Introduction to Databases
Exam of 29/01/2020
Solutions

Diego Calvanese

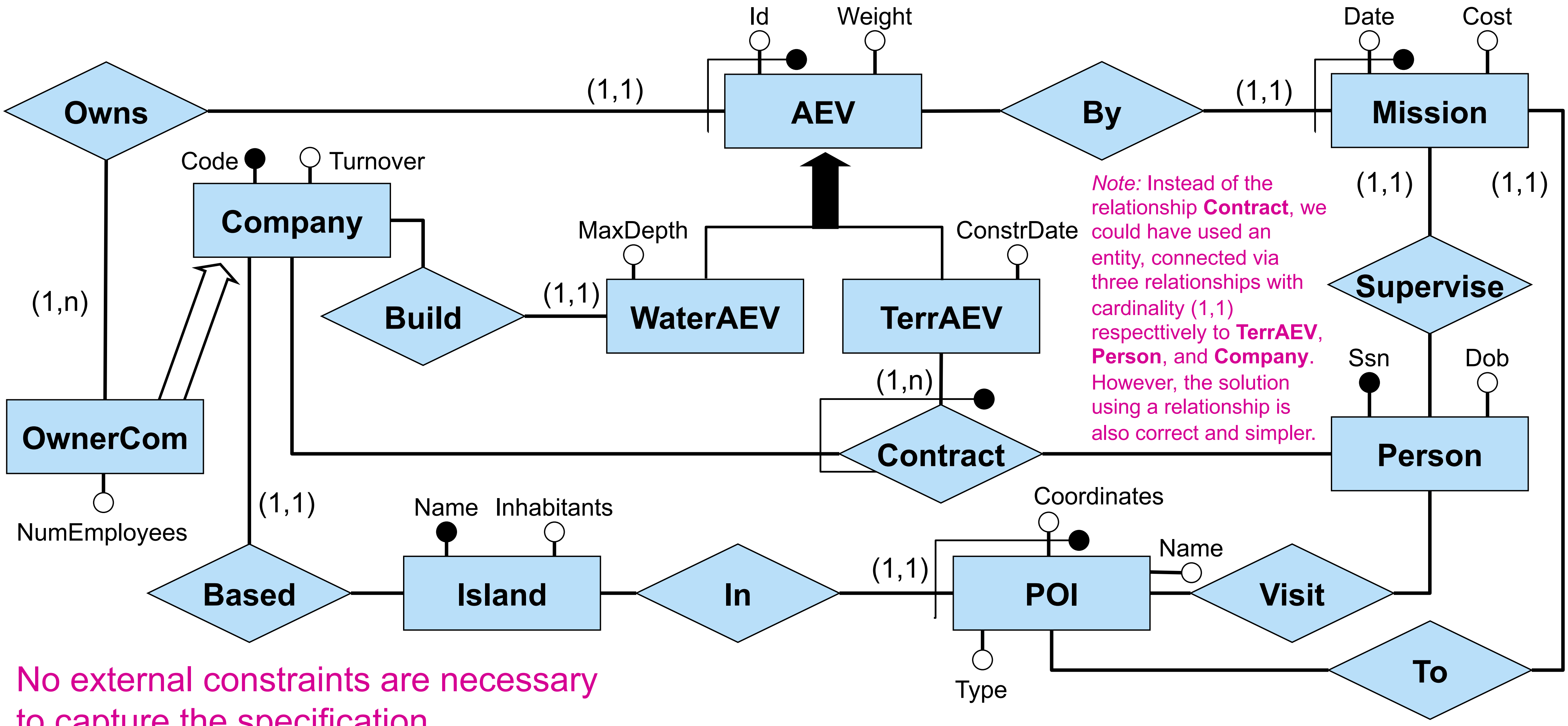
Bachelor in Computer Science
Faculty of Computer Science
Free University of Bozen-Bolzano

