Introduction to Databases

Exam of 25/09/2024

With Solutions

Diego Calvanese

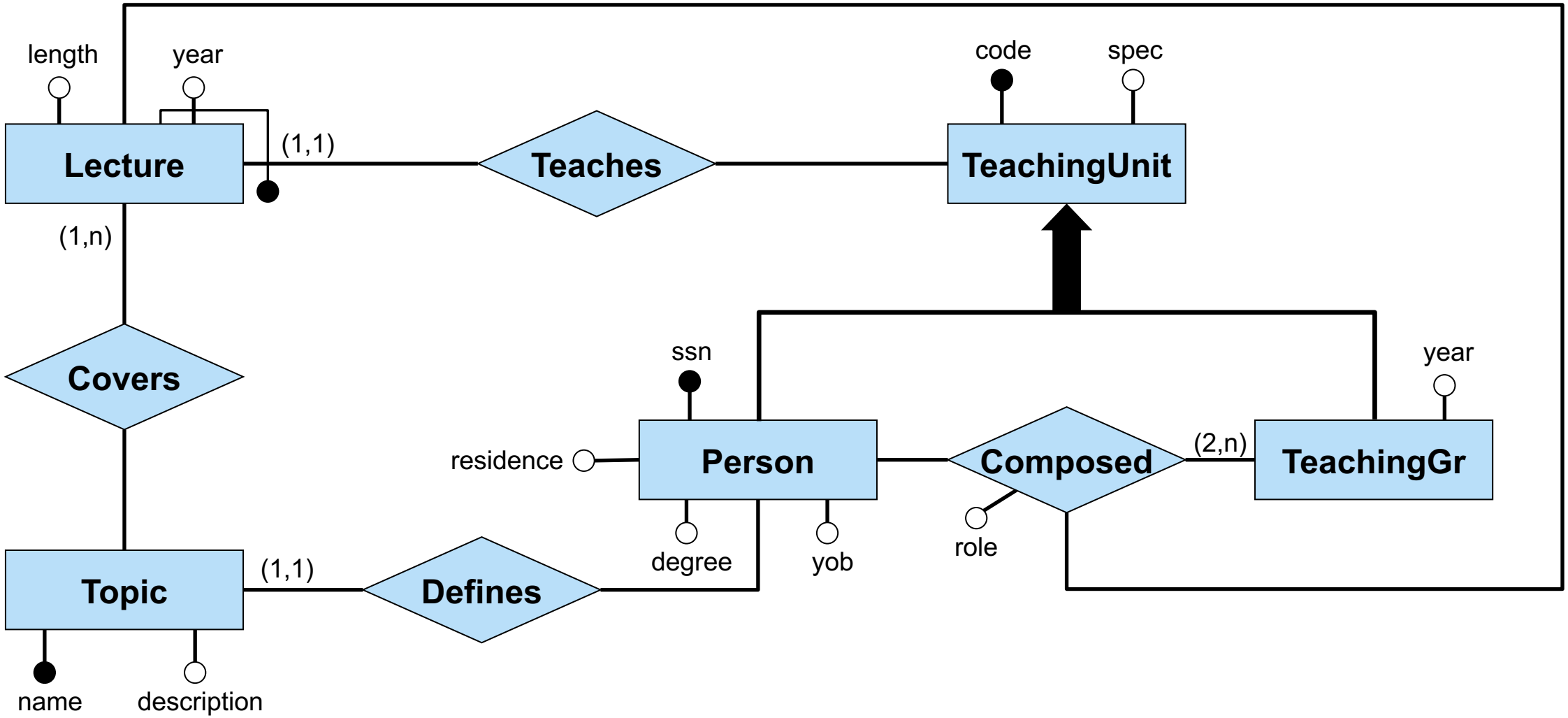
Bachelor in Computer Science
Faculty of Engineering
Free University of Bozen-Bolzano

