Introduction to Databases
Exam of 8/2/2019
Solutions

Diego Calvanese

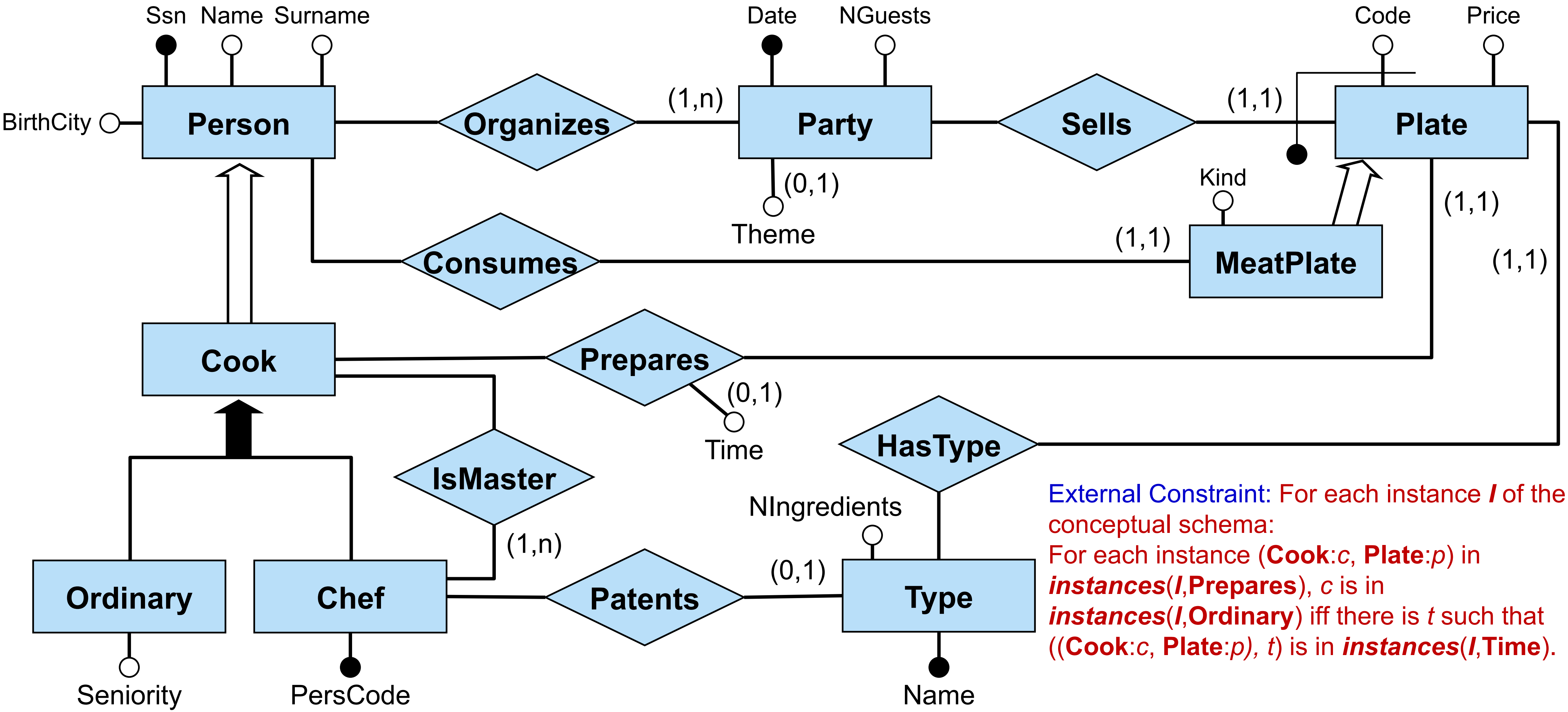
Bachelor in Computer Science
Faculty of Computer Science
Free University of Bozen-Bolzano

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

Problem 1

Design the Entity Relationship schema of an information system for a restaurant that organizes private parties with invited guests. For each party held in the restaurant, we are interested in the date (identifier), the number of guests who participated to the party on that date, the theme (note that not all parties have a theme), the persons (at least one) who organized the party, and the plates sold at the party. For each person, we are interested in ssn (identifier), name, surname, and city of birth. For each plate, we are interested in the code (unique within the party where the plate was sold), the price, the type, and the cook who has prepared it (note that plates of the same type may have different prices, e.g., due to variations). If the cook who prepared the plate is an ordinary cook, we are also interested in the time needed to prepare the plate. In addition, for each meat plate we are also interested in the kind of meat (beef, pork, lamb, etc.) and the guest who consumed it. For each type of plate, we are interested also in the name (identifier), the number of ingredients, and, in case it has been patented by a cook, also the cook who patented it, with the caveat that only cooks who are chefs can patent types of plates. There are exactly two kinds of cooks: ordinary ones and chef-cooks. Of ordinary cooks we are interested in the seniority, and of chef-cooks we are interested in the personal code (unique among chef-cooks), and the cooks (at least one) who have been their masters.

