Introduction to Databases

Exam of 09/02/2021

With Solutions

Diego Calvanese

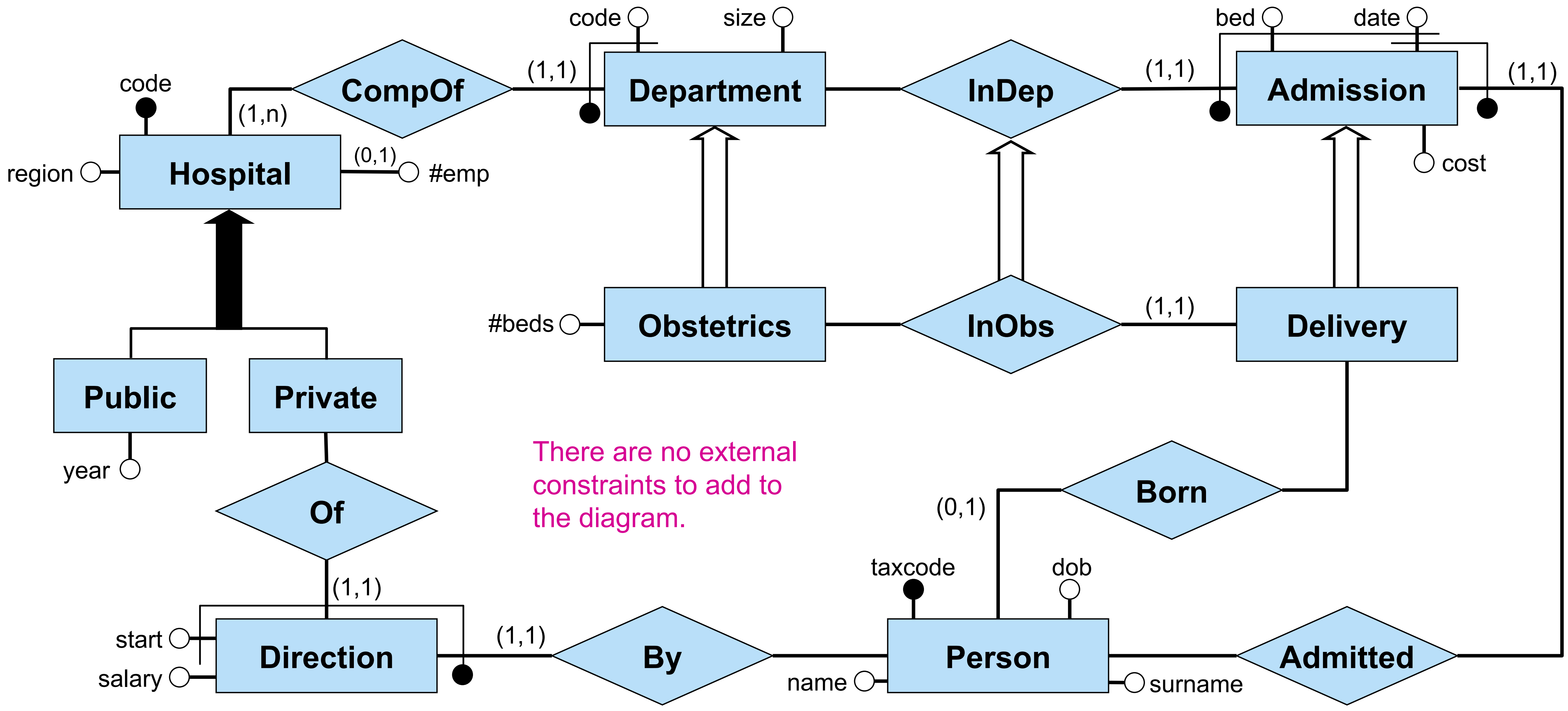
Bachelor in Computer Science
Faculty of Computer Science
Free University of Bozen-Bolzano

