

Profesor:	Francisco Javier Calle Gómez	Grupo	85
Alumno/a:	Iván Sebastián Loor Weir	NIA:	100448737
Alumno/a:	Álvaro González Fúnez	NIA:	100451281
Alumno/a:	Arturo Jiménez Tajuelo Pérez	NIA:	100451161

1. Introducción

En este trabajo hemos aplicado sobre la base de datos relacional diseñada en la práctica anterior una serie de elementos pertenecientes al lenguaje SQL para poder mejorar la BBDD. Entre ellas, hemos implementado consultas avanzadas sobre las tablas existentes, procedimientos, diseño externo mediante la creación de vistas para un usuario y finalmente la creación de disparadores o triggers.

2. Consultas

Consulta 1

a) su diseño en álgebra relacional

Con el fin de realizar la consulta correctamente, nos valemos de la tabla Orders_Clients que se abrevia como OC, la tabla Products que se abrevia P, Varietals que se abrevia V y References que se abrevia R. Con el fin de facilitar más la comprensión del álgebra relacional, utilizamos renombrados. En el primer renombrado, seleccionamos la variedad junto al número de países, filtrando las filas de la relación basadas en una condición de fecha de pedido correspondiente al año anterior al año actual. Luego, utilizamos varios join para combinar múltiples tablas. Posteriormente, llevamos a cabo una operación de agrupación para agrupar los datos por la variedad de producto y calcular la suma de las cantidades de líneas de cliente para cada variedad. Ya después, empleamos una operación de división para dividir el resultado de la agrupación por la variedad del producto. En cambio, en el renombrado B seleccionamos las columnas pedidas en el enunciado, filtrando los datos de la tabla de pedidos para incluir solo aquellos realizados durante el año anterior al año actual. También, volvemos a combinar las tablas usando joins y un left join con otra tabla, garantizando que todas las variedades de productos tengan una “correspondencia” en esta tabla adicional, y si no, se rellenan con ceros. Finalmente, se agrupan los datos según la variedad de producto, el país y el número potencial de consumidores, aplicando funciones de agregación como contar, sumar y promediar.

$$B = \Pi_{p_name, oc_country, count} (DISTINCT oc_username, sum(d_quantity), sum(d_price * d_quantity), AVG(d_quantity), COALESCE(A.Pots.Cons, 0))$$

b) su implementación en SQL

Tratamos de que hiciera el mismo uso que las divisiones planteadas en el álgebra relacional de arriba, modificando un poco la sintaxis y haciendo uso de varias funciones sql. Por ejemplo, la función COALESCE para manejar valores nulos. En este caso, se utilizó para mostrar 0 cuando no hay potenciales consumidores para una variedad particular. También utilizamos EXTRACT para extraer el año de la fecha de los pedidos de clientes o del año anterior al actual, además de utilizar las funciones de agregación en los demás atributos. Luego, en la zona de HAVING calculamos un límite para determinar si un país se considera un potencial consumidor para una variedad específica, siendo del 1% del total de unidades vendidas para esa variedad en el año anterior. Los resultados se agruparon por variedad, país y potenciales consumidores, y luego se ordenaron por variedad, país y el total de unidades vendidas en orden descendente.

```
SELECT
    p.varietal AS Variedad,
    oc.country AS Pais,
    COUNT(DISTINCT oc.username) AS Tot_Compradores,
    SUM(cl.quantity) AS Tot_Unids_Vends,
    SUM(cl.price * cl.quantity) AS Tot_Ingresos,
    AVG(cl.quantity) AS Prom_Unids_Vends_Referenc,
    COALESCE(pc.Pots_Consumidores, 0) AS Pots_Consumidores
FROM
    Orders_Clients oc
    JOIN Client_Lines cl ON oc.orderdate = cl.orderdate
    JOIN References r ON cl.barcode = r.barcode
    JOIN Products p ON r.product = p.product
    JOIN Varietals v ON p.varietal = v.name
    JOIN Origins o ON p.origin = o.name
    LEFT JOIN (
        SELECT
            p.varietal,
            COUNT(DISTINCT oc.country) AS Pots_Consumidores
        FROM
            Orders_Clients oc
            JOIN Client_Lines cl ON oc.orderdate = cl.orderdate
            JOIN References r ON cl.barcode = r.barcode
            JOIN Products p ON r.product = p.product
            JOIN Varietals v ON p.varietal = v.name
            JOIN Origins o ON p.origin = o.name
        WHERE
            EXTRACT(YEAR FROM oc.orderdate) = EXTRACT(YEAR FROM SYSDATE)
    ) pc
GROUP BY
    p.varietal
```

```
HAVING
    SUM(cl.quantity) > (
        SELECT SUM(Tot_Unids_Vends) * 0.01
        FROM (
            SELECT
                SUM(cl.quantity) AS Tot_Unids_Vends
            FROM
                Orders_Clients oc
                JOIN Client_Lines cl ON oc.orderdate =
cl.orderdate
                JOIN References r ON cl.barcode = r.barcode
                JOIN Products p ON r.product = p.product
                JOIN Varietals v ON p.varietal = v.name
                JOIN Origins o ON p.origin = o.name
            WHERE
                EXTRACT(YEAR FROM oc.orderdate) = EXTRACT(YEAR
FROM SYSDATE) - 1
            GROUP BY
                p.varietal, oc.country
        )
        WHERE
            varietal = p.varietal
    )
    ) pc ON p.varietal = pc.varietal
WHERE
    EXTRACT(YEAR FROM oc.orderdate) = EXTRACT(YEAR FROM SYSDATE) - 1
GROUP BY
    p.varietal,
    oc.country,
    COALESCE(pc.Pots_Consumidores, 0)
ORDER BY
    Variedad,
    Pais,
    Tot_Unids_Vends DESC;
```

c) las pruebas realizadas para demostrar que funciona correctamente

La consulta original nos devuelve:

VARIEDAD	PAIS	TOT_COMPRADORES	TOT_UNIDS_VENDS	TOT_INGRESOS	PROM_UNIDS_VENDS_REFERENC	POTS_CONSUMIDORES
Yirgacheffe	Portugal	1	4	374,2	4	80
Yirgacheffe	Qatar	2	17	633,75	8,5	80
Yirgacheffe	Reunion	1	17	1125,74	17	80
Yirgacheffe	Rwanda	1	5	124,85	5	80
Yirgacheffe	Saint Pierre and Miquelon	1	1	96,5	1	80
Yirgacheffe	Samoa	1	16	303,2	16	80
Yirgacheffe	San Marino	1	7	20,93	7	80
Yirgacheffe	Seychelles	1	10	263,1	5	80
Yirgacheffe	Solomon Islands	1	15	284,25	15	80
Yirgacheffe	South Georgia and the South Sandwich Islands	1	28	83,72	28	80
Yirgacheffe	Sudan	2	25	761,41	8,33333333	80
Yirgacheffe	Swaziland	1	1	18,95	1	80
Yirgacheffe	United Arab Emirates	1	3	78,93	3	80
Yirgacheffe	United States Minor Outlying Islands	2	18	453,48	9	80
Yirgacheffe	Uruguay	1	1	54,85	1	80
Yirgacheffe	Uzbekistan	1	15	44,85	15	80
Yirgacheffe	Venezuela	1	6	157,86	6	80
Yirgacheffe	Viet Nam	1	15	465	15	80
Yirgacheffe	Virgin Islands, U.S.	1	6	22,5	6	80
Yirgacheffe	Western Sahara	2	29	460,00	9,66666667	80
Yirgacheffe	Zambia	1	12	299,64	12	80

4498 filas seleccionadas.

Vamos a hacer una prueba, que consiste en agregar una compra de la variedad Yirgacheffe con nuestro usuario y el pais Spain (el cual no posee compras de esa variedad en el ultimo año).
 Primero que todo, tuvimos que insertar datos necesarios del cliente:

```

INSERT INTO Clients (username, reg_datetime, user_passw, name, surn1, surn2, email)
VALUES ('FSDB181', '05/05/23', 'cr7', 'ivan', 'tajuelo', 'funez', '@uc3m.es');

INSERT INTO Client_Addresses (username, waytype, wayname, ZIP, town, country)
VALUES ('FSDB181', 'calle', 'Puebla', '28021', 'Madrid', 'Spain');

INSERT INTO Client_Cards (cardnum, username, card_comp, card_holder, card_expir)
VALUES (9.9585E+11, 'FSDB181', 'Santander', 'Ivan S', '07/08/26');

INSERT INTO Orders_Clients (orderdate, username, town, country, bill_town,
bill_country)
VALUES ('22/11/23', 'FSDB181', 'Madrid', 'Spain', 'Madrid', 'Spain');
  
```

Después, buscamos haciendo consultas una referencia de la variedad a comprar:

```

SQL> select * from References where barcode='QIQ136980722513';

BARCODE      PRODUCT                                F PACK_TYPE  PACK_UNIT  QUANTITY  PRICE  CUR_STOCK  MIN_STOCK  MAX_STOCK
-----
QIQ136980722513  Equivocado de consentida             P cup       ml.         200       2,99    1618       180       3640

SQL> select * from Products where product='Equivocado de consentida';

PRODUCT                                C VARIETAL  ORIGIN  R D
-----
Equivocado de consentida                A Yirgacheffe  Etiop?a  N N
  
```

Realizamos la inserción:


```
INSERT INTO Client_Lines (orderdate, username, town, country, barcode, cardnum,
price, quantity, pay_type)
VALUES ('22/11/23', 'FSDB181', 'Madrid', 'Spain', 'QIQ136900722513', 9.9585E+11,
2.99, 15, 'credit card');
```

Finalmente, volvemos a lanzar la consulta original y observamos como ahora se ha agregado nuestra compra junto a los datos pedidos, debido a que fue realizada en el ultimo año:

