Introduction to Databases

Exam of 20/02/2025

With Solutions

Diego Calvanese

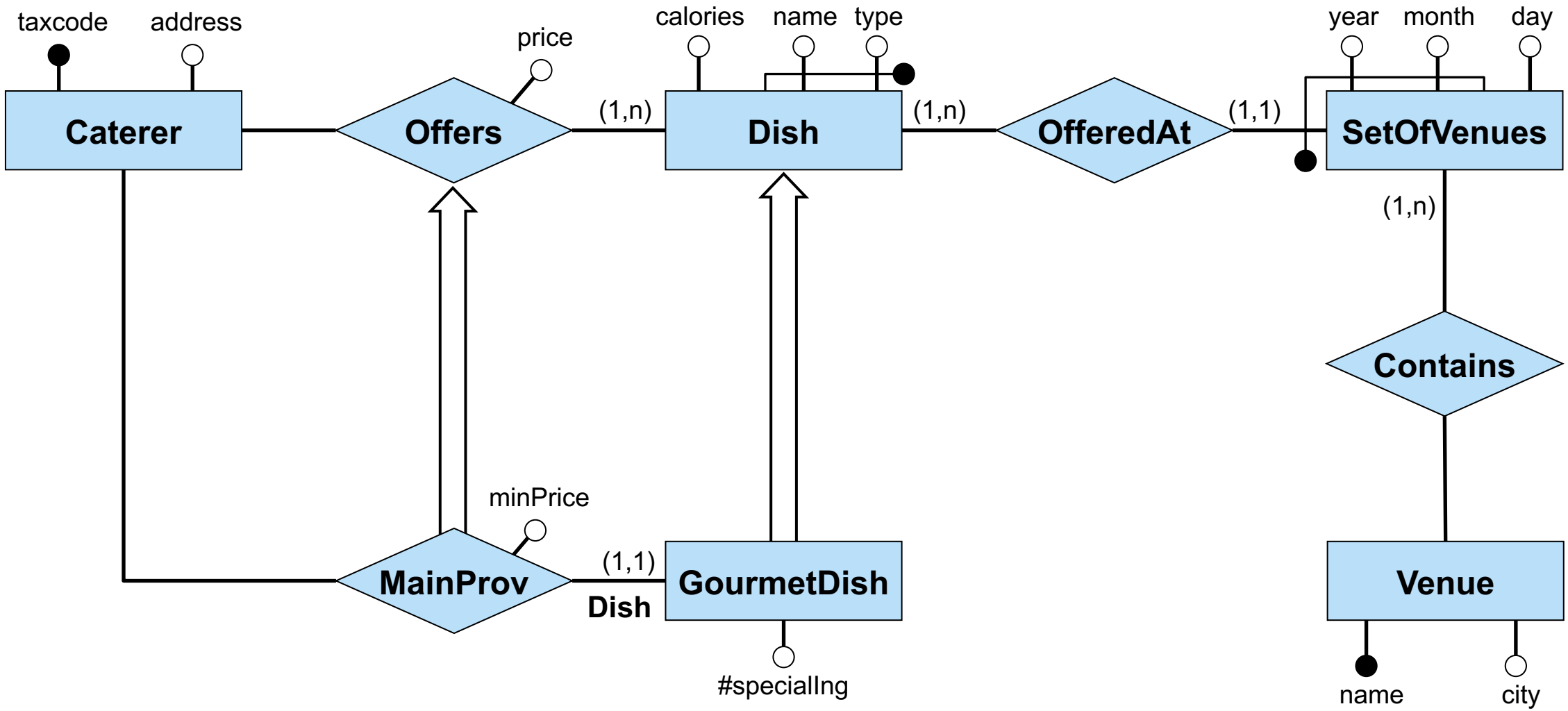
Bachelor in Computer Science
Faculty of Engineering
Free University of Bozen-Bolzano

