Exercises involving Views, Procedures, Functions and Triggers

Consider the instance implemented and populated in SQL in the last activity for the LIBRARY relational database schema in the next page, which is used to keep track of books, borrowers, and book loans. Referential integrity constraints are shown as directed arcs in the figure.

Implement the following exercises using both PostgreSQL (PL/pgSQL) and Oracle Database (PL/SQL) a sequence of SQL instructions to perform the following actions granting that no stored data is lost or damaged. Submit a file with the script for PostgreSQL and another file for the Oracle's script, clearly delimiting the exercises in each script.

1. Create a materialized view month_borrowers that shows the data (card_no, name, address, and phone) from the borrowers who had (or have) more than one loan whose length (i.e., the difference between the date out and the due date) is greater than or equal to 30 days. The view should also show the loan length, the book title and the branch name of these loans. Besides, perform updates to the base tables, eventually run statements to make the DBMS update the view, and show the state of the up-to-date materialized view.

2. Assume that after having deployed the database and loaded it with data, the library manager decided to store book copies individually. From now on, the database should store an id, the acquisition date, and the current conditions (fine, good, fair, or poor) for each copy. Book loans should refer to a specific book copy. The attribute no_of_copies should no longer be stored in the database. However, existing applications should "see" the database as if the schema had not been updated for backward compatibility.

- Create a temporary table to save the current number of copies of each book in each branch.
- Implement the commands to perform the necessary change to the database schema.
- Write a query or a function that returns/shows the inconsistencies between the number of copies of a book in a branch in the updated schema and the respective number of copies in the temporary table. That is, identify the cases these numbers do not match.
- Insert book copies into the updated schema until there is no more such an inconsistency. Commit the changes and drop the temporary table.
- Create a view with the same name as the old table (book_copies), showing the same content. That is, an application could interact with the view as it was the old table.
    - A delete from the view should trigger the deletion of the tuples corresponding to the book copies.
    - An update on the view to attributes book_id or branch_id should be automatically redirected to the base tables.
    - An update on the view reducing the number of copies cannot be accepted.
    - An insert into the view or an update on the view increasing the number of copies should trigger the insertion of book copy tuples such that the number of copies matches the value provided to the update statement. The new tuples' id should be set using a sequence, and their acquisition date should be the current date.
- Show the database state before and after executing inserts, updates and deletes on the view, showing that the triggered actions satisfy all the specifications' points.