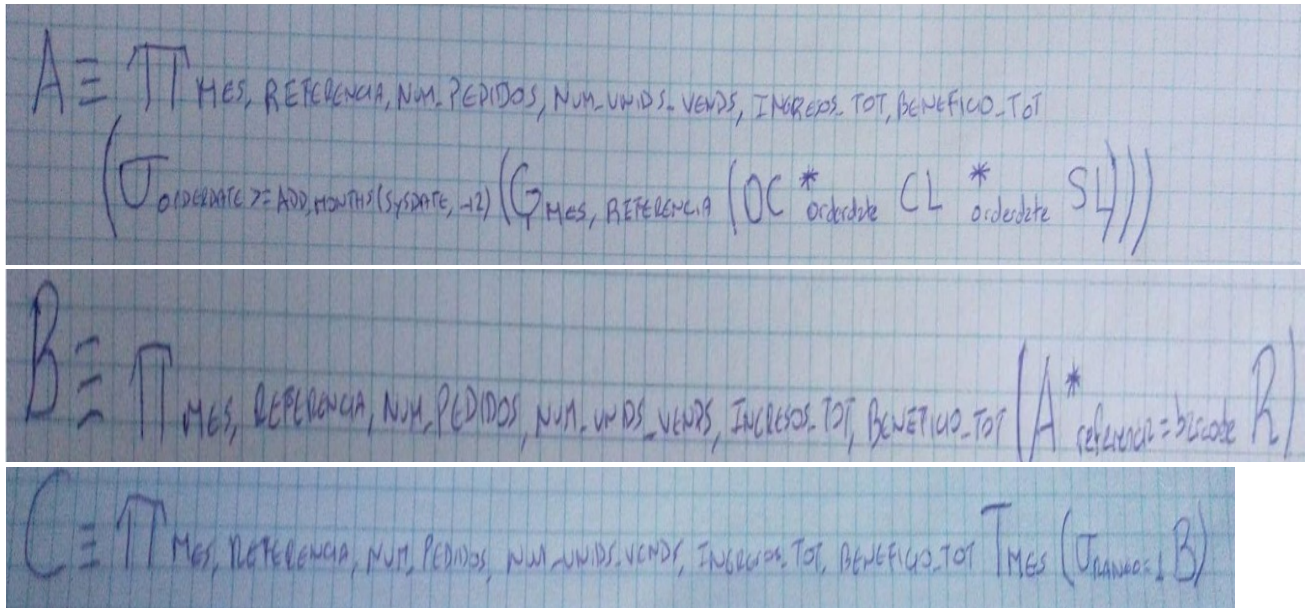
Virgacheffe	Portugal	1	4	374,2	4	81
Virgacheffe	Qatar	2	17	633,75	8,5	81
Virgacheffe	Reunion	1	17	1125,74	17	81
Virgacheffe	Rwanda	1	5	124,85	5	81
Virgacheffe	Saint Pierre and Miquelon	1	1	96,5	1	81
Virgacheffe	Samoa	1	16	303,2	16	81
Virgacheffe	San Marino	1	7	20,93	7	81
Virgacheffe	Seychelles	1	10	263,1	5	81
Virgacheffe	Solomon Islands	1	15	284,25	15	81
Virgacheffe	South Georgia and the South Sandwich Islands	1	28	83,72	28	81
Virgacheffe	Spain	1	15	44,85	15	81
VARIEDAD	PAIS	TOT_COMPRADORES	TOT_UNIDS_VENDS	TOT_INGRESOS	PROM_UNIDS_VENDS_REFERENC	POTS_CONSUMIDORES
Virgacheffe	Sudan	2	25	761,41	8,33333333	81
Virgacheffe	Swaziland	1	1	18,95	1	81
Virgacheffe	United Arab Emirates	1	3	78,93	3	81
Virgacheffe	United States Minor Outlying Islands	2	18	453,48	9	81
Virgacheffe	Uruguay	1	1	54,85	1	81
Virgacheffe	Uzbekistan	1	15	44,85	15	81
Virgacheffe	Venezuela	1	6	157,86	6	81
Virgacheffe	Viet Nam	1	15	465	15	81
Virgacheffe	Virgin Islands, U.S.	1	6	22,5	6	81
Virgacheffe	Western Sahara	2	29	460,09	9,66666667	81
Virgacheffe	Zambia	1	12	299,64	12	81

4499 filas seleccionadas.

Consulta 2

a) su diseño en álgebra relacional

Para hacer la consulta apropiadamente, nos valemos de la tabla Orders_Clients que se abrevia como OC, la tabla Client_Lines que se abrevia CL, Supply_Lines que se abrevia SL y References que se abrevia R. En este caso, también vamos a utilizar renombrados. En el símbolo A calculamos el mes, referencia, número de pedidos, número de unidades vendidas, ingresos totales y beneficio total que poseen los datos de las tablas OC, CL y SL combinadas en el último año. Para la siguiente subconsulta, usamos el símbolo B, de donde se almacenan los mismos atributos que en la subconsulta A, pero con la diferencia de que provienen de una combinación natural entre A y la tabla R. Al final, partimos ya de esos mismos atributos pero esta vez ordenados por mes y provenientes de la subconsulta B filtrada solo con las filas con el rango igual a 1, lo que significa que selecciona solo las referencias que tienen el mayor número de unidades vendidas en cada mes.



b) su implementación en SQL

Para calcular el beneficio total, restamos el costo de suministro (sl.cost) del precio de venta (cl.price). Luego, también nos servimos de la función ROW_NUMBER() para asignar un rango a cada referencia dentro de cada mes, ordenado por la cantidad de unidades vendidas en orden descendente.

```

SELECT
  ms.Mes AS Mes,
  r.barcode AS Referencia,
  Num_Pedidos,
  Num_Unids_Vends,
  Ingresos_Tot,
  Beneficio_Tot
FROM (
  SELECT
    TO_CHAR(oc.orderdate, 'YYYY-MM') AS Mes,
    cl.barcode AS Referencia,
    COUNT(cl.barcode) AS Num_Pedidos,
    SUM(cl.quantity) AS Num_Unids_Vends,
    SUM(cl.price) AS Ingresos_Tot,
    SUM(cl.price - sl.cost) AS Beneficio_Tot,
    ROW_NUMBER() OVER (PARTITION BY TO_CHAR(oc.orderdate, 'YYYY-MM')
    ORDER BY SUM(cl.quantity) DESC) AS Rango
  FROM
    Orders_Clients oc
    JOIN Client_Lines cl ON oc.orderdate = cl.orderdate
    JOIN Supply_Lines sl ON cl.barcode = sl.barcode
  WHERE
  
```

```

        oc.orderdate >= ADD_MONTHS(SYSDATE, -12)
    GROUP BY
        TO_CHAR(oc.orderdate, 'YYYY-MM'),
        cl.barcode
    ) ms
    JOIN References r ON ms.Referencia = r.barcode
    WHERE
        ms.Rango = 1
    ORDER BY
        ms.Mes;
    
```

c) las pruebas realizadas para demostrar que funciona correctamente

El objetivo de esta prueba es comprobar que, si en un mes específico, dejamos de escoger al producto con más unidades vendidas, se escoge al siguiente producto con más unidades vendidas de ese mes.

La salida de la consulta original es:

MES	REFERENCIA	NUM_PEDIDOS	NUM_UNIDS_VENDS	INGRESOS_TOT	BENEFICIO_TOT
2023-04	OII22831Q738220	18	417	404,1	69,66
2023-05	OI0822780772699	15	465	408,3	50,4
2023-06	O0I351740300052	18	396	1266,3	152,04
2023-07	III29452Q587105	16	468	528,8	59,56
2023-08	OQI460610493876	12	336	289,68	33,84
2023-09	QQQ78705Q533142	24	556	105,6	11,4
2023-10	QI096770Q227716	27	459	17,28	2,16
2023-11	QQQ451240782773	9	243	62,82	6,93
2023-12	QQQ96063I470990	18	366	227,7	26,28
2024-01	OQI778450117018	18	540	108	14,04
2024-02	QQQ281740580725	10	322	167,7	23,55
2024-03	OQQ11246Q708278	9	162	527,4	75,63

12 filas seleccionadas.

Ahora, buscamos todos los datos de los productos en un mes dado (por ejemplo, '2023-09') con este código:

```

SELECT
    ms.Mes AS Mes,
    r.barcode AS Referencia,
    Num_Pedidos,
    Num_Unids_Vends,
    Ingresos_Tot,
    Beneficio_Tot
    
```



```

FROM (
  SELECT
    TO_CHAR(oc.orderdate, 'YYYY-MM') AS Mes,
    cl.barcode AS Referencia,
    COUNT(cl.barcode) AS Num_Pedidos,
    SUM(cl.quantity) AS Num_Unids_Vends,
    SUM(cl.price) AS Ingresos_Tot,
    SUM(cl.price - sl.cost) AS Beneficio_Tot,
    ROW_NUMBER() OVER (PARTITION BY TO_CHAR(oc.orderdate, 'YYYY-MM')
ORDER BY SUM(cl.quantity) DESC) AS Rango
  FROM
    Orders_Clients oc
    JOIN Client_Lines cl ON oc.orderdate = cl.orderdate
    JOIN Supply_Lines sl ON cl.barcode = sl.barcode
  WHERE
    TO_CHAR(oc.orderdate, 'YYYY-MM') = '2023-09'
  GROUP BY
    TO_CHAR(oc.orderdate, 'YYYY-MM'),
    cl.barcode
) ms
JOIN References r ON ms.Referencia = r.barcode
ORDER BY
  Num_Unids_Vends ASC;
  
```

La salida de esta consulta es:

MES	REFERENCIA	NUM_PEDIDOS	NUM_UNIDS_VENDS	INGRESOS_TOT	BENEFICIO_TOT
2023-09	OQI33356I363274	3	111	74,25	8,67
2023-09	QII557630411657	6	126	351,6	43,98
2023-09	III29452Q587105	12	128	396,6	44,67
2023-09	IOI09067I423900	3	132	324,18	39,99
2023-09	IOI37783I294862	6	135	29,1	3,72
2023-09	III62697Q978531	20	140	2643	297,45
2023-09	III070570402995	10	140	321,9	32,2
2023-09	IIQ95937Q965197	9	153	1303,65	147,81
2023-09	OII82040Q650945	12	162	270	31,08
2023-09	QQO41416Q877187	15	180	159	21,8
2023-09	QIQ082390436198	8	192	20,8	2,26
2023-09	IOO73843I494054	9	195	25,92	4,71
2023-09	OII64337Q396083	12	198	68,4	11,64
2023-09	IQQ05972I512753	9	216	26,37	2,97
2023-09	OII22831Q738220	9	222	202,05	34,83
2023-09	OOI79081I593125	20	240	241,2	44,15
2023-09	OOI63477Q670918	12	254	8,4	1,14
2023-09	QQQ98657I562583	12	254	508,2	78,78
2023-09	OQI45394I246879	18	270	2753,1	376,38
2023-09	QIO685220597279	15	405	18	2,1
2023-09	QQQ78705Q533142	24	556	105,6	11,4

285 filas seleccionadas.

