course:

Database Systems (NDBI025)

SS2017/18

lecture 1:

Conteptual database modeling

doc. RNDr. Tomáš Skopal, Ph.D.
Mgr. Martin Nečaský, Ph.D.
RNDr. Michal Kopecký, Ph.D.
Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague

Organizational stuff

- student duties
  - credit test (≥ 60% points)
  - exam test
  - attendance strongly recommended (but not mandatory)
    - the slides alone are not comprehensive
    - other sources
      - textbook:
        Ramakrishnan, Gehrke: Database Systems Management, McGraw-Hill, 2003 (available in faculty library)
      - software:
        see references in particular lectures and also the web site below
- web: http://www.ms.mff.cuni.cz/~kopecky/vyuka/ndbi025/

What is the course (not) about?

- it is about
  - conceptual data modeling
  - relational model
  - physical implementation of database management
  - transactional processing
  - introduction to database applications, multimedia and XML databases
- it is NOT about
  - data mining
  - text databases
  - data warehousing, OLAP
  - cloud computing

Follow-up courses

- other topics are subject to follow-up courses
  - Database languages I, II
  - Datalog
  - Database applications
  - Oracle and MS SQL Server administration
  - Transactions
  - Stochastic methods in databases
  - Searching the web and multimedia databases
  - Retrieval of multimedia content on the web
  - XML technology
  - NoSQL databases

Today's lecture outline

- what is a database?
- conceptual modeling
  - ER
  - UML
  - and beyond (OCL)
    - See e.g.
      - Cabot, Jordi; Gogolla, Martin: Object Constraint Language (OCL): a Definitive Guide (https://modeling-languages.com/wp-content/uploads/2012/03/OCLChapter.pdf)
      - https://www.slideshare.net/jcabot/ocl-tutorial

Database management system

- database
  - logically ordered collection of related data instances
  - self-describing, meta-data stored together with data
- database management system
  - general software system for shared access to database
  - provides mechanisms to ensure security, reliability and integrity of the stored data
- database administrator
  - the necessary human factor

Why database systems?

- means of data sharing and reusability
- unified interface and languages for data definition and data manipulation
- data consistency and correctness
- redundancy elimination (compact storage)

Basic Terminology
Model vs. Schema vs. Diagram

- **model** = modeling language
  - set of constructs you can use to express something
  - e.g., UML model = {class, attribute, association}
  - e.g., relational model = {relation schema, attribute}
- **schema** = modeling language expression
  - instance of a model
  - e.g., relational schema = {Person(name, email)}
- **diagram** = schema visualization

Basic Terminology
Stakeholder

- stakeholder is any person which is relevant for your application(s).
  - e.g., application user, investor, owner, domain expert, etc.
- you have to communicate with all stakeholders and balance their requirements when developing a (database) application.

Basic Terminology
Three layers of database modeling

- conceptual layer
  - models a part of the real world relevant for applications built on top of your database
    - real world part = **real-world entities and relationships between them**
  - different conceptual models (e.g. ER, UML)
- logical layer
  - specifies how conceptual components are represented in database structures
  - different logical models (e.g. relational, object-relational, XML, graph, multimedia, etc.)
- physical model
  - specifies how logical database structures are implemented in a specific technical environment
  - data files, index structures (e.g. B+ trees), etc.

Conceptual Modeling Process

- process of creating a conceptual schema
  of an application (or applications) in a selected conceptual model
  on the base of given requirements of various stakeholders
- in fact you do not create only one conceptual schema but multiple
  - each schema describes the application(s) from
    a different point of view
    - there is a different conceptual model suitable for each viewpoint
- two basic viewpoints
  - **conceptual data viewpoint** (this lecture)
  - conceptual functional viewpoint (different courses, e.g., NSWI041)

Conceptual Modeling Process

Requirements analysis

- identify types of entities
- identify types of relationships
- identify characteristics

Model identified types

- choose modeling language
- create conceptual schema

Iteratively adapt your schemas to requirements changing over time

Conceptual Data Modeling Process
STEP 1: Requirements Analysis

- start with requirements of different stakeholders
  - usually expressed in a natural language
  - informal discussions, inquiries
- identify important types of real-world entities, their characteristics, and types of relationships between them
  - ambiguous process
    (because of informal requirements)

Conceptual Data Modeling Process
STEP 1.1: Identify types of entities

"Our social network consists of **<u>person</u>**s which may have other persons as their colleagues. A person can also be a member of several **<u>research team</u>**s. And, she can work on various **<u>research project</u>**s. A team consists of persons which cooperate together. Each team has a leader who must be an **<u>academic professor</u>** (**<u>assistant</u>**, **<u>associate</u>** or **<u>full</u>**). A team acts as an individual entity which can cooperate with other teams. Usually, it is formally part of an **<u>official institution</u>**, e.g., a **<u>university department</u>**. A project consists of persons working on a project but only as research team members."

- identified types of entities
  - **Person**
  - Research **Team**
  - Research **Project**
  - Academic **Professor**
  - **Assistant Professor**
  - **Associate Professor**
  - **Full Professor**
  - Official **Institution**
  - University **Department**

Conceptual Data Modeling Process
STEP 1.2: Identify types of relationships

"Our social network consists of ***person***s which may have ***other person***s as their colleagues. A ***person*** can also be a member of several ***research team***s. And, she (***person***) can work on various research ***project***s. A ***team*** consists of ***person***s which cooperate together. Each ***team*** has a leader who must be an ***academic professor*** (***assistant***, ***associate*** or ***full***). A ***team*** acts as an individual entity which can cooperate with ***other team***s. Usually, it (***team***) is formally part of an ***official institution***, e.g. a ***university department***. A *project* consists of *person*s working on a project but only as research team members."

- identified types of relationships
  - Person is colleague of Person
  - Person is member of Research Team
  - Person works on Project
  - Team consists of Person
  - Team has leader Professor
  - Team cooperates with Team
  - Team is part of Official Institution
  - Project consists of Person who is a member of Project

Conceptual Data Modeling Process
STEP 1.3: Identify characteristics of types

“Each person has a **name** and is identified by its **personal number**. A person can be called to its registered **phone numbers**. We need to know at least one phone number. We also need to send her **emails**.”

- identified characteristics of type **Person:**
  - **personal number**
  - **name**
  - *one or more* **phone numbers**
  - **email**

Conceptual Data Modeling Process
STEP 1.3: Identify characteristics of types

"We need to know **when** a person **became** a member of a project and **when** she **finished** her membership."

