| **Lecturer:** | **Javier Calle** | | |
| --- | --- | --- | --- |
| **Group:** | **286** | **Lab User** |  |
| **Student:** | **Alberto** | **NIA:** | **100495775** |
| **Student:** | **Silvia Bayo Martínez** | **NIA:** | **100495811** |
| **Student:** | [**CARMEN SERRANO PEREZ**](mailto:100495711@alumnos.uc3m.es) | **NIA:** | **100495711** |

# INDEX

[**INDEX**](#_heading=) **1**

[**1 Introduction**](#_heading=) **1**

[**2 Queries**](#_heading=) **2**

[QUERY 1: BoreBooks](#_heading=h.e8u5mxcbagz0) 2

[QUERY 2: Reports on Employees](#_heading=h.wrqn5cdx5t14) 4

[**3 Package**](#_heading=) **4**

[**4 External Design**](#_heading=) **4**

[**5 Explicitly required Triggers**](#_heading=) **5**

[**6 Concluding Remarks**](#_heading=) **5**

# Introduction

This report documents the design, implementation, and testing of different SQL tools applied to the FOUNDICU® relational database as part of the second assignment. These tools include SQL queries, procedures, views, and triggers.

The objective of this task is to consolidate and expand on the relational database created in the first assignment, by introducing practical SQL functionalities that enhance querying and operability.

The document is divided into four main sections:

* Queries: Includes relational algebra, SQL implementation, and test design.
* Packages: Procedure implementations with logic and tests.
* External Design: Implementation of user profile views and their operativity.
* Triggers: Implementation and validation of required triggers.

Each of these sections follows a consistent structure: Design → SQL Implementation → Tests.

* Some part of the code can be found in the public github repository: <https://github.com/Albrtito/UC3M-FILES-P2.git>

# Queries

This section is devoted to describing the mathematical solution to the queries proposed in the problem description (in Relational Algebra). It should also include the SQL translation for each query, and the description and results of the tests designed to check the validity of each solution. Thus, this chapter is divided in several sections (one per query) and each section has three items:

* Relational Algebra
* SQL
* Tests

## QUERY 1: BoreBooks

Books with at least three different language editions and no recorded loans.

Relational Algebra

| LangCount ← γ title, author; COUNT(DISTINCT language) → lang\_count (EDITIONS)  ValidBooks ← σ lang\_count ≥ 3 (LangCount)  Loaned ← π title, author (  σ l.type = 'L' (  ((EDITIONS ⨝ EDITIONS.isbn = COPIES.isbn COPIES)  ⨝ COPIES.signature = LOANS.signature LOANS)  )  )  ValidNotLoaned ← π b.title, b.author (  (BOOKS ⨝ BOOKS.title = ValidBooks.title ∧ BOOKS.author = ValidBooks.author)  ) − Loaned |
| --- |

SQL

****WITH Books\_Lang\_Count AS (

SELECT title, author, COUNT(DISTINCT language) AS lang\_count

FROM editions

GROUP BY title, author

),

Valid\_Books AS (

SELECT title, author

FROM Books\_Lang\_Count

WHERE lang\_count >= 3

),

Loaned\_Books AS (

SELECT DISTINCT e.title, e.author

FROM editions e

JOIN copies c ON e.isbn = c.isbn

JOIN loans l ON c.signature = l.signature

WHERE l.type = 'L'

)

SELECT DISTINCT b.title, b.author

FROM books b

JOIN Valid\_Books vb ON b.title = vb.title AND b.author = vb.author

WHERE NOT EXISTS (

SELECT 1

FROM Loaned\_Books lb

WHERE lb.title = b.title AND lb.author = b.author

)

ORDER BY b.title;



Tests

**Test 1 – Book with 3 editions (different languages), no loans**

****-- Insert book

INSERT INTO books (title, author, country, language, pub\_date, topic)

VALUES ('TEST1', 'Serrano, Carmen', 'Spain', 'Spanish', 2010, 'Fiction');

-- 3 editions (different languages)

INSERT INTO editions (isbn, title, author, language, national\_lib\_id)