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

Problem 1

Design the Entity-Relationship schema of an application for managing the assignment of lectures to contract teachers within a private school, for which the following information is of interest. Of each **lecture**, we are interested in the year in which it is held, the length (in hours), and the teaching unit that teaches it, where a teaching unit may teach at most one lecture per year. Each lecture covers one or more topics. Each **topic** is identified by its name, and of each topic we are also interested in the description and the main person who defined it. Notice that a topic may be covered by zero, one, or more lectures. Each **teaching unit** is identified by a code (assigned by the school for payment purposes), and we are interested in its specialization. There are exactly two types of teaching units: persons, and teaching groups. Of each **person** we are interested in the ssn (identifier), the year of birth, the residence, and the type of degree (BSc, MSc, PhD, MA,etc.). Of each **teaching group** we are interested in the year in which it was formed and its composition for the various lectures. Notice that the composition of a teaching group may vary between lectures, and for each lecture it is simply given by the persons (at least two) that are part of the teaching group for that lecture, together with the role played by the person (leader, assistant, translator, etc.).

Problem 1: Conceptual schema – Diagram



Problem 1: Conceptual schema – External constraints

For each instance *I* of the schema:

1. "For each teaching group and each lecture that determines its composition, the teaching group is composed of at least two persons."
   More formally: For each (TeachingGroup:*tg*, Lecture:*le*, Person:*p1*) ∈ *instances*(*I*,**Composed**), there is a *p2* ∈ *instances*(*I*,**Person**) with *p2* ≠ *p1* such that (TeachingGroup:*tg*, Lecture:*le*, Person:*p2*) ∈ *instances*(*I*,**Composed**).

2. "For each teaching group and each lecture that determines its composition, the lecture is taught by that teaching group."
   More formally: For each (TeachingGroup:*tg*, Lecture:*le*, Person:*p*) ∈ *instances*(*I*,**Composed**), we have that (TeachingUnit:*tg*, Lecture:*le*) ∈ *instances*(*I*,**Teaches**).

Problem 2

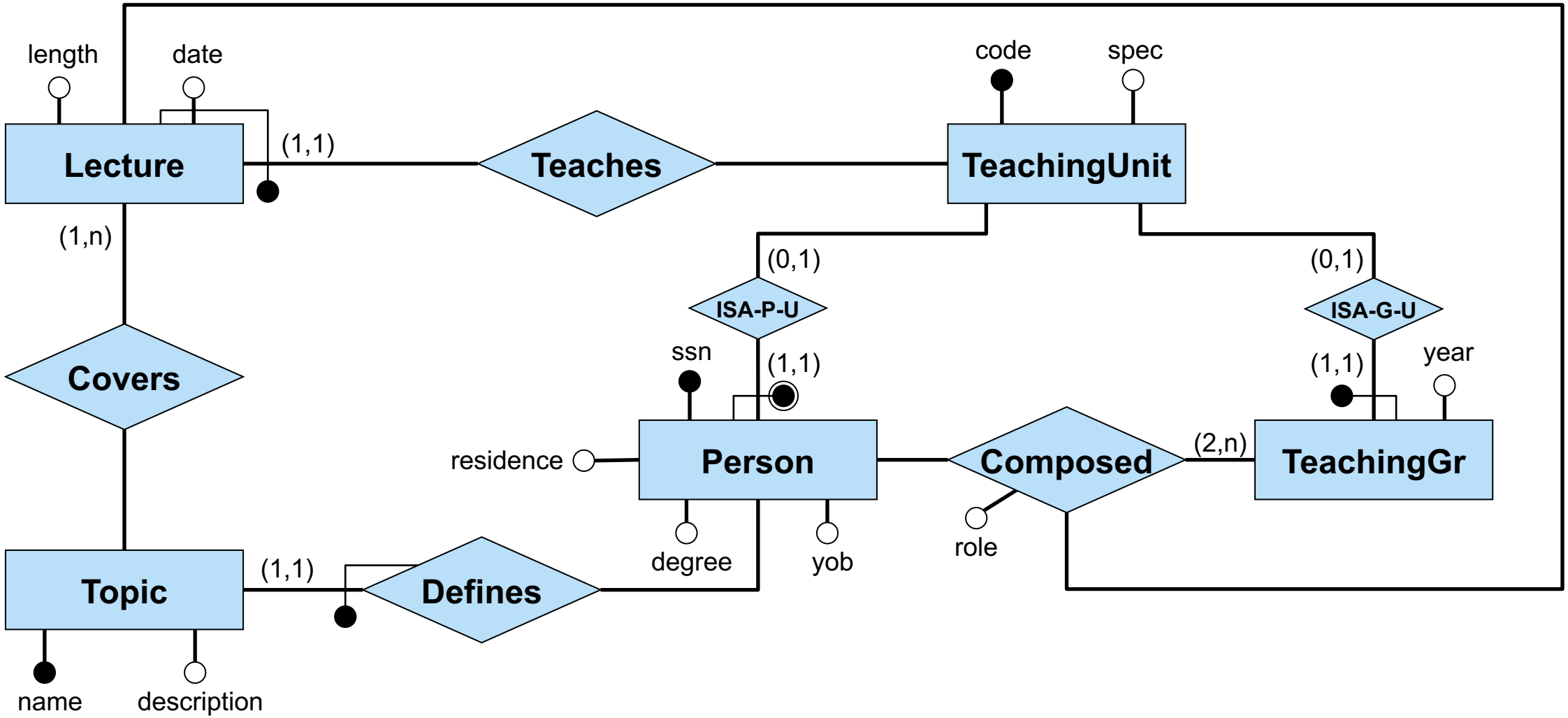
Carry out the logical design of the database, producing the complete relational schema with constraints, taking into account that when we access a topic, we also want to know the main person who defined it.