- identified characteristics of type is member of:
  - **from**
  - **to**

Conceptual Data Modeling Process
STEP 2: Model Identified Types

- model your types using a suitable conceptual data model (i.e., create conceptual data schema) and visualize it as a diagram
- you can use various tools for modeling, so-called **Case Tools**, e.g.,
  - commercial Enterprise Architect, IBM Rational Rose
  - academic eXolutio

Conceptual Data Modeling Process
STEP 2.1: Choose suitable modeling language

- there are various languages for modeling conceptual data schemas
  - each is associated with a well-established visualization in diagrams
- in this lecture, you will see the model UML class diagrams (shortly **UML**) and Entity-Relationship model (shortly **ER**)
- there are also others (out of scope of this lecture)
  - Object Constraints Language (OCL)
  - Object-Role Model (ORM)
  - Web Ontology Language (OWL)
  - Predicate Logic
  - Description Logic

used in practice

rarely used in practice,
used to define formal background
and properties of modeling languages

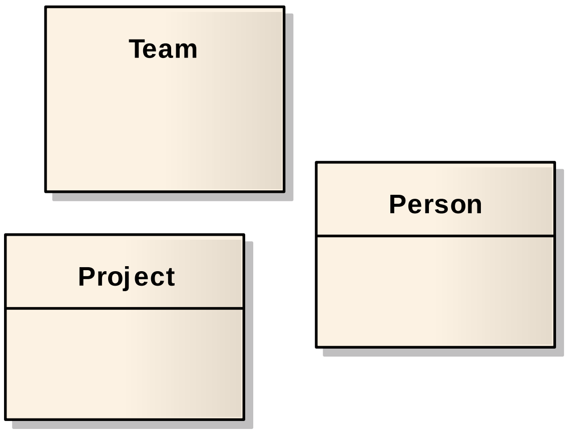
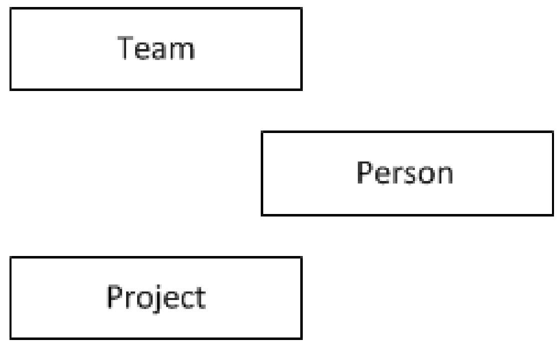
Conceptual Data Modeling Process
STEP 2.2: Create conceptual schema

- express your identified types of entities, relationships and their characteristics with constructs offered by the selected conceptual modeling language
- **UML:** classes, associations, attributes
- **ER:** entity types, relationship types, attributes
  - ER is more oriented to **data design**
  - UML is more oriented to **code design**
    - but is suitable for data design as well (wide scope language)
  - both used in practice,
    UML has became more popular in recent years

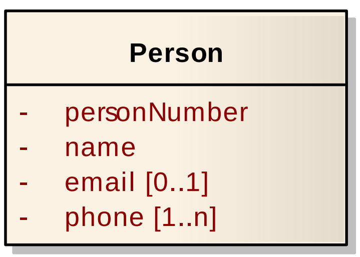
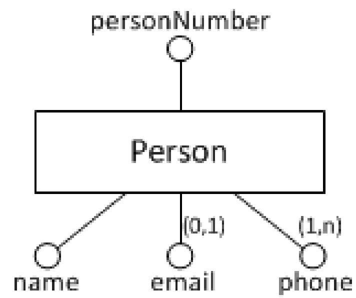
Conceptual Data Modeling Process ER vs. UML

- ER model
  - not standardized, various notations, various extensions (e.g., IS-A hierarchy)
- UML
  - family of models, e.g., **class diagrams**, use case diagrams, state diagrams,
  - standardized by OMG (object management group), current version UML 2.5 (Jul 2015)

UML and ER Basic Constructs
How to model types of entities?

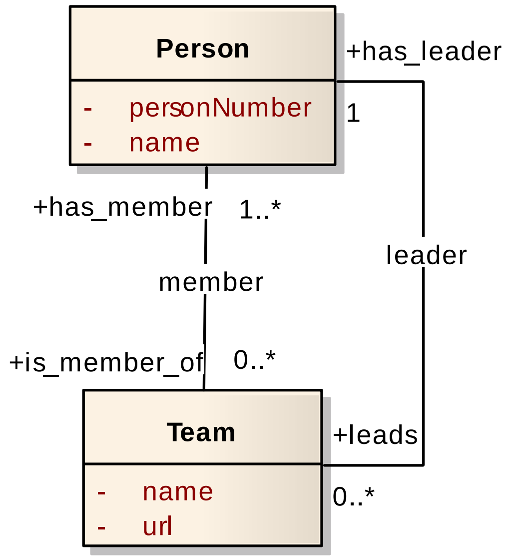
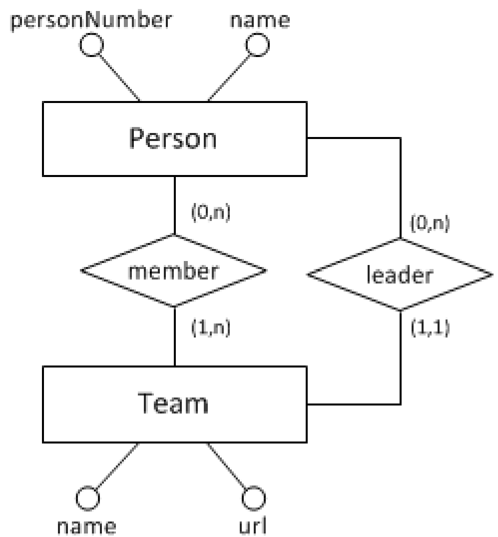
UML	ER	Real-world
		Real-world persons, research teams and research projects.
UML construct: **Class** • name	ER construct: **Entity type** • name	Type of real-world entities

UML and ER Basic Constructs
How to model characteristics of entities?

<table>
<tr><th>UML</th><th>ER</th><th>Real-world</th></tr>
<tr>
<td>
</td>
<td>
</td>
<td>A person is characterized by its personal number, name, optional email and one or more phone numbers.</td>
</tr>
<tr>
<td>UML construct:
Attribute of class
name
cardinality</td>
<td>ER construct:
Attribute of entity type
name
cardinality</td>
<td>Characteristics of a type of real-world entities</td>
</tr>
</table>

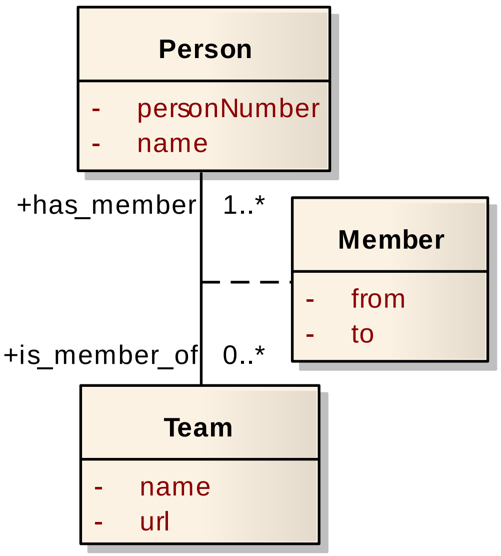
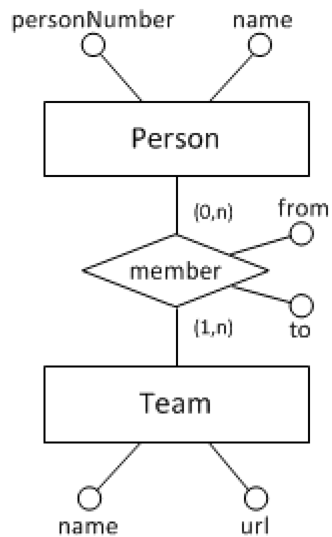
UML and ER Basic Constructs

How to model types of relationships?