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

Problem 1

Design the Entity-Relationship schema of an information system relating to the hospitalisation of patients. Of each **hospital**, we are interested in the code (identifier), the number of employees (but this information is not always available), the region in which it is located, and the departments of which it is composed (at least one). There are two types of hospitals: public and private. Of **public hospitals**, we are interested also in the year of foundation, and of **private hospitals**, also in the various persons who have directed them, each with the starting date and the salary. Note that on each date and in each hospital, at most one person begins their **direction** role. Note also that the same person may direct the same hospital in different periods. Of each **department**, we are interested in the code (unique within the hospital in which it is located), the size in square meters and, in the case of an **obstetrics department**, also the number of beds. For each **admission** of a person to a department, various kinds of information are recorded; those of interest for the database are: the date, the occupied bed (positive integer number unique within the department), the person admitted, and the cost of the admission. When the admission is for a **delivery**, the department must be the obstetrics department, and in this case the information about the persons born from the delivery (possibly more than one) is also recorded. Note that a bed can be occupied by a maximum of one person per day and the same person cannot be admitted more than once per day. Finally, of each **person**, we are interested in the tax code (identifier), the name, the surname, and the date of birth.

Problem 1: Conceptual schema



There are no external constraints to add to the diagram.

Problem 2

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

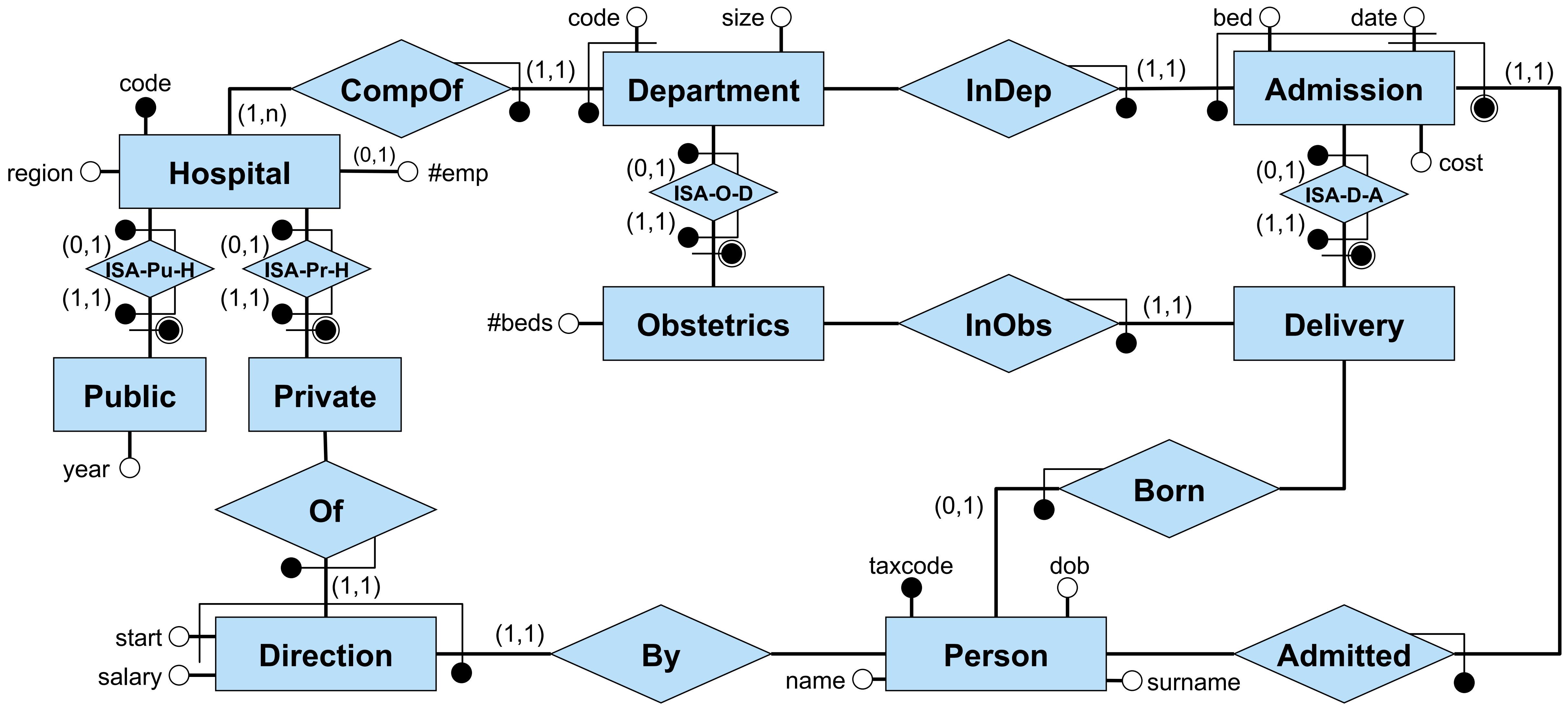
1. We want to avoid null values in the database.
2. For an admission, we are always interested in knowing the person who has been admitted.

