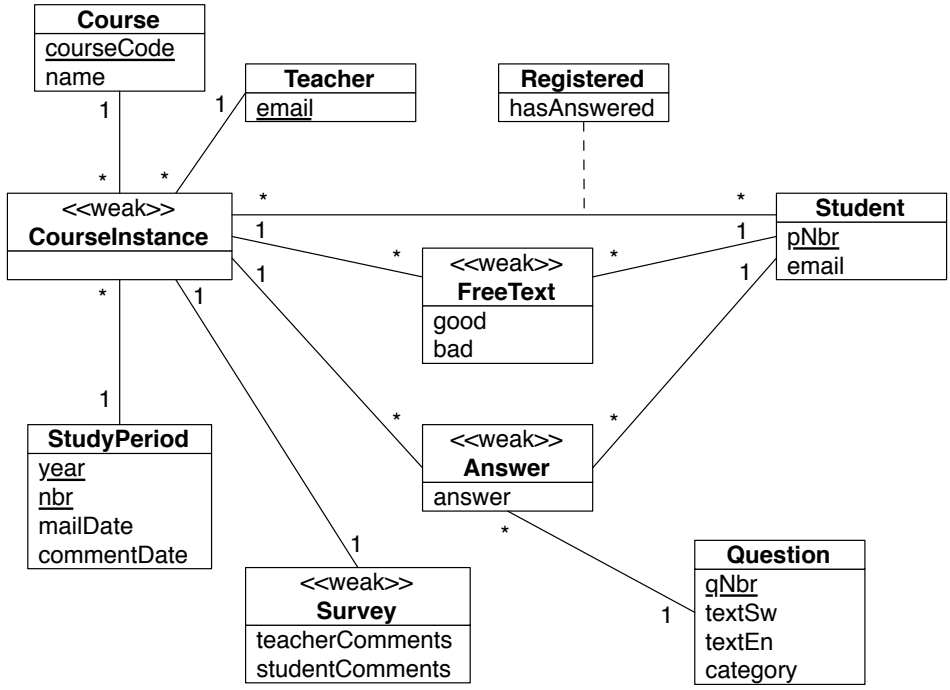
Solutions, Database Technology Examination

2012–03–05

1. E/R diagram:



Comments: The weak entity set CourseInstance has the key courseCode + year + periodNbr. This is too long: we invent the key instanceId. CourseInstances and Surveys could be combined. hasAnswered isn't necessary (can instead check if the student has an entry in Answers).

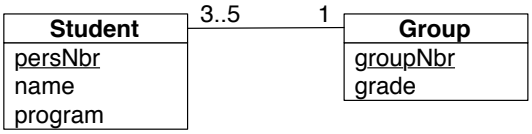
Relations:

Courses(courseCode, name)
StudyPeriods(year, nbr, mailDate, commentDate)
CourseInstances(instanceId, *courseCode*, *year*, *periodNbr*, *teacherEmail*)
Teachers(email)
Students(pNbr, email)
Registered(*pNbr*, *instanceId*, hasAnswered)
Questions(qNbr, textSw, textEn, category)
Surveys(*instanceId*, teacherComments, studentComments)
FreeTexts(*pNbr*, *instanceId*, good, bad)
Answers(*pNbr*, *instanceId*, *qNbr*, answer)

There are no other functional dependencies except for the key dependencies, so the relations are in BCNF. Average of Q19 for EDA-courses:

```
select courseCode, avg(answer)
from Answers natural join CourseInstances
where qNbr = 19
      and courseCode like 'EDA%'
      and year = 2012
group by courseCode;
```

2. E/R diagram:




```
create table Students {
    persNbr  char(11),
    name     varchar(30) not null,
    program  char(2) not null,
    groupNbr int not null,
    primary key (persNbr),
    foreign key (groupNbr) references Groups(groupNbr)
);

update Students
set groupNbr = 9
where name = 'Bo Ek' and groupNbr = 7;

select groupNbr, name, program, grade
from Students natural join Groups
order by groupNbr, name;

select name, program, grade
from Students natural join Groups
where grade = (select max(grade) from Groups);

select program, avg(grade)
from Students natural join Groups
where grade > 0
group by program
order by program;
```

3. We have the following functional dependencies:

   FD1. $AC \rightarrow D$
   FD2. $BC \rightarrow D$
   FD3. $A \rightarrow B$
   FD4. $BD \rightarrow A$

   FD1 can be derived from FD3 and FD2:

   $$\{AC\}^+ \Rightarrow \{AC\} \overset{FD3}{\Rightarrow} \{ACB\} \overset{FD2}{\Rightarrow} \{ACBD\}$$

   Note that you cannot derive FD2 from the other FD's. Here's a counter example where FD1, FD3, and FD4 hold, but FD2 doesn't hold:

A	B	C	D
1	1	3	4
2	1	3	5

   Keys: the attribute C doesn't appear on the right-hand side of any of the dependencies, so it must be included in the key. Closure of this attribute:

   $$\{C\}^+ \Rightarrow \{C\}$$

Two-attribute subsets which include C:

$$\{AC\}^+ \Rightarrow \{AC\} \overset{FD3}{\Rightarrow} \{ACB\} \overset{FD2}{\Rightarrow} \{ACBD\}$$
$$\{BC\}^+ \Rightarrow \{BC\} \overset{FD2}{\Rightarrow} \{BCD\} \overset{FD4}{\Rightarrow} \{BCDA\}$$
$$\{CD\}^+ \Rightarrow \{CD\}$$

{AC} and {BC} are keys. There are no three-attribute subsets that include C but not AC or BC, so they are the only keys.

FD3 and FD4 violate the BCNF condition, since the left-hand sides of these dependencies aren't superkeys. But the right-hand sides of these dependencies, B and A, are parts of a key, so the relation is in 3NF.

We decompose starting from FD3, $A \rightarrow B$, and get:

R1(A, B)	in BCNF, $A \rightarrow B$
R2(A, C, D)	in BCNF, $AC \rightarrow D$

4. An unnormalized relation contains redundancy (the same thing expressed in more than one place). A relation is normalized by splitting it in smaller relations.

   When normalized relations are used, a query must reconstruct the data by joining the smaller relations. This takes time, which may be a reason to use unnormalized relations. A typical example is a relation that stores a person's person number, name and address: Person(persNbr, name, postalCode, street, city). This relation isn't normalized, but you can live with the redundancy (that the postal code for a street is mentioned in many places). And it has the advantage that all address data is in one place, which probably often is good.

5. ```
   start transaction;
   select x from A lock in share mode;
   select y from B for update;
   commit; -- or rollback
   ```

6. a) The function must join the tables Competitions and Results, and then sum the points attribute. It does all this in Java code ... It can be done with one SQL statement:

   ```
   select sum(points)
   from Competitions natural join Results
   where sport = 'parameter 2' and contestant = 'parameter 1';
   ```

   b) The function is vulnerable to SQL injection, and it should use prepared statements instead of regular statements.
   ```