http://www.inf.unibz.it/~calvanese/teaching/exams/idb/

Problem 1

Design the Entity Relationship schema of an information system relating to missions of autonomous exploration vehicles (AEVs) owned by companies. Of each **company**, we are interested in the code (identifier), the turnover, and the island where the company is based. Of each company that owns AEVs, we are also interested in the number of employees. Of each **AEV**, we are interested in the company that owns it, the id (unique within the company that owns it), and the weight. There are exactly two types of AEVs: underwater and terrestrial ones. Of each underwater AEV, we are interested in the company that built it and the maximum depth at which it can dive. Of each terrestrial AEV, we are interested in the date of construction and in the contracts for its maintenance (at least one). Every **maintenance contract** for a terrestrial AEV is characterized by the company that carries out the maintenance and by the person that is responsible for it. Given an AEV and a company that has a contract for its maintenance, there is exactly one person responsible for that contract. Of each **mission**, we are interested in the AEV that performed it, the date, the cost, the person who supervised it, and the point of interest (POI) to which the AEV must travel to carry out the mission itself. Note that an AEV can carry out at most one mission per day. Of each **POI** we are interested in its type (e.g., flatland, mountain, underwater, etc.), its name, the island to which it belongs, and the coordinates within the island. There are no two different POIs with the same coordinates belonging to the same island. Of each **island**, we are interested in the name (identifier) and the number of inhabitants. Of each **person**, we are interested in the ssn (identifier), the date of birth, and the POIs the person has visited.

Problem 1: Conceptual schema – Diagram



Note: Instead of the relationship **Contract**, we could have used an entity, connected via three relationships with cardinality (1,1) respecttively to **TerrAEV**, **Person**, and **Company**. However, the solution using a relationship is also correct and simpler.

No external constraints are necessary to capture the specification.

Problem 2

Carry out the logical design of the database, producing the complete relational schema with constraints, taking into account the following indications:

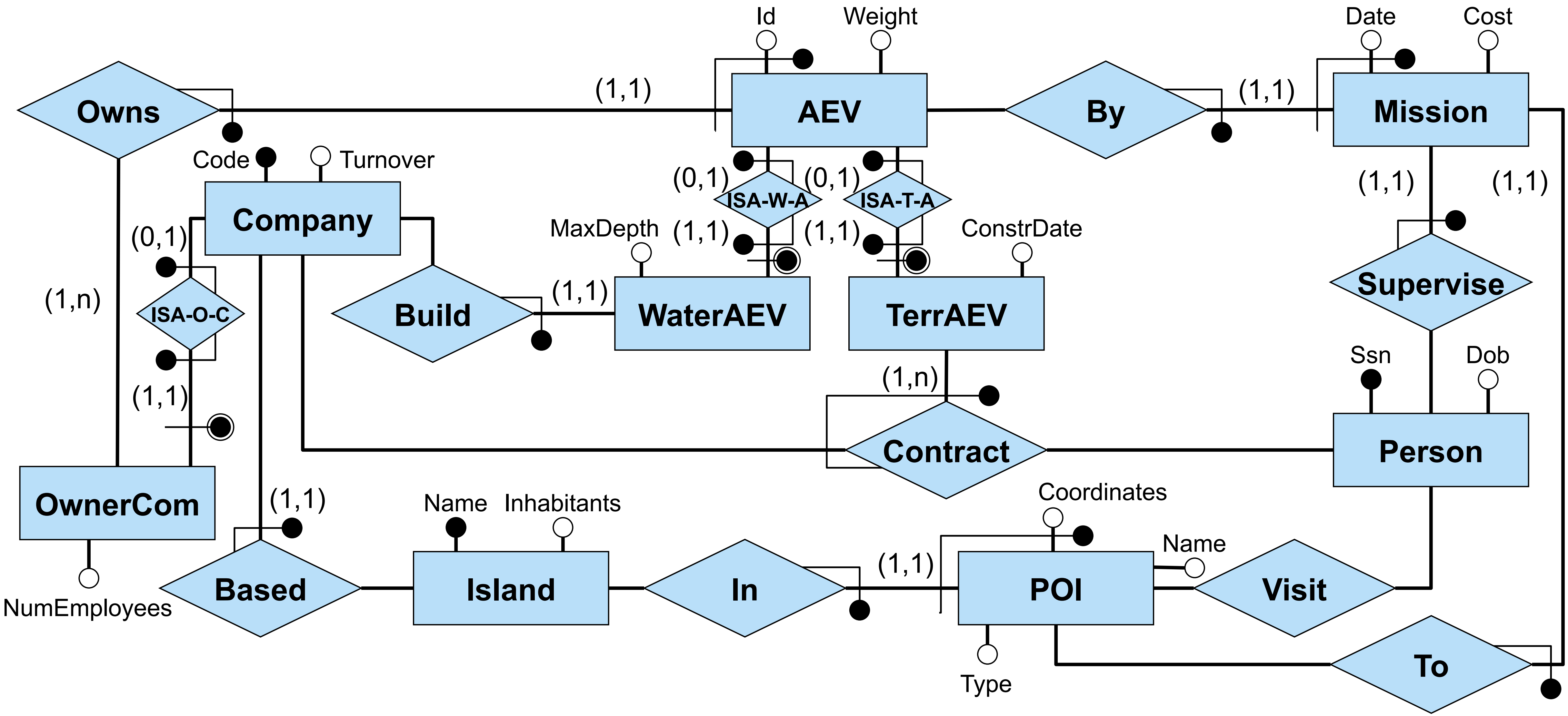
1. When accessing the data of a mission, we always want to know the person who has supervised it.
2. Terrestrial AEVs are accessed separately from underwater AEVs.
3. When accessing a (terrestrial or underwater) AEV, we are always interested in knowing its weight.