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 – Diagram



Problem 2: Restructured conceptual schema – External constraints

1) Generalization constraint resulting from the elimination of the complete generalization hierarchy for **Hospital**:
   For each instance of the conceptual schema, each instance of **Hospital** participates either to **ISA-Pu-H** or to **ISA-Pr-H**, but not to both.

2) Constraint resulting from the elimination of ISA between the relationships **InOps** and **InDep**:
   For each instance ***I*** of the conceptual schema, if <Delivery:*d*,Obstetrics:*o*> ∈ ***instances***(***I***,**InOps**), *a* is the unique instance of **Admission** such that <Delivery:*d*,Admission:*a*> ∈ ***instances***(***I***,**ISA-D-A**), and *p* is the unique instance of **Department** such that <Obstetrics:*o*,Department:*p*> ∈ ***instances***(***I***,**ISA-O-D**), then <Admission:*a*,Department:*p*> ∈ ***instances***(***I***,**InOps**).

Problem 2: Direct translation (1/2)

Hospital(code, region, #emp*)
  inclusion: Hospital[code] ⊆ Department[hospital]
  generalization constraint: Hospital[code] ⊆ Public[code] ∪ Private[code]
  Public[code] ∩ Private[code] = ∅

Public(code, year)
  foreign key: Public[code] ⊆ Hospital[code]

Private(code)
  foreign key: Private[code] ⊆ Hospital[code]

Department(code, hospital, size)
  foreign key: Department[hospital] ⊆ Hospital[code]

Admission(person, date, bed, cost)
  foreign key: Admission[person] ⊆ Person[taxcode]
  join constraint: in the join between Admission and InDep, done by equating the attributes person and date, the attributes (bed, date, department, hospital) form a key

InDep(person, date, department, hospital)
  foreign key: InDep[person,date] ⊆ Admission[person,date]
  foreign key: InDep[department,hospital] ⊆ Department[code,hospital]

Problem 2: Direct translation (2/2)

Obstetrics(code, hospital, #beds)
  foreign key: Obstetrics[code,hospital] $\subseteq$ Department[code,hospital]

Delivery(person, date)
  foreign key: Delivery[person,date] $\subseteq$ Admission[person,date]

InObs(person, date, department, hospital)
  foreign key: InObs[person,date] $\subseteq$ Delivery[person,date]
  foreign key: InObs[department,hospital] $\subseteq$ Obstetrics[code,hospital]
  foreign key: InObs[person,date,department,hospital] $\subseteq$ InDep[person,date,department,hospital]

Person(taxcode, name, surname, dob)

Born(person, date, child)
  foreign key: Born[person,date] $\subseteq$ Delivery[person,date]
  foreign key: Born[child] $\subseteq$ Person[taxcode]

Direction(hospital, director, start, salary)
  foreign key: Direction[hospital] $\subseteq$ Private[code]
  foreign key: Direction[director] $\subseteq$ Person[taxcode]

Problem 2: Direct translation – External constraints

1) The generalization constraints resulting from the elimination of the complete hierarchy for **Hospital** are already expressed in the schema.
2) The constraint resulting from the elimination of ISA between the relationships **InOps** and **InDep** has been expressed as a foreign key from **InOps** to **InDep**.

Problem 2: Restructuring of the relational schema

1. We want to avoid null values in the database.
2. For an admission, we are always interested in knowing the person who has been admitted.

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

- We take into account indication 1 by performing a vertical partition of **Hospital** into two relations, where the first one contains the attributes **code** and **region**, and the second one contains, in addition to the primary key **code**, the nullable attribute **#emp**.
- Indication 2 has already been taken into account by selecting as primary identifier for the entity **Admission** the attribues (**person**, **date**).

Problem 2: Restructured relational schema

We specify here only the relations with their constraints that have been changed with respect to the schema obtained through the direct translation.

Hospital(code, region)

inclusion: Hospital[code] $\subseteq$ Department[hospital]

generalization constraint: Hospital[code] $\subseteq$ Public[code] $\cup$ Private[code]

Public[code] $\cap$ Private[code] = $\emptyset$

Employees(code, #emp)

foreign key: Employees[code] $\subseteq$ Hospital[code]

No additional (external) contraints are introduced by restructuring the relational schema.