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

Problem 1

Design the Entity-Relationship schema of an application for managing dishes offered by caterers who deliver them to venues, for which the following information is of interest. Of each **caterer**, we are interested in the taxcode (identifier) and the address. Of each **dish**, we are interested in the name and type (normal,vegetarian, vegan, etc.), which together identify the dish, the calories, and the caterers that offer the dish, each with the price at which they offer it. It should be noted that a dish must be offered by at least one caterer and can be offered by multiple caterers. Additionally, of each dish, we are interested in the **set of venues** where it is offered. This set must not be empty but can change over time, at most once per month, and we are interested in the date (i.e., the day, in addition to the month and year) since when the set of venues offers the dish. **Gourmet dishes** are a type of dish with special ingredients, for which one of the offering caterers acts as the main provider, who specifies a minimum price for the dish. Of each gourmet dish we are interested in the number of special ingredients, the main provider, and the minimum price he has specified for the dish. Notice that no caterer might offer a gourmet dish at a lower price than the minimum price set by the main provider. Finally, of each **venue** we are interested in the name (identifier) and the city where it is located.

Problem 1: Conceptual schema – Diagram



Problem 1: Conceptual schema – External constraints

"No caterer might offer a gourmet dish at a lower price than the minimum price set by the main provider."

Formally: For each instance I of the schema, if(Caterer:mc, Dish:d) $\in$ $instances(I,$**MainProv**$)$, ((Caterer:mc,Dish:d), mp) $\in$ $instances(I,$**minPrice**$)$, (Caterer:c,Dish:d) $\in$ $instances(I,$**Offers**$)$, and ((Caterer:c,Dish:d), p) $\in$ $instances(I,$**price**$)$, then $mp \leq p$.

Problem 2

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

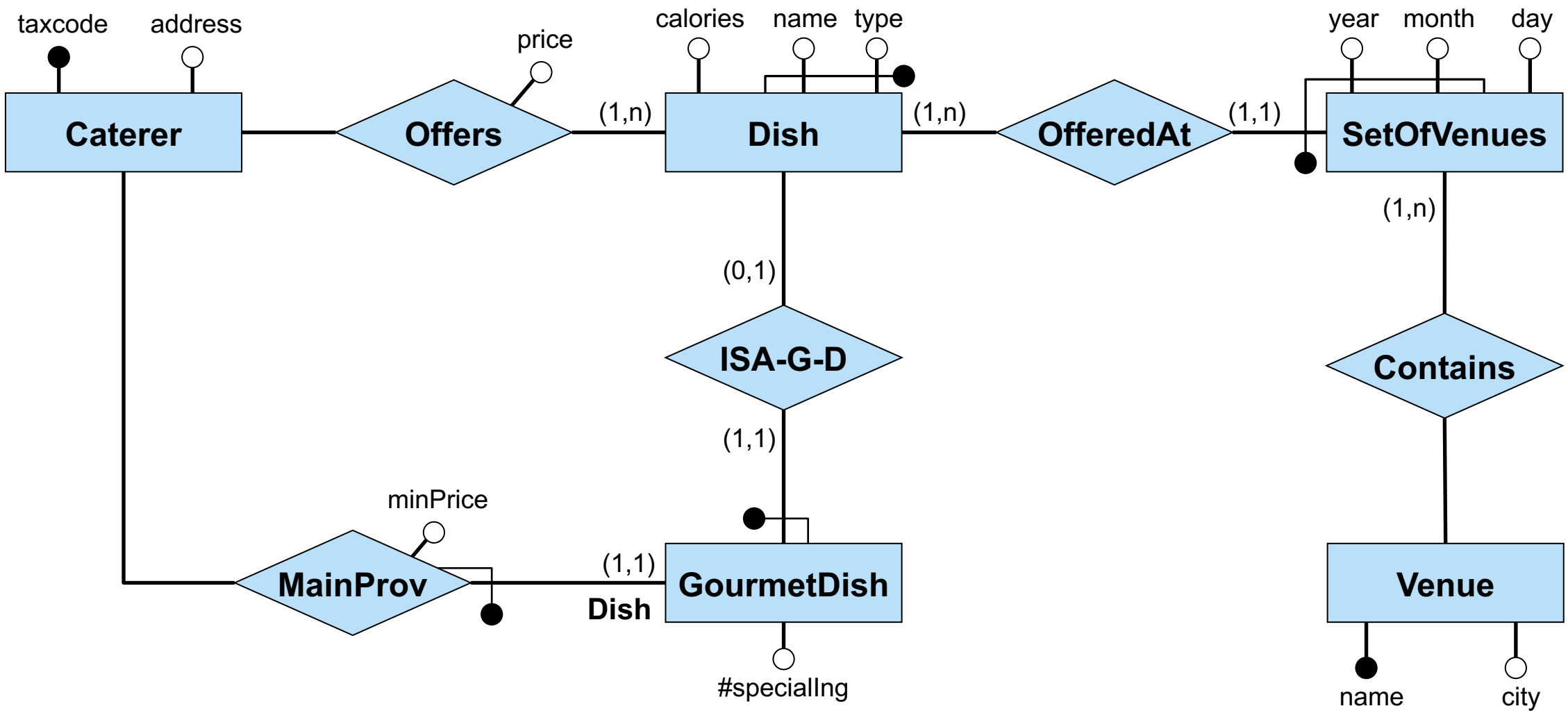
1. We access gourmet dishes separately from ordinary dishes.
2. Every time we access a gourmet dish, we always want to know who is the main provider, the number of special ingredients, and also the calories.