Vemos que la consulta original escoge correctamente la última referencia con valor 556 en el número de unidades vendidas. Por lo tanto, vamos a realizar la misma consulta original, pero

con la diferencia de que ya no vamos a incluir la última referencia, sino que ahora la consulta escogerá la siguiente referencia, o sea, la que posee 405 unidades vendidas.

```
SELECT
    ms.Mes AS Mes,
    r.barcode AS Referencia,
    Num_Pedidos,
    Num_Unids_Vends,
    Ingresos_Tot,
    Beneficio_Tot
FROM (
    SELECT
        TO_CHAR(oc.orderdate, 'YYYY-MM') AS Mes,
        cl.barcode AS Referencia,
        COUNT(cl.barcode) AS Num_Pedidos,
        SUM(cl.quantity) AS Num_Unids_Vends,
        SUM(cl.price) AS Ingresos_Tot,
        SUM(cl.price - sl.cost) AS Beneficio_Tot,
        ROW_NUMBER() OVER (PARTITION BY TO_CHAR(oc.orderdate, 'YYYY-MM')
        ORDER BY SUM(cl.quantity) DESC) AS Rango
    FROM
        Orders_Clients oc
        JOIN Client_Lines cl ON oc.orderdate = cl.orderdate
        JOIN Supply_Lines sl ON cl.barcode = sl.barcode
    WHERE
        oc.orderdate >= ADD_MONTHS(SYSDATE, -12)
        and cl.barcode != 'QQQ78705Q533142'
    GROUP BY
        TO_CHAR(oc.orderdate, 'YYYY-MM'),
        cl.barcode
) ms
JOIN References r ON ms.Referencia = r.barcode
WHERE
    ms.Rango = 1
ORDER BY
    ms.Mes;
```

La salida de esta consulta es:

MES	REFERENCIA	NUM_PEDIDOS	NUM_UNIDS_VENDS	INGRESOS_TOT	BENEFICIO_TOT
2023-04	OII22831Q738220	18	417	404,1	69,66
2023-05	OIO822780772699	15	465	408,3	50,4
2023-06	OII351740300052	18	396	1266,3	152,04
2023-07	III29452Q587105	16	468	528,8	59,56
2023-08	OQI460610493876	12	336	289,68	33,84
2023-09	QIO685220597279	15	405	18	2,1
2023-10	QIO96770Q227716	27	459	17,28	2,16
2023-11	QQQ451240782773	9	243	62,82	6,93
2023-12	QQQ96063I470990	18	366	227,7	26,28
2024-01	OQI778450117018	18	540	108	14,04
2024-02	QQQ281740580725	10	322	167,7	23,55
12 filas seleccionadas.					
MES	REFERENCIA	NUM_PEDIDOS	NUM_UNIDS_VENDS	INGRESOS_TOT	BENEFICIO_TOT
2024-03	OQQ11246Q708278	9	162	527,4	75,63

Se comprueba que ha cambiado la referencia con más unidades vendidas de ese mes a la segunda referencia con más unidades vendidas de ese mismo mes y damos por válida la prueba.

3. Paquete

- a) su diseño (entradas, salidas, lógica del bloque principal), y en caso de haber necesitado hacer uso de elementos auxiliares (consultas, vistas, otros procedimientos/funciones...) también se debe incluir el diseño de estos (a menos que sean consultas triviales).

Procedimiento 1:

Entradas: CIF del proveedor

Salidas: utilizando DBMS_OUTPUT.PUT_LINE, incluyendo el número de pedidos confirmados, el número de pedidos completados y el promedio del tiempo de entrega para los pedidos confirmados.

Lógica del bloque principal:

Se obtiene el taxID (identificador fiscal) del proveedor correspondiente al CIF proporcionado como entrada.

Se cuentan el número de pedidos confirmados y completados del proveedor en el último año, filtrando por estado "P" (pendiente) y "F" (finalizado), respectivamente.

Se calcula el promedio del tiempo de entrega para los pedidos confirmados del proveedor en el último año.

Se imprimen las estadísticas calculadas en la salida estándar utilizando DBMS_OUTPUT.PUT_LINE.

Para cada referencia de producto que el proveedor suministra, se calcula: el coste actual, el coste mínimo y máximo durante el último año, la diferencia del coste actual respecto al promedio de costes de todas las ofertas, la diferencia del coste actual respecto a la mejor oferta para el producto.

Se imprime el detalle de la oferta para cada referencia de producto, incluyendo el coste actual, el coste mínimo y máximo durante el último año, así como las diferencias calculadas.

Procedimiento 2:

El procedimiento opera sobre la tabla ‘Replacements’ y produce mensajes de éxito o error mediante DBMS_OUTPUT.PUT_LINE.

En cuanto a la lógica del bloque principal:

Se actualiza la columna status de la tabla Replacements, estableciendo el estado de los reemplazos de borrador a confirmado. Esto se hace mediante la sentencia UPDATE.

Posteriormente, se emite un mensaje de éxito indicando que los borradores de pedidos se han convertido en pedidos confirmados. Además, se manejan las excepciones emitiendo un mensaje de error en caso de que ocurra algún problema durante la ejecución del procedimiento.

Tanto en el procedimiento 1 como en el procedimiento 2 nos basamos en el enunciado de la práctica 1 para usar las letras correspondientes al status, utilizamos funciones de agregación y una función ADD_MONTHS para retroceder en meses del año actual.

b) su implementación en SQL

```
CREATE OR REPLACE PACKAGE caffeine AS
    PROCEDURE Informe_Proveedor (cif_provider IN VARCHAR2);
    PROCEDURE Set_Replacement_Orders;
END caffeine;
/

CREATE OR REPLACE PACKAGE BODY caffeine AS
    PROCEDURE Informe_Proveedor (cif_provider IN VARCHAR2) AS
        v_provider_taxid Providers.taxID%TYPE;
        v_total_orders NUMBER;
        v_completed_orders NUMBER;
        v_avg_delivery_time NUMBER;
    BEGIN
        -- Obtener el identificador fiscal del proveedor
        SELECT taxID INTO v_provider_taxid FROM Providers WHERE taxID =
cif_provider;

        -- Contar el número de pedidos confirmados del proveedor en el
último año
        SELECT COUNT(*) INTO v_total_orders
        FROM Replacements r
        WHERE r.taxID = v_provider_taxid
        AND r.orderdate >= ADD_MONTHS(SYSDATE, -12)
        AND r.status = 'P';

        -- Contar el número de pedidos completados del proveedor en el
último año
        SELECT COUNT(*) INTO v_completed_orders
        FROM Replacements r
        WHERE r.taxID = v_provider_taxid
        AND r.orderdate >= ADD_MONTHS(SYSDATE, -12)
        AND r.status = 'F';

        -- Calcular el promedio del tiempo de entrega para ofertas
confirmadas
        SELECT AVG(deldate - orderdate) INTO v_avg_delivery_time
        FROM Replacements r
        WHERE r.taxID = v_provider_taxid
        AND r.orderdate >= ADD_MONTHS(SYSDATE, -12)
        AND r.status = 'P';

        -- Mostrar estadísticas
        DBMS_OUTPUT.PUT_LINE('Numero de pedidos confirmados en el
ultimo anyo: ' || v_total_orders);
        DBMS_OUTPUT.PUT_LINE('Numero de pedidos completados en el
ultimo anyo: ' || v_completed_orders);
```



```
DBMS_OUTPUT.PUT_LINE('Promedio del tiempo de entrega para
ofertas confirmadas: ' || ROUND(v_avg_delivery_time, 2) || '
dias');

-- Mostrar detalle de ofertas
FOR ref IN (SELECT DISTINCT r.barCode FROM Replacements r WHERE
r.taxID = v_provider_taxid) LOOP
    DECLARE
        v_coste_actual NUMBER;
        v_coste_minimo NUMBER;
        v_coste_maximo NUMBER;
        v_diferencia_coste NUMBER;
        v_diferencia_mejor_oferta NUMBER;
    BEGIN
        -- Obtener el coste actual
        SELECT cost INTO v_coste_actual
        FROM Supply_Lines
        WHERE taxID = v_provider_taxid
        AND barCode = ref.barCode;

        -- Obtener el coste mínimo durante el último año
        SELECT MIN(re.payment) INTO v_coste_minimo
        FROM Supply_Lines s JOIN Replacements re ON s.taxID =
re.taxID
        WHERE s.barCode = ref.barCode
        AND re.barCode = ref.barCode
        AND s.taxID = v_provider_taxid
        AND re.orderdate >= ADD_MONTHS(SYSDATE, -12);

        -- Obtener el coste máximo durante el último año
        SELECT MAX(re.payment) INTO v_coste_maximo
        FROM Supply_Lines s JOIN Replacements re ON s.taxID =
re.taxID
        WHERE s.barCode = ref.barCode
        AND re.barCode = ref.barCode
        AND s.taxID = v_provider_taxid
        AND re.orderdate >= ADD_MONTHS(SYSDATE, -12);

        -- Calcular la diferencia del coste actual respecto al
promedio de costes de todas las ofertas
        SELECT AVG(re.payment) INTO v_diferencia_coste
        FROM Supply_Lines s JOIN Replacements re ON s.taxID =
re.taxID
        WHERE s.barCode = ref.barCode
        AND re.barCode = ref.barCode
        AND s.taxID = v_provider_taxid
        AND re.orderdate >= ADD_MONTHS(SYSDATE, -12);
        v_diferencia_coste := v_coste_actual - v_diferencia_coste;
```

```
-- Calcular la diferencia del coste actual respecto a la
mejor oferta para el producto
SELECT MIN(payment) INTO v_diferencia_mejor_oferta
FROM (SELECT payment FROM Replacements WHERE barCode =
ref.barCode AND taxID = v_provider_taxid ORDER BY payment);
IF v_coste_actual = v_diferencia_mejor_oferta THEN
SELECT MIN(payment) INTO v_diferencia_mejor_oferta
FROM (SELECT payment FROM Replacements WHERE barCode =
ref.barCode AND taxID = v_provider_taxid ORDER BY payment OFFSET 1
ROW);
END IF;
v_diferencia_mejor_oferta := v_coste_actual -
v_diferencia_mejor_oferta;

-- Mostrar detalle de oferta
DBMS_OUTPUT.PUT_LINE('Detalle de oferta para la referencia
' || ref.barCode || ':');
DBMS_OUTPUT.PUT_LINE(' Coste actual: ' || v_coste_actual);
DBMS_OUTPUT.PUT_LINE(' Coste minimo durante el ultimo
anyo: ' || v_coste_minimo);
DBMS_OUTPUT.PUT_LINE(' Coste maximo durante el ultimo
anyo: ' || v_coste_maximo);
DBMS_OUTPUT.PUT_LINE(' Diferencia del coste actual
respecto al promedio: ' || ROUND(v_diferencia_coste, 2));
DBMS_OUTPUT.PUT_LINE(' Diferencia del coste actual
respecto a la mejor oferta: ' || ROUND(v_diferencia_mejor_oferta,
2));
END;
END LOOP;
END Informe_Proveedor;

PROCEDURE Set_Replacement_Orders AS
BEGIN
-- Actualizar el estado de los reemplazos de 'D' (Borrador) a
'P' (Confirmado)
UPDATE Replacements
SET status = 'P'
WHERE status = 'D';

-- Mostrar mensaje de éxito
DBMS_OUTPUT.PUT_LINE('Los borradores de pedidos se han
convertido en pedidos confirmados exitosamente.');
```

EXCEPTION

```
WHEN OTHERS THEN
-- Mostrar mensaje de error
DBMS_OUTPUT.PUT_LINE('Error al convertir los borradores de
pedidos en pedidos confirmados.');
```

