

Profesor:	Daniel Esteban Villamil Sierra	Grupo	85
Alumno/a:	Álvaro González Fúnez	NIA:	100451281
Alumno/a:	Jhonatan Barcos Gambaro	NIA:	100548615

1. Introducción

Esta práctica tiene como objetivo integrar consultas, procedimientos, vistas y disparadores que optimicen la gestión de información relacionada con las operaciones de la fundación Foundicu.org. Se aplican principios del álgebra relacional, diseño SQL y pruebas para garantizar un sistema eficiente y coherente. Todo ello busca mejorar la integridad y rendimiento del sistema de información de la organización.

2. Consultas

2.1. BoreBooks:

a) su diseño en álgebra relacional

En primer lugar, agrupamos los libros que tengan ediciones de más de 3 idiomas:
 $\text{BooksPlusLanguages} = \pi\{\text{title, author}\} (\sigma\{\text{COUNT(DISTINCT language)} \geq 3\} (\text{Books} \bowtie \text{Editions}))$

Seguidamente, agrupamos los libros de los cuales se ha prestado alguna copia:
 $\text{BooksWithLoans} = \pi\{\text{title, author}\} (\text{Books} \bowtie \text{Copies} \bowtie \text{Loans})$

Finalmente, como nuestro objetivo es encontrar aquellos libros que tengan ediciones de más de 3 idiomas de los cuales no se haya prestado ninguna copia, realizaremos la diferencia entre BooksPlusLanguages y BooksWithLoans para hallar la consulta BoreBooks:
 $\text{BoreBooks} = \text{BooksPlusLanguages} - \text{BooksWithLoans}$

b) su implementación en SQL

En primer lugar, agrupamos los libros que tengan ediciones de más de 3 idiomas:

```
Unset
WITH BooksPlusLanguages AS (
    SELECT e.title, e.author
    FROM Editions e
    GROUP BY e.title, e.author
    HAVING COUNT(DISTINCT e.language) >= 3
),
```

Seguidamente, agrupamos los libros de los cuales se ha prestado alguna copia.

```
Unset
BooksWithLoans AS (
    SELECT DISTINCT e.title, e.author
    FROM Loans l
    JOIN Copies c ON l.signature = c.signature
    JOIN Editions e ON c.isbn = e.isbn
)
```

Finalmente, como nuestro objetivo es encontrar aquellos libros que tengan ediciones de más de 3 idiomas de los cuales no se haya prestado ninguna copia, realizaremos la diferencia entre BooksPlusLanguages y BooksWithLoans para hallar la consulta BoreBooks:

```
Unset
SELECT b.*
FROM Books b
JOIN BooksPlusLanguages mlb ON b.title = mlb.title AND b.author = mlb.author
LEFT JOIN BooksWithLoans lb ON mlb.title = lb.title AND mlb.author = lb.author
WHERE lb.title IS NULL;
```

c) las pruebas realizadas para demostrar que funciona correctamente

Para probar que nuestra consulta es correcta, hemos realizado una inserción de nuevos libros, tal que uno cumpla las condiciones para aparecer en la consulta y otro no. Se adjunta en el archivo “Consultas.txt” las inserciones realizadas.

Por una parte hemos insertado el libro “El Quijote” con 3 ediciones de distintos idiomas y con copias no prestadas. Así pues, al realizar dicha inserción y ejecutar de nuevo la consulta, el libro “El Quijote” aparece en dicha consulta.

Por otra parte, hemos insertado el libro “1984” con 2 ediciones de distintos idiomas y por lo tanto, al realizar la inserción y ejecutar de nuevo la consulta, el libro “1984” no aparece en dicha consulta.

Por último, hemos insertado el libro “Cien años de soledad” con 3 ediciones de distintos idiomas pero con alguna copia prestada, por lo que al realizar la inserción y ejecutar de nuevo la consulta, el libro no aparece en dicha consulta.

Así pues, tras ejecutar la consulta inicialmente, tenemos 60 filas:

```
GaudÃ-  
  
TITLE  
-----  
CONTENT  
-----  
AWARDS  
-----  
  
60 rows selected.
```

Sin embargo, tras la inserción, tenemos 61, pues el libro “El Quijote” cumple con las condiciones de dicha consulta:

```
TITLE
-----
CONTENT
-----
AWARDS
-----

El Quijote

61 rows selected.
```

2.2. Informe de Empleados:

a) su diseño en álgebra relacional

En primer lugar, calculamos la edad del conductor teniendo en cuenta la fecha actual y su cumpleaños:

$$\text{Edad} = \pi\{\text{FULLNAME}, \text{FLOOR}(\text{MONTHS_BETWEEN}(\text{SYSDATE}, \text{BIRTHDATE}) / 12)\}$$

(drivers)

Seguidamente, calculamos la antigüedad de su contrato:

$$\text{Antigüedad_Contrato} = \pi\{\text{FULLNAME}, \text{FLOOR}(\text{MONTHS_BETWEEN}(\text{SYSDATE}, \text{CONT_START}) / 12)\}$$

(drivers)

Así mismo, calculamos los años en activo del conductor:

$$\text{Años_Activo} = \pi\{\text{FULLNAME}, \text{COUNT}(\text{DISTINCT EXTRACT}(\text{YEAR FROM TASKDATE}))\}$$

(drivers \bowtie assign_drv \bowtie services)

Por otra parte, calculamos la media de paradas por año activo:

$$\text{Media_Paradas_Por_Año} = \pi\{\text{FULLNAME}, \text{ROUND}(\text{COUNT}(\text{TOWN}) / \text{Años_Activo}, 2)\}$$

(drivers \bowtie assign_drv \bowtie services)

Calculamos la media de préstamos por año activo:

$$\text{Media_Préstamos_Por_Año} = \pi\{\text{FULLNAME}, \text{ROUND}(\text{COUNT}(\text{SIGNATURE}) / \text{Años_Activo}, 2)\}$$

(drivers \bowtie assign_drv \bowtie services \bowtie loans)

Finalmente, calculamos el porcentaje de préstamos no devueltos:

$$\text{Porcentaje_No_Devueltos} = \pi\{\text{FULLNAME}, \text{ROUND}(100 * \text{COUNT}(\sigma\{\text{RETURN IS NULL}\}(\text{loans})) / \text{COUNT}(\text{SIGNATURE}), 2)\} (\text{drivers} \bowtie \text{assign_drv} \bowtie \text{services} \bowtie \text{loans})$$

b) su implementación en SQL

En primer lugar, calculamos la edad del conductor teniendo en cuenta la fecha actual y su cumpleaños:

```
Unset
SELECT
    d.FULLNAME AS Nombre_Completo,
    FLOOR(MONTHS_BETWEEN(SYSDATE, d.BIRTHDATE) / 12) AS Edad,
```

Seguidamente, calculamos la antigüedad de su contrato:

```
Unset
    FLOOR(MONTHS_BETWEEN(SYSDATE, d.CONT_START) / 12) AS Antigüedad_Contrato,
```

Así mismo, calculamos los años en activo del conductor:

```
Unset
    COUNT(DISTINCT EXTRACT(YEAR FROM s.TASKDATE)) AS Años_Activo,
```

Por otra parte, calculamos la media de paradas por año activo:

Unset

```
ROUND(COUNT(s.TOWN) / NULLIF(COUNT(DISTINCT EXTRACT(YEAR FROM  
s.TASKDATE)), 0), 2) AS Media_Paradas_Por_Año,
```

Calculamos la media de préstamos por año activo:

Unset

```
ROUND(COUNT(l.SIGNATURE) / NULLIF(COUNT(DISTINCT EXTRACT(YEAR FROM  
s.TASKDATE)), 0), 2) AS Media_Préstamos_Por_Año,
```

Calculamos el porcentaje de préstamos no devueltos:

Unset

```
ROUND(100 * COUNT(CASE WHEN l.RETURN IS NULL THEN 1 END) /  
NULLIF(COUNT(l.SIGNATURE), 0), 2) AS Porcentaje_No_Devueltos
```

Finalmente unimos las tablas y las agrupamos:

Unset

```
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.FULLNAME, d.BIRTHDATE, d.CONT_START;
```

c) las pruebas realizadas para demostrar que funciona correctamente

Para demostrar que la consulta es correcta se ha realizado la inserción de nuevos conductores y se comprobará manualmente que los datos de la consulta acerca de estos son correctos.

En primer lugar, antes de realizar la inserción y ejecutar la consulta observamos que tenemos 13 filas, es decir, un total de 13 conductores en nuestra BBDD:

NOMBRE_COMPLETO	EDAD	ANTIGÜEDAD_CONTRATO	AÑOS_ACTIVO	MEDIA_PARADAS_POR_AÑO
-----	-----	-----	-----	-----
María José Chal Abrigado	44	2	1	1806
César Pareja Feliz	55	3	1	55
Zacarías Rico Morcón	42	7	1	2146
Fuencisla Maja San Frutos	33	5	1	2604
José Pepe Perez Quitasol	43	2	1	1495
Amaia Itsasoko Gaztelua	35	4	1	1810
Frutos Campo Dorado	41	2	1	2683
Octavio Sansegundo Quinto	43	6	1	1781
Eulalia Puig Castella	39	4	1	2566
Celso Hórreo Madera	44	7	1	2301
Carlos Tercero Madrid	45	2	1	1176
-----	-----	-----	-----	-----
NOMBRE_COMPLETO	EDAD	ANTIGÜEDAD_CONTRATO	AÑOS_ACTIVO	MEDIA_PARADAS_POR_AÑO
-----	-----	-----	-----	-----
Victoria Rojo Blanco	38	2	1	1670
María Pensamiento Calma Grata	42	7	1	1799

13 rows selected.

A continuación, se inserta el siguiente conductor:

Unset

```

INSERT INTO drivers (PASSPORT, EMAIL, FULLNAME, BIRTHDATE, PHONE, ADDRESS,
CONT_START, CONT_END)
VALUES ('A12345678', 'juan.perez@example.com', 'Juan Pérez',
TO_DATE('1985-07-15', 'YYYY-MM-DD'), 612345678, 'Calle Ficticia 123',
TO_DATE('2020-01-01', 'YYYY-MM-DD'), NULL);
  
```

Tras ejecutar de nuevo la consulta, observamos como se ha añadido correctamente el nuevo conductor a la visualización y todos sus datos son correctos:

NOMBRE_COMPLETO	EDAD	ANTIGÜEDAD_CONTRATO	AÑOS_ACTIVO	MEDIA_PARADAS_POR_AÑO
-----	-----	-----	-----	-----
María José Chal Abrigado	44	2	1	1806
César Pareja Feliz	55	3	1	55
Zacarías Rico Morcón	42	7	1	2146
Fuencisla Maja San Frutos	33	5	1	2604
José Pepe Perez Quitasol	43	2	1	1495
Amaia Itsasoko Gaztelua	35	4	1	1810
Frutos Campo Dorado	41	2	1	2683
Octavio Sansegundo Quinto	43	6	1	1781
Juan Pérez	39	5	0	
Eulalia Puig Castella	39	4	1	2566
Celso Hórreo Madera	44	7	1	2301
-----	-----	-----	-----	-----
NOMBRE_COMPLETO	EDAD	ANTIGÜEDAD_CONTRATO	AÑOS_ACTIVO	MEDIA_PARADAS_POR_AÑO
-----	-----	-----	-----	-----
Carlos Tercero Madrid	45	2	1	1176
Victoria Rojo Blanco	38	2	1	1670
María Pensamiento Calma Grata	42	7	1	1799

14 rows selected.

3. Paquete

Hemos creado el paquete *foundicu*, como un conjunto de procedimientos que permiten realizar una gestión óptima de nuevos préstamos, reservas y devoluciones de ejemplares existentes en nuestra BBDD.

Para ello, se han diseñado tres procedimientos: InsertarPrestamo, InsertarReserva y RegistrarDevolucion, en los cuales tendremos en cuenta las reglas dadas por el enunciado como la verificación de disponibilidad, límite de préstamos y las limitaciones de usuarios sancionados. A continuación se presenta una subsección para cada uno de estos procedimientos:

3.1. InsertarPrestamo:

a) su diseño (entradas, salidas, lógica del bloque principal)

- Entradas: *signature* - Identificador unívoco de un ejemplar.
- Salidas: Se devolverán distintos prints para la gestión de errores y la verificación de si se ha registrado correctamente:
 - ‘Error de verificación 1: Usuario no encontrado.’
 - ‘Error de verificación 2: Límite de préstamos alcanzado.’
 - ‘Error de verificación 3: El ejemplar ya está prestado.’
 - ‘El préstamo se ha registrado correctamente.’
- Lógica del bloque principal:
 1. Se verifica si el usuario que intenta registrar el préstamo de un ejemplar está registrado en la BBDD.
 2. Se verifica si existe una reserva de dicho ejemplar está reservado por dicho usuario a fecha de hoy.
 3. Se verifica si el usuario tiene alguna sanción actual.
 4. Se verifica si el usuario ha alcanzado el límite de préstamos.
 5. Se registra el préstamo (en caso de que se cumplan todas las verificaciones anteriores).

b) su implementación en SQL

```
CREATE TABLE current_user (  
    email VARCHAR2(100)  
);
```

Creamos el paquete:


```
CREATE OR REPLACE PACKAGE foundicu AS
    PROCEDURE InsertarPrestamo(p_signature CHAR);
END foundicu;
/
```