In your design you should follow the methodology adopted in the course, and 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).

You should motivate explicitly how the above indications affect your design.

Problem 2: Restructured conceptual schema



Problem 2: Restructured conceptual schema – External constraints

1. "No caterer might offer a gourmet dish at a lower price than the minimum price set by the main provider."
   Formally: For each instance *I* of the schema, if (Caterer:*mc*, Dish:*gd*) ∈ *instances*(*I*,**MainProv**), ((Caterer:*mc*,Dish:*gd*), *mp*) ∈ *instances*(*I*,**minPrice**), (GourmetDish:*gd*, Dish:*d*) ∈ *instances*(*I*,**ISA-G-D**), (Caterer:*c*,Dish:*d*) ∈ *instances*(*I*,**Offers**), and ((Caterer:*c*,Dish:*d*), *p*) ∈ *instances*(*I*,**price**), then $mp \leq p$.
2. Constraint due to the elimination of ISA between **MainProv** and **Offer**.
   Formally: For each instance *I* of the schema, if (Caterer:*c*, Dish:*gd*) ∈ *instances*(*I*,**MainProv**) and (GourmetDish:*gd*, Dish:*d*) ∈ *instances*(*I*,**ISA-G-D**), then (Caterer:*c*,Dish:*d*) ∈ *instances*(*I*,**Offers**),

Problem 2: Direct translation

Caterer(taxcode, address)

Dish(name, type, calories)
- inclusion: Dish[name,type] $\subseteq$ Offers[dishName,dishType]
- inclusion: Dish[name,type] $\subseteq$ SetOfVenues[dishName,dishType]

GourmetDish(name, type, #specialIng)
- foreign key: GourmetDish[name,type] $\subseteq$ Dish[name,type]
- foreign key: GourmetDish[name,type] $\subseteq$ MainProv[dishName,dishType]

Venue(name, city)

SetOfVenues(dishName, dishType, year, month, day)
- foreign key: SetOfVenues[dishName,dishType] $\subseteq$ Dish[name,type]
- inclusion: SetOfVenues[dishName,dishType,year,month] $\subseteq$ Contains[dishName,dishType,year,month]

Offers(caterer, dishName, dishType, price)
- foreign key: Offers[caterer] $\subseteq$ Caterer[taxcode]
- foreign key: Offers[dishName,dishType] $\subseteq$ Dish[name,type]

MainProv(caterer, dishName, dishType, minPrice)
- foreign key: MainProv[caterer] $\subseteq$ Caterer[taxcode]
- foreign key: MainProv[dishName,dishType] $\subseteq$ GourmetDish[name,type]
- foreign key: MainProv[caterer,dishName,dishType] $\subseteq$ Offers[caterer,dishName,dishType]

Contains(dishName, dishType, year, month, venue)
- foreign key: Contains[dishName,dishType,year,month] $\subseteq$ SetOfVenues[dishName,dishType,year,month]
- foreign key: Contains[venue] $\subseteq$ Venue[name]

Problem 2: Direct translation – External constraints

1. “No caterer might offer a gourmet dish at a lower price than the minimum price set by the main provider.”
   We capture this through a **CHECK** constraint on **MainProv**:

```
CHECK (minPrice <=
  ALL (SELECT O.price from Offers O
       WHERE dishName = O.dishName AND
             dishType = O.dishType))
```

2. External constraint 2, introduced due to the elimination of ISA between **MainProv** and **Offer**, is captured by the last foreign key on **MainProv**.