Problem 3

Consider a database that includes the relations `Municipality` and `Residence`. For each municipality, the relation `Municipality(name,province,year)` stores the name of the municipality, the name of the province in which the municipality is located and the year in which the last municipal election was held. The relation `Residence(code,municipality,salary)` stores, for each person, the tax code, the municipality of residence, and the annual salary.

1. We call a province "obsolete" if all municipalities with names different from the one of the province itself had their last municipal election before 2016. Write a *query in relational algebra* that returns the provinces that are obsolete.

2. Given a province, we often want to know the name of the municipality (or municipalities, if there are more than one), among those in that province, in which lives the person (or the persons, if there are more than one) with the highest salary within the province. Write a *SQL query* that returns this information for each province.

Problem 3: Solution (1/2)

`Municipality(name,province,year)`

1. We call a province "obsolete" if all municipalities with names different from the one of the province itself had their last municipal election before 2016. Write a *query in relational algebra* that returns the provinces that are obsolete.

$\text{PROJ}_{\text{province}}(\text{Municipality})$

$-$

$\text{PROJ}_{\text{province}}(\text{SEL}_{\text{name <> province AND year >= 2016}}(\text{Municipality}))$

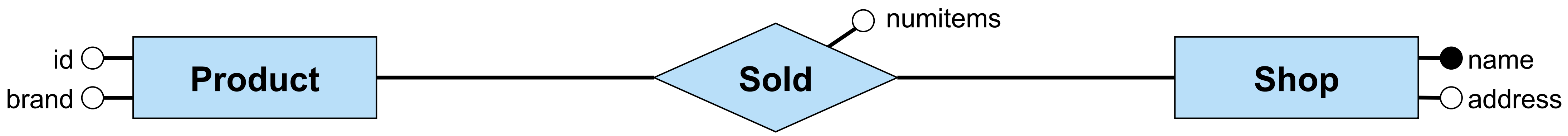
Problem 3: Solution (2/2)

```
Municipality(name,province,year)
Residence(code,municipality,salary)
```

2. Given a province, we often want to know the name of the municipality (or municipalities, if there are more than one), among those in that province, in which lives the person (or the persons, if there are more than one) with the highest salary within the province. Write a *SQL query* that returns this information for each province.

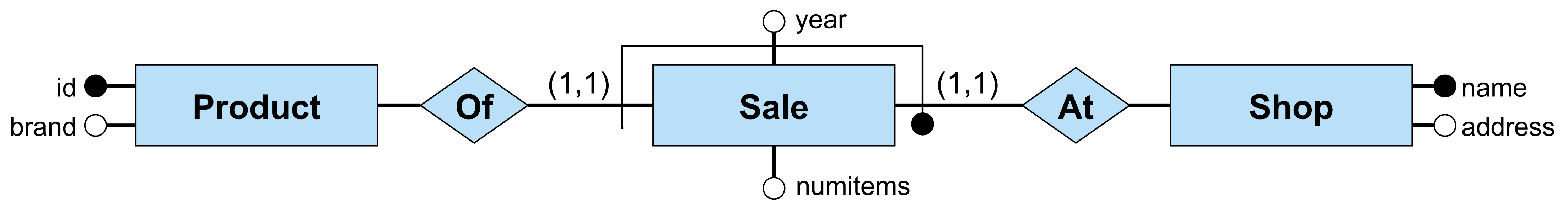
```
SELECT DISTINCT M.province, R.municipality
FROM Municipality M, Residence R
WHERE M.name = R.municipality AND
      R.salary = (SELECT MAX(R2.salary)
                  FROM Municipality M2, Residence R2
                  WHERE M2.name = R2.municipality AND
                        M2.province = M.province)
```

Problem 4



Consider the conceptual schema shown above, which models how many items of each product have been sold by each shop. Suppose that, due to a revision of the requirements, it has become of interest to know the number of items of each product sold by each shop also in past years, once per year. Show the conceptual schema resulting from the revision.

Problem 4: Solution



We adopt the standard approach of historizing the **Sold** relationship. For doing so we need to first transform it into an entity (which we call **Sale**) and then we add the attribute **year** and make it part of the identifier of **Sale**.