1 PROBLEM DESCRIPTION

On the DB designed for CONGATE, some queries, views, procedures, and triggers must be developed. The lecturers will provide a solution to 1st assignment that will serve as a common starting point for all students (so they have equal opportunities).

The first step will be, therefore, to run the scripts provided by the lecturers (new DB creation script, and script for inserting data into those new tables). Once this is done, a series of elements (composing this assignment) must be developed and documented.

The description of these elements is as follows:

1.1 Queries

a. For each product currently offered (in use), report about how many doctors could be appointed with that product (counting all the specialties covered and all the affiliated hospitals). Outputs: company name, company tax id, product name, version, (number of) coverages, (number of) doctors.
b. Products (currently in use) offering some coverage that they cannot satisfy (because it is not included among the services of any of the hospitals with which the company has a contract). Outputs: Company name, company tax id, product name, version, list of unsatisfied coverages (separated by the semicolon character ';').
c. For each specialty, minimum and maximum waiting periods, and brief desc of the product (including the name of the company that offers it, the name of the product and its version). If there are several "tied" companies, the product with the earlier release date will be chosen (if still tied, any of the products is chosen). Outputs: specialty, type of row (either 'minimum' or 'maximum' period), period in days, company name, company tax id, product name, and version. Sort the output alphabetically by specialty.

1.2 Operability (package with procedures/functions)

Define and create a package that contains at least the following elements:

- Variable *curr_user* (active or "current user")
- Procedure that allows assigning a value to the "current user" variable (it must be verified that the user, identified by his passport, is registered in the client table; the success of the operation must be reported on the display).
- Procedure to insert a new product for the current customer (active user). The procedure will allow specifying both the company and the product name, and will assign the most recent version of it. If the product is withdrawn, it should not insert it, and in any case it will report the result of the operation on the shell.
- Procedure to insert a new appointment for the current user, with the indicated specialty covered by the given policy, on the date provided, and with the specified doctor and

hospital. Before insertion, the validity of the appointment will be checked (the policy is valid on that date and covers that specialty, the hospital and the doctor are accessible with that company, and there is no other appointment with that doctor overlapping with the new one within ±15 minutes).

1.3 External Design: "*user*" *profile*

Users must have access to the *Overlaps*, *My_Coverages*, and *Recommendations* tables. These tables contain information derived from other tables in the DB, and should be implemented as (logical) views. All of them give access only to the data corresponding to the current user (whose identifier is currently stored in the "current user" variable of the package created in the former section).

- Overlaps: informs about overlapping coverages (as of today) related to the current user's policies (that is, whenever s/he has the same coverage in two products contracted by her/him and active today). This view should be "read only".
- My_Coverages: list of products contracted today (company, product, version) with their coverages for the current user. This view will be operational, allowing the insertion of a row: if that product has that coverage, what is inserted is a new policy for this user and that product, so that from now on the row will appear in this view (the row inserted); it will also allow the deletion of a row (the policy will be deleted, so that the row will no longer belong to the view); updates won't have effect on this view (no change made).
- Recommendations: list coverages that the current user does not have, and any current product which latest (active) version has that coverage. This view is also "read-only".

1.4 Active Databases

Implement one or more triggers to address each of the following needs:

A. Anytime a new version of a product is added, that version becomes the current one (without an specific expiration date) and all other previous versions will be obsolete (with a withdrawal date prior to or equal to the present time). The new version must be the highest value of version (if the value provided is not the highest, the insert is rejected). In addition, a client must be prevented from contracting an obsolete version.

B. Every time a client inserts a new appointment for any specialty, it will be inserted in the database only if there was no other previous appointment of that client for the same specialty. In case that appointment already exists, a new row won't be inserted, but the existing row will be modified (the date, the hospital, the doctor, ... will be changed by the values that new row had). For tracking appointment modifications, it is necessary to store (along with the appointments) the dates of creation (date on which it is inserted as a new appointment) and last modification (date on which the existing row is updated). These two new columns must be added to the Appointments table.

2 SUPPORTING MATERIALS

Apart from classes and tutoring sessions, students can count on the following resources:

- Documents: assignment statement (this doc);
  - class slides;
  - relational graph with solution to 1st assignment
  - template for writing the assignment report.
- Audiovisual resources: video classes to acquire specific knowledge about the use of the tools that will be used in the laboratories (console management and pl/sql syntax) in the 'inverted class' style.
- Sw Resources: user account on RDBMS Oracle (accessible from all computer rooms in the University, and from Aula Virtual), with enough privileges for all required operations and reading privileges on the obsolete DB's tables.
  - Script for creating the tables that implement the new Database.
  - Data migration script from the obsolete DB to the new DB.

3 TO DO

All the results of this ASSIGNMENT (designs, code, tests) will be collected in a single document (*assignment report*, in PDF format). For each section of the statement (worth 2.5/10 points each) establish a chapter in memory. Within each chapter, include a separate section for each element required in the statement. These sections must follow the same order that has been established in the statement.

Each of these sections will be presented with the description of the element to be developed (statement), and its resolution with three subsections, as described below: design, implementation, and testing.

- Design: queries and views should be described in relational algebra; as for code blocks, you have to describe their logic (working); finally, the parameterization of the ECA rules (triggers) must be justified. The design will be accompanied by relevant comments about the implicit semantics incorporated or the explicit not reflected (wherever necessary).
- Implementation: PL/SQL code that implements that element. The code must be collected in text format (so that it can be easily transferred onto the sql*plus console with copy-paste operations) and properly indented to improve its readability.

- Tests: description of the actions to be carried out to verify the correct working of the implemented element, description of the expected result, and description of the result obtained (accompany by screenshots to illustrate that result). When the result obtained differs from that expected, explain the deviation, the problem detected (if any) and the actions necessary to correct.

Examples:

- A query can be verified by running it several times on as many different states of the DB, and establishing in advance the differences between the results that should be obtained. Note that a simple execution is not a test (it is not known whether its result is correct or not). But by changing the state of the DB is changed (inserting, deleting and/or updating data) so that the result of the query change, and so that the result can be either predicted or characterized in advance, then useful test cases can be established. Exhaustive testing is not requested in this work (just include some examples of testing).
- Checking views is analogous to checking queries.
- Blocks (procedures) can be tested by running with different parameters (checking results).
- A trigger can be checked by causing it to be activated in different cases (forcing the triggering event) and checking its effect. If there is a risk of a mutating table error, it is convenient to check it by means of activations that involve more than one row.

Document all the work carried out by means of the pertinent *Labwork Report*, for which writing a template is provided. Apart from including the requested elements with their development stages (design, implementation, test), make sure that all design decisions are conveniently justified and thus reflected in the report. Save the document as **.pdf** file (portable document format), name it as ***nia1_nia2_nia3_LW2.pdf*** , and submit it through Aula Global.