VALUES ('ISBN1001', 'TEST1', 'Serrano, Carmen', 'Spanish', 'NL1001');

INSERT INTO editions (isbn, title, author, language, national\_lib\_id)

VALUES ('ISBN1002', 'TEST1', 'Serrano, Carmen', 'English', 'NL1002');

INSERT INTO editions (isbn, title, author, language, national\_lib\_id)

VALUES ('ISBN1003', 'TEST1', 'Serrano, Carmen', 'French', 'NL1003');

-- Copies with no loans

INSERT INTO copies (signature, isbn, condition)

VALUES ('SIG01', 'ISBN1001', 'G');

INSERT INTO copies (signature, isbn, condition)

VALUES ('SIG02', 'ISBN1002', 'G');

INSERT INTO copies (signature, isbn, condition)

VALUES ('SIG03', 'ISBN1003', 'G');

Expected result: The book “TEST1” should appear in the result.

Result:

**Test 2 – Add a loan to the previously created copy**

****-- Insert user

INSERT INTO users (

user\_id, id\_card, name, surname1, surname2,

birthdate, town, province, address, email,

phone, type, ban\_up2

)

VALUES (

'USR0001', '11223344A00000001', 'Lucía', 'Ortega', 'Salas',

TO\_DATE('1999-09-09', 'YYYY-MM-DD'),

'Villaverde', 'Madrid',

'Av. de los Libros 15', 'lucia@books.com',

600123456, 'R', NULL

);

-- Insert book and editions (3 languages)

INSERT INTO books (title, author, country, language, pub\_date, topic)

VALUES ('TEST1', 'Serrano, Carmen', 'Spain', 'Spanish', 2010, 'Fiction');

INSERT INTO editions (isbn, title, author, language, national\_lib\_id)

VALUES ('RISBN001', 'TEST1', 'Serrano, Carmen', 'Spanish', 'NL101');

INSERT INTO editions (isbn, title, author, language, national\_lib\_id)

VALUES ('RISBN002', 'TEST1', 'Serrano, Carmen', 'English', 'NL102');

INSERT INTO editions (isbn, title, author, language, national\_lib\_id)

VALUES ('RISBN003', 'TEST1', 'Serrano, Carmen', 'French', 'NL103');

-- Insert copies

INSERT INTO copies (signature, isbn, condition)

VALUES ('RSIG1', 'RISBN001', 'G');

INSERT INTO copies (signature, isbn, condition)

VALUES ('RSIG2', 'RISBN002', 'G');

INSERT INTO copies (signature, isbn, condition)

VALUES ('RSIG3', 'RISBN003', 'G');

-- Insert a loan (signature RSIG1)

INSERT INTO loans (signature, user\_id, stopdate, town, province, type, time)

VALUES (

'RSIG1',

'USR0001',

TO\_DATE('22-11-2024', 'DD-MM-YYYY'),

'Sotomelones de las Caballerizas',

'Albacete',

'L',

20

);

Expected result: “TEST1” should no longer appear in the result.

Result:



**Test 3 – Book with only 2 languajes**

****-- Book with just 2 languajes

INSERT INTO books (title, author, country, language, pub\_date, topic)

VALUES ('TEST2', 'Carlos Ruiz', 'Spain', 'Spanish', 2012, 'Psychology');

INSERT INTO editions (isbn, title, author, language, national\_lib\_id)

VALUES ('ISBN2001', 'TEST2', 'Carlos Ruiz', 'Spanish', 'NL2001');

INSERT INTO editions (isbn, title, author, language, national\_lib\_id)

VALUES ('ISBN2002', 'TEST2', 'Carlos Ruiz', 'Catalan', 'NL2002');

INSERT INTO copies (signature, isbn, condition)

VALUES ('SIG04', 'ISBN2001', 'G');

INSERT INTO copies (signature, isbn, condition)

VALUES ('SIG05', 'ISBN2002', 'G');

Expected result: “TEST2” should not appear in the result.