Creamos el paquete:

```
CREATE OR REPLACE PACKAGE BODY foundicu AS

    PROCEDURE InsertarPrestamo(p_signature CHAR) IS
        user_id CHAR(10);
        num_prestamos NUMBER;
        prestamos_activos NUMBER;
        v_email VARCHAR2(100);
    BEGIN
        -- Obtener el correo electrónico del usuario actual desde la
        tabla auxiliar
        BEGIN
            SELECT email INTO v_email
            FROM current_user;
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE('Error: No se encontró el correo
electrónico del usuario actual.');
```

```
                RETURN;
            END;

            -- Obtener el ID del usuario actual utilizando el correo
electrónico
            BEGIN
                SELECT user_id INTO user_id
                FROM users
                WHERE email = v_email;
            EXCEPTION
                WHEN NO_DATA_FOUND THEN
                    DBMS_OUTPUT.PUT_LINE('Error: Usuario no encontrado.');
```

```
        RETURN;  
  
    END;  
  
    -- Contar el número de préstamos activos del usuario  
    SELECT COUNT(*) INTO num_prestamos  
    FROM loans  
    WHERE user_id = user_id AND return IS NULL;  
  
    -- Verificar si el usuario ha alcanzado el límite de préstamos  
    IF num_prestamos >= 5 THEN  
        DBMS_OUTPUT.PUT_LINE('Error: Límite de préstamos  
alcanzado.');
```

```
        RETURN;  
    END IF;  
  
    -- Verificar si el ejemplar está disponible (sin préstamos  
activos)  
    SELECT COUNT(*) INTO prestamos_activos  
    FROM loans  
    WHERE signature = p_signature AND return IS NULL;  
  
    IF prestamos_activos > 0 THEN  
        DBMS_OUTPUT.PUT_LINE('Error: El ejemplar ya está prestado.');
```

```
        RETURN;  
    END IF;  
  
    -- Registrar el préstamo  
    INSERT INTO loans (signature, user_id, stopdate, town, province,  
type, time, return)  
    VALUES (p_signature, user_id, SYSDATE, 'Madrid', 'Madrid', 'L',  
0, NULL);  
  
    DBMS_OUTPUT.PUT_LINE('El préstamo se ha registrado  
correctamente.');
```

```
END InsertarPrestamo;
```

```
END foundicu;  
/
```

- c) las pruebas realizadas para demostrar que funciona correctamente (LOS DATOS ESTÁN BASADOS EN LOS INSERTS DE PRUEBA ESTABLECIDOS EN LA VISTA MY_RESERVATIONS (PUNTO 4.3), QUE HICIMOS PREVIAMENTE, ANTES DE EJECUTAR LO QUE VIENE A CONTINUACIÓN, ASEGURARSE DE QUE ESTÁN HECHOS TODOS LOS INSERTS)

```
INSERT INTO current_user (email) VALUES  
( 'carlos.lopez@example.com' );
```

Y obtenemos los siguientes resultados:

```
SQL> BEGIN  
2     foundicu.InsertarPrestamo(p_signature => 'P003');  
3 END;  
4 /  
Error: El ejemplar ya está prestado.  
  
PL/SQL procedure successfully completed.
```

3.2. InsertarReserva:

d) su diseño (entradas, salidas, lógica del bloque principal)

- Entradas: *ISBN* y *DATE* - ISBN del libro a reservar y fecha de reserva.
- Salidas: Se devolverán distintos prints para la gestión de errores y la verificación de si se ha registrado correctamente:
 - 'Error de verificación 1: Usuario no encontrado.'
 - 'Error de verificación 2: Límite de préstamos alcanzado.'
 - 'Error de verificación 3: No hay ejemplares disponibles para reservar.'
 - 'La reserva se ha registrado correctamente.'
- Lógica del bloque principal:
 1. Se verifica si el usuario que intenta registrar el préstamo de un ejemplar está registrado en la BBDD.

2. Se verifica que el usuario no haya alcanzado el límite superior de préstamos y que no esté sancionado.
3. Se verifica la disponibilidad de un ejemplar para el ISBN especificado durante las próximas dos semanas.
4. Se registra la reserva (en caso de que se cumplan todas las verificaciones anteriores).

e) su implementación en SQL

Creamos una tabla auxiliar SI NO SE HA CREADO ANTES:

```
CREATE TABLE current_user (  
    email VARCHAR2(100)  
);
```

Creamos el paquete

```
CREATE OR REPLACE PACKAGE foundicu AS  
    PROCEDURE InsertarReserva(p_isbn VARCHAR2, p_date DATE);  
END foundicu;  
/
```

Y el cuerpo:

```
CREATE OR REPLACE PACKAGE BODY foundicu AS  
  
    PROCEDURE InsertarReserva(p_isbn VARCHAR2, p_date DATE) IS  
        user CHAR(10);  
        num_prestamos NUMBER;  
        id_prestamo CHAR(5);  
    BEGIN  
        -- Obtener el ID del usuario actual desde la tabla auxiliar  
        BEGIN  
            SELECT user_id INTO user  
            FROM users  
            WHERE email = (SELECT email FROM current_user)  
            AND ROWNUM = 1; -- Aseguramos que obtenemos solo una  
fila  
        EXCEPTION  
            WHEN NO_DATA_FOUND THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Error de verificación 1:
Usuario no encontrado.');
```

```
        RETURN;
```

```
    END;
```

```
    -- Verificar el número de préstamos actuales del usuario
    SELECT COUNT(*) INTO num_prestamos
    FROM loans
    WHERE user_id = user;
```

```
    IF num_prestamos >= 5 THEN
        DBMS_OUTPUT.PUT_LINE('Error de verificación 2: Límite
de préstamos alcanzado.');
```

```
        RETURN;
```

```
    END IF;
```

```
    -- Verificamos la disponibilidad de un ejemplar con el ISBN
especificado
    BEGIN
        SELECT c.signature INTO id_prestamo
        FROM copies c
        WHERE c.isbn = p_isbn
        AND NOT EXISTS (
            SELECT 1 FROM loans l
            WHERE l.signature = c.signature
            AND l.return IS NULL
        )
        AND ROWNUM = 1; -- Aseguramos que obtenemos solo una
fila
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Error de verificación 3: No
hay ejemplares disponibles para reservar.');
```

```
        RETURN;
```

```
    END;
```

```
-- Registramos la reserva
INSERT INTO loans (signature, user_id, stopdate, type,
town, province, time)
VALUES (id_prestamo, user, p_date, 'R', 'Desconocido',
'Desconocido', 0);

DBMS_OUTPUT.PUT_LINE('La reserva se ha registrado
correctamente..');

END InsertarReserva;

END foundicu;
/
```

- f) las pruebas realizadas para demostrar que funciona correctamente (LOS DATOS ESTÁN BASADOS EN LOS INSERTS DE PRUEBA ESTABLECIDOS EN LA VISTA MY_RESERVATIONS (PUNTO 4.3), QUE HICIMOS PREVIAMENTE, ANTES DE EJECUTAR LO QUE VIENE A CONTINUACIÓN, ASEGURARSE DE QUE ESTÁN HECHOS TODOS LOS INSERTS)

Hacemos lo siguiente SI NO SE HA HECHO ANTES:

```
INSERT INTO current_user (email) VALUES
('carlos.lopez@example.com');
```

Y obtenemos este resultado:

```
SQL> BEGIN
2   foundicu.InsertarReserva(p_isbn => 'ISBN98765', p_date => TO_DATE('02-FEB-2025', 'DD-MON-YYYY'));
3 END;
4 /
Error de verificación 3: No hay ejemplares disponibles para reservar.
PL/SQL procedure successfully completed.
```

3.3. RegistrarDevolucion:

- g) su diseño (entradas, salidas, lógica del bloque principal)

- Entradas: *signature* - Identificador unívoco de un ejemplar.
- Salidas: Se devolverán distintos prints para la gestión de errores y la verificación de si se ha registrado correctamente:
 - ‘Error de verificación 1: Usuario no encontrado.’
 - ‘Error de verificación 2: No existe un préstamo registrado para este usuario y ejemplar.’
 - ‘Error de verificación 3: El ejemplar ya está prestado.’
 - ‘La devolución se ha registrado correctamente.’
- Lógica del bloque principal:
 1. Se verifica si el usuario que intenta registrar el préstamo de un ejemplar está registrado en la BBDD.
 2. Se verifica que el usuario tiene el préstamo del ejemplar para poder llevar a cabo la devolución.
 3. Se registra la devolución (en caso de que se cumplan todas las verificaciones anteriores).

