1. – PROBLEM DESCRIPTION

On the relational database already designed for the Bib-buses foundation (*FOUNDICU*® Org.), it is necessary to develop some operative elements. Specifically queries, views, procedures, and triggers. The description of these elements is as follows:

1.1.- Queries (2 points, 1 point each)

a) BoreBooks: books with editions in at least three different languages, of which no copy has ever been loaned.

b) Reports on Employees: for each driver, provide their full name, age, seniority contracted (whole years), active years (years on road), number of stops per active year, number of loans per active year, percentage of unreturned loans.

1.2.- Operativity (package 'foundicu' containing these public procedures) (2.5 points, distributed as 1+1+0.5 respectively)

- Insert Loan Procedure: receives a signature; checks current USER exists, and that there is a hold on the copy (signature) for the current user as of today, and turns that reservation into a loan; otherwise (no reservation), checks that the copy is available for two weeks, and that the current user has not reached the upper limit for loans (nor they are sanctioned); when loan is possible inserts a new loan row, else reports about the problem.
- Insert Reservation Procedure: receives an ISBN and a date; checks that the current USER exists and has quota for reserving (has not reached the upper borrowing limit, and they are not sanctioned); checks the availability of a copy of that edition for two weeks (14 days) from the date provided, and then places the hold (else, reports the hinder).
- Record Books Returning Procedure: receives a signature; checks current USER has borrowed that book , and records its return (else, reports on the impediment).

1.3.- External Design: *bibuses services user* profile (2.5 points, respectively 0.5+1+1).

For this section you need to define the concept of '*current user*'. You can decide either to take the value provided by nullary function USER (inserting that 'user' for testing) or introduce into the former section's (1.2) package a public variable (for example, *current_user CHAR(10)*, with a procedure for assigning and a function for returning).

- **`my_data`** (*read only*) view: reads current user's personal data (id, full name, address, …)
- **`my_loans`** (*operable*) view: returns all past loans from current USER, along with their posts if any (when no post on a loan, null values will be returned). Apart from reading, updating 'post' attribute is allowed (if so, post_date is automatically set to current date and time; if former post on that loan was null, likes and dislikes are automatically set to zero). No other column can be updated. Deletions and insertions are also not allowed.

only allowed operation my_loans set post to .....

- `my_resevations` (*operable*) view: shows all current users reservations. Allows inserting, deleting, and changing dates (provided that the book, or any other copy of the same ISBN, is available).

1.4.- Active Databases/Triggering (3 points, one point each**)**
Design, implement, and test a solution to **three** of the following four trigger problems*:

a) Prevent 'posts' from institutional users (municipal libraries).

b) When the condition of a copy is set to 'deteriorated', the 'deregistration date' is automatically set to 'current date and time'.

history_users
history_loans

c) Create a 'history table' of both users and loans (not views, but another two tables). When a user is removed, create a history record of that user, and move all their loans to the history loans.

d) Add a column 'reads' to Books. When a book is loaned, update the number of reads.

(*) Note: you should choose three problems to solve (discard one); each of these problems will require the creation of at least one trigger (you can create several of them for any problem if you deem it appropriate)

2. – STARTING POINT

For the development of this practical work, a completely new database will be available, already populated with the data from the previously available database. To replicate this database, creation and loading scripts will be provided, along with a brief documentation on this design (relational graph, and a subset of the most relevant semantic comments).

The first step in carrying out this work is therefore to run these scripts to replicate the development environment (recreating this database in your Oracle DB account).

3. – SUPPORTING MATERIALS

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

- Documents:
  - assignment statement (this doc);
  - class slides;
  - solution to the first assignment (design and comments)
  - template for writing the assignment report (.docx format).

- Audiovisual resources: video classes to acquire specific knowledge about the use of the tools that will be used in the laboratories (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. Scripts for replicating environment:
    - Script for creating the tables that implement the new Database.
    - Data migration script from the obsolete DB to the new DB.

4. – 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 first item in the statement (queries, package, external design, triggers) include a chapter in your assignment's report. 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**. Notice that those subsections are worth **30%**, **40%** and **30%** of the **score** (respectively), so neglecting any of them can involve severe score loss.

A. **Design**: queries should be described in relational algebra (views and subqueries must also be documented); 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). In case, new/modified elements (such as, tables) will be documented.

B. **Implementation**: PL/SQL code that implements that element. The code must be collected in text format (so that it can be easily transferred by copying from the .pdf document and pasting onto the sql*plus console for eventual running checks). Besides, the code has to be properly indented to improve its readability, and commented in any aspect not included in the former section (design).

C. **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.

no exhausting testing needed: one good test per subsection is enough.

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.

Just one student from each team should deliver.

Deadline: April 4, 2025