Result:



## 

## QUERY 2: 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.

Relational Algebra

* The arrow ← is being used as the renaming operator

| #Select the important characteristics from the drivers table and rename it driver\_base driver\_base ← π d.passport, d.fullname, d,birthdate, d.cond\_start, d.cont\_end TRUNC(MONTHS\_BETWEEN(SYSDATE, d.birthdate)/12)→ age, TRUNC(MONTHS\_BETWEEN(NVL(d.cond\_end, SYSDATE))/12) → seniority\_years,  TRUNC(MONTHS\_BETWEEN(NVL(d.cond\_end, SYSDATE))/12) → active\_years (drivers d)  driver\_stops ← π d.passport, COUNT(DISTICT(town || province) → total\_stops (drivers d ⨝ assign\_drv(passport) ⨝ services (passport AND taskdate)) GROUP BY d.passport  driver\_loans ← π d.passport, COUNT(l.signature) → total\_loans, COUNT(l.return) → unreturned\_loans (drivers d ⨝ services s(d.passport = s.passport AND ad.taskdate = s.taskdate) ⨝ loans l (s.town = l.town AND s.province = l.province AND s.taskdate = l.stopdate) GROUP BY d.passport.  #Begin select query with the subqueries. The functions used to handle division by 0 errors are not included in the relational algebra  π db.fullname, db.age, db.seniority\_years, db.active\_years, ds.total\_stops/db.active\_years, dl.total\_loans/db.active\_years, dl.unreturned\_loans x 100 / dl.total\_loans ( driver\_base db ⨝ driver\_stops (passport) ⨝ driver\_loans(passport) ORDER BY db.fullname |
| --- |

SQL

****-- For each driver, provide:

-- 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

-- Get the stops per active year for each driver

WITH

-- Get the info that we need from the drivers table into an smaller table

-- passport (so that we can still identify), Full name, birthdate age(needs to be computed), seniority(needs to be computed), active years(needs to be computed), we also maintain the contract start and end dates

driver\_base AS (

SELECT

d.passport,

d.fullname,

d.birthdate,

d.cont\_start,

d.cont\_end,

-- Calculate age in years

TRUNC(MONTHS\_BETWEEN(SYSDATE, d.birthdate) / 12) AS age,

-- Calculate seniority (contract length) in years.

-- Use NVL to handle NULL end dates, if no end date, then use SYSDATE

TRUNC(MONTHS\_BETWEEN(NVL(d.cont\_end, SYSDATE), d.cont\_start) / 12) AS seniority\_years,

-- Dont get what's the difference between this and the one above, so add it here as the same thing

TRUNC(MONTHS\_BETWEEN(NVL(d.cont\_end, SYSDATE), d.cont\_start) / 12) AS active\_years

FROM

drivers d

),

driver\_stops AS (

SELECT

d.passport,

COUNT(DISTINCT s.town || s.province) AS total\_stops

FROM

drivers d

LEFT JOIN

assign\_drv ad ON d.passport = ad.passport

LEFT JOIN

services s ON ad.passport = s.passport AND ad.taskdate = s.taskdate

GROUP BY

d.passport

),

-- Get the stops per active year for each driver: Finally this is not used

-- because the computation is done directly in the final select query

driver\_stops\_average AS(

SELECT

ds.passport,

db.fullname,

db.active\_years,

ds.total\_stops / db.active\_years AS stops\_per\_active\_year

FROM

driver\_stops ds

LEFT JOIN

driver\_base db ON ds.passport = db.passport

),

-- Count loans and unreturned loans for each driver

driver\_loans AS (

SELECT

d.passport,

COUNT(l.signature) AS total\_loans,

COUNT(CASE WHEN l.return IS NULL THEN 1 END) AS unreturned\_loans

FROM

drivers d

LEFT JOIN

assign\_drv ad ON d.passport = ad.passport

LEFT JOIN

services s ON ad.passport = s.passport AND ad.taskdate = s.taskdate

LEFT JOIN

loans l ON s.town = l.town AND s.province = l.province AND s.taskdate = l.stopdate

GROUP BY

d.passport

)

-- Query using all subqueries defined above

-- For each driver, provide:

-- 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

SELECT

db.fullname AS "Driver Name",

db.age AS "Age",

db.seniority\_years AS "Seniority (Years)",

db.active\_years AS "Active Years",

-- Calculate stops per active year

CASE

-- Handle division by zero for zero active years. Just been there some months

WHEN db.active\_years = 0 THEN ds.total\_stops

ELSE ROUND(ds.total\_stops / db.active\_years, 2)

END AS "Stops per Active Year",

-- Calculate loans per active year

CASE

-- Handle division by zero for zero active years. Just been there some months

WHEN db.active\_years = 0 THEN dl.total\_loans

ELSE ROUND(dl.total\_loans / db.active\_years, 2)

END AS "Loans per Active Year",

CASE

-- Same thing, but checking if the total loans is null instead of 0

WHEN NVL(dl.total\_loans, 0) = 0 THEN 0

ELSE ROUND(dl.unreturned\_loans \* 100 / dl.total\_loans, 2)

END AS "Unreturned Loans (%)"

FROM

driver\_base db

LEFT JOIN

driver\_stops ds ON db.passport = ds.passport

LEFT JOIN

driver\_loans dl ON db.passport = dl.passport

-- Order the results by the driver's full name attribute

ORDER BY

db.fullname

;



Tests

**Test 1 – Single tests, show data inserted correctly**

As this query only shows the data there already is, the main tests to show that it is working should be to just insert some test data with some expected results and check that the results shown are actually the ones expected.

Insert into drivers a driver with same initial and end contract date

****-- insert a driver with the same cont start and end date

insert into drivers

(passport, email, fullname, birthdate, phone, address, cont\_start, cont\_end)

VALUES ('000000000000', 'test1@email.com', 'Test1',

TO\_DATE('15-05-2025', 'DD-MM-YYYY'), 600111222,

'Calle Mayor 10, Madrid',

TO\_DATE('15-05-2025', 'DD-MM-YYYY'), NULL);  
-- Then perform the selection

Expected result: Test1 should appear with all values equal 0 and all other values should appear as expected

Result:



# Package

Include an introduction with the structure of the package, and a subsection for each procedure or function that it includes. For each procedure, you must describe:

1. its design (inputs, outputs, logic of the main block), and in case of having needed to make use of auxiliary elements (queries, views, other procedures/functions...) their design and implementation must also be included (unless they are trivial queries).
2. its implementation in SQL
3. tests

—------------------------------------------------------------------------------------------------------

The foundicu package implements the basic user functionalities for the FOUNDICU® system, including loaning, reserving, and returning copies.

## Structure of the Package

The package is composed of the following functions and Procedures:

* insert\_loan(p\_signature VARCHAR2): Allows a user to borrow a copy.
* insert\_reservation(p\_isbn VARCHAR2, p\_date DATE): Allows a user to reserve a copy.
* record\_return(p\_signature VARCHAR2): Registers the return of a copy.

Each of these components is documented below in detail

## Package implementation

CREATE OR REPLACE PACKAGE foundicu AS

current\_user\_id VARCHAR2(20);

PROCEDURE set\_current\_user(p\_userid VARCHAR2);

FUNCTION get\_current\_user RETURN VARCHAR2;

PROCEDURE insert\_loan(p\_signature VARCHAR2);

PROCEDURE insert\_reservation(p\_isbn VARCHAR2, p\_date DATE);

PROCEDURE record\_return(p\_signature VARCHAR2);

END foundicu;

/

CREATE OR REPLACE PACKAGE BODY foundicu AS

PROCEDURE set\_current\_user(p\_userid VARCHAR2) IS

BEGIN

current\_user\_id := p\_userid;

END;

FUNCTION get\_current\_user RETURN VARCHAR2 IS

BEGIN

RETURN current\_user\_id;

END;

-- INSERT RESERVATION

PROCEDURE insert\_reservation(p\_isbn VARCHAR2, p\_date DATE) IS

v\_exists INTEGER;

v\_ban\_date DATE;

v\_loans INTEGER;

v\_signature VARCHAR2(5);

BEGIN

-- Check that the user exists

SELECT COUNT(\*) INTO v\_exists FROM USERS WHERE USER\_ID = current\_user\_id;