<table>
<tr><th>UML</th><th>ER</th><th>Real-world</th></tr>
<tr>
<td>
</td>
<td>
</td>
<td>A team has one or more members. A person can be a member of zero or more teams.

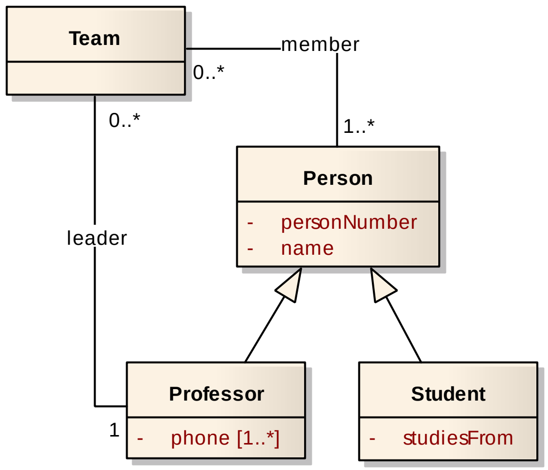
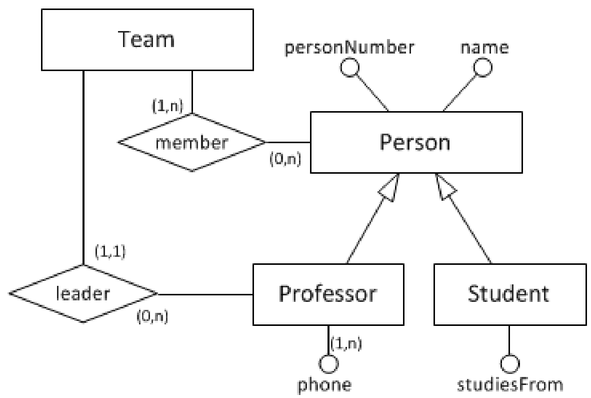
A team has exactly one leader. A person can be a leader of zero or more teams.</td>
</tr>
<tr>
<td>UML construct:
Binary association
• optional name
• two participants with optional name and cardinality</td>
<td>ER construct:
Binary relationship type
• name
• two participants with cardinality</td>
<td>A type of relationship between two real-world entities.</td>
</tr>
</table>

UML and ER Basic Constructs
How to model characteristics of relationships?

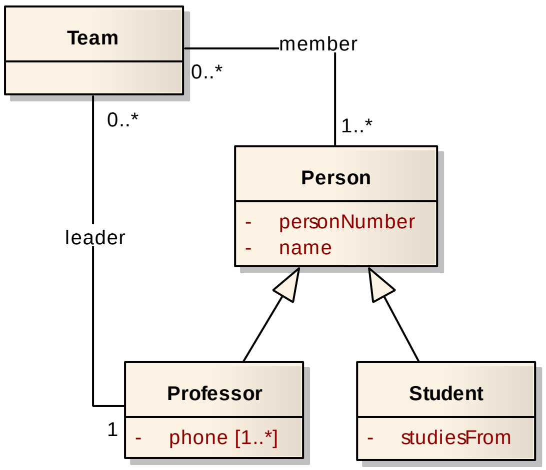
<table>
<tr><th>UML</th><th>ER</th><th>Real-world</th></tr>
<tr>
<td>
</td>
<td>
</td>
<td>A person is a team member in a given time period.</td>
</tr>
<tr>
<td>UML construct:
Attribute of binary association class
combination of class and binary association</td>
<td>ER construct:
Attribute of relationship type
name
cardinality</td>
<td>Characteristics of a type or relationships between two real-world entities.</td>
</tr>
</table>

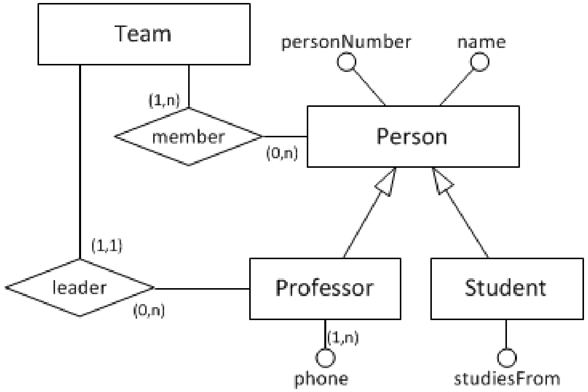
UML and ER Basic Constructs

How to model generalizations/specializations?

<table>
<tr><th>UML</th><th>ER</th><th>Real-world</th></tr>
<tr>
<td>
</td>
<td>
</td>
<td>A professor is a person which, in addition to a number and name, has one or more phones and can lead teams.
A student is a person which, in addition to a number and name, has a data from which (s)he studies.</td>
</tr>
<tr>
<td>Construct:
Specific kind of binary association called generalization
no name, roles and cardinalities</td>
<td>Construct:
IS-A relationship
no name and cardinalities</td>
<td>A type of real-world entities which is a specialization of another type.</td>
</tr>
</table>

UML and ER Basic Constructs
How to model generalizations/specializations?





Additional constraints
(independently of the modeling language used):

Covering Constraint:

$$Professors \cup Students = Persons$$

- each person is either professor or student

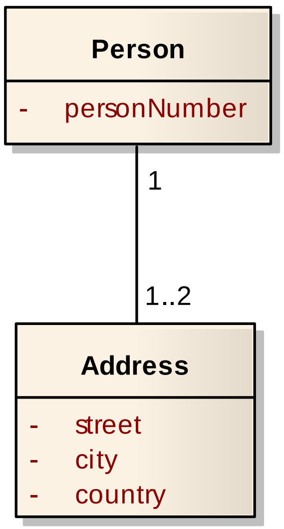
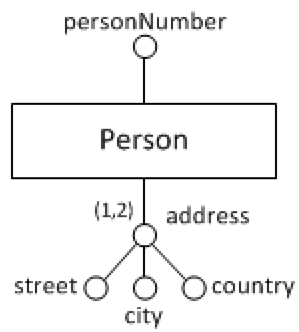
Overlap Constraint:

$$Professors \cap Students = \emptyset$$

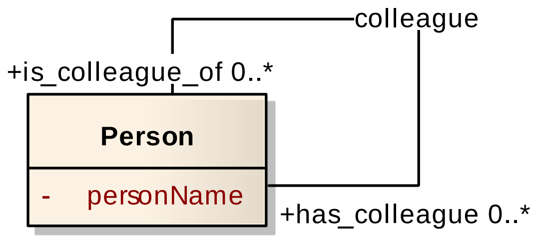
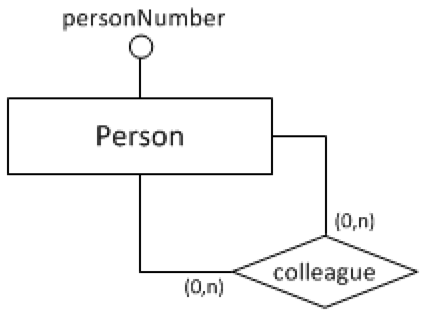
- there is no person which is both professor and student