Problem 2: Restructuring of the relational schema

Indications:

1. We access gourmet dishes separately from ordinary dishes.
2. Every time we access a gourmet dish, we always want to know who is the main provider, the number of special ingredients, and also the calories.

- We take into account Indication 1 by performing a horizontal decomposition of relation **Dish** into **DishGourmet** into **DishNonGourmet**.
- We take into account Indication 2 by merging **GourmetDish** and **DishGourmet**, (which are strongly coupled), into a new relation (for which we chose the name **DishGourmet**), and by further merging **MainProv** and **DishGourmet** (which are also strongly coupled).

Problem 2: Restructured relational schema

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

Specifically, the relations Caterer, Venue, and Contains are not modified, and the relations Dish, GourmetDish, and MainProv are replaced by the relations DishNonGourmet and DishGourmet, defined as follows:

DishNonGourmet(name, type, calories)
- inclusion: DishNonGourmet[name,type] $\subseteq$ Offers[dishName,dishType]
- inclusion: DishNonGourmet[name,type] $\subseteq$ SetOfVenues[dishName,dishType]

DishGourmet(name, type, calories, #specialIng, mainProv, minPrice)
- inclusion: DishGourmet[name,type] $\subseteq$ Offers[dishName,dishType]
- inclusion: DishGourmet[name,type] $\subseteq$ SetOfVenues[dishName,dishType]
- foreign key: DishGourmet[mainProv] $\subseteq$ Caterer[taxcode]
- foreign key: DishGourmet[mainProv,name,type] $\subseteq$ Offers[caterer,dishName,dishType]
- constraint: DishGourmet[name,type] $\cap$ DishNonGourmet[name,type] = $\emptyset$

SetOfVenues(dishName, dishType, year, month, year)
- constraint: SetOfVenues[dishName,dishType] $\subseteq$ DishGourmet[name,type] $\cup$ DishNonGourmet[name,type]
- inclusion: SetOfVenues[dishName,dishType,year,month] $\subseteq$ Contains[dishName,dishType,year,month]

Offers(caterer, dishName, dishType, price)
- foreign key: Offers[caterer] $\subseteq$ Caterer[taxcode]
- constraint: Offers[dishName,dishType] $\subseteq$ DishGourmet[name,type] $\cup$ DishNonGourmet[name,type]

Problem 2: Restructured relational schema – External constraints

1. "No caterer might offer a gourmet dish at a lower price than the minimum price set by the main provider."
   We capture this through a **CHECK** constraint on **DishGourmet**:

```
CHECK (minPrice <=
  ALL (SELECT O.price from Offers O
       WHERE name = O.dishName AND
             type = O.dishType))
```

2. External constraint 2, introduced due to the elimination of ISA between **MainProv** and **Offer**, is captured by the last foreign key on **DishGourmet**.

Problem 3

Consider a database *D* containing the two relations:

i. `Activity(code,type)`, which stores the code and the type (a string) of the activities offered by a resort;

ii. `Schedule(guest,activitycode,time)`, with a foreign key constraint from `activitycode` to `code` of `Activity`, which stores the activities that guests have scheduled to perform at the resort.

Note that the set of all activity types is given by all types that appear as the type of some activity that is being offered by the resort.

1. Write in **relational algebra** a query over *D* that computes all pairs (*g*,*t*) such that no activity of type *t* appears in the schedule of guest *g*.
2. Write in **SQL** a query over *D* that computes all guests who have in their schedule at least one activity of each type.

Problem 3: Solution 1

`Activity(code,type)` `Schedule(guest,activitycode,time)`

1. Write in **relational algebra** a query over *D* that computes all pairs (*g*,*t*) such that no activity of type *t* appears in the schedule of guest *g*.

$$\mathbf{PROJ}_{guest,type}\ (\text{Schedule}\ \mathbf{JOIN}\ \text{Activity}) - \mathbf{PROJ}_{guest,type}\ (\text{Schedule}\ \mathbf{JOIN}_{activitycode\ =\ code}\ \text{Activity})$$

Notice that the first join is actually a cartesian product, returning all possible pairs of a schedule and an activity.