h) su implementación en SQL

Creamos el paquete:

```
CREATE OR REPLACE PACKAGE foundicu AS
    PROCEDURE RegistrarDevolucion(p_email VARCHAR2, p_signature
    VARCHAR2);
END foundicu;
/
```

Creamos su cuerpo:

```
-- Cuerpo del paquete foundicu
CREATE OR REPLACE PACKAGE BODY foundicu AS

    PROCEDURE RegistrarDevolucion(p_email VARCHAR2, p_signature
    VARCHAR2) IS
        user_id users.USER_ID%TYPE; -- Variable que hereda el tipo
de datos de USER_ID
        num_prestamos NUMBER; -- Contador de préstamos activos para
el ejemplar
```

```
BEGIN
    -- Intentar obtener el user_id basado en el email
proporcionado
    BEGIN
        SELECT USER_ID INTO user_id
        FROM users
        WHERE LOWER(TRIM(email)) = LOWER(TRIM(p_email))
        AND ROWNUM = 1;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Error: No se ha encontrado un
usuario con el email proporcionado.');
```

```
        RETURN;
    END;

    -- Verificar que el usuario tiene el préstamo del ejemplar
SELECT COUNT(*) INTO num_prestamos
FROM loans
WHERE signature = p_signature
AND user_id = user_id
AND return IS NULL; -- Verifica si el préstamo está
activo

IF num_prestamos = 0 THEN -- Si no hay préstamos activos,
error
    DBMS_OUTPUT.PUT_LINE('Error: El usuario no tiene este
libro prestado o ya fue devuelto.');
```

```
    RETURN;
END IF;

    -- Registrar la devolución actualizando el campo 'return'
con la fecha actual
UPDATE loans
SET return = SYSDATE
WHERE signature = p_signature
AND user_id = user_id
```



```
        AND return IS NULL; -- Se asegura que solo actualice los  
préstamos no devueltos  
  
        DBMS_OUTPUT.PUT_LINE('La devolución se ha registrado  
correctamente.');
```

END RegistrarDevolucion;

END foundicu;

/

- i) las pruebas realizadas para demostrar que funciona correctamente (LOS DATOS ESTÁN BASADOS EN LOS INSERTS DE PRUEBA ESTABLECIDOS EN LA VISTA MY_RESERVATIONS (PUNTO 4.3), QUE HICIMOS PREVIAMENTE, ANTES DE EJECUTAR LO QUE VIENE A CONTINUACIÓN, ASEGURARSE DE QUE ESTÁN HECHOS TODOS LOS INSERTS)

```
SQL> BEGIN  
2   foundicu.RegistrarDevolucion('carlos.lopez@example.com', 'P003');  
3 END;  
4 /  
Error: El usuario no tiene este libro prestado o ya fue devuelto.  
PL/SQL procedure successfully completed.
```

4. Diseño externo

- vista my_data (read only)

Contiene los datos personales del usuario actual. Esta vista se apoya en un paquete PL/SQL llamado foundicu, que actúa como un mecanismo para guardar y obtener el usuario activo mediante las funciones SetCurrentUser y GetCurrentUser.

- a) su diseño en álgebra relacional

$\pi\{\text{user_id, full_name, address, email, phone}\}(\sigma\{\text{user_id} = \text{GetCurrentUser()}\}(\text{Users}))$

b) su implementación en SQL

```
-- Crear el paquete foundicu
CREATE OR REPLACE PACKAGE foundicu AS
    current_user CHAR(10);
    PROCEDURE SetCurrentUser(p_user CHAR);
    FUNCTION GetCurrentUser RETURN CHAR;
END foundicu;
/
```

Creamos el cuerpo del paquete foundicu:

```
CREATE OR REPLACE PACKAGE BODY foundicu AS
    PROCEDURE SetCurrentUser(p_user CHAR) IS
    BEGIN
        current_user := p_user; -- Asignar el usuario actual
    END SetCurrentUser;

    FUNCTION GetCurrentUser RETURN CHAR IS
    BEGIN
        RETURN current_user; -- Devolver el usuario actual
    END GetCurrentUser;
END foundicu;
/
```

Creamos la vista:

```
CREATE OR REPLACE VIEW my_data AS
SELECT user_id,
       name || ' ' || surname1 || ' ' || NVL(surname2, '') AS
full_name,
       address,
       email,
       phone
FROM users
WHERE user_id = foundicu.GetCurrentUser()
WITH READ ONLY;
```

c) Pruebas:

```
BEGIN
```

```
foundicu.SetCurrentUser('0010126950'); -- Poner el usuario que  
se quiere probar  
END;  
/
```

Observamos lo que tenemos en la vista:

```
SELECT * FROM my_data;
```

Comprobamos los resultados:

```
SQL> SELECT * FROM my_data;  
  
USER_ID    FULL_NAME  
-----  
0010126950 Juana Díaz
```

La fila tiene una longitud tan grande que no podemos mostrar todos los datos que muestra en la misma captura.

- ★ La vista my_data soporta la operación SELECT (consultar los datos personales del usuario actual) y no soporta ninguna más, sin embargo, en esta primera vista, no es necesario crear disparadores, ya que la vista es de solo lectura y el gestor maneja adecuadamente las restricciones mencionadas.

- vista my_loans (operable)

muestra todos préstamos del usuario, junto con sus publicaciones si las hubiera (si no hay publicaciones sobre un préstamo, se devolverán valores nulos). Además de la consulta, la vista debe permitir actualizar el atributo 'post': al hacer esto, el atributo post_date se asigna automáticamente con la fecha y hora actuales; si no existía un post anterior (el valor anterior del post era nulo) los atributos 'likes' y 'dislikes' se inicializan a cero.

a) su diseño en álgebra relacional

Seleccionamos los préstamos:

```
 $\sigma\{user\_id = GetCurrentUser()\}(Loans)$ 
```

Hacemos el LEFT JOIN:

```
 $\sigma\{user\_id = GetCurrentUser()\}(Loans) \bowtie \{Loans.signature = Posts.signature\} Posts$ 
```

Proyectamos las columnas:

```
 $\pi\{signature, user\_id, stopdate, return, text AS post, post\_date, likes, dislikes\}$ 
```

```
 $(\sigma\{user\_id = GetCurrentUser()\}(Loans) \bowtie \{Loans.signature = Posts.signature\} Posts)$ 
```

b) su implementación en SQL

```
-- Crear la vista my_loans
CREATE OR REPLACE VIEW my_loans AS
SELECT
    l.signature,
    l.user_id,
    l.stopdate,
    l.return,
    p.text AS post,
    p.post_date,
    p.likes,
    p.dislikes
FROM loans l
LEFT JOIN posts p ON l.signature = p.signature
WHERE l.user_id = foundicu.GetCurrentUser()
WITH CHECK OPTION;
```