Problem 1: Conceptual schema



External Constraint: For each instance ***I*** of the conceptual schema:
For each instance (**Cook**:*c*, **Plate**:*p*) in ***instances***(***I***,**Prepares**), *c* is in ***instances***(***I***,**Ordinary**) iff there is *t* such that ((**Cook**:*c*, **Plate**:*p*), *t*) is in ***instances***(***I***,**Time**).

Problem 2

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

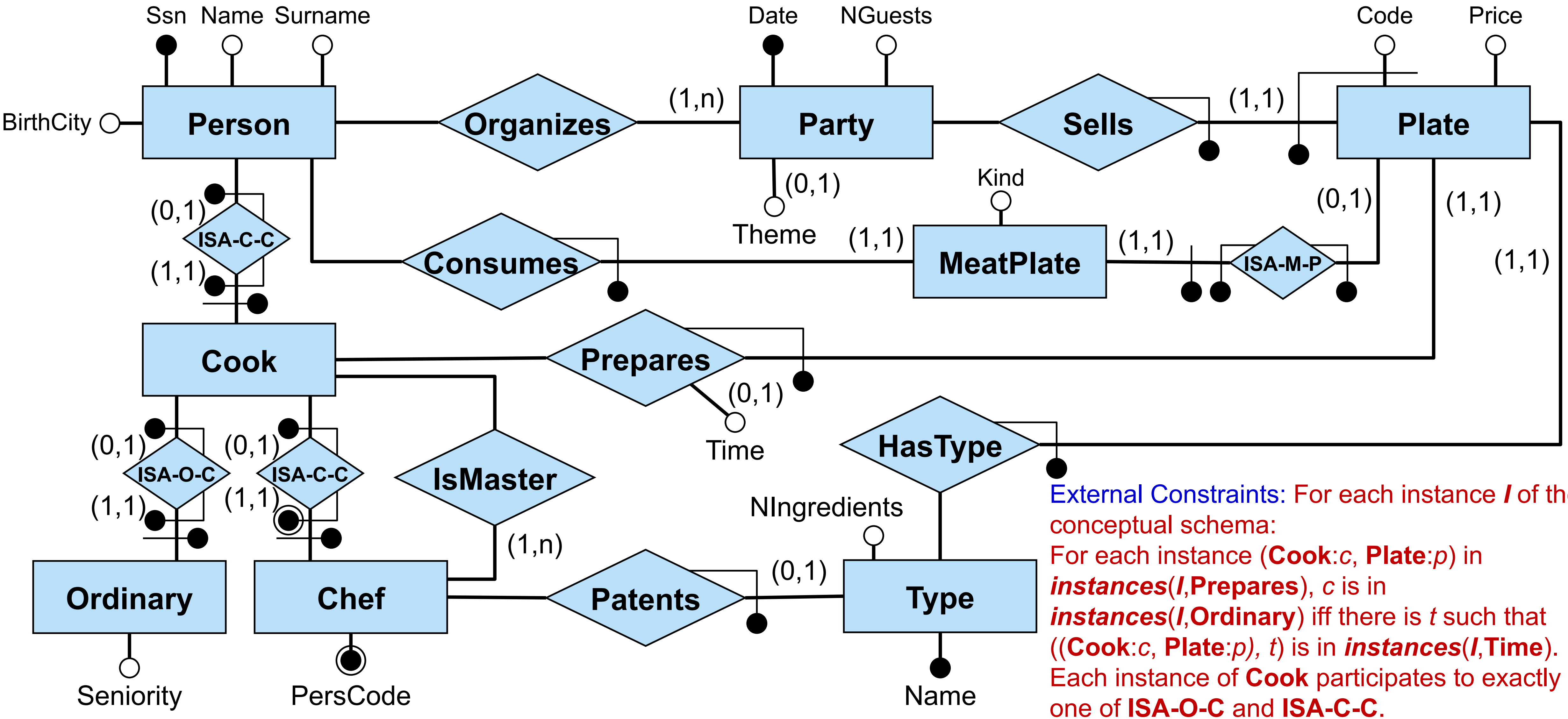
1. chef-cooks are accessed by means of the personal code;
2. when accessing a type of plate, we are often asked also the cook who has patented it;
3. null values in the database have to be avoided.