As steps in your design you should produce:

- the restructured ER schema (possibly with external constraints),
- the direct translation into the relational model (possibly with external constraints), and
- the restructured relational schema (again with constraints).

Motivate explicitly how the above indications affect your design.

Problem 2: Restructured conceptual schema



Problem 2: Restructured conceptual schema – External constraints

External Constraint: We only need to introduce the generalization constraint resulting from the elimination of the complete hierarchy:

For each instance ***I*** of the conceptual schema, each instance of **AEV** participates either to **ISA-W-A** or to **ISA-T-A**, but not to both.

Problem 2: Direct translation (1/3)

AEV(<u>Id</u>, <u>Owner</u>, Weight)
  foreign key: AEV[Owner] $\subseteq$ OwnerCompany[Code]
  generalization constraint: AEV[Id,Owner] $\subseteq$ WaterAEV[Id,Owner] $\cup$ TerrAEV[Id,Owner]

WaterAEV(<u>Id</u>, <u>Owner</u>, MaxDepth)
  foreign key: WaterAEV[Id,Owner] $\subseteq$ AEV[Id,Owner]
  inclusion: WaterAEV[Id,Owner] $\subseteq$ Build[AEV,Owner]

TerrAEV(<u>Id</u>, <u>Owner</u>, ConstrDate)
  foreign key: TerrAEV[Id,Owner] $\subseteq$ AEV[Id,Owner]
  inclusion: TerrAEV[Id,Owner] $\subseteq$ Contract[TerrAEV,Owner]
  generalization constraint: WaterAEV[Id,Ownder] $\cap$ TerrAEV[Id,Owner] = $\emptyset$

Company(<u>Code</u>, Turnover)
  foreign key: Company[Code] $\subseteq$ Based[Company]

OwnerCompany(<u>Code</u>, NumEmployees)
  foreign key: OwnerCompany[Code] $\subseteq$ Company[Code]
  inclusion: OwnerCompany[Code] $\subseteq$ AEV[Owner]

Problem 2: Direct translation (2/3)

Build(<u>AEV</u>, <u>Owner</u>, Builder)
  foreign key: Build[AEV,Owner] $\subseteq$ WaterAEV[Id,Owner]
  foreign key: Build[Builder] $\subseteq$ Company[Code]