Creamos el trigger (en la sección de pruebas explicamos el funcionamiento):

```
CREATE OR REPLACE TRIGGER trg_update_my_loans
INSTEAD OF UPDATE ON my_loans
FOR EACH ROW
BEGIN
    -- Si el nuevo post no es nulo y el antiguo es nulo
    IF :NEW.post IS NOT NULL AND :OLD.post IS NULL THEN
        -- Insertar el nuevo post con los valores predeterminados
        INSERT INTO posts (signature, user_id, stopdate, post_date,
text, likes, dislikes)
            VALUES (:OLD.signature, foundicu.GetCurrentUser(),
:OLD.stopdate, SYSDATE, :NEW.post, 0, 0);

    -- Si el nuevo post no es nulo y ya existía un post
    ELSIF :NEW.post IS NOT NULL THEN
        -- Actualizar el texto del post y la fecha
        UPDATE posts
        SET text = :NEW.post, post_date = SYSDATE
        WHERE signature = :OLD.signature;
```

```
-- Si el nuevo post es nulo, no se actualiza el post
ELSE
    RAISE_APPLICATION_ERROR(-20001, 'Solo se permite actualizar
el campo post.');
```

```
-- No se permite la actualización de ninguna otra columna, como
'signature', 'user_id', 'stopdate', etc.
IF :NEW.signature != :OLD.signature OR :NEW.user_id !=
:OLD.user_id OR :NEW.stopdate != :OLD.stopdate THEN
    RAISE_APPLICATION_ERROR(-20002, 'No se puede actualizar
este campo.');
```

```
END IF;
END;
/
```

c) Pruebas:

```
BEGIN
    -- Asignar el usuario actual
    foundicu.SetCurrentUser('0010126950');
```

```
END;
/
```

Realizar un update:

```
UPDATE my_loans
SET post = 'Este es un comentario sobre el préstamo.'
WHERE signature = 'KJ006' AND user_id = foundicu.GetCurrentUser();
```

Ver el resultado de la vista:

```
select * from my_loans;
```

Comprobar el resultado:

```
SQL> select * from my_loans;
```

SIGNA	USER_ID	STOPDATE	RETURN	POST
KJ006	0010126950	21-NOV-24	05-DEC-24	Este es un comentario sobre el préstamo.

No entra toda la fila en la misma captura, insertamos otra con los datos restantes de suma importancia:

POST_DATE	LIKES	DISLIKES
04-APR-25	0	0

- ★ La vista `my_loans` soporta la operación `SELECT`, permitiendo consultar los préstamos del usuario actual junto con sus publicaciones asociadas. Para la implementación de las demás operaciones, se ejecuta un disparador (`trg_update_my_loans`):

- **Evento que lo dispara:** Se activa en lugar de una operación de actualización (`UPDATE`) en la vista `my_loans`.
- **Temporalidad:** Se ejecuta en lugar de la operación de actualización solicitada.
- **Granularidad:** Se ejecuta para cada fila afectada por la operación.
- **Condiciones y acciones:**
 - Si el nuevo valor de `post` no es nulo y el valor anterior era nulo, inserta un nuevo registro en la tabla `posts` con `likes` y `dislikes` inicializados a cero.
 - Si el nuevo valor de `post` no es nulo y ya existía un post, actualiza el texto y la fecha de la publicación existente.
 - Si el nuevo valor de `post` es nulo, genera un error indicando que solo se permite actualizar el campo `post`.
 - Si se intenta actualizar cualquier otra columna de la vista, genera un error indicando que no se pueden actualizar estos campos.

- **vista `my_reservations` (operable)**

a) su álgebra relacional

Seleccionar préstamos del usuario actual

$\sigma_{\{user_id = u\}}(LOANS)$

Proceso JOIN

$\sigma_{\{user_id = u\}}(LOANS) \bowtie \{LOANS.signature = COPIES.signature\} COPIES$

Proyección de atributos:

$\pi\{LOANS.signature, LOANS.user_id, stopdate, town, province, type, time, return, COPIES.isbn\}$

$(\sigma_{\{user_id = u\}}(LOANS) \bowtie \{LOANS.signature = COPIES.signature\} COPIES)$

b) su implementación en SQL

```
CREATE OR REPLACE VIEW my_reservations AS
SELECT
    l.signature,
    l.user_id,
    l.stopdate,
    l.town,
    l.province,
    l.type,
    l.time,
    l.return,
    c.isbn
FROM
    loans l
JOIN
    copies c ON l.signature = c.signature;
```

Hacemos el trigger para **insertar** (Explicado en la sección ‘Pruebas’):

```
CREATE OR REPLACE TRIGGER trg_insert_my_reservations
INSTEAD OF INSERT ON my_reservations
FOR EACH ROW
DECLARE
    v_available_signature VARCHAR2(50);
BEGIN
    -- Verificar disponibilidad de copias del libro
    BEGIN
        SELECT signature INTO v_available_signature
        FROM copies
        WHERE isbn = :NEW.isbn
        AND NOT EXISTS (
            SELECT 1
            FROM loans
            WHERE loans.signature = copies.signature
            AND return IS NULL
        )
    )
```

```
        FETCH FIRST 1 ROW ONLY;  
    EXCEPTION  
        WHEN NO_DATA_FOUND THEN  
            RAISE_APPLICATION_ERROR(-20001, 'No hay copias  
disponibles para reservar.');
```

```
        END;  
  
        -- Insertar la nueva reserva  
        INSERT INTO loans (signature, user_id, stopdate, town,  
province, type, time, return)  
        VALUES (v_available_signature, foundicu.GetCurrentUser(),  
:NEW.stopdate, :NEW.town, :NEW.province, 'R', 0, NULL);  
    END;
```

Hacemos el trigger para **eliminar** (Explicado en la sección ‘Pruebas’):

```
CREATE OR REPLACE TRIGGER trg_delete_my_reservations  
INSTEAD OF DELETE ON my_reservations  
FOR EACH ROW  
BEGIN  
    -- Eliminar la reserva correspondiente en la tabla loans  
    DELETE FROM loans  
    WHERE signature = :OLD.signature  
    AND stopdate = :OLD.stopdate;  
END;  
/
```

Hacemos el paquete y el trigger para **actualizar** (Explicado en la sección ‘Pruebas’):

```
CREATE OR REPLACE PACKAGE trigger_control_pkg IS  
    trigger_executed BOOLEAN := FALSE;  
END trigger_control_pkg;  
/
```

El cuerpo del paquete:

```
CREATE OR REPLACE PACKAGE BODY trigger_control_pkg IS  
END trigger_control_pkg;
```


/

El trigger:

```
CREATE OR REPLACE TRIGGER trg_update_my_reservations
INSTEAD OF UPDATE ON my_reservations
FOR EACH ROW
DECLARE
    v_available_signature VARCHAR2(50);
BEGIN
    -- Verificar si el trigger ya ha sido ejecutado en esta
    transacción
    IF NOT trigger_control_pkg.trigger_executed THEN
        -- Marcar el trigger como ejecutado
        trigger_control_pkg.trigger_executed := TRUE;

        -- Verificar disponibilidad de copias del libro
        BEGIN
            SELECT signature INTO v_available_signature
            FROM copies
            WHERE isbn = :NEW.isbn
            AND NOT EXISTS (
                SELECT 1
                FROM loans
                WHERE loans.signature = copies.signature
                AND return IS NULL
            )
            FETCH FIRST 1 ROW ONLY;
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                RAISE_APPLICATION_ERROR(-20001, 'No hay copias
disponibles para reservar.');
```

```
        town = :NEW.town,  
        province = :NEW.province,  
        type = 'R',  
        time = 0,  
        return = NULL  
    WHERE signature = :OLD.signature  
    AND stopdate = :OLD.stopdate;  
END IF;  
END;  
/
```

c) Pruebas:

Probamos la **inserción** de una nueva fila, para ello hacemos inserts en todas las tablas dependientes:

```
INSERT INTO municipalities (TOWN, PROVINCE, POPULATION)  
VALUES ('Granada', 'Granada', 1000);  
  
INSERT INTO routes (ROUTE_ID)  
VALUES ('R002');  
  
INSERT INTO drivers (PASSPORT, EMAIL, FULLNAME, BIRTHDATE, PHONE,  
ADDRESS, CONT_START)  
VALUES ('PASSPORT002', 'driver2@example.com', 'Ana Gómez',  
TO_DATE('1985-07-20', 'YYYY-MM-DD'), 987654321, 'Calle Verdadera  
789', TO_DATE('2024-02-01', 'YYYY-MM-DD'));  
  
INSERT INTO assign_drv (PASSPORT, TASKDATE, ROUTE_ID)  
VALUES ('PASSPORT002', TO_DATE('2025-01-01', 'YYYY-MM-DD'),  
'R002');  
  
INSERT INTO stops (TOWN, PROVINCE, ADDRESS, ROUTE_ID, STOPTIME)  
VALUES ('Granada', 'Granada', 'Avenida Principal 123', 'R002', 45);  
  
INSERT INTO bibuses (PLATE, LAST_ITV, NEXT_ITV)
```

```
VALUES ('M001', TO_DATE('2024-06-01', 'YYYY-MM-DD'),
TO_DATE('2025-06-01', 'YYYY-MM-DD'));

INSERT INTO assign_bus (PLATE, TASKDATE, ROUTE_ID)
VALUES ('M001', TO_DATE('2025-01-01', 'YYYY-MM-DD'), 'R002');

INSERT INTO services (TOWN, PROVINCE, BUS, TASKDATE, PASSPORT)
VALUES ('Granada', 'Granada', 'M001', TO_DATE('2025-01-01',
'YYYY-MM-DD'), 'PASSPORT002');

INSERT INTO books (TITLE, AUTHOR, COUNTRY, LANGUAGE, PUB_DATE,
ALT_TITLE, TOPIC, CONTENT, AWARDS)
VALUES ('Libro de Ejemplo', 'Autor Ejemplo', 'España', 'Español',
2025, NULL, 'Educativo', 'Contenido de ejemplo', NULL);

INSERT INTO editions (ISBN, TITLE, AUTHOR, LANGUAGE,
NATIONAL_LIB_ID)
VALUES ('ISBN98765', 'Libro de Ejemplo', 'Autor Ejemplo',
'Español', 'LIB2');

INSERT INTO copies (SIGNATURE, ISBN, CONDITION, COMMENTS,
DEREGISTERED)
VALUES ('P003', 'ISBN98765', 'N', 'Estado nuevo', NULL);

INSERT INTO users (USER_ID, ID_CARD, NAME, SURNAME1, SURNAME2,
BIRTHDATE, TOWN, PROVINCE, ADDRESS, EMAIL, PHONE, TYPE)
VALUES ('USR002', 'ID987654', 'Carlos', 'López', 'Martínez',
TO_DATE('1990-03-15', 'YYYY-MM-DD'), 'Granada', 'Granada', 'Calle
de la Biblioteca 2', 'carlos.lopez@example.com', 912345678, 'U');

INSERT INTO my_reservations (signature, user_id, stopdate, town,
province, type, time, return, isbn) VALUES ('P003', 'USR002',
TO_DATE('2025-01-01', 'YYYY-MM-DD'), 'Granada', 'Granada', 'R', 0,
NULL, 'ISBN98765');
```

Y hacemos el siguiente procedimiento:

```
BEGIN
    foundicu.SetCurrentUser('USR002');
END;
/
```

El resultado en la vista my_reservations es:

```
SQL> select * from my_reservations where signature = 'P003';
```

SIGNA	USER_ID	STOPDATE	TOWN	PROVINCE	T	TIME	RETUR
P003	USR002	01-JAN-25	Granada	Granada	R	0	01-MA

Y el resultado, por consiguiente, en loans es:

```
SQL> select * from loans where signature = 'P003';
```

SIGNA	USER_ID	STOPDATE	TOWN	PROVINCE	T
P003	USR002	01-JAN-25	Granada	Granada	R

Ahora probamos a **eliminar** la fila creada y observamos el resultado:

```
SQL> DELETE FROM my_reservations
2 WHERE signature = 'P003'
3 AND user_id = 'USR002'
4 AND stopdate = TO_DATE('2025-01-01', 'YYYY-MM-DD');

1 row deleted.

SQL> SELECT * FROM loans
2 WHERE signature = 'P003'
3 AND user_id = 'USR002'
4 AND stopdate = TO_DATE('2025-01-01', 'YYYY-MM-DD');

no rows selected
```

Para **actualizar** utilizamos el siguiente comando:

```
UPDATE my_reservations
SET return = TO_DATE('2025-04-04', 'YYYY-MM-DD')
WHERE signature = 'P003';
```

Y obtenemos este resultado:

```
SQL> select return from my_reservations where signature = 'P003';  
  
RETURN  
-----  
04-APR-25  
  
SQL> select return from loans where signature = 'P003';  
  
RETURN  
-----  
04-APR-25
```

- ★ La vista my_reservations sólo recoge la operación SELECT
Sus disparadores tienen las siguientes características:
 - **Temporalidad:** Se ejecutan en lugar de la operación de eliminación solicitada.
 - **Granularidad:** Se ejecutan para cada fila afectada por la operación.

5. Disparadores

Se van a realizar los 3 primeros disparadores:

a) Evitar los "posts" de usuarios institucionales (bibliotecas municipales).

- Este trigger previene que los usuarios institucionales (identificados por tener el campo type igual a 'L' en la tabla users) puedan insertar nuevas entradas en la tabla posts.
 - **Tabla asociada:** El trigger está asociado a la tabla posts.
 - **Evento que lo dispara:** Se activa en el evento INSERT sobre la tabla posts.
 - **Temporalidad:** Es un trigger de tipo BEFORE, lo que significa que se ejecuta antes de que la operación INSERT se realice en la tabla.
 - **Granularidad:** Opera con granularidad FOR EACH ROW, es decir, se ejecuta una vez por cada fila que se intenta insertar en la tabla posts.
 - **Condición:** Dentro del cuerpo del trigger, se verifica si el campo type del usuario que realiza la inserción es igual a 'L'. Si esta condición se cumple, se ejecuta la acción definida en el trigger.
 - **Acción:** Si la condición anterior se cumple (es decir, si el usuario es de tipo 'L'), el trigger lanza un error utilizando RAISE_APPLICATION_ERROR con el

código -20001 y el mensaje 'Usuarios institucionales no pueden hacer posts.'. Esto impide que la inserción se complete para esos usuarios específicos.