END Set_Replacement_Orders;

```
END caffeine;  
/
```

c) pruebas

Procedimiento 1

Primero, realizamos una serie de inserciones en la tabla correspondiente con datos aleatorios, pero con status='D'

```
INSERT INTO Replacements (taxID, barCode, orderdate, status, units,  
deldate, payment)  
VALUES ('E22742064D', 'IQI60766I602281', '12/03/15', 'D', 15,  
'14/03/15', 54.5);
```

```
INSERT INTO Replacements (taxID, barCode, orderdate, status, units,  
deldate, payment)  
VALUES ('E22742064D', 'IQI60766I602281', '13/03/15', 'D', 15,  
'15/03/15', 60.2);
```

```
INSERT INTO Replacements (taxID, barCode, orderdate, status, units,  
deldate, payment)  
VALUES ('M86843422E', 'IQQ46127Q226614', '03/08/22', 'D', 50,  
'17/09/22', 44.8);
```

Entonces, al consultar los datos de Replacements obtenemos:

TAXID	BARCODE	ORDERDATE	STATUS	UNITS	DELDATE	PAYMENT
E22742064D	IQI60766I602281	12/03/15	D	15	14/03/15	54,5
E22742064D	IQI60766I602281	13/03/15	D	15	15/03/15	60,2
M86843422E	IQQ46127Q226614	03/08/22	D	50	17/09/22	44,8

Ahora llamamos al procedimiento del paquete:

```
EXECUTE caffeine.Set_Replacement_Orders;
```

La salida esperada es:

```
SQL> EXECUTE caffeine.Set_Replacement_Orders;  
Los borradores de pedidos se han convertido en pedidos confirmados exitosamente.  
  
Procedimiento PL/SQL terminado correctamente.  
  
SQL> select * from Replacements;
```

TAXID	BARCODE	ORDERDATE	STATUS	UNITS	DELDATE	PAYMENT
E22742064D	IQI60766I602281	12/03/15	P	15	14/03/15	54,5
E22742064D	IQI60766I602281	13/03/15	P	15	15/03/15	60,2
M86843422E	IQQ46127Q226614	03/08/22	P	50	17/09/22	44,8

Procedimiento 2

Realizamos más inserciones con datos aleatorios:

```
INSERT INTO Replacements (taxID, barCode, orderdate, status, units,  
deldate, payment)  
VALUES ('E22742064D', 'IQI60766I602281', '24/05/23', 'P', 30,  
'27/05/23', 57.45);
```

```
INSERT INTO Replacements (taxID, barCode, orderdate, status, units,  
deldate, payment)  
VALUES ('E22742064D', 'IQI60766I602281', '09/02/24', 'P', 6, '14/02/24',  
54.58);
```

```
INSERT INTO Replacements (taxID, barCode, orderdate, status, units,  
deldate, payment)  
VALUES ('E22742064D', 'IQI60766I602281', '12/03/24', 'P', 32,  
'18/03/24', 49.30);
```

```
INSERT INTO Replacements (taxID, barCode, orderdate, status, units,  
deldate, payment)  
VALUES ('E22742064D', 'QQO961040656170', '11/01/24', 'P', 26,  
'01/02/24', 21.64);
```

```
INSERT INTO Replacements (taxID, barCode, orderdate, status, units,  
deldate, payment)  
VALUES ('E22742064D', 'QQO961040656170', '15/03/24', 'P', 41,  
'29/03/24', 22.56);
```

```
INSERT INTO Replacements (taxID, barCode, orderdate, status, units,  
deldate, payment)  
VALUES ('E22742064D', 'QQO961040656170', '03/07/23', 'P', 16,  
'24/07/23', 28.48);
```

```
INSERT INTO Replacements (taxID, barCode, orderdate, status, units,  
deldate, payment)
```

```
VALUES ('E22742064D', 'QQO961040656170', '25/05/22', 'F', 3, '30/05/22', 20.05);
```

```
INSERT INTO Replacements (taxID, barCode, orderdate, status, units, deldate, payment)
VALUES ('E22742064D', 'QQO961040656170', '01/08/23', 'F', 25, '09/08/23', 25.98);
```

```
INSERT INTO Replacements (taxID, barCode, orderdate, status, units, deldate, payment)
VALUES ('E22742064D', 'QQO961040656170', '04/10/23', 'F', 34, '12/10/23', 22.56);
```

```
INSERT INTO Replacements (taxID, barCode, orderdate, status, units, deldate, payment)
VALUES ('E22742064D', 'QQO961040656170', '02/01/24', 'F', 25, '06/01/24', 21.35);
```

Observamos los datos a evaluar en la consulta sobre la tabla de borradores:

```
SQL> select * from Replacements;
```

TAXID	BARCODE	ORDERDATE	STATUS	UNITS	DELDATE	PAYMENT
E22742064D	IQI60766I602281	12/03/15	P	15	14/03/15	54,5
E22742064D	IQI60766I602281	13/03/15	P	15	15/03/15	60,2
M86843422E	IQQ46127Q226614	03/08/22	P	50	17/09/22	44,8
E22742064D	IQI60766I602281	24/05/23	P	30	27/05/23	57,45
E22742064D	IQI60766I602281	09/02/24	P	6	14/02/24	54,58
E22742064D	IQI60766I602281	12/03/24	P	32	18/03/24	49,3
E22742064D	QQO961040656170	11/01/24	P	26	01/02/24	21,64
E22742064D	QQO961040656170	15/03/24	P	41	29/03/24	22,56
E22742064D	QQO961040656170	03/07/23	P	16	24/07/23	28,48
E22742064D	QQO961040656170	25/05/22	F	3	30/05/22	20,05
E22742064D	QQO961040656170	01/08/23	F	25	09/08/23	25,98
E22742064D	QQO961040656170	04/10/23	F	34	12/10/23	22,56
E22742064D	QQO961040656170	02/01/24	F	25	06/01/24	21,35

13 filas seleccionadas.

Luego, probamos a llamar al procedimiento 2 utilizando la referencia E22742064D:

```
EXECUTE caffeine.Informe_Proveedor('E22742064D');
```

Esta ejecución nos muestra como resultado el informe correspondiente:


```
SQL> EXECUTE caffeine.Informe_Proveedor('E22742064D');
Numero de pedidos confirmados en el ultimo año: 6
Numero de pedidos completados en el ultimo año: 3
Promedio del tiempo de entrega para ofertas confirmadas: 11,67 dias
Detalle de oferta para la referencia IQI60766I602281:
Coste actual: 54,58
Coste minimo durante el ultimo año: 49,3
Coste maximo durante el ultimo año: 57,45
Diferencia del coste actual respecto al promedio: ,8
Diferencia del coste actual respecto a la mejor oferta: 5,28
Detalle de oferta para la referencia QQO961040656170:
Coste actual: 22,56
Coste minimo durante el ultimo año: 21,35
Coste maximo durante el ultimo año: 28,48
Diferencia del coste actual respecto al promedio: -1,2
Diferencia del coste actual respecto a la mejor oferta: 2,51

Procedimiento PL/SQL terminado correctamente.
```

En cambio, si decidimos hacer el informe con la otra referencia de la cual solo había un borrador y que además no era del último año, nos devuelve:

```
SQL> EXECUTE caffeine.Informe_Proveedor('M86843422E');
Numero de pedidos confirmados en el ultimo año: 0
Numero de pedidos completados en el ultimo año: 0
Promedio del tiempo de entrega para ofertas confirmadas: dias
Detalle de oferta para la referencia IQQ46127Q226614:
Coste actual: 9,3
Coste minimo durante el ultimo año:
Coste maximo durante el ultimo año:
Diferencia del coste actual respecto al promedio:
Diferencia del coste actual respecto a la mejor oferta: -35,5

Procedimiento PL/SQL terminado correctamente.
```

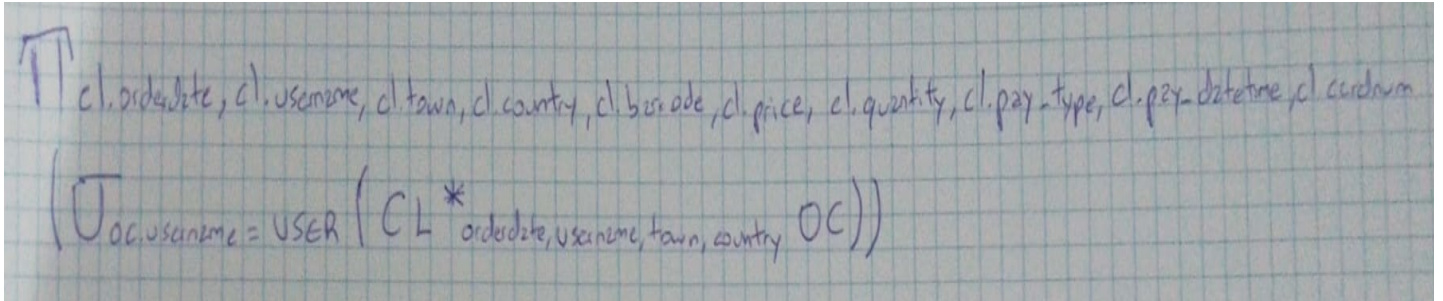
4. Diseño externo

Vista 1

su diseño en álgebra relacional:

Para empezar, proyectamos fecha de pedido, nombre de usuario, ciudad, país, código de barras, precio, cantidad, tipo de pago, fecha y hora de pago, y número de tarjeta, mientras se filtran las tuplas de Client_Lines que coinciden con el nombre de usuario actual, mediante la operación de selección, considerando la condición de que el nombre de usuario en Orders_Clients sea igual al nombre de usuario actual. Luego, se

unen estas tuplas filtradas con las correspondientes de Orders_Clients, basándose en la igualdad de las fechas de pedido, nombres de usuario, ciudades y países.



su implementación en SQL:

```
CREATE OR REPLACE VIEW Mis_compras AS (
SELECT cl.orderdate,
      cl.username,
      cl.town,
      cl.country,
      cl.barcode,
      cl.price,
      cl.quantity,
      cl.pay_type,
      cl.pay_datetime,
      cl.cardnum
FROM Client_Lines cl
JOIN Orders_Clients oc ON cl.orderdate = oc.orderdate
                        AND cl.username = oc.username
                        AND cl.town = oc.town
                        AND cl.country = oc.country
WHERE oc.username = USER)
WITH READ ONLY;
```

Pruebas:

Antes de mostrar los datos de la vista, necesitamos crear todos los datos del cliente de nuestro usuario actual, ya que la vista no contiene nada por ahora. Insertamos datos en todas las tablas necesarias para la ejecución de nuestra tarea.

```
INSERT INTO Clients (username, reg_datetime, user_passw, name, surn1,
surn2, email)
VALUES('FSDB181', '05/05/23', 'cr7', 'ivan', 'tajuelo', 'funez',
'@uc3m.es');
```

```
INSERT INTO Client_Addresses (username, waytype, wayname, ZIP, town,
country)
VALUES('FSDB181', 'calle', 'Puebla', '28021', 'Madrid', 'Spain');

INSERT INTO Client_Cards (cardnum, username, card_comp, card_holder,
card_expir)
VALUES(9.9585E+11, 'FSDB181', 'Santander', 'Ivan S', '07/08/26');

INSERT INTO Orders_Clients (orderdate, username, town, country,
bill_town, bill_country)
VALUES ('21/10/23', 'FSDB181', 'Madrid', 'Spain', 'Madrid', 'Spain');

INSERT INTO Client_Lines (orderdate, username, town, country, barcode,
cardnum, price, quantity,
pay_type)
VALUES ('21/10/23', 'FSDB181', 'Madrid', 'Spain', 'OIQ26181I381053',
9.9585E+11, 5.46, 80, 'credit card');
```

Una vez hecho esto, seleccionamos lo que contiene nuestra vista y nos muestra:

```
SQL> select * from Mis_Compras;
```

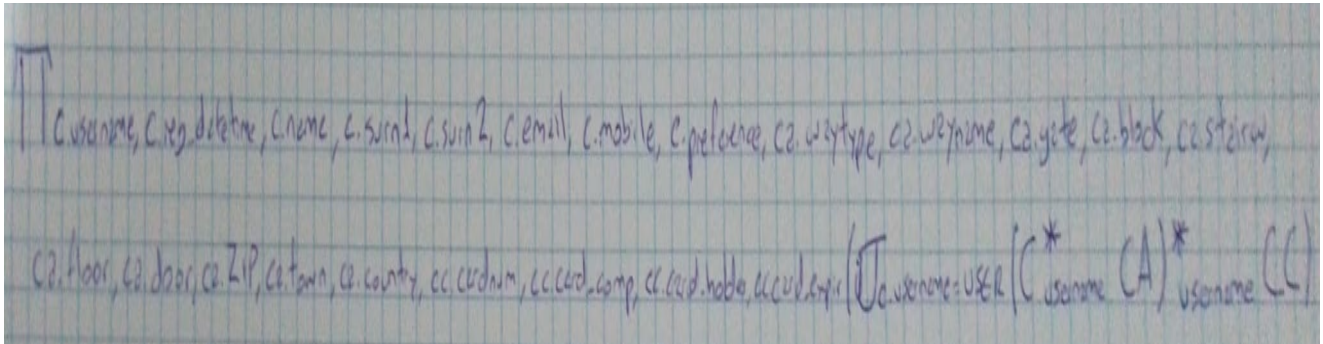
ORDERDAT	USERNAME	TOWN	COUNTRY
21/10/23	FSDB181	Madrid	Spain

COUNTRY	BARCODE	PRICE	QU	PAY_TYPE	PAY_DATE	CARDNUM
Spain	OIQ26181I381053	5,46	80	credit card		9,9585E+11

Vista 2

su diseño en álgebra relacional:

Se comienza seleccionando todas las columnas pertinentes de la tabla Clients junto con información de direcciones y tarjetas asociadas. Para ello, se realiza una unión entre Clients y Client_Addresses basada en el campo username. Posteriormente, se une el resultado de esta operación con la tabla Client_Cards también a través del campo username. Luego, se aplica una operación de proyección para seleccionar solo las columnas especificadas en la consulta original. Al final, se aplica una operación de selección para filtrar las filas donde el nombre de usuario coincide con el usuario actual (USER).



su implementación en SQL:

A diferencia del algebra relacional, tuvimos que usar left joins, sobretodo para que se incluyan todas las filas de la tabla Clients, incluso si no tienen correspondencia en las tablas Client_Addresses y Client_Cards.

```
CREATE OR REPLACE VIEW Mi_perfil AS(  
SELECT c.username,  
       c.reg_datetime,  
       c.name,  
       c.surn1,  
       c.surn2,  
       c.email,  
       c.mobile,  
       c.preference,  
       ca.waytype AS address_waytype,  
       ca.wayname AS address_wayname,  
       ca.gate AS address_gate,  
       ca.block AS address_block,  
       ca.stairw AS address_stairw,  
       ca.floor AS address_floor,  
       ca.door AS address_door,  
       ca.ZIP AS address_ZIP,  
       ca.town AS address_town,  
       ca.country AS address_country,  
       cc.cardnum,  
       cc.card_comp,  
       cc.card_holder,  
       cc.card_expir  
FROM Clients c  
LEFT JOIN Client_Addresses ca ON c.username = ca.username  
LEFT JOIN Client_Cards cc ON c.username = cc.username  
WHERE c.username = USER)  
WITH READ ONLY;
```

Pruebas:

Aprovechando que ya se tenían los datos insertados desde la vista 1, mostramos por pantalla lo que contiene nuestra vista:

```
SQL> select * from Mi_Perfil;
```

USERNAME	REG_DATE	NAME	SURN1	SURN2
FSDB181	05/05/23	ivan	tajuelo	funez

EMAIL	MOBILE	PREFERENCE	ADDRESS_WA	ADDRESS_WAYNAME
@uc3m.es		EMAIL	calle	Puebla

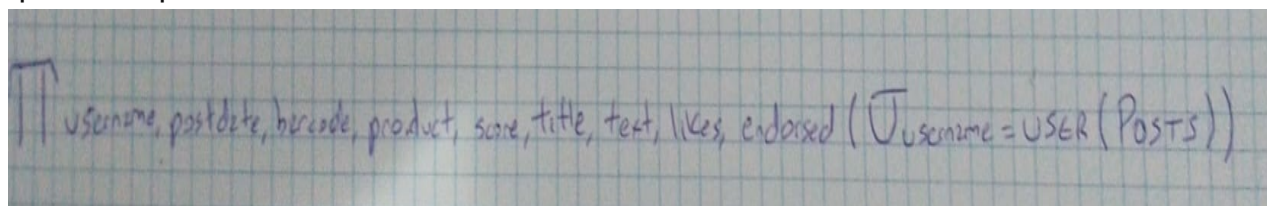
ADD A	AD	ADDRESS	AD	ADDRE	ADDRESS_TOWN	ADDRESS_COUNTRY	CARD
		28021			Madrid	Spain	9,9585E

CARDNUM	CARD_COMP	CARD HOLDER	CARD_EXP
9,9585E+11	Santander	Ivan S	07/08/26

Vista 3

su diseño en álgebra relacional:

Seleccionamos las columnas especificadas, que son todas las pertenecientes a la tabla Posts que corresponden al usuario actual con la variable USER.



$\pi_{username, postdate, barcode, product, score, title, text, likes, endorsed} (\sigma_{username = USER(Posts)})$

su implementación en SQL :

Creamos una vista sencilla que opera sobre la tabla Posts

```
CREATE OR REPLACE VIEW mis_comentarios AS
SELECT *
FROM Posts
WHERE username = USER;
```


Para permitir que la vista cumpla con todos los requisitos, creamos tres triggers.

El primer trigger:

```
CREATE OR REPLACE TRIGGER insertar_comentario_trigger
INSTEAD OF INSERT ON mis_comentarios
FOR EACH ROW
BEGIN
    INSERT INTO Posts (username, postdate, barcode, product, score,
title, text, likes)
VALUES
(:NEW.username, :NEW.postdate, :NEW.barcode, :NEW.product, :NEW.score, :N
EW.title, :NEW.text, :NEW.likes);
END insertar_comentario_trigger;
/
```

El segundo trigger:

```
CREATE OR REPLACE TRIGGER eliminar_comentario_trigger
INSTEAD OF DELETE ON mis_comentarios
FOR EACH ROW
BEGIN
    IF :OLD.likes > 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'No puedes eliminar comentarios
con likes mayores a cero');
    ELSE
        DELETE FROM Posts WHERE username = :OLD.username AND postdate
= :OLD.postdate;
    END IF;
END eliminar_comentario_trigger;
/
```

El tercer trigger:

```
CREATE OR REPLACE TRIGGER actualizar_comentario_trigger
INSTEAD OF UPDATE ON mis_comentarios
FOR EACH ROW
BEGIN
    IF :OLD.likes = 0 THEN
        UPDATE Posts SET text = :NEW.text WHERE username = :OLD.username
AND postdate = :OLD.postdate;
    ELSE
        RAISE_APPLICATION_ERROR(-20001, 'No puedes actualizar comentarios
con likes que no sean cero');
    END IF;
END actualizar_comentario_trigger;
/
```

Pruebas:

Una vez realizados estos pasos previos, verificamos que en Posts no exista ningún cliente asociado a nuestro usuario. Después, insertamos en la vista un comentario y verificamos que también se haya insertado en la tabla Posts correctamente.

Comentario a insertar en la vista:

```
INSERT INTO mis_comentarios (username, postdate, barCode, product, score,
title, text, likes)
VALUES ('FSDB181', '23/10/23', 'OIQ26181I381053', 'Cazador de pena', 3,
'Sobrevalorado', 'normalito sin mas', 50);
```

La prueba realizada en la terminal:

```
SQL> select * from Posts where username=USER;
ninguna fila seleccionada

SQL> INSERT INTO mis_comentarios (username, postdate, barCode, product, score, title, text, likes)
  2 VALUES ('FSDB181', '23/10/23', 'OIQ26181I381053', 'Cazador de pena', 3, 'Sobrevalorado', 'normalito sin mas', 50);
1 fila creada.

SQL> select * from mis_comentarios;
USERNAME          POSTDATE BARCODE          PRODUCT          SCORE T
-----
FSDB181           23/10/23 OIQ26181I381053  Cazador de pena          3 S

SQL> select * from Posts where username=USER;
USERNAME          POSTDATE BARCODE          PRODUCT          SCORE T
-----
FSDB181           23/10/23 OIQ26181I381053  Cazador de pena          3 S
```

A posteriori, insertamos otro comentario más en la vista, pero con la condición de que este tiene 0 likes, a diferencia del anterior que tenía 50 likes.

```
INSERT INTO mis_comentarios (username, postdate, barCode, product, score,
title, text, likes)
VALUES ('FSDB181', '09/11/23', 'OIQ26181I381053', 'Cazador de pena', 4,
'Mejorado', 'mejor version', 0);
```