IF v\_exists = 0 THEN

RAISE\_APPLICATION\_ERROR(-20101, 'User does not exist');

END IF;

-- Check if the user is banned

SELECT BAN\_UP2 INTO v\_ban\_date FROM USERS WHERE USER\_ID = current\_user\_id;

IF v\_ban\_date IS NOT NULL AND v\_ban\_date > SYSDATE THEN

RAISE\_APPLICATION\_ERROR(-20102, 'User is banned');

END IF;

-- Check if the user has fewer than 2 active loans

SELECT COUNT(\*) INTO v\_loans

FROM LOANS

WHERE USER\_ID = current\_user\_id

AND TYPE = 'L'

AND RETURN IS NULL;

IF v\_loans >= 2 THEN

RAISE\_APPLICATION\_ERROR(-20103, 'Loan limit reached');

END IF;

-- Search for an available copy of the ISBN

SELECT c.SIGNATURE INTO v\_signature

FROM COPIES c

WHERE c.ISBN = p\_isbn

AND c.CONDITION != 'D'

AND NOT EXISTS (

SELECT 1 FROM LOANS l

WHERE l.SIGNATURE = c.SIGNATURE

AND l.STOPDATE BETWEEN p\_date AND p\_date + 14

)

AND ROWNUM = 1;

-- Insert the reservation

INSERT INTO LOANS(SIGNATURE, USER\_ID, TYPE, STOPDATE)

VALUES(v\_signature, current\_user\_id, 'R', p\_date);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20104, 'No available copy to reserve');

END insert\_reservation;

-- RECORD RETURN

PROCEDURE record\_return(p\_signature VARCHAR2) IS

v\_exists INTEGER;

BEGIN

-- Check if there is an active loan

SELECT COUNT(\*) INTO v\_exists

FROM LOANS

WHERE SIGNATURE = p\_signature

AND USER\_ID = current\_user\_id

AND TYPE = 'L'

AND RETURN IS NULL;

IF v\_exists = 0 THEN

RAISE\_APPLICATION\_ERROR(-20201, 'No active loan found');

END IF;

-- Register the return

UPDATE LOANS

SET RETURN = SYSDATE

WHERE SIGNATURE = p\_signature

AND USER\_ID = current\_user\_id

AND TYPE = 'L'

AND RETURN IS NULL;

END record\_return;

END foundicu;

/



## Tests

# External Design

Describe the views and carry out their design, implementation, and tests (in a similar way to how the queries were made in section 2 but developing their operativity completeness where required). Include a subsection for each view you develop, outlining:

1. its design in relational algebra
2. its implementation in SQL
3. Tests: notice that it must be checked that the view is properly defined (like a query), as well as the operativity of the read and write views: it is necessary to establish which operations (insertion/deletion/modification) the manager resolves itself, and which other operations it does not.

Operations on views not automatically supported by the manager must be resolved using triggers (of type *instead of*), which must also be described, implemented and tested in this section.

- - - - – - - - - – - - - - – - - - - – - - - – - - - - – - - - - - - - - - – - - - - - - – - - – - - - - – - - -

In this section we describe the external design through the development of the my\_data, my\_loans, and my\_reservations views. Each view will include the design in relational algebra, its SQL implementation and lastly tests to check that the view is defined as it should, as well as the operativity of the operable views.





**View : my\_data:**

1. Relational Algebra

my\_data(uid, ...) :=

σ USER\_ID = current\_user(USERS)



1. SQL implementation

CREATE OR REPLACE VIEW my\_data AS

SELECT USER\_ID, ID\_CARD, NAME, SURNAME1, SURNAME2, BIRTHDATE, TOWN, PROVINCE, ADDRESS, EMAIL, PHONE, TYPE, BAN\_UP2

FROM USERS

WHERE USER\_ID = foundicu.get\_current\_user();



1. Test

SELECT \* FROM my\_data



# Explicitly required Triggers

For each resolved trigger, include a subsection containing:

1. Description of the design: Table to which it is associated, Event or events in which it is triggered, Temporality (before, after or instead of), Granularity (by row or statement), Condition (if it has one) and Action (description in natural language).
2. Code (PL/SQL)
3. Tests

## Trigger1: Institutional users (those with type=L) cannot post (insert into the posts table)

Description of the Design

* Table: POSTS
* Event: INSERT of a column
* Temporality: BEFORE
* Granularity: FOR EACH ROW
* Condition: The value of :NEW.user\_ID related to some user on the user table must not be related to the type = “L”, an user of type Library for what matters.
* Action: When some user is actually of type Library the trigger ensures that no insertion is done and an error is raised (‘INSTITUTIONAL USERS CANNOT CREATE POSTS’). If the user is not type L then the insertion is done normally without raising any errors.

Code (PL/SQL)

-- Description: Prevents the insertion of posts of institutional users (municipal libraries). The trigger is used before the insertion of a new post in the posttable, checking the type of user by the userID provided for the insert.

-- If the user.type is 'L' then it is a library

CREATE OR REPLACE TRIGGER trg\_prevent\_institutional\_posts

-- Before the insert

BEFORE INSERT ON posts

FOR EACH ROW

DECLARE

v\_user\_type users.type%TYPE;

BEGIN

-- Get the user type

select type into v\_user\_type

from users

WHERE user\_ID = :NEW.user\_ID;

-- Check if the user type is 'L' (library)

IF v\_user\_type = 'L' THEN

-- Raise an error if the user is a library

RAISE\_APPLICATION\_ERROR(-20001, 'Institutional users cannot create posts.');

END IF;

END;

/

Tests

Creation of rows for the tests:

****

-- Create all necessary tables for the test

INSERT INTO municipalities(TOWN, PROVINCE, POPULATION)

VALUES ('MADRID', 'MADRID', 50000);

INSERT INTO municipalities(TOWN, PROVINCE, POPULATION)

VALUES ('BARCELONA', 'BARCELONA', 45000);

INSERT INTO municipalities(TOWN, PROVINCE, POPULATION)

VALUES ('VALENCIA', 'VALENCIA', 35000);

INSERT INTO routes(ROUTE\_ID)

VALUES ('R001');

INSERT INTO drivers(PASSPORT, EMAIL, FULLNAME, BIRTHDATE, PHONE, ADDRESS, CONT\_START, CONT\_END)

VALUES ('ABC123456789', 'conductor1@email.com', 'Juan Pérez García',

TO\_DATE('15-05-1980', 'DD-MM-YYYY'), 600111222,

'Calle Mayor 10, Madrid',

TO\_DATE('01-01-2020', 'DD-MM-YYYY'), NULL);

INSERT INTO bibuses(PLATE, LAST\_ITV, NEXT\_ITV)

VALUES ('1234ABC', TO\_DATE('01-01-2023', 'DD-MM-YYYY'),

TO\_DATE('01-01-2024', 'DD-MM-YYYY'));

INSERT INTO assign\_drv(PASSPORT, TASKDATE, ROUTE\_ID)

VALUES ('ABC123456789', TO\_DATE('10-02-2023', 'DD-MM-YYYY'), 'R001');

INSERT INTO assign\_bus(PLATE, TASKDATE, ROUTE\_ID)

VALUES ('1234ABC', TO\_DATE('10-02-2023', 'DD-MM-YYYY'), 'R001');

INSERT INTO stops(TOWN, PROVINCE, ADDRESS, ROUTE\_ID, STOPTIME)

VALUES ('MADRID', 'MADRID', 'Plaza Mayor 1', 'R001', 1000);

INSERT INTO stops(TOWN, PROVINCE, ADDRESS, ROUTE\_ID, STOPTIME)

VALUES ('BARCELONA', 'BARCELONA', 'Rambla 25', 'R001', 1200);

INSERT INTO services(TOWN, PROVINCE, BUS, TASKDATE, PASSPORT)

VALUES ('MADRID', 'MADRID', '1234ABC', TO\_DATE('10-02-2023', 'DD-MM-YYYY'), 'ABC123456789');