UML and ER Advanced Constructs

Composite attributes

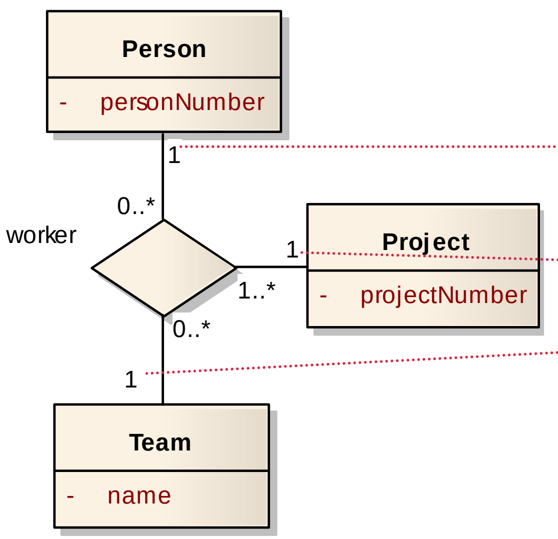
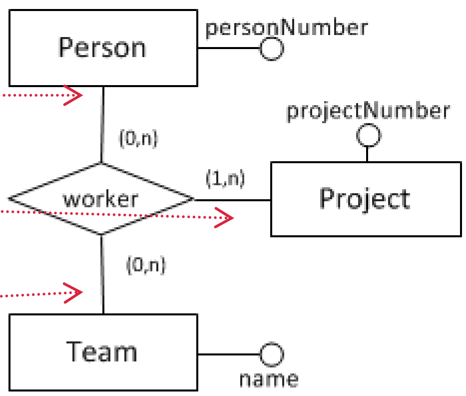
<table>
<tr><th>UML</th><th>ER</th><th>Real-world</th></tr>
<tr>
<td>
</td>
<td>
</td>
<td>A person has one or two addresses comprising street, city and country.</td>
</tr>
<tr>
<td>UML construct:
No specific construct; composite attributes can be expressed with an auxiliary class.</td>
<td>ER construct:
Composite attribute
• name
• cardinality
• sub-attributes</td>
<td></td>
</tr>
</table>

UML and ER Advanced Constructs
Recursive associations

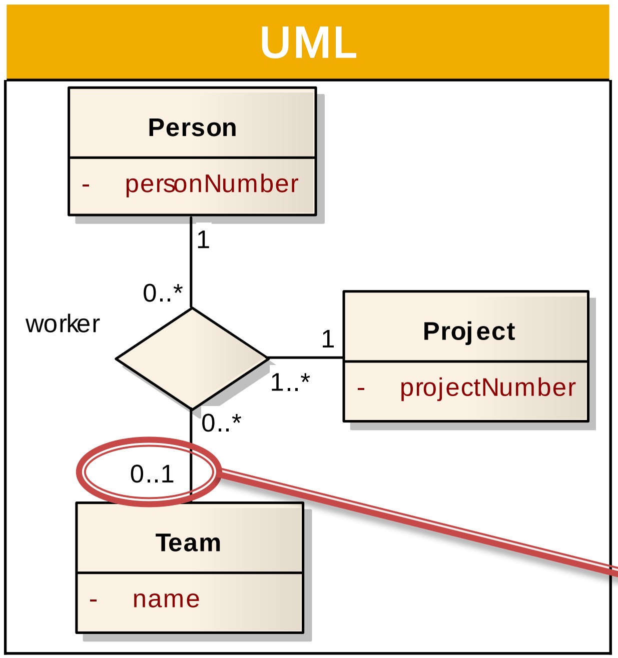
UML	ER	Real-world
		A person has zero or more colleagues.
UML construct: Normal association with the same participants.	ER construct: Normal relationship type with the same participants.	

UML and ER Advanced Constructs

N-ary associations

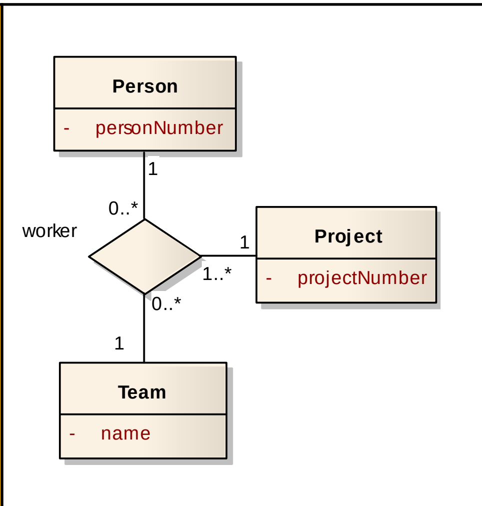
<table>
<tr><th>UML</th><th>ER</th><th>Real-world</th></tr>
<tr>
<td>
</td>
<td>
</td>
<td>A person works on a project but only as a team member.</td>
</tr>
<tr>
<td>Construct:
N-ary association
Similar to binary association but with three or more participants.</td>
<td>Construct:
N-ary relationship type
Similar to binary relationship type but with three or more participants.</td>
<td>NOTE: Attributes can be expressed in the same way as for binary variants.</td>
</tr>
</table>

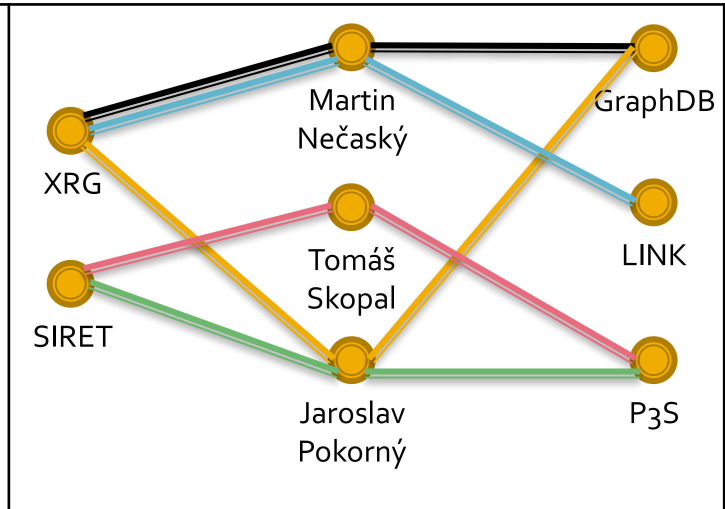
UML and ER Advanced Constructs
N-ary associations



- UML n-ary associations have stronger expressive power in their cardinalities than ER n-ary relationship types
- A person can also work on a project as an individual person without a relationship to a team.