As intermediate 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



External Constraints: For each instance ***I*** of the conceptual schema:
For each instance (**Cook**:*c*, **Plate**:*p*) in ***instances***(***I***,**Prepares**), *c* is in ***instances***(***I***,**Ordinary**) iff there is *t* such that ((**Cook**:*c*, **Plate**:*p*), *t*) is in ***instances***(***I***,**Time**).
Each instance of **Cook** participates to exactly one of **ISA-O-C** and **ISA-C-C**.

Problem 2: Direct translation (1/3)

Person(<u>Ssn</u>, Name, Surname, BirthCity)

Cook(<u>Ssn</u>)
  foreign key: Cook[Ssn] $\subseteq$ Person[Ssn]
  generalization constraint: Cook[Ssn] $\subseteq$ Ordinary[Ssn] $\cup$ ISA-C-C[Cook]

Ordinary(<u>Ssn</u>, Seniority)
  foreign key: Ordinary[Ssn] $\subseteq$ Cook[Ssn]
  generalization constraint: Ordinary[Ssn] $\cap$ ISA-C-C[Cook] = $\emptyset$

Chef(<u>PersCode</u>)
  foreign key: Chef[PersCode] $\subseteq$ ISA-C-C[Chef]
  inclusion: Chef[PersCode] $\subseteq$ IsMaster[Chef]

ISA-C-C(<u>Chef</u>, Cook)
  foreign key: ISA-C-C[Chef] $\subseteq$ Chef[PersCode]
  foreign key: ISA-C-C[Cook] $\subseteq$ Cook[Ssn]
  key: Cook

Problem 2: Direct translation (2/3)

IsMaster(Chef, Cook)
  foreign key: IsMaster[Chef] $\subseteq$ Chef[PersCode]
  foreign key: IsMaster[Cook] $\subseteq$ Cook[Ssn]

Party(Date, NGuests, Theme*)
  inclusion: Party[Date] $\subseteq$ Organizes[Party]

Organizes(Person, Party)
  foreign key: Organizes[Person] $\subseteq$ Person[Ssn]
  foreign key: Organizes[Party] $\subseteq$ Party[Date]

Plate(Code, Party, Price)
  foreign key: Plate[Party] $\subseteq$ Party[Date]
  foreign key: Plate[Code,Party] $\subseteq$ Prepares[PlateCode,PlateParty]
  foreign key: Plate[Codel,Party] $\subseteq$ HasType[PlateCode,PlateParty]

MeatPlate(Code, Party, Kind)
  foreign key: MeatPlate[Code,Party] $\subseteq$ Plate[Code,Party]
  foreign key: MeatPlate[Code,Party] $\subseteq$ Consumes[PlateCode,PlateParty]

Problem 2: Direct translation (3/3)

Consumes(Person, <u>PlateCode</u>, <u>PlateParty</u>)
  foreign key: Consumes[Person] $\subseteq$ Person[Ssn]
  foreign key: Consumes[PlateCode,PlateParty] $\subseteq$ MeatPlate[Code,Party]

Prepares(Cook, <u>PlateCode</u>, <u>PlateParty</u>, Time*)
  foreign key: Prepares[Cook] $\subseteq$ Cook[Ssn]
  foreign key: Prepares[PlateCode,PlateParty] $\subseteq$ Plate[Code,Party]

Type(<u>Name</u>, NIngredients)

HasType(<u>PlateCode</u>, <u>PlateParty</u>, Type)
  foreign key: HasType[PlateCode,PlateParty] $\subseteq$ Plate[Code,Party]
  foreign key: HasType[Type] $\subseteq$ Type[Name]

Patents(Chef, <u>Type</u>)
  foreign key: Patents[Chef] $\subseteq$ Chef[PersCode]
  foreign key: Patents[Type] $\subseteq$ Type[Name]

Constraint: For each tuple (*cook*, *c*, *p*, *time*) in **Prepares**, if there is an *s* such that the tuple (*cook*, *s*) is in **Ordinary**, then *time* ≠ NULL.

Problem 2: Restructuring of the relational schema

1. Chef-cooks are accessed by means of the personal code.
2. When accessing a type of plate, we are often asked also the cook who has patented it.
3. Null values in the database have to be avoided.