- Código en SQL

```
CREATE OR REPLACE TRIGGER prevent_institutional_posts
BEFORE INSERT ON posts
FOR EACH ROW
DECLARE
    v_user_type CHAR(1);
BEGIN
    SELECT type INTO v_user_type FROM users WHERE user_id =
:NEW.user_id;
    IF v_user_type = 'L' THEN
        RAISE_APPLICATION_ERROR(-20001, 'Usuarios institucionales
no pueden hacer posts.');
```

- Pruebas

Las pruebas consisten en intentar insertar en 'posts' un usuario institucional (una biblioteca), para ello, hemos creado un usuario nuevo e intentado que haga un 'post'. Para ello, debido a la relación de dependencia entre las tablas, hemos tenido que hacer la siguiente secuencia:

```
INSERT INTO municipalities (TOWN, PROVINCE, POPULATION)
VALUES ('Barcelona', 'Barcelona', 1000);

INSERT INTO routes (ROUTE_ID)
VALUES ('R001');
```

```
INSERT INTO drivers (PASSPORT, EMAIL, FULLNAME, BIRTHDATE, PHONE,
ADDRESS, CONT_START)
```

```
VALUES ('PASSPORT001', 'driver@example.com', 'Juan Pérez',
TO_DATE('1980-06-15', 'YYYY-MM-DD'), 123456789, 'Calle Falsa 456',
TO_DATE('2025-01-01', 'YYYY-MM-DD'));

INSERT INTO assign_drv (PASSPORT, TASKDATE, ROUTE_ID)
VALUES ('PASSPORT001', TO_DATE('2024-11-23', 'YYYY-MM-DD'), 'R001');

INSERT INTO stops (TOWN, PROVINCE, ADDRESS, ROUTE_ID, STOPTIME)
VALUES ('Barcelona', 'Barcelona', 'Calle Falsa 123', 'R001', 30);

INSERT INTO bibuses (PLATE, LAST_ITV, NEXT_ITV)
VALUES ('B001', TO_DATE('2024-01-01', 'YYYY-MM-DD'),
TO_DATE('2025-01-01', 'YYYY-MM-DD'));

INSERT INTO assign_bus (PLATE, TASKDATE, ROUTE_ID)
VALUES ('B001', TO_DATE('2024-11-23', 'YYYY-MM-DD'), 'R001');

INSERT INTO services (TOWN, PROVINCE, BUS, TASKDATE, PASSPORT)
VALUES ('Barcelona', 'Barcelona', 'B001', TO_DATE('2024-11-23',
'YYYY-MM-DD'), 'PASSPORT001');

INSERT INTO books (TITLE, AUTHOR, COUNTRY, LANGUAGE, PUB_DATE, ALT_TITLE,
TOPIC, CONTENT, AWARDS)
VALUES ('Libro de prueba', 'Autor de prueba', 'España', 'Español', 2024,
NULL, 'Ficción', 'Contenido de prueba', NULL);

INSERT INTO editions (ISBN, TITLE, AUTHOR, LANGUAGE, NATIONAL_LIB_ID)
VALUES ('ISBN123456789', 'Libro de prueba', 'Autor de prueba', 'Español',
'LIB001');

INSERT INTO copies (SIGNATURE, ISBN, CONDITION, COMMENTS, DEREGISTERED)
VALUES ('P002', 'ISBN123456789', 'N', 'Nuevo estado', NULL)
```

```
INSERT INTO users (USER_ID, ID_CARD, NAME, SURNAME1, SURNAME2, BIRTHDATE,
TOWN, PROVINCE, ADDRESS, EMAIL, PHONE, TYPE)

VALUES ('BIB001', 'BIB123456', 'Biblioteca', 'Central', NULL,
TO_DATE('2000-01-01', 'YYYY-MM-DD'), 'Barcelona', 'Barcelona', 'Avenida
de la Biblioteca 1', 'contacto@biblioteca.es', 934567890, 'L');

INSERT INTO loans (SIGNATURE, USER_ID, STOPDATE, TOWN, PROVINCE, TYPE,
TIME, RETURN)

VALUES ('P002', 'BIB001', TO_DATE('2024-11-23', 'YYYY-MM-DD'),
'Barcelona', 'Barcelona', 'L', 0, NULL);

INSERT INTO posts (SIGNATURE, USER_ID, STOPDATE, POST_DATE, TEXT, LIKES,
DISLIKES)

VALUES ('P002', 'BIB001', TO_DATE('2025-04-02', 'YYYY-MM-DD'),
TO_DATE('2025-04-03', 'YYYY-MM-DD'), 'Texto de prueba de usuario
institucional', 0, 0);
```

Después de realizar estos inserts, creando una nueva biblioteca, no debe dejarnos hacer el último insert, no debe dejar insertar en posts:

```
INSERT INTO posts (SIGNATURE, USER_ID, STOPDATE, POST_DATE, TEXT, LIKES, DISLIKES)
*
ERROR at line 1:
ORA-20001: Usuarios institucionales no pueden hacer posts.
```

b) - Cuando el estado de una copia se establece como "deteriorado", la "fecha de baja" se establece automáticamente en "fecha y hora actuales"

- Este trigger se asegura de que, cuando el campo condition de una fila en la tabla copies se actualiza a 'D' desde un valor diferente, el campo deregistered se establezca automáticamente con la fecha y hora actuales, registrando así el momento en que se realizó dicha actualización.

- **Tabla asociada:** El trigger está asociado a la tabla copies.
- **Evento que lo dispara:** Se activa en el evento UPDATE sobre la tabla copies.
- **Temporalidad:** Es un trigger de tipo BEFORE, lo que significa que se ejecuta antes de que la operación UPDATE se realice en la tabla.

- **Granularidad:** Opera con granularidad FOR EACH ROW, es decir, se ejecuta una vez por cada fila que se actualiza en la tabla copies.
 - **Condición:** Dentro del cuerpo del trigger, se evalúa si el nuevo valor de la columna condition es igual a 'D' y el valor antiguo de condition es diferente de 'D'. Si esta condición se cumple, se ejecuta la acción definida en el trigger.
 - **Acción:** Cuando la condición se cumple (es decir, cuando el campo condition cambia de un valor distinto de 'D' a 'D'), el trigger asigna la fecha y hora actuales al campo deregistered de la fila que se está actualizando, utilizando SYSDATE.
- Código SQL

```
CREATE OR REPLACE TRIGGER update_deregistration_date
BEFORE UPDATE ON copies
FOR EACH ROW
BEGIN
    IF :NEW.condition = 'D' AND :OLD.condition <> 'D' THEN
        SELECT SYSDATE INTO :NEW.deregistered FROM dual;
    END IF;
END;
/
```

- Pruebas

Ya que en el primer trigger hemos insertado una fila en 'copies', la utilizaremos en este apartado:

Comprobamos que existe esa fila en copies:

```
SELECT * FROM copies WHERE SIGNATURE = 'P002';
```

Si aparece 'no rows selected', insertar el siguiente fragmento:

```
INSERT INTO copies (SIGNATURE, ISBN, CONDITION, COMMENTS,  
DEREGISTERED)  
  
VALUES ('P002', 'ISBN123456789', 'N', 'Nuevo estado', NULL);
```

IMPORTANTE: No utilizar el código de arriba en caso de haberlo empleado previamente en el primer trigger.

```
SELECT * FROM copies WHERE SIGNATURE = 'P002';
```

