CS252.HACD — Solutions to Lecture Exercises (in Section 9)

Slide 5

Exercise: Write a **Tutorial D** constraint declaration to ensure that every `StudentId` value appearing in `IS_ENROLLED_ON` also appears as a `StudentId` value in `IS_CALLED`.

The simplest way is just

```
CONSTRAINT Only_known_students_are_enrolled
IS_EMPTY ( IS_ENROLLED_ON NOT MATCHING IS_CALLED ) ;
```

Slide 6

Exercise: Write a **Tutorial D** constraint declaration to enforce the FD `{StudentId}→{Name}` to hold in `ENROLMENT`.

There are several solutions. The logician might try:

```
WITH ENROLMENT { StudentId, Name } AS ESN :
IS_EMPTY ( ( ESN RENAME ( Name AS N1 ) JOIN
             ESN RENAME ( Name AS N2 ) )
           WHERE N1 <> N2 )
```

But rather slicker is

```
IS_EMPTY ( ENROLMENT GROUP ({ALL BUT StudentId} AS CN)
           WHERE COUNT(CN{Name}) > 1 )
```

Here we obtain, by grouping, for each `StudentId` value, a set of `{CourseId, Name}` pairs. The FD means that in every such set the `Name` value must be constant.

Perhaps even simpler:

```
WITH ENROLMENT { StudentId, Name } AS ESN :
COUNT ( ESN ) = COUNT ( ESN { StudentId } )
```

If any `StudentId` value appears with different Name values in distinct tuples of `ENROLMENT`, then these counts will not be the same; otherwise they will be.

Slide 24

(a) Write down the nontrivial functional dependencies (FDs) that hold in `fixture`. For each one, state whether its determinant is a key of `fixture`.

`{Date}` → `{Against, Venue, GoalsFor, GoalsAgainst, Result}`
`{Date}` is a key.

`{Against, Venue}` → `{Date, GoalsFor, GoalsAgainst, Result}`
`{Against, Venue}` is a key.

`{GoalsFor, GoalsAgainst}` ↦ `{Result}`
`{GoalsFor, GoalsAgainst}` is not a key.

(b) Decompose `fixture` into as few relvars as possible such that each one is in BCNF. Use Boris's notation for the relvars and state the key(s) of each one. For each BCNF relvar, state whether it is in 6NF.

`fixture {Against, Venue, Date, GoalsFor, GoalsAgainst}`
`{Date}` and `{Against, Venue}` are both keys.
`fixture` is not in 6NF.

`outcome {GoalsFor, GoalsAgainst, Result}`
`{GoalsFor, GoalsAgainst}` is the only key.
`outcome` is in 6NF.

(c) What problem, originally observed by Anne, has been solved in the BCNF design?

Because the score determines the result, two or more matches with the same score must always have the same result. The database would be inconsistent if, for example, a draw was given for a certain score of 0-0 but a win to Wargs for another 0-0 score. (Of course we know which one is incorrect, but BCNF alone does not provide a fix for that.)

(d) For any BCNF relvar that is not in 6NF, decompose it into an equivalent set of 6NF relvars having the same key.

I decompose `fixture` as follows, `Date` being the key in each case:

```
fixA  {Against, Date}
fixV  {Venue, Date}
fixGF {GoalsFor, Date}
fixGA {GoalsAgainst, Date}
```

(e) What problem, originally observed by Anne and remaining in the BCNF design, has been solved in the 6NF design?

They had agreed that it is desirable to be able to record the date and venue of a fixture before the match is actually played. Boris's original design makes that impossible except by means of a contrivance such as assigning values such as -1, -1 and "not played yet" to `GoalsFor`, `GoalsAgainst`, and `Result`, respectively. Such contrivances lead to needless complications for users of the database and are best avoided.

(f) What problem, originally observed by Anne, remains in the complete 6NF design?

The `Result` attribute is completely redundant because, as Boris explained, its value can always be deduced from `GoalsFor` and `GoalsAgainst`. Unless we declare the constraint that would be needed to ensure that the result is consistent with the score, the database is at risk of being inconsistent and therefore incorrect.

(g) What new problems did Boris observe with the 6NF design?

1. The constraint expressed by `{Against, Venue}` being a key of `fixture` has been "lost". The 6NF design permits any number of matches to be played against the same opponents at either venue.
2. Unless appropriate constraints are added, the design allows a tuple to exist in either of `fixA` and `fixV` for a certain `Date` without there being also a tuple for that `Date` in the other.
3. Similarly, unless appropriate constraints are added, the design allows a tuple to exist in either of `fixGF` and `fixGA` for a certain `Date` without there being also a tuple for that `Date` in the other.

(h) What compromise do you think Anne and Boris settled on, to solve all the problems they had both observed? Give the relvars and their keys, and state any other constraints that you think needs to be declared for this design.

They recombined the 6NF relvars into just two, discarding the `Result` attribute altogether, as follows:

`fixture {Against, Venue, Date}`
`{Date}` and `{Against, Venue}` are both keys.
`outcome {GoalsFor, GoalsAgainst, Date}`
`{Date}` is the only key and is also a foreign key referencing `fixture`. As **Tutorial D** doesn't support FOREIGN KEY syntax, the constraint must be declared like this:

```
CONSTRAINT outcomeFK
     IS_EMPTY (outcome NOT MATCHING fixture);
```

End of solutions