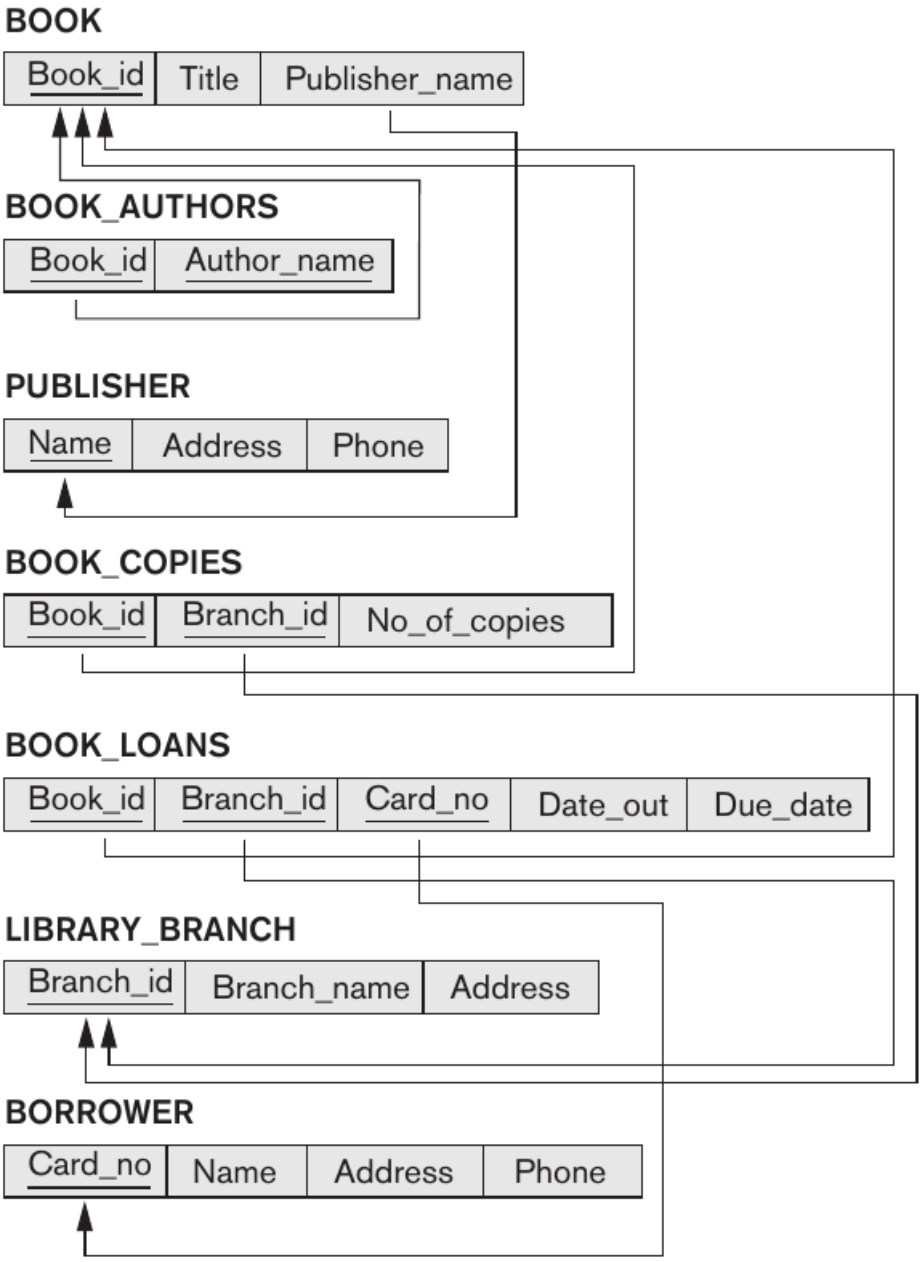
3. Write a procedure to detect and reconcile authors who have inconsistent entries in the book authors table. This procedure is an example of entity matching, which is a complex task but will be quite simplified for this exercise.

Assume that authors can have inconsistent entries due to typos (e.g., John Joseph Powell and John Joseph Powel) or abbreviations (e.g., John Joseph Powell and John J. Powell). Assume also that typos are constrained to at most two non-matching characters. This contraint allows detecting missing (e.g., Powel) or exceeding caracters (e.g., Poweell), and character switches (e.g., Pwoell). Also, assume that author names are either: i) given name, middle name, and surname (e.g., John Joseph Powell); or ii) given name, middle initial, and surname (e.g., John J. Powell). Duplicates regarding middle name or middle initial must match the first letter (i.e., John Joseph Powell and John A. Powell are not duplicates).

The procedure should:

- detect duplicates;
- identify the most frequent between the duplicated values and eliminate the duplication by updating the least frequent value with the most frequent one (e.g., if there is 5 occurrences of John J. Powell in the database and only one occurrence of John J. Pwoell, then the procedure should update John J. Pwoell to John J. Powell);
- log every duplicate elimination, saving the book id, the old author name, the new author name, and the update timestamp to a log table.

Notice that your procedure implementation can rely on cursors, call built-in or user-defined functions, use triggers, etc. Use the DBMS resources as needed. Submit a script containing all statements needed to implement and test the procedure (i.e., the script should include proper data insertions to test the procedure).


BOOK
Book_id
Title
Publisher_name
BOOK_AUTHORS
Book_id
Author_name
PUBLISHER
Name
Address
Phone
BOOK_COPIES
Book_id
Branch_id
No_of_copies
BOOK_LOANS
Book_id
Branch_id
Card_no
Date_out
Due_date
LIBRARY_BRANCH
Branch_id
Branch_name
Address
BORROWER
Card_no
Name
Address
Phone