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

For each instance *I* of the schema:

1. For each (TeachingGroup:*tg*, Lecture:*le*, Person:*p1*) ∈ *instances*(*I*,**Composed**), there is a *p2* ∈ *instances*(*I*,**Person**) with *p2* ≠ *p1* such that (TeachingGroup:*tg*, Lecture:*le*, Person:*p2*) ∈ *instances*(*I*,**Composed**).

2. For each (TeachingGroup:*tg*, Lecture:*le*, Person:*p*) ∈ *instances*(*I*,**Composed**), we have that (TeachingUnit:*tu*, Lecture:*le*) ∈ *instances*(*I*,**Teaches**), where (TeachingGroup:*tg*, TeachingUnit:*tu*) ∈ *instances*(*I*,**ISA-G-U**).

3. For each *tu* ∈ *instances*(*I*,**TeachingUnit**), either there is a *p* such that (Person:*p*, TeachingUnit:*tu*) ∈ *instances*(*I*,**ISA-P-U**), or there is a *tg* such that (TeachingGroup:*tg*, TeachingUnit:*tu*) ∈ *instances*(*I*,**ISA-G-U**), but not both.

Problem 2: Direct translation

TeachingUnit(code, spec)

Person(code, ssn, yob, degree, residence)
- foreign key: Person[code] ⊆ TeachingUnit[code]
- key: ssn

TeachingGr(code, year)
- foreign key: TeachingGr[code] ⊆ TeachingUnit[code]
- inclusion: TeachingGr[code] ⊆ Composed[teachingGr]

Lecture(date, teachingUnit, length)
- foreign key: Lecture[teachingUnit] ⊆ TeachingUnit[code]
- inclusion: Lecture[date,teachingUnit] ⊆ Covers[lectureDate,teachingUnit]

Topic(name, description)
- foreign key: Topic[name] ⊆ Defines[topic]

Defines(topic, person)
- foreign key: Defines[topic] ⊆ Topic[name]
- inclusion: Defines[person] ⊆ Person[code]

Covers(lectureDate, teachingUnit, topic)
- foreign key: Covers[lectureData,teachingUnit] ⊆ Lecture[date,teachingUnit]
- foreign key: Covers[topic] ⊆ Topic[name]