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

- Indication 1 has already been considered by making PersCode the primary identifier of the Chef entity, and hence by translating ISA-C-C as a separate relation.
- We take into account indication 2 by merging Type with Patents. However, due to indication 3, we need to first carry out a horizontal decomposition of Type, into Type and PatentedType, and then we can merge PatentedType with Patents.
- Due to indication 3, we do a vertical decomposition of Party.
- To take into account indication 3 for Prepares, considering that the preparation time is specified when the cook is ordinary, we do a horizontal decomposition.

Problem 2: Restructured relational schema (1/3)

Person(<u>Ssn</u>, Name, Surname, BirthCity)

Cook(<u>Ssn</u>)
- foreign key: Cook[Ssn] ⊆ Person[Ssn]
- generalization constraint: Cook[Ssn] ⊆ Ordinary[Ssn] ∪ ISA-C-C[Cook]

Ordinary(<u>Ssn</u>, Seniority)
- foreign key: Ordinary[Ssn] ⊆ Cook[Ssn]
- generalization constraint: Ordinary[Ssn] ∩ ISA-C-C[Cook] = ∅

Chef(<u>PersCode</u>)
- foreign key: Chef[PersCode] ⊆ ISA-C-C[Chef]
- inclusion: Chef[PersCode] ⊆ IsMaster[Chef]

ISA-C-C(<u>Chef</u>, Cook)
- foreign key: ISA-C-C[Chef] ⊆ Chef[PersCode]
- foreign key: ISA-C-C[Cook] ⊆ Cook[Ssn]
- key: Cook

IsMaster(<u>Chef</u>, <u>Cook</u>)
- foreign key: IsMaster[Chef] ⊆ Chef[PersCode]
- foreign key: IsMaster[Cook] ⊆ Cook[Ssn]

Problem 2: Restructured relational schema (2/3)

Party(Date, NGuests)
  inclusion: Party[Date] ⊆ Organizes[Party]

PartyTheme(Party, Theme)
  foreign key: PartyTheme[Party] ⊆ Party[Date]

Organizes(Person, Party)
  foreign key: Organizes[Person] ⊆ Person[Ssn]
  foreign key: Organizes[Party] ⊆ Party[Date]

Plate(Code, Party, Price)
  foreign key: Plate[Party] ⊆ Party[Date]
  constraint: Plate[Code,Party] ⊆ PreparesChef[PlateCode,PlateParty] ∪
    PreparesOrdinary[PlateCode,PlateParty]
  foreign key: Plate[Codel,Party] ⊆ HasType[PlateCode,PlateParty]

MeatPlate(Code, Party, Kind)
  foreign key: MeatPlate[Code,Party] ⊆ Plate[Code,Party]
  foreign key: MeatPlate[Code,Party] ⊆ Consumes[PlateCode,PlateParty]

Problem 2: Restructured relational schema (3/3)

Consumes(Person, <u>PlateCode</u>, <u>PlateParty</u>)
  foreign key: Consumes[Person] $\subseteq$ Person[Ssn]
  foreign key: Consumes[PlateCode,PlateParty] $\subseteq$ MeatPlate[Code,Party]

PreparesChef(Cook, <u>PlateCode</u>, <u>PlateParty</u>)
  foreign key: PreparesChef[Cook] $\subseteq$ Chef[Ssn]
  foreign key: PreparesChef[PlateCode,PlateParty] $\subseteq$ Plate[Code,Party]

PreparesOrdinary(Cook, <u>PlateCode</u>, <u>PlateParty</u>, Time)
  foreign key: PreparesOrdinary[Cook] $\subseteq$ Ordinary[Ssn]
  foreign key: PreparesOrdinary[PlateCode,PlateParty] $\subseteq$ Plate[Code,Party]

Type(<u>Name</u>, NIngredients)

PatentedType(<u>Name</u>, NIngredients, Chef)
  foreign key: PatentedType[Chef] $\subseteq$ Chef[PersCode]
  foreign key: PatentedType[Type] $\subseteq$ Type[Name]

HasType(<u>PlateCode</u>, <u>PlateParty</u>, Type)
  foreign key: HasType[PlateCode,PlateParty] $\subseteq$ Plate[Code,Party]
  constraint: HasType[Type] $\subseteq$ Type[Name] $\cup$ PatentedType[Name]

Problem 3

The relation `Teaches(teacher,class,year)` specifies which teachers have taught in which classes in the various years, while the relation `Class(code,size,school)` stores, for each class, the code (identifier), the size, and the school to which it belongs.