Problem 3: Solution 2

`Activity(code,type)` `Schedule(guest,activitycode,time)`

2. Write in **SQL** a query over *D* that computes all guests who have in their schedule at least one activity of each type.

```
SELECT S.guest
FROM Schedule S
WHERE NOT EXISTS
    (SELECT type
     FROM Activity
     WHERE type NOT IN
        (SELECT type
         FROM Schedule JOIN Activity ON activitycode = code
         WHERE guest = S.guest))
```

Problem 4

Provide the definition of integrity constraint and of referential integrity constraint.
Then consider the following relational schema ***S*** (in which primary keys are underlined and attributes with an asterisk may contain NULL):

R(A, B*, C)
tuple constraint: B is not NULL implies B = A

Q(D, E, F)
foreign key: Q[D,E,F] $\subseteq$ R[A,B,C]

For each of the following three items, say whether there exists at least one database instance that is correct with respect to schema ***S*** with the corresponding property:

1. the set { ***B*** } is not a superkey for ***R***;
2. the set { ***C*** } is not a superkey for ***R***;
3. the set { ***D***, ***E*** } is not a superkey for ***Q***.

For each of the three items listed above, if the answer is positive, then show the corresponding database instance; if the answer is negative, then describe in detail the reason why a database instance that is correct with respect to schema ***S*** does not exist.

Problem 4: Solution of Item 1

For the definition of integrity constraint and of referential integrity constraint, we refer to the course slides.

The answer is **no**. We show that there is no database instance that is correct w.r.t. schema ***S*** in which the set { ***B*** } is not a superkey for ***R***, by showing that a database instance ***I*** in which { ***B*** } is not a superkey for ***R*** is not correct w.r.t. ***S***.

Indeed, in ***I***, relation ***R*** would contain two distinct tuples with the same non-null value for ***B***, say (*a1*,*b*,*c1*) and (*a2*,*b*,*c2*), with $b \neq$ **`NULL`**.

Then, either one or both of the two tuples do not satify the tuple constraint for ***R***, i.e., $a1 \neq b$ or $a2 \neq b$, and then ***I*** is not correct w.r.t. ***S***.

Or they both satisfy the tuple constraint, i.e., $a1 = a2 = b$, and since the two tuples are distinct, $c1 \neq c2$. Then we have in ***R*** two tuples with the same value *b* for attribute ***A*** but different values for attribute ***C***, which constitutes a violation of the primary key constraint for ***R***. Hence, ***I*** is not correct w.r.t. ***S***.

Problem 4: Solution of Item 2

The answer is **yes**. We show that there exists a database instance ***I*** that is correct w.r.t. schema ***S*** in which the set { ***C*** } is not a superkey for ***R***.

Indeed, it suffices to define ***I*** as the database instance where relation ***Q*** is empty and relation ***R*** contains two tuples (*a1*,*a1*,*c*) and (*a2*,*a2*,*c*). It is immediate to see that ***I*** is correct w.r.t. schema ***S***. Moreover, since ***R*** contains two tuples with the same value *c* for attribute ***C***, but different values for attribute ***A*** (and for attribute ***B***), the set { ***C*** } is not a superkey for ***R***.

Problem 4: Solution of Item 3

The answer is **no**. We show that there is no database instance that is correct w.r.t. schema ***S*** in which the set { ***D***, ***E*** } is not a superkey for ***Q***, by showing that a database instance ***I*** in which { ***D***, ***E*** } is not a superkey for ***Q*** is not correct w.r.t. ***S***.

Indeed, in ***I***, relation ***Q*** would contain two distinct tuples, say (*d*,*e*,*f1*) and (*d*,*e*,*f2*), with the same (non-null) values for both ***D*** and ***E***, and different values *f1* ≠ *f2 for* ***F***. Then, either ***I*** does not satify the foreign key constraint for ***Q***, and then ***I*** is not correct w.r.t. ***S***. Or ***I*** satifies the foreign key constraint for ***Q***, and then relation ***R*** contains the two tuples (*d*,*e*,*f1*) and (*d*,*e*,*f2*). But these two tuples in ***R*** have the same value *d* for attribute ***A*** and different values *f1* and *f2* for attribute ***C***. Hence, they violate the primary key constraint for relation ***R***, and ***I*** is not correct w.r.t. ***S***.