String url = "jdbc:odbc:" + args[0];

Connection db_connection = DriverManager.getConnection (url, "dba", "sql");

Statement db_statement = db_connection.createStatement();

db_statement.executeUpdate("create table employee { int id, char(50) name };");
db_statement.executeUpdate("insert into employee values (1, 'Jan Kowlski');");
db_connection.commit();
```

http://www.javacoffeebreak.com/articles/jdbc/

RBD w języku Java (2)

- JDBC – cont.

```
// [...]

// Execute the statement
ResultSet result = db_statement.executeQuery("select * from employee");

// Process the result
while (result.next() )
{
        System.out.println ("ID : " + result.getInt("ID"));

        System.out.println ("Name : " + result.getString("Name"));
        System.out.println ();
}
```

http://www.javacoffeebreak.com/articles/jdbc/

- As it can be seen it is not the easiest way of working with data.
- Especially if we compare it to an object-oriented approach, e.g. ObjectPlus.

- *To be continued…*