INSERT INTO users(USER\_ID, ID\_CARD, NAME, SURNAME1, SURNAME2, BIRTHDATE, TOWN, PROVINCE, ADDRESS, EMAIL, PHONE, TYPE)

VALUES ('USR001', '12345678A', 'Ana', 'López', 'Martínez',

TO\_DATE('20-03-1990', 'DD-MM-YYYY'), 'MADRID', 'MADRID',

'Calle Gran Vía 25', 'ana.lopez@email.com', 600222333, 'P');

INSERT INTO users(USER\_ID, ID\_CARD, NAME, SURNAME1, SURNAME2, BIRTHDATE, TOWN, PROVINCE, ADDRESS, EMAIL, PHONE, TYPE)

VALUES ('USR002', '87654321B', 'Carlos', 'García', 'Rodríguez',

TO\_DATE('15-07-1985', 'DD-MM-YYYY'), 'BARCELONA', 'BARCELONA',

'Paseo de Gracia 40', 'carlos.garcia@email.com', 600333444, 'L');

INSERT INTO books(TITLE, AUTHOR, COUNTRY, LANGUAGE, PUB\_DATE, TOPIC, CONTENT)

VALUES ('Don Quijote de la Mancha', 'Miguel de Cervantes', 'España', 'Español',

1605, 'Novela de caballería', 'En un lugar de la Mancha, de cuyo nombre no quiero acordarme...');

INSERT INTO editions(ISBN, TITLE, AUTHOR, LANGUAGE, PUBLISHER, NATIONAL\_LIB\_ID)

VALUES ('9788424105662', 'Don Quijote de la Mancha', 'Miguel de Cervantes', 'Español',

'Editorial Cátedra', 'BNE123456');

INSERT INTO copies(SIGNATURE, ISBN, CONDITION)

VALUES ('CP001', '9788424105662', 'G');

INSERT INTO copies(SIGNATURE, ISBN, CONDITION)

VALUES ('CP002', '9788424105662', 'G');

INSERT INTO loans(SIGNATURE, USER\_ID, STOPDATE, TOWN, PROVINCE, TYPE, TIME, RETURN)

VALUES ('CP001', 'USR001', TO\_DATE('10-02-2023', 'DD-MM-YYYY'),

'MADRID', 'MADRID', 'P', 30, TO\_DATE('10-03-2023', 'DD-MM-YYYY'));

INSERT INTO loans(SIGNATURE, USER\_ID, STOPDATE, TOWN, PROVINCE, TYPE, TIME, RETURN)

VALUES ('CP002', 'USR002', TO\_DATE('10-02-2023', 'DD-MM-YYYY'),

'MADRID', 'MADRID', 'L', 30, TO\_DATE('10-03-2023', 'DD-MM-YYYY'));



**Test 1 – When a personal user tries to make a post, there should be no error.**

* Initial condition: User with type=P makes a posts, a new entry for posts is inserted with the corresponding user\_ID.

Action:

* All tables should be properly created in order for this insertion to be made.

****INSERT INTO posts(SIGNATURE, USER\_ID, STOPDATE, POST\_DATE, TEXT, LIKES, DISLIKES)

VALUES ('CP001', 'USR001', TO\_DATE('10-02-2023', 'DD-MM-YYYY'),

TO\_DATE('20-02-2023', 'DD-MM-YYYY'),

'some post text',

15, 2);

Verification:

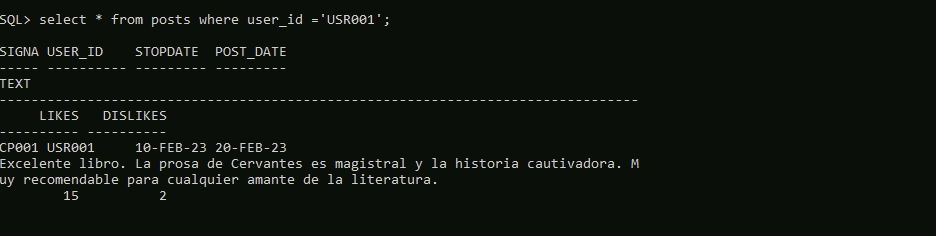
****-- Check that the post of user USR001 (type P) was CREATE DATABASE

SELECT \* FROM posts WHERE USER\_ID = 'USR001';



Expected Result: The posts of the user should have been properly created

Result:



**Test 2 – When a library/municipal user tries to make a post, there should be an error.**

* Initial condition: User with type=L makes a posts, a new entry for posts triggers the trigger when trying to insert with the corresponding user\_ID.