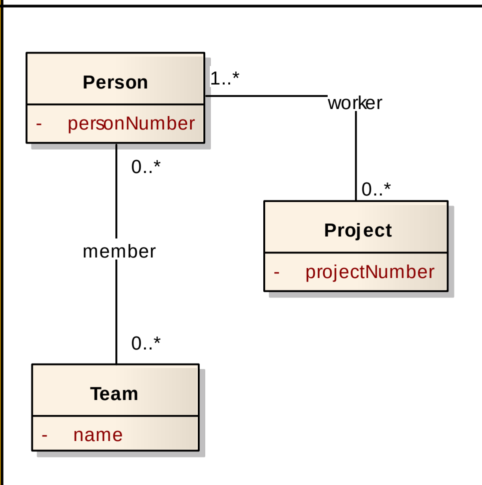
UML and ER Advanced Constructs
N-ary associations vs. binary associations

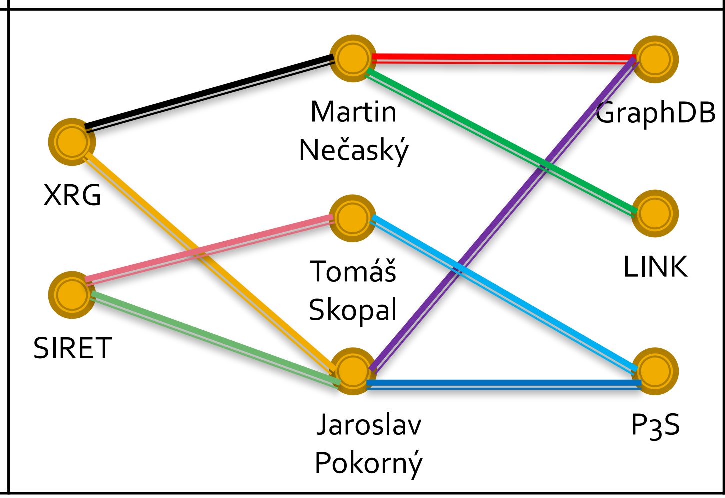




Which projects does JP work on as a member of SIRET?

P3S.



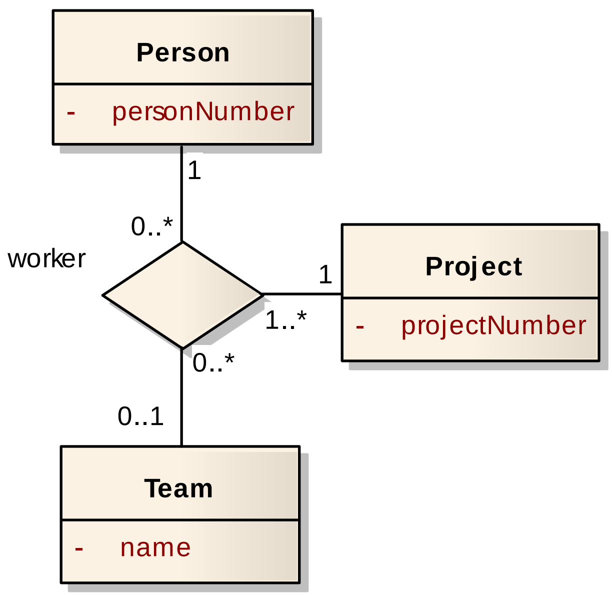


Which projects does JP work on as a member of SIRET?

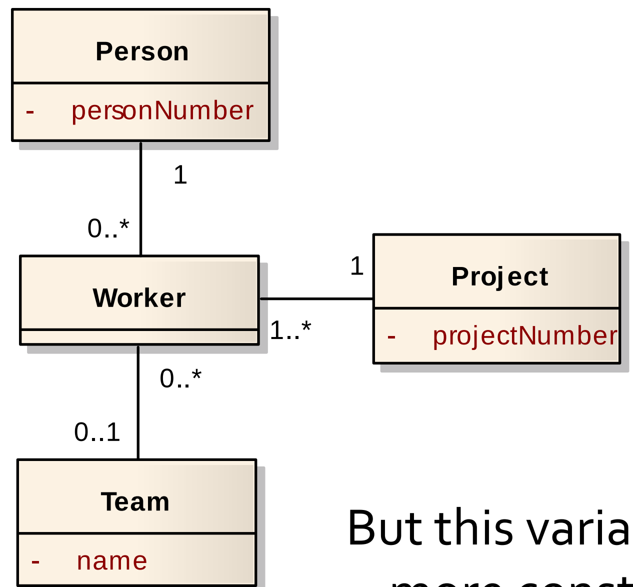
GraphDB or P3S?

UML and ER Advanced Constructs
N-ary associations vs. binary associations

- n-ary association = class + separate binary association for each original participant



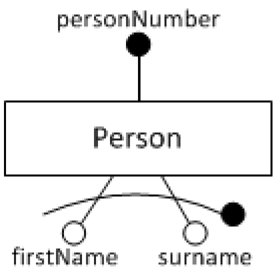
=



But this variant uses more constructs

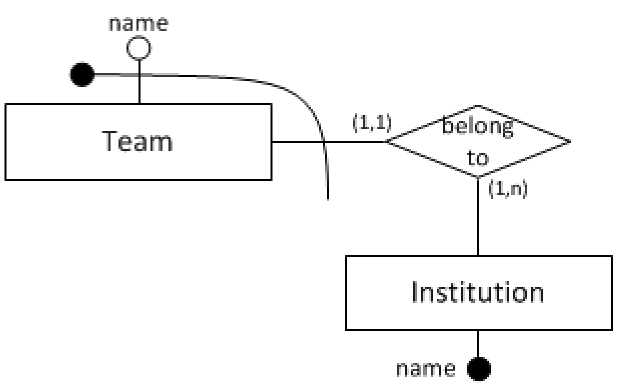
UML and ER Advanced Constructs

Identifiers

UML	ER	Real-world
		A person is identified by its personal number or by a combination of its first name and surname. $(\forall p_1, p_2 \in Persons)$ $(p_1.personNumber = p_2.personNumber \rightarrow p_1 = p_2)$ $(\forall p_1, p_2 \in Persons)$ $\left(\begin{array}{c} (p_1.firstName = p_2.firstName \ \wedge \\ p_1.surname = p_2.surname) \rightarrow p_1 = p_2 \end{array} \right)$
n/a	**Construct:** Attribute or a group of attributes marked as **identifier**.	