Composed(teachingGr, person, lectureDate, teachingUnit, role)
- foreign key: Composed[teachingGr] ⊆ TeachingGr[code]
- foreign key: Composed[person] ⊆ Person[code]
- foreign key: Composed[lectureDate,teachingUnit] ⊆ Lecture[date,teachingUnit]
- tuple constraint: teachingGr = teachingUnit

External constraints:

1.
```
CHECK (2 <=
    ALL (SELECT COUNT(person)
         FROM Composed
         GROUP BY teachingGr,
                  lectureDate))
```
   Notice that we do not need to include also **teachingUnit** in the `GROUP BY` clause, due to the tuple constraint on **Composed**.
2. External constraint 2 is captured by the tuple constraint on **Composed**.
3. Person[code] ∩ TeachingGr[code] = ∅
   TeachingUnit[code] ⊆
   Person[code] ∪ TeachingGr[code]

Problem 2: Restructuring of the relational schema

Indication: When we access a topic, we also want to know the main person who defined it.

- We take into account the indication by merging relation **Defines** into **Topic**.
- Due to the tuple constraint on relation **Composed**, we can remove attribute **teachingUnit** and use **teachingGr** instead of **teachingUnit** in the foreign key constraint.

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.

The relation **Defines** is removed and the relation **Topic** is replaced by the following one:

Topic(name, description, person)
  foreign key: Topic[person] $\subseteq$ Person[code]

The relation **Composed** is replaced by the following one:

Composed(teachingGr, person, lectureDate, role)
  foreign key: Composed[teachingGr] $\subseteq$ TeachingGr[code]
  foreign key: Composed[person] $\subseteq$ Person[code]
  foreign key: Composed[lectureDate,teachingGr] $\subseteq$ Lecture[date,teachingUnit]

Problem 3

Consider a database *B* containing the two relations:

i. `Member(code,level)`, which stores the player code and level (an integer) of the members of a chess club;

ii. `Game(winner,loser)`, which stores the result of the games played between members of the club (where both `winner` and `loser` are foreign keys to `code` of `Member`).

We call "wow" a member who has won at least one game and *all* the games they have won are with members of equal or greater level than their own.

1. Assuming that there are no null values in the database, write a SQL query that computes the code of all wow members of the club.
2. Assuming that there may be null values in the database, but only for the attribute `level`, say whether the query written for Item 1 is correct or not. If the answer is yes, give reasons for the answer. If the answer is no, write the correct SQL query.

Problem 3: Solution 1 (1/2)

`Member(code,level)` `Game(winner,loser)`

1. Assuming that there are no null values in the database, write a SQL query that computes the code of all wow members of the club.

```
SELECT winner
FROM Game
  EXCEPT
SELECT winner
FROM Member W, Game G, Member L
WHERE W.code = G.winner AND
      L.code = G.loser AND
      L.level < W.level
```

Problem 3: Solution 1 (2/2)

`Member(code,level)` `Game(winner,loser)`

2. Assuming that there may be null values in the database, but only for the attribute `level`, say whether the query written for Item 1 is correct or not. If the answer is yes, give reasons for the answer. If the answer is no, write the correct SQL query.

The query for Item 1 is not correct when the database may contain null values for attribute `level`, since a member whose level is null is not returned by the query following `EXCEPT`, but such member might have won a game against a member of a lower level.

The query that correctly takes into account null values for attribute `level` is the following:

```
SELECT winner
FROM Game
  EXCEPT
SELECT winner
FROM Member W, Game G, Member L
WHERE W.code = G.winner AND L.code = G.loser AND
      (L.level < W.level OR W.level IS NULL OR L.level IS NULL)
```

Problem 3: Solution 2 (1/2)

`Member(code,level)` `Game(winner,loser)`

1. Assuming that there are no null values in the database, write a SQL query that computes the code of all wow members of the club.

```
SELECT DISTINCT code
FROM Member W JOIN Game G ON W.code = G.winner
WHERE W.level <= ALL
        (SELECT L.level
         FROM Member L JOIN Game G2 ON L.code = G2.loser
         WHERE W.code = G2.winner)
```

Problem 3: Solution 2 (2/2)

`Member(code,level)` `Game(winner,loser)`

2. Assuming that there may be null values in the database, but only for the attribute `level`, say whether the query written for Item 1 is correct or not. If the answer is yes, give reasons for the answer. If the answer is no, write the correct SQL query.

