Exercises 6 – SQL (updates and triggers)

Bases de Dados, FCT-NOVA

Ano letivo 2015/16

Group 1. The university database of the previous exercises does not store any information about the non-teaching staff of the departments. In this group we will change the database to allow for that.

Of each staff member of a department we'd like to store the name, gender (that can only be `M` or `F`), and date in which the member started to work in the department. Each staff member should also have an identification number, but we don't care too much about what the number is. The best would be for the system to automatically take care of the numbers, e.g. assigning to each a sequencial number.

1. Create the table for staff, with the attributes as described above.
2. Impose the required integrity constrains (do not forget primary and foreign keys)
3. Create an SQL sequence for automatically assigning the id numbers of staff member. For creating a sequence use:

   ```
   CREATE SEQUENCE name_seq
   START WITH value
   INCREMENT BY step;
   ```

   The current value of an expression can be obtained with the pseudo column `CURRVAL`, and the next value with the pseudo column `NEXTVAL`.
4. Insert some staff into the database. To test the constraints, try to insert a staff member with gender `'N'` and a member of the staff of a department with code `26`.

Group 2. The database is also not prepared to store relationships between professors and courses. In particular, it is not prepared to store information about who is the professor responsible for each course, nor about who are the professores that teach in each course.

In this group we will prepare the database to store such information.

1. Create a table `teaching` to store who teaches in each course. Do not forget to define the appropriate integrity constraints. E.g. a professor cannot teach a non existing course, and a course cannot be taught by a non existing professor. Add some tuples to the table.
2. Change the database so that it allows to store who is the (single) responsible for each course. Note that the responsible for a course is not necessarily a professor that teaches in that course.
3. For each course in the database, add information about who is its responsible. Do it in such a way that the responsible for any given course is always a professor of the department of that course.
4. Add now integrity constraints that guarantee that the professor of each course is always a professor of the department of that course. To test the constraints, try to say that the professor with code `4` (who is from the Math department) is responsible for the `Bases de Dados` course.

Group 3. Let's now analyse how the current and previous categories of each professor are stored in the database. It doesn't make sense, does it?!

Note that there is nothing in the database that relates the current category with the previous ones! For instance, in the table (`historico_categorias`) with the previous categories of professor 4, there are entries for categories 2, 3, and 4. But the current category of professor 4 is 2!!

Moreover, by just storing one date in each previous category, it is impossible to know when the professor was in that category without assuming that she was always in a category, and the she always progressed from one category to the one immediately above.

Let's change that! We propose the following model for storing the current and previous categories of professors:

- In the `docentes` table, we will store the information of her current category (as it is now in the database).
- Each tuple of the `historico_categorias` table denotes a past category of a professor, with a reference for which is that category, which is the professor, the date in which she started and ended being in that category.
- The tuple with the current category of a professor has `null` in the ending date.
- The `historico_categorias` table can only be indirectly updated via updates to the current category of the professor made in the `docentes` table.
  - when a new professor is added, in `historico_categorias` one must add a tuple with her starting date;
  - when the attribute with the category of the professor is updated, one must register (in `historico_categorias`) that the professor started in that day to have that category, and that she ended having the previous category in the day before;
  - when a professor is deleted (from the database, of course!), one must delete all the corresponding information from `historico_categorias`.

To implement this in our database, there is an extra problem: what to do with the data that is already there? What we propose is that you assume the following

- The correct data is in the `historico_categorias` table (i.e. the categories in the `docentes` table should be ignored).
- The date now in the `historico_categorias` table is the starting date in that category. If there is a subsequent category in the table, one assumes that the ending date of the previous category is the day before the starting date of the subsequent category.
- The professors' progression were always without any interruptions, from one category to the one immediately above.

Let's do this, step by step...

1. Start by adding the attribute `end_date` to the table `historico_categorias`, and an integrity constraint to assure that that date is greater than the starting date.
2. To help in filling in the new attribute, create a view that outputs the category changes of professors. More precisely, create a view that, for each professor that ever changed (to a subsequent) category, returns: her code, the code of the previous category, the date in which she started in the subsequent category, and the date it ended having the previous category (the day before the one in which she started in the subsequent category).
3. Using the view you've just created, fill in the new attribute of `historico_categorias`.
4. Update the category of each professor in `docentes`, with her last category.

5. Create a trigger that, whenever a new professor is added, inserts the corresponding information in both the `historico_categorias` and the `docentes` table. Test the trigger by adding a new professor, a checking what ended up in both those tables.

6. Create a trigger guaranteeing that whenever a professor is deleted, all her information is deleted from both tables.

7. Finally, create a trigger that, whenever the category attribute in `docentes` is updated, inserts and changes the information in `historico_categorias` accordingly. I.e. registers the ending date in the previous category, and insert a new tuple for the new category. Test the trigger by promoting the professors 1 and 2 to the category immediately above the one in which they are now.