Entonces, consultamos la cantidad de comentarios que tiene la vista y la tabla con el usuario actual e intentamos borrar el segundo comentario (0 likes). Al ver que lo hace con éxito tanto en la vista como en la tabla, decidimos también eliminar el primer comentario (con 50 likes), pero salta un mensaje del disparador notificandonos que no podemos eliminar comentarios con un número de likes mayor que cero. Por último, comprobamos que no se haya eliminado el comentario a pesar del aviso y damos por exitosa la prueba:

```
SQL> INSERT INTO mis_comentarios (username, postdate, barCode, product, score, title, text, likes)VALUES ('FSDB181', '09/11/23', 'OIQ26181I381053', 'Cazador de pena', 4, 'Mejorado', 'Cazador de pena', 0);
1 fila creada.

SQL> select * from mis_comentarios;

USERNAME          POSTDATE BARCODE          PRODUCT          SCORE T
-----
FSDB181            23/10/23 OIQ26181I381053 Cazador de pena    3 S
FSDB181            09/11/23 OIQ26181I381053 Cazador de pena    4 M

SQL> select * from Posts where username=USER;

USERNAME          POSTDATE BARCODE          PRODUCT          SCORE T
-----
FSDB181            23/10/23 OIQ26181I381053 Cazador de pena    3 S
FSDB181            09/11/23 OIQ26181I381053 Cazador de pena    4 M

SQL> delete from mis_comentarios where likes=0;
1 fila suprimida.

SQL> select * from mis_comentarios;

USERNAME          POSTDATE BARCODE          PRODUCT          SCORE T
-----
FSDB181            23/10/23 OIQ26181I381053 Cazador de pena    3 S

SQL> select * from Posts where username=USER;

USERNAME          POSTDATE BARCODE          PRODUCT          SCORE T
-----
FSDB181            23/10/23 OIQ26181I381053 Cazador de pena    3 S

SQL> delete from mis_comentarios where likes=50;
delete from mis_comentarios where likes=50
*
ERROR en línea 1:
ORA-20001: No puedes eliminar comentarios con likes mayores a cero
ORA-06512: en "FSDB181.ELIMINAR_COMENTARIO_TRIGGER", línea 3
ORA-04088: error durante la ejecucion del disparador 'FSDB181.ELIMINAR_COMENTARIO_TRIGGER'

SQL> select * from mis_comentarios;

USERNAME          POSTDATE BARCODE          PRODUCT          SCORE T
-----
FSDB181            23/10/23 OIQ26181I381053 Cazador de pena    3 S

SQL> select * from Posts where username=USER;

USERNAME          POSTDATE BARCODE          PRODUCT          SCORE T
-----
FSDB181            23/10/23 OIQ26181I381053 Cazador de pena    3 S
```

Después, volvemos a insertar el comentario borrado anteriormente y probamos a cambiar el texto del comentario con 50 likes, provocando que el trigger se dispare mostrando en pantalla que no se pueden actualizar comentarios con likes que no sean cero. Verificamos que no haya actualizado sin querer el comentario haciendo las consultas correspondientes y luego probamos a eliminar el siguiente comentario (el de 0 likes). En efecto, se actualiza tanto en la vista como en la tabla Posts el contenido del comentario pasado y damos por terminada la ejecución de todas las pruebas sobre las vistas.

```
update mis_comentarios set text='a' where title='Sobrevalorado';
update mis_comentarios set text='a' where title='Mejorado';
```

```
SQL> select title, text, likes from mis_comentarios;

TITLE                                     TEXT
-----
Sobrevalorado                           normalito sin mas
Mejorado                                mejor version

SQL> select title, text, likes from Posts where username=USER;

TITLE                                     TEXT
-----
Sobrevalorado                           normalito sin mas
Mejorado                                mejor version

SQL> update mis_comentarios set text='a' where title='Sobrevalorado';
update mis_comentarios set text='a' where title='Sobrevalorado'
*
ERROR en línea 1:
ORA-20001: No puedes actualizar comentarios con likes que no sean cero
ORA-06512: en "FSDB181.ACTUALIZAR_COMENTARIO_TRIGGER", línea 5
ORA-04088: error durante la ejecución del disparador 'FSDB181.ACTUALIZAR_COMENTARIO_TRIGGER'

SQL> select title, text, likes from mis_comentarios;

TITLE                                     TEXT
-----
Sobrevalorado                           normalito sin mas
Mejorado                                mejor version

SQL> select title, text, likes from Posts where username=USER;

TITLE                                     TEXT
-----
Sobrevalorado                           normalito sin mas
Mejorado                                mejor version

SQL> update mis_comentarios set text='a' where title='Mejorado';

1 fila actualizada.

SQL> select title, text, likes from mis_comentarios;

TITLE                                     TEXT
-----
Sobrevalorado                           normalito sin mas
Mejorado                                a

SQL> select title, text, likes from Posts where username=USER;

TITLE                                     TEXT
-----
Sobrevalorado                           normalito sin mas
Mejorado                                a
```

5. Disparadores

Trigger 1

- Descripción del diseño: Tabla a la que está asociado, Evento o eventos en los que se dispara, Temporalidad (antes, después o en vez de), Granularidad (por fila o sentencia), Condición (si la tiene) y Acción (descripción en lenguaje natural).

- Tabla a la que está asociado: Posts
- Evento o eventos en los que se dispara: BEFORE INSERT OR UPDATE
- Temporalidad: Antes (BEFORE)
- Granularidad: Por fila (FOR EACH ROW)
- Condición: No tiene una condición explícita, pero se ejecuta antes de la inserción o actualización en la tabla Posts.
- Acción (descripción en lenguaje natural):

Este trigger se dispara antes de que se inserte o actualice una fila en la tabla Posts. Su objetivo es determinar si el usuario que está realizando la acción ha comprado previamente el producto asociado a la publicación (barcode). Para esto, busca en la tabla Orders_Clients y Client_Lines coincidencias entre el usuario (username) y el producto (barcode) en las órdenes de compra y líneas de cliente respectivamente. Si encuentra alguna coincidencia, establece el valor de endorsed en 'Y' (indicando que el usuario ha comprado el producto anteriormente); de lo contrario, establece el valor en 'N'. Luego, actualiza el atributo endorsed en la fila que se está insertando o actualizando en la tabla Posts con el valor determinado.

- Código (en SQL)

Primero, realizamos una modificación de la tabla Posts para que se permita el cambio del valor de endorsed que realiza el trigger, debido a que esta poseía un formato tipo DATE en ese atributo que no ayudaba a insertar bien un dato de tipo CHAR como lo es el pedido por el enunciado:

```
ALTER TABLE Posts MODIFY endorsed CHAR(1);
```

El código del trigger:

```
CREATE OR REPLACE TRIGGER trg_update_endorsed
BEFORE INSERT OR UPDATE ON Posts
FOR EACH ROW
DECLARE
    v_endorsed_value CHAR(1);
BEGIN
    -- Determinar si el usuario ha comprado el producto antes
    SELECT CASE WHEN COUNT(*) > 0 THEN 'Y' ELSE 'N' END
    INTO v_endorsed_value
    FROM Orders_Clients oc
    INNER JOIN Client_Lines cl ON oc.orderdate = cl.orderdate
    WHERE oc.username = :NEW.username
    AND cl.barcode = :NEW.barcode;

    -- Actualizar el atributo "endorsed"
    :NEW.endorsed := v_endorsed_value;
END;
```


/

- Pruebas

Después de cambiar el formato de endorsed en la tabla Posts, procedemos a insertar un comentario de un usuario llamado 'gh' que ya había comprado el producto referido anteriormente:

```
INSERT INTO Posts (username, postdate, barCode, product, score, title,
text, likes)
VALUES ('gh', '14/03/21', 'QII204750241087', 'Mercurio', 4, 'hola',
'adioos', 2821);
```

Después de hacer la inserción, mostramos por pantalla como el valor de endorsed ahora si tiene un valor que es 'Y', a diferencia de los anteriores comentarios que aún no tenían la nueva herramienta que nos proporciona el trigger:

```
SQL> select username, postdate, endorsed from Posts where username='gh';
```

USERNAME	POSTDATE	E
gh	09/02/15	-
gh	09/05/16	-
gh	10/09/18	-
gh	30/09/18	-
gh	11/10/18	-
gh	18/12/18	-
gh	14/03/21	-
gh	15/06/22	-
gh	24/06/22	-
gh	30/06/22	-
gh	04/07/22	-

USERNAME	POSTDATE	E
gh	05/07/22	-

12 filas seleccionadas.

```
SQL> INSERT INTO Posts (username, postdate, barCode, product, score, title, text, likes)
      2 VALUES ('gh', '14/03/21', 'QII204750241087', 'Mercurio', 4, 'hola', 'adioos', 2821);
```

1 fila creada.

```
SQL> select username, postdate, endorsed from Posts where username='gh';
```

USERNAME	POSTDATE	E
gh	09/02/15	-
gh	09/05/16	-
gh	10/09/18	-
gh	30/09/18	-
gh	11/10/18	-
gh	18/12/18	-
gh	14/03/21	Y
gh	14/03/21	-
gh	15/06/22	-
gh	24/06/22	-
gh	30/06/22	-

USERNAME	POSTDATE	E
gh	04/07/22	-
gh	05/07/22	-

13 filas seleccionadas.

Ahora, para probar que hace lo mismo en el caso contrario, utilizamos al mismo usuario para hacer un post pero con un producto que nunca haya comprado:

```
INSERT INTO Posts (username, postdate, barCode, product, score, title, text, likes)
VALUES ('gh', '17/05/22', 'IOO50302I916906', 'Dados', 2, 'holaaa', 'adiooooo', 165);
```

Después de insertar, vemos que en Posts se le asigna el valor 'N' a su endorsed correspondiente:

```
SQL> INSERT INTO Posts (username, postdate, barCode, product, score, title, text, likes)
2 VALUES ('gh', '17/05/22', 'I0050302I916906', 'Datos', 2, 'holaaa', 'adiooooo', 165);

1 fila creada.

SQL> select username, postdate, endorsed from Posts where username='gh';

USERNAME                                POSTDATE E
-----
gh                                09/02/15
gh                                09/05/16
gh                                10/09/18
gh                                30/09/18
gh                                11/10/18
gh                                18/12/18
gh                                14/03/21 Y
gh                                14/03/21
gh                                17/05/22 N
gh                                15/06/22
gh                                24/06/22

USERNAME                                POSTDATE E
-----
gh                                30/06/22
gh                                04/07/22
gh                                05/07/22

14 filas seleccionadas.
```

Trigger 2

- Descripción del diseño:

- Tabla a la que está asociado: Clients
- Evento o eventos en los que se dispara: AFTER DELETE
- Temporalidad: Después (AFTER)
- Granularidad: Por fila (FOR EACH ROW)
- Condición: No tiene una condición explícita, pero se ejecuta después de que se elimine una fila en la tabla Clients.
- Acción (descripción en lenguaje natural):

Este trigger se dispara después de que se elimina un cliente de la tabla Clients. Su objetivo es mover las compras asociadas a ese cliente a la tabla de compras anónimas (Orders_Anonym), así como también mover los comentarios que haya realizado a la tabla de comentarios anónimos (AnonyPosts). Además, elimina las compras y comentarios del cliente dado de baja de las tablas correspondientes. Primero, inserta en la tabla Orders_Anonym la información de las compras realizadas por el cliente eliminado, teniendo en cuenta las direcciones de facturación y de entrega. Luego, inserta en la tabla Lines_Anonym los detalles de las líneas de compra asociadas a esas compras, incluyendo la información de

la tarjeta de crédito utilizada para el pago.