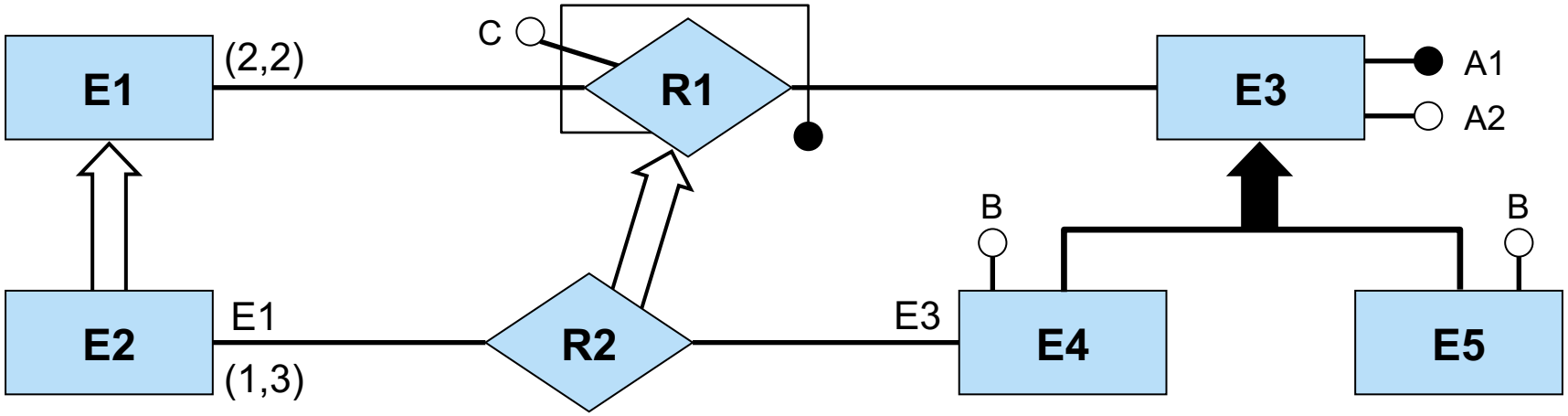
The query for Item 1 is correct also when the database may contain null values for attribute `level`, for the following reasons:

- A member whose level is null should not be returned by the query, since we cannot know for sure that they have won only games with members of equal or greater level than their own. And indeed such member is not returned, since the comparison `NULL <= ALL (...)` is false.
- Also, if a member has won a game with another member whose level is null, we cannot know for sure that the level of the loser was equal or greater than that of the winner, and the winner should not be returned by the query. And also in this case the comparison `W.level <= ALL (...)` is false, since `W.level <= NULL` is false.

Problem 4

Consider the ER schema *S* shown below and answer the following questions:

1. What problems, if any, does schema *S* suffer from that impair its quality?
2. What transformations should be performed on schema *S* to obtain a schema *S'* equivalent to *S* in which quality is maximized?



Problem 4: Solution

Schema S has the following problems, which impair its quality:

1. The identifier of relationship *R1* is not essential, since it includes the attribute *C*, in addition to *all* roles of the relationship. It should be removed, so that the only identifier of *R1* is the implicit one.
2. The two attributes *B* of entities *E4* and *E5* are intensionally redudant, since *E4* and *E5* are the only two child entities of a complete hierarchy. They should be replaced by an attribute *B* of *E3*.
3. The maximum cardinality 3 for the participation of *E2* to relationship *R2* is not tight, since *E2* inherits from *E1* the maximum cardinality 2 for the participation to *R1*, hence also to *R2*. Hence, the maximum cardinality 3 should be replaced by 2.