Note that:

- more than one teacher can teach in the same class in the same year,
- a teacher can teach in several classes in the same year, and
- a teacher can teach in the same class in several years.

Express in SQL the following queries over the above two relations:

1. For each teacher, compute the schools where the teacher has been teaching since 2010.
2. For each teacher, compute the maximum size within the classes where the teacher has worked since 2005, but only if the number of such classes is greater than 10.
3. For each class in the "`Liceo Carducci`" school, compute the average annual number of teachers who taught there.

Problem 3: Solution (1/2)

Teaches(teacher, class, year)

Class(<u>code</u>, size, school)

1. For each teacher, compute the schools where the teacher has been teaching since 2010.

```
SELECT DISTINCT T.teacher, C.school
FROM Teaches T JOIN Class C ON T.class = C.code
WHERE T.year >= 2010
```

2. For each teacher, compute the maximum size within the classes where the teacher has worked since 2005, but only if the number of such classes is greater than 10.

```
SELECT T.teacher, MAX(C.size)
FROM Teaches T JOIN Class C ON T.class = C.code
WHERE T.year >= 2005
GROUP BY T.teacher
HAVING COUNT(C.code) > 10
```

Problem 3: Solution (2/2)

Teaches(teacher, class, year)

Class(code, size, school)

3. For each class in the "Liceo Carducci" school, compute the average annual number of teachers who taught there.

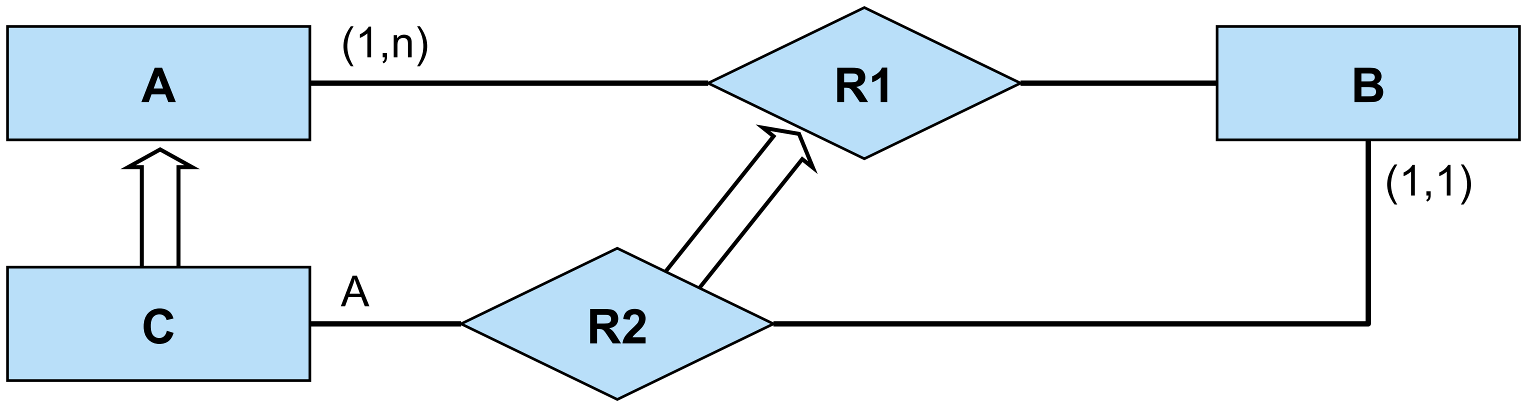
```
SELECT V.class, AVG(V.numTeachers) avgNumTeachers
FROM (SELECT T.class, T.year, COUNT(T.teacher) AS numTeachers
      FROM Teaches T JOIN Class C ON T.class = C.code
      WHERE C.school = 'Liceo Carducci'
      GROUP BY T.class, T.year) V
GROUP BY V.class
```

Problem 4

Consider the conceptual schema *S* shown below and say if there is an instance of schema *S* that contains exactly one instance of *A* that is **not** an instance of *C*.

If the answer is positive, show such an instance of *S*.

If the answer is negative, explain in detail why such an instance does not exist.



Problem 4: Solution

The following instance *I* satisfies the specified condition:

instances(*I*,*A*) = { a, c },

instances(*I*,*B*) = { b },

instances(*I*,*C*) = { c },

instances(*I*,*R1*) = { (*A*:a, *B*:b), (*A*:c, *B*:b) },

instances(*I*,*R2*) = { (*A*:c, *B*:b) },