Después, mueve los comentarios realizados por el cliente eliminado a la tabla AnonyPosts.

Finalmente, elimina las filas correspondientes a las compras y comentarios del cliente dado de baja de las tablas Client_Lines y Posts, respectivamente.

- Código (en SQL)

Código del trigger

```
CREATE OR REPLACE TRIGGER Move_To_Anonymous
AFTER DELETE ON Clients
FOR EACH ROW
BEGIN
    -- Mover compras a compras anónimas
    INSERT INTO Orders_Anonym (orderdate, contact, contact2,
        dliv_datetime, name, surn1, surn2, bill_waytype,
        bill_wayname, bill_gate, bill_block,
        bill_stairw, bill_floor, bill_door, bill_ZIP,
        bill_town, bill_country, dliv_waytype,
        dliv_wayname, dliv_gate, dliv_block,
        dliv_stairw, dliv_floor, dliv_door, dliv_ZIP,
        dliv_town, dliv_country)
        select distinct oc.orderdate, nvl(c.email, c.mobile), nvl2(c.email,
        c.mobile, null), oc.dliv_datetime, c.name, c.surn1, c.surn2, ca.waytype,
        ca.wayname,
        ca.gate, ca.block, ca.stairw, ca.floor, ca.door, ca.ZIP, ca.town,
        ca.country,
        a.waytype, a.wayname, a.gate, a.block, a.stairw, a.floor, a.door,
        a.ZIP, a.town, a.country
        from Orders_Clients oc join Client_Addresses ca on
        oc.username=ca.username
        join Clients c on c.username=oc.username and c.username=ca.username
        join (select oc.username username, ca.waytype waytype, ca.wayname
        wayname, ca.gate gate, ca.block block, ca.stairw stairw, ca.floor floor,
        ca.door door,
        ca.ZIP ZIP, ca.town town, ca.country country from Orders_Clients oc
        join Client_Addresses ca on oc.username=ca.username
        where oc.town=ca.town and oc.country=ca.country and
        oc.username=:old.username) a on a.username=oc.username
        where oc.bill_town=ca.town and oc.bill_country=ca.country
        and oc.username=:old.username ;

    INSERT INTO Lines_Anonym (orderdate, contact, dliv_town,
        dliv_country, barcode, price, quantity,
        pay_type, pay_datetime, card_comp, card_num,
        card_holder, card_expir)
        SELECT distinct o.orderdate, o.contact, o.dliv_town, o.dliv_country,
        cl.barcode, cl.price, cl.quantity, cl.pay_type, cl.pay_datetime,
        cc.card_comp, cl.cardnum, cc.card_holder, cc.card_expir
        FROM Orders_Anonym o join Clients c on nvl(c.email, c.mobile) =
        o.contact
        join Client_Lines cl on cl.username = c.username join Client_Cards cc
        on c.username=cc.username
```

```
WHERE c.username = :old.username;

-- Mover comentarios a comentarios anónimos
INSERT INTO AnonyPosts (postdate, barcode, product, score, title,
text, likes, endorsed)
SELECT postdate, barcode, product, score, title, text, likes,
endorsed
FROM Posts
WHERE username = :old.username;

-- Eliminar las compras y comentarios del cliente dado de baja
DELETE FROM Client_Lines WHERE username = :old.username;
DELETE FROM Posts WHERE username = :old.username;
END;
/
```

- Pruebas:

No hemos conseguido solventar el error de tabla mutante, creemos que se debe a que estamos intentando acceder a la tabla 'Clients' dentro de un trigger que se dispara después de que se elimina una fila de la misma tabla. Esto puede causar conflictos ya que la tabla 'Clients' está siendo modificada y aún no se ha confirmado la eliminación. Intentamos probar con un trigger compuesto que hiciera las inserciones con la temporalidad:granularidad BEFORE STATEMENT, y que hiciera las eliminaciones con la temporalidad:granularidad AFTER EACH ROW, pero no conseguimos dar con la solución a falta de tiempo.

Trigger 3

- Descripción del diseño:

- Tabla a la que está asociado: LINES_ANONYM
- Evento o eventos en los que se dispara: BEFORE INSERT
- Temporalidad: Antes (BEFORE)
- Granularidad: Por fila (FOR EACH ROW)
- Condición: Verifica si el campo card_num en la fila nueva (:NEW.card_num) no es nulo.
- Acción (descripción en lenguaje natural):
Este trigger se dispara antes de que se inserte una nueva fila en la tabla LINES_ANONYM. Su objetivo es prevenir la inserción de compras anónimas que utilicen tarjetas de crédito asociadas a usuarios registrados.
Cuando se proporciona un número de tarjeta de crédito (card_num), el trigger verifica si esta tarjeta está asociada a algún usuario registrado en la tabla Client_Cards. Si encuentra alguna coincidencia, el trigger genera un error y evita la inserción de la compra anónima, indicando que no se permite realizar compras anónimas con una tarjeta de crédito asociada a un usuario registrado.

- Código (en SQL)

Tuvimos un problema a la hora de seleccionar el número de tarjeta de la tabla Client_Cards y consistía en que si le pasábamos el número correcto aun así no detectaba la fila aunque existiera en la tabla. Entonces, optamos por seleccionar en la condición where todos los atributos de la tarjeta excepto su número para hacer el conteo bien y que no hubiera problemas.

Código del trigger:

```
CREATE OR REPLACE TRIGGER Prevent_Anonymous_Purchase
BEFORE INSERT ON LINES_ANONYM
FOR EACH ROW
DECLARE
    v_cardnum NUMBER(20);
BEGIN
    -- Verificar si se proporciona un número de tarjeta de crédito
    IF :NEW.card_num IS NOT NULL THEN
        -- Si se proporciona un número de tarjeta de crédito, verificar
        si está asociado a un usuario registrado
        SELECT COUNT(*)
        INTO v_cardnum
        FROM Client_Cards
        WHERE card_comp = :NEW.card_comp
        AND card_holder = :NEW.card_holder
        AND card_expir = :NEW.card_expir;
        -- Si se encuentra la tarjeta de crédito en la tabla de tarjetas
        de crédito de los clientes,
        -- se produce un error y se impide la inserción de la compra
        anónima
        IF v_cardnum > 0 THEN
            RAISE_APPLICATION_ERROR(-20001, 'No se permite realizar
            compras anónimas con una tarjeta de crédito asociada a un usuario
            registrado.');
```

- Pruebas

Antes de realizar la compra anónima, necesitamos insertarla también en la tabla de pedidos anónimos.

```
INSERT INTO Orders_Anonym (orderdate, contact, name, surn1, bill_waytype,
bill_wayname, bill_ZIP,
bill_town, bill_country, dliv_waytype,
dliv_wayname, dliv_ZIP, dliv_town, dliv_country)
VALUES('05/10/23', 'ivi@gmail', 'ivan', 'loor', 'Puebla', 'Sanabria',
'23028', 'Madrid', 'Espanya', 'Calle', 'DDD', '20821', 'Madrid',
'Espanya');
```

```
INSERT INTO LINES_ANONYM (orderdate, contact, dliv_town, dliv_country,
barcode, price, quantity,
```



```
pay_type, card_comp, card_num, card_holder, card_expir)  
VALUES('05/10/23', 'ivi@gmail', 'Madrid', 'Espanya', 'OIQ93476Q682030',  
24.15, 8, 'credit card', 'Cabestro', 8.3278E+11, 'C Roberto Almira',  
'01/03/25');
```

```
INSERT INTO LINES_ANONYM (orderdate, contact, dliv_town, dliv_country,  
barcode, price, quantity,  
pay_type, card_comp, card_num, card_holder, card_expir)  
VALUES('05/10/23', 'ivi@gmail', 'Madrid', 'Espanya', 'OIQ93476Q682030',  
24.15, 8, 'credit card', 'Cab', 914592799351, 'C Rob', '01/03/25');
```

Al realizar la segunda inserción, que es la que corresponde a la compra anónima, el disparador salta debido a que ya habíamos verificado que la tarjeta insertada existía anteriormente en la tabla Client_Cards para probar esta sección. En cambio, en la última inserción metemos una tarjeta que no pertenecía a ningún cliente y la inserta en la compra anónima sin problema alguno:

```
SQL> INSERT INTO LINES_ANONYM (orderdate, contact, dliv_town, dliv_country, barcode, price, quantity,  
2 pay_type, card_comp, card_num, card_holder, card_expir)  
3 VALUES('05/10/23', 'ivi@gmail', 'Madrid', 'Espanya', 'OIQ93476Q682030', 24.15, 8, 'credit card', 'Cabestro', 8.3278E+11, 'C Roberto Almira', '0  
INSERT INTO LINES_ANONYM (orderdate, contact, dliv_town, dliv_country, barcode, price, quantity,  
*)  
ERROR en línea 1:  
ORA-20001: No se permite realizar compras anonimas con una tarjeta de credito asociada a un usuario registrado.  
ORA-06512: en "FSD8181.PREVENT_ANONYMOUS_PURCHASE", línea 16  
ORA-04088: error durante la ejecución del disparador 'FSD8181.PREVENT_ANONYMOUS_PURCHASE'  
  
SQL> select * from Lines_Anonym where contact = 'ivi@gmail';  
  
ninguna fila seleccionada  
  
SQL> INSERT INTO LINES_ANONYM (orderdate, contact, dliv_town, dliv_country, barcode, price, quantity,  
2 pay_type, card_comp, card_num, card_holder, card_expir)  
3 VALUES('05/10/23', 'ivi@gmail', 'Madrid', 'Espanya', 'OIQ93476Q682030', 24.15, 8, 'credit card', 'Cab', 914592799351, 'C Rob', '01/03/25');  
  
1 fila creada.  
  
SQL> select * from Lines_Anonym where contact = 'ivi@gmail';  
  
ORDERDAT CONTACT DLIV_TOWN DLIV_COUNTRY  
-----  
05/10/23 ivi@gmail Madrid Espanya
```

Trigger 4

- Descripción del diseño: Tabla a la que está asociado, Evento o eventos en los que se dispara, Temporalidad (antes, después o en vez de), Granularidad (por fila o sentencia), Condición (si la tiene) y Acción (descripción en lenguaje natural).
 - Tabla a la que está asociado: Client_Lines
 - Evento o eventos en los que se dispara: AFTER INSERT
 - Temporalidad: Después (AFTER)
 - Granularidad: Por fila (FOR EACH ROW)

- Condición: No tiene una condición explícita, pero se ejecuta después de la inserción en la tabla Client_Lines.
- Acción (descripción en lenguaje natural):

Este trigger se dispara después de que se inserta una nueva fila en la tabla Client_Lines. Su objetivo es actualizar el stock de los productos comprados y gestionar el proceso de reposición de stock si es necesario.

Primero, obtiene el código de barras del producto comprado y luego recupera el stock actual, el stock mínimo y el stock máximo para ese producto desde la tabla References. Calcula el nuevo stock después de la compra y actualiza el valor en la tabla References.