Action:

* Again, see the initial part of these tests to check that the user with id US002 is actually of type=L

****INSERT INTO posts(SIGNATURE, USER\_ID, STOPDATE, POST\_DATE, TEXT, LIKES, DISLIKES)

VALUES ('CP002', 'USR002', TO\_DATE('10-02-2023', 'DD-MM-YYYY'),

TO\_DATE('25-02-2023', 'DD-MM-YYYY'),

'some other post text',

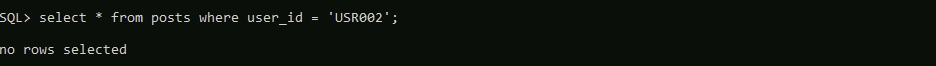
20, 1);

Verification: Check that the row has not been created

****-- Check that no post of user USR002 (type L) was created

SELECT \* FROM posts WHERE USER\_ID = 'USR002';

Expected Result: NO rows with that characteristic



## Trigger2: Deregistration When Copy Becomes Deteriorated

Description of the Design

* Table: COPIES
* Event: UPDATE of the column condition
* Temporality: BEFORE
* Granularity: FOR EACH ROW
* Condition: The new value of :NEW.condition must be 'D' (for “Deteriorated”)
* Action: This trigger ensures that every time a copy of a book is marked as deteriorated (condition = 'D'), the system automatically sets the deregistered column to the current system date (SYSDATE). This provides a reliable way to track when copies were removed from circulation due to poor condition.

Code (PL/SQL)

CREATE OR REPLACE TRIGGER trg\_deregistration\_on\_deteriorated

BEFORE UPDATE OF condition ON copies

FOR EACH ROW

WHEN (NEW.condition = 'D')

BEGIN

:NEW.deregistered := SYSDATE;

END;

/



Tests

**Test 1 – When a copy is deteriorated, the deregistration date should be set**

* Initial condition: Copy with signature 'FI430' exists with condition = 'G' and deregistered IS NULL.

Action:

****UPDATE copies

SET condition = 'D'

WHERE signature = 'FI430';

Verification:

****SELECT signature, condition, deregistered

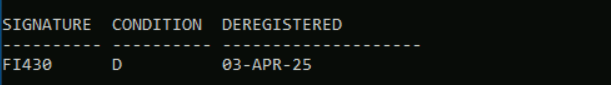
FROM copies

WHERE signature = 'FI430';



Expected Result: The deregistered field should be automatically updated with the current system date and time.

Result:



**Test 2 – When condition is changed to another value, the deregistration date must remain NULL**

Action:

****UPDATE copies

SET condition = 'G', deregistered = NULL

WHERE signature = 'FI430';

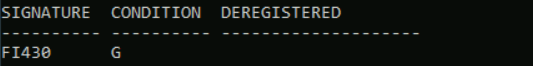
Verification:

****SELECT signature, condition, deregistered

FROM copies

WHERE signature = 'FI430';

Expected Result: The deregistered field must remain unchanged (NULL), confirming the trigger was not activated.



# Concluding Remarks

Firstly, you have to defend the achieved result, emphasizing the goodness of the semantic coverage, usage (comment unfeasible queries, in case), documentation, etc.

After stating your results, comment your achievement through this labwork: required effort (how much time you spent), knowledge gain, progress, etc. You can also propose improvements for further editions (size of the problem, requested items, deadlines, supporting materials, etc.).

## Queries:

* For query 2, the difference between seniority years and active years could not be obtained/deduced and therefore both of them obtain the same value.
* For query 2, it was assumed that the active years referred to the number of years not the years as a date. When computing the number of loans or stops per active years the result of the average number of stops per year/ loans per year is given.