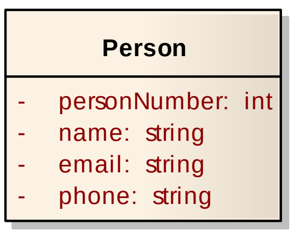
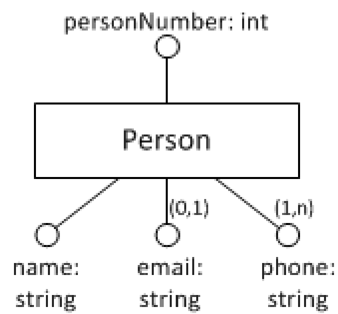
UML and ER Advanced Constructs

Weak entity types

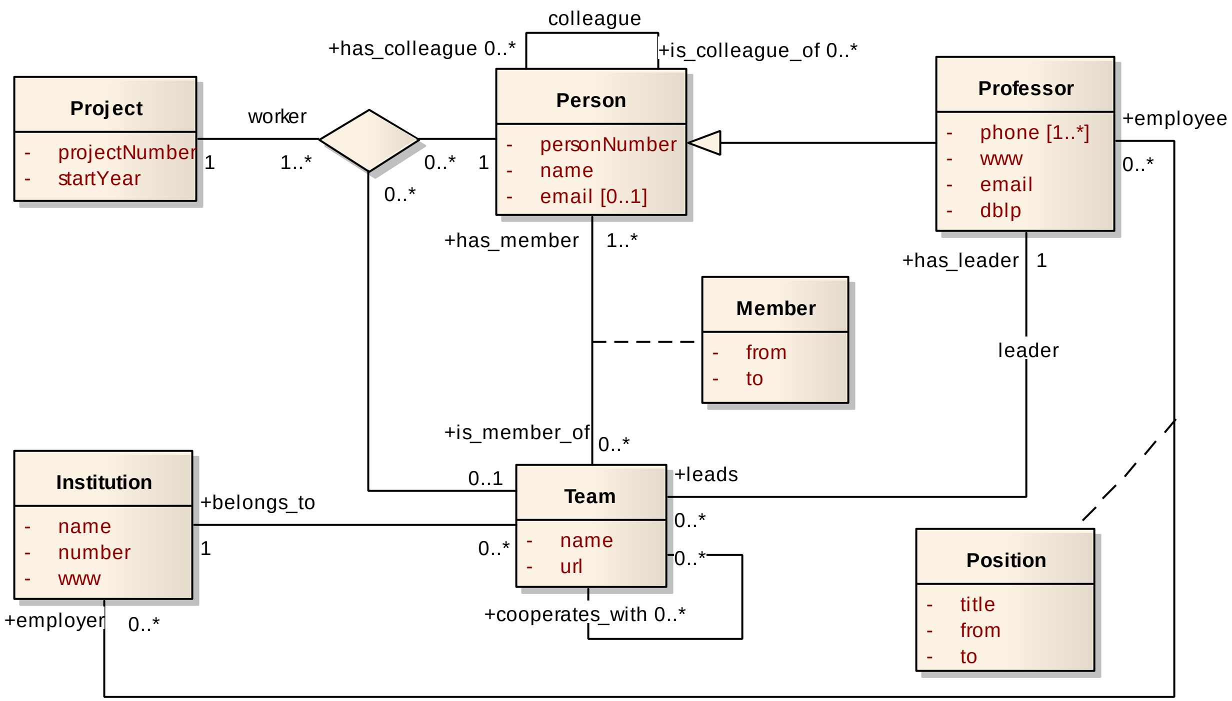
UML	ER	Real-world
		A team is identified by a combination of its name and a name of its institution. $$(\forall t_1, t_2 \in Teams)\left(\begin{array}{c}(t_1.Institution.name = t_2.Institution.name\ \wedge \\ t_1.name = t_2.name) \rightarrow t_1 = t_2\end{array}\right)$$
n/a	**Construct:** Weak entity type = entity type which participates in a relationship type with card. (1,1) and the relationship is a part of its identifier.	

UML and ER Advanced Constructs

Data types

UML	ER	Real-world
		A person has a person number which is an integer and name, email and phone which are strings.
Construct: Attribute of class may have a data type.	**Construct:** Attribute of entity type may have a data type.	NOTE: 1. Set of data types not specified strictly. 2. Data types are not very important at the conceptual layer.

Complete Example in UML



Object Constraint Language (OCL)

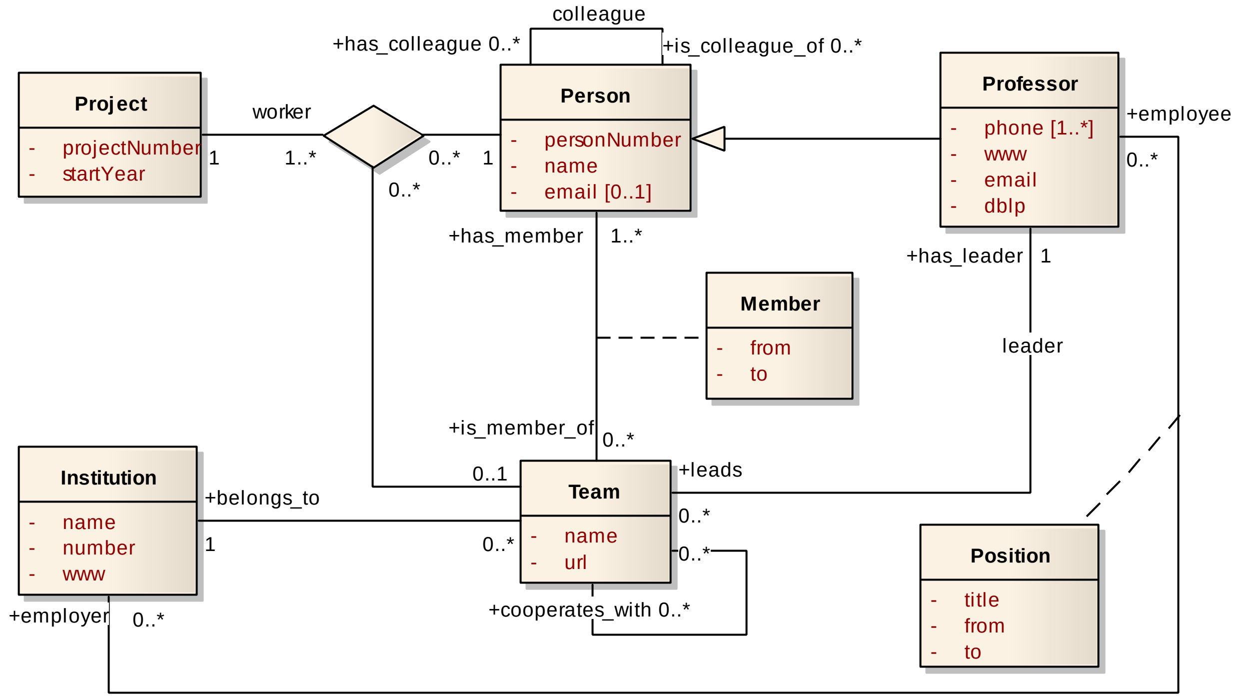
- language for formal specification of advanced integrity constraints
- supports invariants, derived values, method pre- and post-conditions, etc.
- we focus on invariants

```
context variable : Class inv
    ... constraint ...
```