Vemos el estado de la fila, nos fijamos en la columna 'Deregistered':

```
SQL> SELECT * FROM copies WHERE SIGNATURE = 'P002';
```

SIGNA	ISBN	C	COMMENTS	DEREGISTE
P002	ISBN123456789	N	Nuevo estado	

Ahora actualizamos el valor de 'condition':

```
UPDATE copies SET CONDITION = 'D' WHERE SIGNATURE = 'P002';
```

Por último, hacemos la comprobación final:

```
SELECT * FROM copies WHERE SIGNATURE = 'P002';
```

Y vemos que la fecha se ha actualizado:

```
SQL> select * from copies where signature = 'P002';
```

SIGNA	ISBN	C	COMMENTS	DEREGISTE
P002	ISBN123456789	D	Nuevo estado	04-APR-25

- c) Crear "tablas de históricos" tanto para usuarios como para préstamos (no vistas, sino otras dos tablas idénticas). Cuando se elimina un usuario, crear

un registro histórico de ese usuario y mover todos sus préstamos al histórico de préstamos.

- El trigger `archive_user_data` se encarga de archivar la información de los usuarios y sus préstamos antes de que un usuario sea eliminado de la base de datos, asegurando la conservación de registros históricos y manteniendo la integridad de los datos.
- **Tabla asociada:** El trigger está asociado a la tabla `users`.
- **Evento que lo dispara:** Se activa en el evento `DELETE` sobre la tabla `users`.
- **Temporalidad:** Es un trigger de tipo `BEFORE`, lo que significa que se ejecuta antes de que la operación `DELETE` se realice en la tabla.
- **Granularidad:** Opera con granularidad `FOR EACH ROW`, es decir, se ejecuta una vez por cada fila que se va a eliminar de la tabla `users`.
- **Condición:** No se especifica una condición explícita en el trigger; se ejecuta para cada eliminación de fila en la tabla `users`.
- **Acción:** Antes de eliminar un usuario de la tabla `users`, el trigger realiza las siguientes acciones:
 - Inserta la información del usuario que se va a eliminar en la tabla `historic_users`, preservando así un registro histórico del usuario.
 - Copia todos los préstamos asociados a ese usuario desde la tabla `loans` a la tabla `historic_loans`, manteniendo un historial de los préstamos del usuario.
 - Elimina los préstamos del usuario de la tabla `loans`, limpiando los registros activos de préstamos asociados al usuario que se va a eliminar.
- Código SQL

Creamos la tabla `historic_users`:

```
CREATE TABLE historic_users AS SELECT * FROM users WHERE 1=0;
```

Ahora creamos la tabla `historic_loans`:

```
CREATE TABLE historic_loans AS SELECT * FROM loans WHERE 1=0;
```

Y por último, el código:

```
CREATE OR REPLACE TRIGGER archive_user_data  
BEFORE DELETE ON users  
FOR EACH ROW
```

```
BEGIN

-- Insertar el usuario en la tabla de históricos

INSERT INTO historic_users

    (USER_ID, ID_CARD, NAME, SURNAME1, SURNAME2, BIRTHDATE, TOWN,
    PROVINCE, ADDRESS, EMAIL, PHONE, TYPE, BAN_UP2)

VALUES

    (:OLD.USER_ID, :OLD.ID_CARD, :OLD.NAME, :OLD.SURNAME1,
:OLD.SURNAME2, :OLD.BIRTHDATE, :OLD.TOWN, :OLD.PROVINCE,

    :OLD.ADDRESS, :OLD.EMAIL, :OLD.PHONE, :OLD.TYPE,
:OLD.BAN_UP2);

-- Insertar los préstamos del usuario en la tabla de históricos

INSERT INTO historic_loans

SELECT * FROM loans WHERE user_id = :OLD.user_id;

-- Eliminar los préstamos del usuario en la tabla original

DELETE FROM loans WHERE user_id = :OLD.user_id;

END;

/
```

- Pruebas

Nos basamos también en los inserts del trigger 1, habíamos hecho muchos inserts para generar nuevas filas.

IMPORTANTE: Si no se ha seguido el proceso, es necesario realizar los inserts del trigger 1.

Llegado este punto, procedemos a eliminar el usuario.

```
DELETE FROM users WHERE USER_ID = 'BIB001';
```

Comprobamos la tabla 'historic_users':

```
SELECT * FROM historic_users;
```

Comprobamos la tabla 'historic_loans':

```
SELECT * FROM historic_loans;
```

Y vemos el resultado:

```
SQL> select * from historic_loans;
```

SIGNA	USER_ID	STOPDATE	TOWN	PROVINCE	T	TIME	RETURN
S001	USER001	23-NOV-24	Madrid	Madrid	L	0	

```
SQL> select * from historic_users;
```

USER_ID	ID_CARD	NAME	SURNAME1
USER001	12345678X	Biblioteca	Central

6. Conclusiones

En primer lugar, hemos dedicado numerosas horas en perfeccionar los códigos SQL para generar la solución requerida:

En las consultas, se puede observar que las soluciones son las esperadas acorde al enunciado del ejercicio.

En los paquetes, hemos tenido que crear una tabla auxiliar para que nos ayude a encontrar el usuario actual, de esta manera hemos obtenido una parte significativa de las pruebas cubiertas, sin embargo, nos hubiese gustado añadir alguna prueba extra para asegurar el correcto funcionamiento de las mismas.

En las vistas, la primera era más fácil ya que era READ ONLY, sin embargo, en las dos últimas, hemos tenido que crear triggers (hasta en tres ocasiones en la tercera). Hemos obtenido los resultados que queríamos a pesar de la dificultad que suponían los nuevos inserts debido a la alta dependencia entre las clases.

En los triggers, también nos hemos visto afectados por la gran dependencia entre clases, y hemos tenido que emplear mucho tiempo haciendo pruebas que no violasen las relaciones, pero hemos llegado a los resultados que buscábamos

Desde nuestra perspectiva, la práctica tiene un tamaño adecuado comparado con la primera entrega, que consideramos que eran demasiadas tablas por crear y además surgían problemas constantes de almacenamiento. En este caso, nos parece correcto que se tengan que implementar todo tipo de procedimientos para progresar en la materia. Aunque, en cierto momento hemos tenido que crear triggers para hacer vistas, y el apartado mismo de triggers, lo que supone una saturación de este tipo de proceso.

Los plazos han sido buenos, se han separado correctamente ambas entregas. Nuestro principal inconveniente es que no hemos podido estar juntos para realizar la práctica en común, como en la primera ocasión, esta vez hemos tenido que dividir trabajo e ir compartiendo el mismo varias veces para compenetrarnos.

Se agradece que se nos haya aportado unas tablas en condiciones para trabajar, lo que ha hecho que no arrastremos errores y nos dediquemos plenamente a lo indicado en el proyecto.

El punto que mejoraríamos de cara al año que viene sería la entrega, nos gustaría entregar a parte un documento .txt para cada apartado, de tal manera que quedaría más limpia la memoria y se podría hacer los inserts en la consola SQL sin inconvenientes. De la manera actual, la información termina siendo ilegible y muy agresiva a la vista, además, pueden ocurrir con más frecuencia errores al cambiar la información de nuestros documentos a la memoria.