Mission(<u>Date</u>, <u>AEV</u>, <u>Owner</u>, Cost)
  foreign key: Mission[AEV,Owner] $\subseteq$ AEV[Id,Owner]
  foreign key: Mission[Date,AEV,Owner] $\subseteq$ To[Mission,AEV,Owner]
  foreign key: Mission[Date,AEV,Owner] $\subseteq$ Supervise[Mission,AEV,Owner]

Person(<u>Ssn</u>, Dob)

Supervise(<u>Mission</u>, <u>AEV</u>, <u>Owner</u>, Person)
  foreign key: Supervise[Mission,AEV,Owner] $\subseteq$ Mission[Date,AEV,Owner]
  foreign key: Supervise[Person] $\subseteq$ Person[Ssn]

Island(<u>Name</u>, Inhabitants)

POI(<u>Coordinates</u>, <u>Island</u>, Name, Type)
  foreign key: POI[Island] $\subseteq$ Island[Name]

Problem 2: Direct translation (3/3)

To(Mission, AEV, Owner, POI, Island)
- foreign key: To[Mission,AEV,Owner] $\subseteq$ Mission[Date,AEV,Owner]
- foreign key: To[POI,Island] $\subseteq$ POI[Coordinates,Island]

Visit(Person, POI, Island)
- foreign key: Visit[Person] $\subseteq$ Person[Ssn]
- foreign key: Visit[POI,Island] $\subseteq$ POI[Coordinates,Island]

Based(Company, Island)
- foreign key: Based[Company] $\subseteq$ Company[Code]
- foreign key: Based[Island] $\subseteq$ Island[Name]

Contract(TerrAEV, Owner, MaintCompany, Person)
- foreign key: Contract[TerrAEV,Owner] $\subseteq$ TerrAEV[Id,Owner]
- foreign key: Contract[MaintCompany] $\subseteq$ Company[Code]
- foreign key: Contract[Person] $\subseteq$ Person[Ssn]

Problem 2: Restructuring of the relational schema

1. When accessing the data of a mission, we always want to know the person who has supervised it.
2. Terrestrial AEVs are accessed separately from underwater AEVs.
3. When accessing a (terrestrial or underwater) AEV, we are always interested in knowing its weight.

We take into account the above indications in the following way:

- We take into account indication 1 by merging **Mission** with **Supervise**.
- We take into account indications 2 and 3 by first performing a horizontal partition of **AEV** into two relations containing the sets of tuples that represent respectively the underwater and the terrestrial AEVs, and then merging these two relations respectively with **WaterAEV** and **TerrAEV**.
- The foreign keys towards **AEV** in **WaterAEV** and **TerrAEV** disappear, and all other foreign keys and inclusions towards **AEV** become constraints of inclusion in the union of **WaterAEV** and **TerrAEV**.

Problem 2: Restructured relational schema (1/3)

WaterAEV(Id, Owner, MaxDepth, Weight)
- foreign key: WaterAEV[Owner] ⊆ OwnerCompany[Code]
- inclusion: WaterAEV[Id,Owner] ⊆ Build[AEV,Owner]

TerrAEV(Id, Owner, ConstrDate, Weight)
- foreign key: TerrAEV[Owner] ⊆ OwnerCompany[Code]
- inclusion: TerrAEV[Id,Owner] ⊆ Contract[TerrAEV,Owner]
- generalization constraint: WaterAEV[Id,Owndet] ∩ TerrAEV[Id,Owner] = ∅

Company(Code, Turnover)
- foreign key: Company[Code] ⊆ Based[Company]

OwnerCompany(Code, NumEmployees)
- foreign key: OwnerCompany[Code] ⊆ Company[Code]
- inclusion: OwnerCompany[Code] ⊆ WaterAEV[Owner] ∪ TerrAEV[Owner]

Problem 2: Restructured relational schema (2/3)