- constraint
  - variables (**`t : Team`**)
  - navigation paths in your conceptual schema (**`t.has_leader.employer`**)
    1. Assigns a team to **`t`**
    2. From **`t`** it goes to the associated collection of leaders of **`t`** (the set contains only one leader because of the cardinality 1)
    3. From the set of leaders it goes to the associated collection of all employers (the set contains zero or more employers of the leader because of the cardinality 0..*)
    4. NOTE: The result of navigation (".") is always a collection
  - logical operators (**`and, or, implies`**)
  - operators on (collection of) instances

```
t.has_leader.employer->size() > 0 //size of a set
t.Project.startYear->min() //the first project of t
t.has_leader.employer->forAll(e | e.www->size()>0)
                                        //size of a string (overloaded)
```

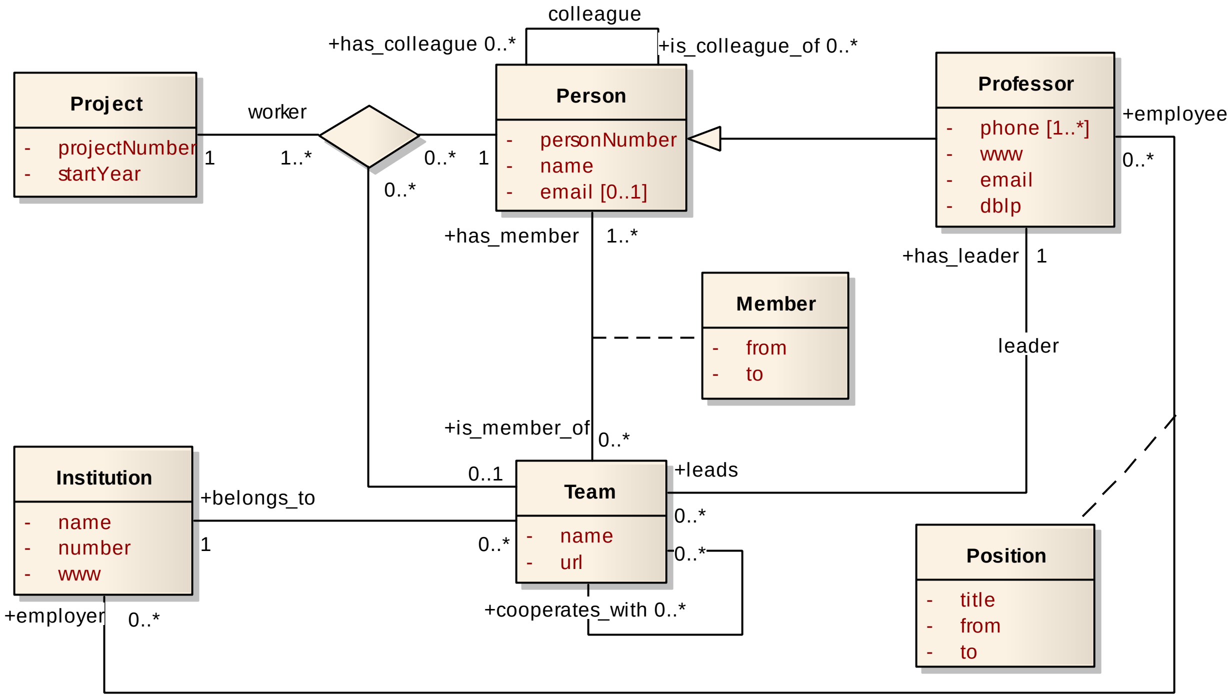
Object Constraint Language (OCL)




```
context p : Project inv
          p.startYear > 1990
```

"Each project must start after 1990."

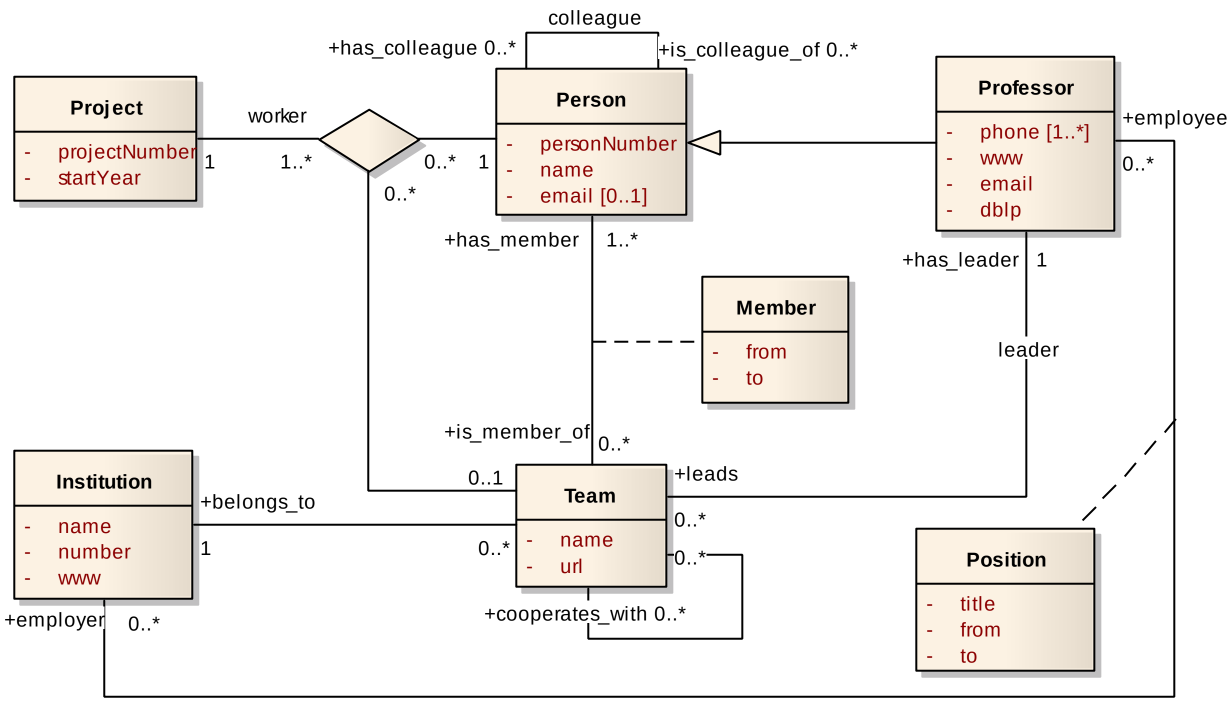
Object Constraint Language (OCL)




```
context Team inv
        self.has_member->size() > 10 implies
            self.worker->size() > 0
```

"Each team with more than 10 members must have a project."