Build(<u>AEV</u>, <u>Owner</u>, Builder)
- foreign key: Build[AEV,Owner] $\subseteq$ WaterAEV[Id,Owner]
- foreign key: Build[Builder] $\subseteq$ Company[Code]

Mission(<u>Date</u>, <u>AEV</u>, <u>Owner</u>, Cost, Supervisor)
- foreign key: Mission[AEV,Owner] $\subseteq$ WaterAEV[Id,Owner] $\cup$ TerrAEV[Id,Owner]
- foreign key: Mission[Date,AEV,Owner] $\subseteq$ To[Mission,AEV,Owner]
- foreign key: Mission[Supervisor] $\subseteq$ Person[Ssn]

Person(<u>Ssn</u>, Dob)

Island(<u>Name</u>, Inhabitants)

POI(<u>Coordinates</u>, <u>Island</u>, Name, Type)
- foreign key: POI[Island] $\subseteq$ Island[Name]

Problem 2: Restructured relational schema (3/3)

To(<u>Mission</u>, <u>AEV</u>, <u>Owner</u>, POI, Island)
  foreign key: To[Mission,AEV,Owner] ⊆ Mission[Date,AEV,Owner]
  foreign key: To[POI,Island] ⊆ POI[Coordinates,Island]

Visit(<u>Person</u>, <u>POI</u>, <u>Island</u>)
  foreign key: Visit[Person] ⊆ Person[Ssn]
  foreign key: Visit[POI,Island] ⊆ POI[Coordinates,Island]

Based(<u>Company</u>, Island)
  foreign key: Based[Company] ⊆ Company[Code]
  foreign key: Based[Island] ⊆ Island[Name]

Contract(<u>TerrAEV</u>, <u>Owner</u>, <u>MaintCompany</u>, Person)
  foreign key: Contract[TerrAEV,Owner] ⊆ TerrAEV[Id,Owner]
  foreign key: Contract[MaintCompany] ⊆ Company[Code]
  foreign key: Contract[Person] ⊆ Person[Ssn]

The generalization constraint is not necessary anymore.

Problem 3

In a database, the relation `Lecture(teacher,student,day,month,year)` stores for each private lecture the ssn of the teacher, the ssn of the student, and the date when the lecture was held, while the relation `Person(ssn,age)` specifies ssn and age of persons (teachers and students).

Express in SQL the following queries over the above relations:

1. For each person, compute all persons to which that person has taught at least one private lecture in the years from 2016 to 2019.
2. For each person, compute to how many different persons who are underage (i.e., less than 18 years old) that person has taught private lectures in 2019.
3. Compute the ssn and the age of each person who has taught lectures exclusively from 2015 onwards, but only if the person has taught more than 50 lectures.

Problem 3: Solution (1/3)

```
Lecture(teacher,student,day,month,year)
Person(ssn,age)
```

1. For each person, compute all persons to which that person has taught at least one private lecture in the years from 2016 to 2019.

```
SELECT DISTINCT teacher, student
FROM Lecture
WHERE year >= 2016 AND year <= 2019
```

Problem 3: Solution (2/3)

```
Lecture(teacher,student,day,month,year)
Person(ssn,age)
```

2. For each person, compute to how many different persons who are underage (i.e., less than 18 years old) that person has taught private lectures in 2019.

```
SELECT teacher, COUNT (DISTINCT student)
  FROM Lecture JOIN Person ON student = ssn
  WHERE age < 18 AND year = 2019
  GROUP BY teacher
UNION
SELECT ssn AS teacher, 0
  FROM Person
  WHERE ssn NOT IN (SELECT teacher FROM Lecture JOIN Person ON student = ssn
                    WHERE age < 18 AND year = 2019)
```

Problem 3: Solution (3/3)

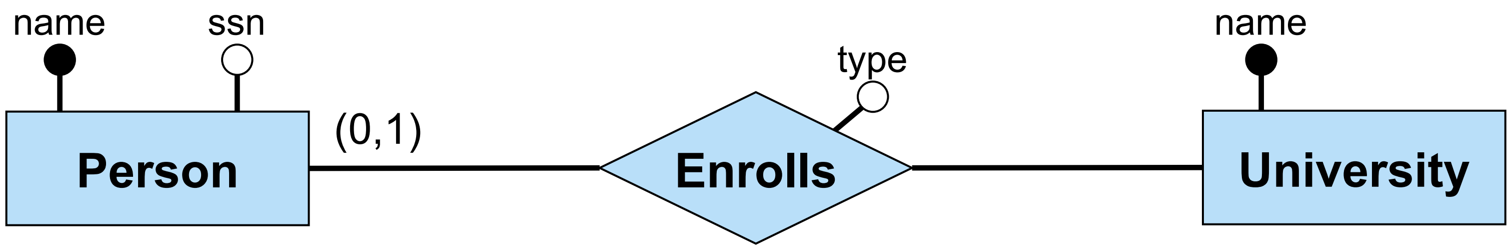
```
Lecture(teacher,student,day,month,year)
Person(ssn,age)
```

3. Compute the ssn and the age of each person who has taught lectures exclusively from 2015 onwards, but only if the person has taught more than 50 lectures.

```
SELECT ssn, age
FROM Person
WHERE ssn NOT IN (SELECT teacher FROM Lecture
                  WHERE year < 2015)
      AND (SELECT COUNT(*) FROM Lecture WHERE teacher = ssn) > 50
```

Problem 4

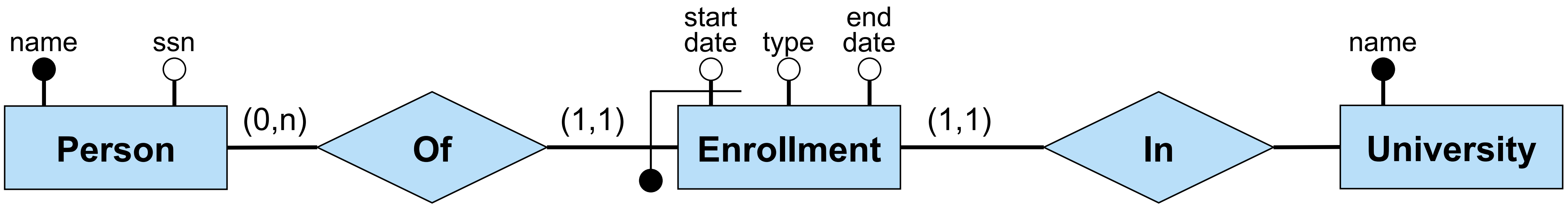
Consider the conceptual schema shown below, representing the fact that a Person is currently enrolled in a university, with the type of enrollment (full-time or part-time).



Suppose now that we want to record not only whether a person is currently enrolled in a university, but also the past enrollments, with their type, start date, and (expected) end date. Consider that a person can be enrolled in at most one university at a time. Restructure the conceptual schema above so that it represents this new state of affairs, and specify also the necessary external constraints (if needed).

Problem 4: Solution

We can restructure the schema according to the modelling pattern that allows for a historicized representation of schema elements. Since we have to historicize the relationship **Enrolls** and this modeling pattern applies to entities, we first have to transform the relationship into an entity. The resulting ER schema is as follows.



We have also to add the external constraint that in every instance ***I*** of the above schema, there are no e_1, e_2, p, sd_1, sd_2, and ed_2 such that *(1)* e_1 and e_2 in ***instances***(***I***,**Enrollment**), *(2)* p in ***instances***(***I***,**Person**), *(3)* (e_1,p) and (e_2,p) in ***instances***(***I***,**Of**), *(4)* (e_1,sd_1) and (e_2,sd_2) in ***instances***(***I***,**startdate**), *(5)* (e_2,ed_2) in ***instances***(***I***,**enddate**), and *(6)* $sd_2 < ed_1$ and $ed_1 < ed_2$.