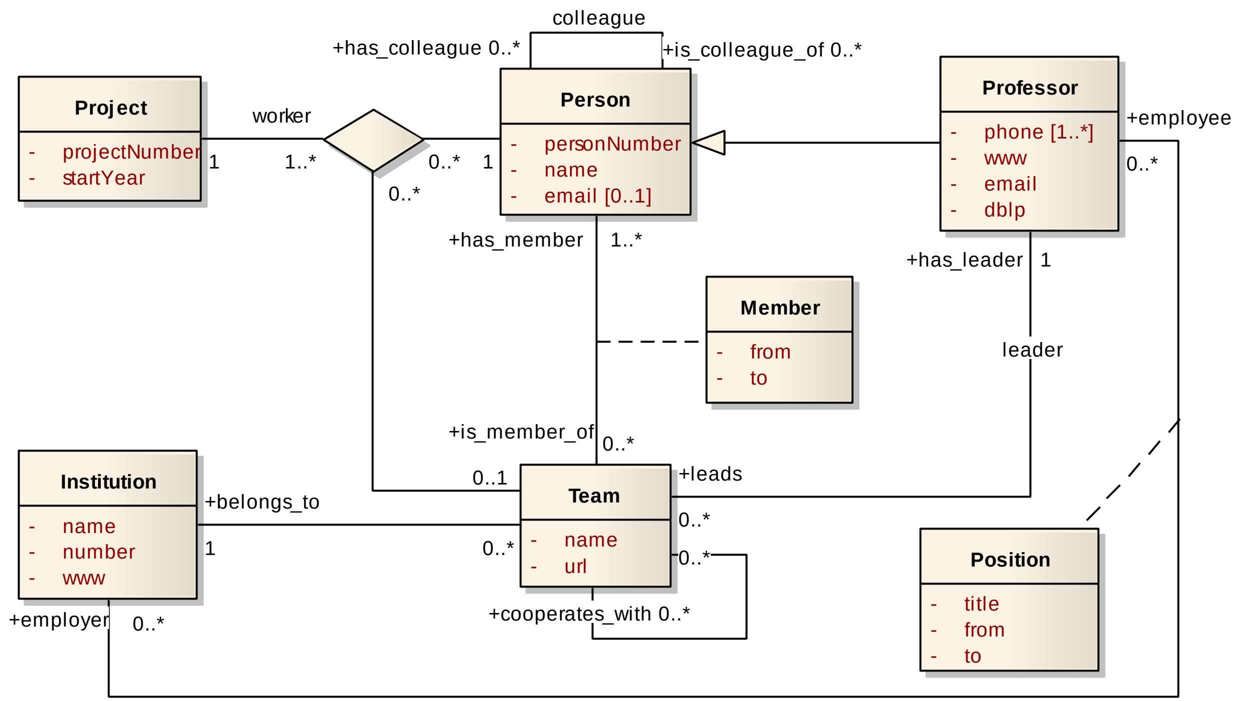
Object Constraint Language (OCL)




```
context p1,p2 : Person inv
        p1.personNumber = p2.personNumber implies p1 = p2
```

"A person is identified by its personal number."

Object Constraint Language (OCL)




```
context t : Team inv
        t.belongs_to.employee->exists(p | p = t.has_leader)
```

"A team leader must be an employee of the institution of the team."

OCL Invariants

- Invariants are defined in the context of a specific type, named the context type of the constraint.
- Its body, the boolean condition to be checked, must be satisfied by all instances of the context type.
- Invariants can:
  - restrict the value of single objects, like
    - ```
      context p : Project inv
                p.startYear > 1990
      ```
    - ```
      context p : Project inv newerProjectsOnly:
                p.startYear > 1990
      ```
- The *self* variable represents an arbitrary instance of the *Project* class and the dot notation is used to access the properties of the *self* object
- All instances of *Project* class must evaluate this condition to true.

OCL Invariants

- Many invariants express more complex conditions limiting the possible relationships between different objects in the system, usually related through association links.
  - For instance
    -
```
context t : Team inv
       t.belongs_to.employee->exists(
              p | p = t.has_leader
              )
```
- Conjuctions
  - not, and, or, xor, implies
  - Boolean comparisons =, <>

OCL Invariants - Boolean

- Three values: null, false, true
- Unary operator not(b)

b	**not(b)**
`null`	`null`
`false`	`true`
`true`	`false`

OCL Invariants - Boolean

- Three values: null, false, true
- Binary operators

a	b	a and b	a or b	a xor b	a implies b	a=b	a<>b
null	null	null	null	null	null	true	false
null	false	false	null	null	null	false	true
null	true	null	true	null	true	false	true
false	null	false	null	null	true	false	true
false	false	false	false	false	true	true	false
false	true	false	true	true	true	false	true
true	null	null	true	null	null	false	true
true	false	false	true	true	false	false	true
true	true	true	true	false	true	true	false

OCL Invariants – Navigation

- Navigation
  - Dot followed by data member
    - Provides – in general – set of values typed by the type of the data member
    - If the set has cardinality 1, it can be maintained as a single value
  - Dot followed by the opposite role name of some association
    - Provides – in general – set of instances of the class on the denoted side of association
    - If the set has cardinality 1, it can be maintained as a single instance
  - For instance – assume t of type Team
    - `t.name` ... String (set of Strings with cardinality one)
    - `t.has_leader` ... *Professor* (set of *Professor* instances with cardinality one)
    - `t.has_member` ... set of *Person* instances
    - `t.belongs_to` ... *Institution* (set of *Institution* instances with cardinality one)
    - `t.belongs_to.employee` ... set of *Professor* instances

OCL Invariants – Set operators

- Uses arrow notation set->operator(...)
- Single values and single instances can be understood as sets
  - `->size()` ... Integer, size of the set
  - `->min()` ... minimal value from the set of values
  - `->max()` ... minimal value from the set of values
  - `->forAll(x | condition)`
    ... big quantifier – all elements in the set must meet the condition
  - `->exists(x | condition)`
    ... small quantifier – at least one element in the set must meet the condition

OCL Invariants

- Navigation
  - Dot followed by data member
    - Provides – in general – set of values typed by the type of the data member
    - If the set has cardinality 1, it can be maintained as a single value
  - Dot followed by the opposite role name of some association
    - Provides – in general – set of instances of the class on the denoted side of association
    - If the set has cardinality 1, it can be maintained as a single instance
  - For instance – assume t of type Team
    - `t.name` ... String (set of Strings with cardinality one)
    - `t.has_leader` ... *Professor* (set of *Professor* instances with cardinality one)
    - `t.has_member` ... set of *Person* instances
    - `t.belongs_to` ... Institution (set of *Institution* instances with cardinality one)
    - `t.belongs_to.employee` ... set of *Professor* instances

Software for practices

- ER
  - ERtoS
- UML
  - trial version of Enterprise architect
    - full version available in course NSWI041
  - trial version of Rational Rose
  - eXolutio