Luego, calcula la cantidad de unidades solicitadas para reponer el stock hasta el máximo. Si la cantidad de proveedores para ese producto es cero, simplemente verifica si se ha alcanzado el stock mínimo y, de ser así, imprime un mensaje indicando que se ha alcanzado y el pedido queda en estado de borrador hasta que esté disponible su proveedor correspondiente.

Si hay uno o más proveedores para el producto y se alcanza el stock mínimo, genera un nuevo pedido de reposición. Para esto, determina el proveedor que ofrece el menor costo para la referencia y lo inserta en la tabla Replacements junto con la cantidad de unidades a solicitar y el costo.

- Código (en SQL)

Nos basamos en el enunciado de la práctica 1 para realizar las acciones en cada caso, con proveedor, muchos proveedores o ninguno, además de las excepciones. Cabe recalcar, que no supimos darle una solución eficiente a cuando no había proveedor, debido a que el enunciado simplemente nos decía que lo guardemos como borrador. Y sumándole los problemas de tener que añadir un nuevo proveedor en la tabla Providers y Supply_Lines o escoger alguno en base a nuestra propia decisión (ya que no estaba explícitamente mencionado en el enunciado) se nos complicó la toma de decisión. Es por eso que preferimos imprimir un mensaje por pantalla anunciando que el pedido estaría en ese estado hasta que tuviese un nuevo proveedor (a pesar de que no lo guardamos como borrador).

```
CREATE OR REPLACE TRIGGER Update_Stocks
AFTER INSERT ON Client_Lines
FOR EACH ROW
DECLARE
    v_cur_stock NUMBER(5);
    v_min_stock NUMBER(5);
    v_max_stock NUMBER(5);
    v_unidades NUMBER(5);
    v_barcode CHAR(15);
    v_proveedor CHAR(10);
    v_cost NUMBER(10,2);
    v_numprov NUMBER(5);
```

```
BEGIN
    -- Obtener el código de barras del producto comprado
    SELECT barcode INTO v_barcode
    FROM References
    WHERE barcode = :NEW.barcode;

    -- Obtener el stock actual y el stock mínimo del producto
    SELECT cur_stock, min_stock, max_stock INTO v_cur_stock, v_min_stock,
v_max_stock
    FROM References
    WHERE barcode = v_barcode;

    -- Calcular el nuevo stock después de la compra
    v_cur_stock := v_cur_stock - :NEW.quantity;

    -- Actualizar el stock en la tabla de referencias
    UPDATE References
    SET cur_stock = v_cur_stock
    WHERE barcode = v_barcode;

    -- Calculo de unidades solicitadas: stock máximo menos el stock
actual
    v_unidades := v_max_stock - v_cur_stock;

    -- Obtener la cantidad de proveedores que tiene la referencia
    select count(*) into v_numprov
    from (select s.taxID, s.cost from Supply_Lines s join References r on
s.barcode=r.barcode where r.barcode=v_barcode)
    ;

    IF v_numprov = 0 THEN
        -- Verificar si es necesario generar un nuevo borrador de pedido de
reposición
        IF v_cur_stock < v_min_stock THEN
            DBMS_OUTPUT.PUT_LINE('Se ha alcanzado el stock minimo para el
producto con codigo de barras ' || v_barcode);
            DBMS_OUTPUT.PUT_LINE('El pedido quedara en estado borrador
hasta que este disponible su proveedor correspondiente');
        END IF;
    ELSIF v_numprov > 0 THEN
        -- Verificar si es necesario generar un nuevo borrador de pedido de
reposición
        IF v_cur_stock < v_min_stock THEN
            DBMS_OUTPUT.PUT_LINE('Se ha alcanzado el stock minimo para el
producto con codigo de barras ' || v_barcode);
            DBMS_OUTPUT.PUT_LINE('Se ha generado un nuevo pedido de
reposicion.');
```

```
-- Obtener el id del proveedor que ofrezca el menor coste
para esa referencia para insertar en Replacements
select * into v_proveedor, v_cost
from (select s.taxID, s.cost from Supply_Lines s join
References r on s.barcode=r.barcode where r.barcode=v_barcode order by
s.cost)
where rownum='1';

-- Generacion de un nuevo pedido de reposición
INSERT INTO Replacements (taxID, barCode, orderdate, status,
units, payment)
VALUES (v_proveedor, v_barcode, :NEW.orderdate, 'P',
v_unidades, v_cost);
END IF;
END IF;
END;
/
```

- Pruebas

```
select barcode, product, quantity, price, cur_stock, min_stock, max_stock
from references where barcode='QIQ88093I746620';

select * from Supply_Lines where barcode='QIQ88093I746620';

INSERT INTO Client_Lines (orderdate, username, town, country, barcode,
cardnum, price, quantity,
pay_type)
VALUES ('21/10/23', 'FSDB181', 'Madrid', 'Spain', 'QIQ88093I746620',
9.9585E+11, 14.83, 95, 'credit card');

select barcode, product, quantity, price, cur_stock, min_stock, max_stock
from references where barcode='QIQ88093I746620';

select * from Replacements where barcode='QIQ88093I746620';
```

Probamos a insertar un producto que posee 1 proveedor:

```
SQL> select barcode, product, quantity, price, cur_stock, min_stock, max_stock from references where barcode='QIQ88093I746620';
```

BARCODE	PRODUCT	QUANTITY	PRICE	CUR_STOCK	MIN_STOCK	MAX_STOCK
QIQ88093I746620	Ojos de paraíso	500	14,83	419	330	3740

```
SQL> select * from Supply_Lines where barcode='QIQ88093I746620';
```

TAXID	BARCODE	COST
L11001725C	QIQ88093I746620	13,34

```
SQL> INSERT INTO Client_Lines (orderdate, username, town, country, barcode, cardnum, price, quantity,
2 pay_type)
3 VALUES ('21/10/23', 'FSDB181', 'Madrid', 'Spain', 'QIQ88093I746620', 9.9585E+11, 14.83, 95, 'credit card');
```

Se ha alcanzado el stock mínimo para el producto con código de barras QIQ88093I746620
Se ha generado un nuevo pedido de reposición.

1 fila creada.

```
SQL> select barcode, product, quantity, price, cur_stock, min_stock, max_stock from references where barcode='QIQ88093I746620';
```

BARCODE	PRODUCT	QUANTITY	PRICE	CUR_STOCK	MIN_STOCK	MAX_STOCK
QIQ88093I746620	Ojos de paraíso	500	14,83	324	330	3740

```
SQL> select * from Replacements where barcode='QIQ88093I746620';
```

TAXID	BARCODE	ORDERDATE	UNITS	DEDATE	PAYMENT
L11001725C	QIQ88093I746620	21/10/23 P	3416		13,34

Probamos a insertar un producto que posee más proveedores, para mostrar que escoge al proveedor más barato:

```
select barcode, product, quantity, price, cur_stock, min_stock, max_stock
from references where barcode='QQQ499620574447';
```

```
select * from Supply_Lines where barcode='QQQ499620574447';
```

```
INSERT INTO Orders_Clients (orderdate, username, town, country,
bill_town, bill_country)
VALUES ('14/06/23', 'FSDB181', 'Madrid', 'Spain', 'Madrid', 'Spain');
```

```
INSERT INTO Client_Lines (orderdate, username, town, country, barcode,
cardnum, price, quantity,
pay_type)
VALUES ('14/06/23', 'FSDB181', 'Madrid', 'Spain', 'QQQ499620574447',
9.9585E+11, 28.4, 50, 'credit card');
```

```
select * from Replacements where barcode='QQQ499620574447';
```

```
select barcode, product, quantity, price, cur_stock, min_stock, max_stock
from references where barcode='QQQ499620574447';
```

Lo cual nos muestra:

```
SQL> select barcode, product, quantity, price, cur_stock, min_stock, max_stock from references where barcode='QQQ499620574447';
```

BARCODE	PRODUCT	QUANTITY	PRICE	CUR_STOCK	MIN_STOCK	MAX_STOCK
QQQ499620574447	Valla	500	28,4	127	80	690

```
SQL> select * from Supply_Lines where barcode='QQQ499620574447';
```

TAXID	BARCODE	COST
Q62114119C	QQQ499620574447	24,99
V63955124J	QQQ499620574447	25,27

```
SQL> INSERT INTO Orders_Clients (orderdate, username, town, country, bill_town, bill_country)
2 VALUES ('14/06/23', 'FSDB181', 'Madrid', 'Spain', 'Madrid', 'Spain');
```

1 fila creada.

```
SQL>
SQL> INSERT INTO Client_Lines (orderdate, username, town, country, barcode, cardnum, price, quantity,
2 pay_type)
3 VALUES ('14/06/23', 'FSDB181', 'Madrid', 'Spain', 'QQQ499620574447', 9.9585E+11, 28.4, 50, 'credit card');
```

Se ha alcanzado el stock mínimo para el producto con código de barras QQQ499620574447
Se ha generado un nuevo pedido de reposición.

1 fila creada.

```
SQL> select * from Replacements where barcode='QQQ499620574447';
```

TAXID	BARCODE	ORDERDATE	UNITS	DEDATE	PAYMENT
Q62114119C	QQQ499620574447	14/06/23	P	613	24,99

```
SQL> select barcode, product, quantity, price, cur_stock, min_stock, max_stock from references where barcode='QQQ499620574447';
```

BARCODE	PRODUCT	QUANTITY	PRICE	CUR_STOCK	MIN_STOCK	MAX_STOCK
QQQ499620574447	Valla	500	28,4	77	80	690

Por último, insertamos un producto sin proveedor alguno:

```
select barcode, product, quantity, price, cur_stock, min_stock, max_stock
from references where barcode='QQQ90712I998588';
```

```
select * from Supply_Lines where barcode='QQQ90712I998588';
```

```
INSERT INTO Orders_Clients (orderdate, username, town, country,
bill_town, bill_country)
VALUES ('03/09/23', 'FSDB181', 'Madrid', 'Spain', 'Madrid', 'Spain');
```

```
INSERT INTO Client_Lines (orderdate, username, town, country, barcode,
cardnum, price, quantity,
pay_type)
VALUES ('03/09/23', 'FSDB181', 'Madrid', 'Spain', 'QQQ90712I998588',
9.9585E+11, 0.95, 98, 'credit card');
```

```
select * from Replacements where barcode='QQQ90712I998588';
```

```
select barcode, product, quantity, price, cur_stock, min_stock, max_stock
```



```
from references where barcode='QOQ90712I998588';
```

Debido a que el enunciado solo nos dice que el pedido quedará en estado ‘borrador’ cuando no tenga proveedores, y no podemos insertar en Replacements sin insertar en Supply_Lines, necesitamos coger un proveedor exacto de Providers o crear uno nuevo para rellenar Supply_Lines. Optamos por dejar que el cliente realice la compra y aparezca solo un aviso de que se resolverá el asunto del proveedor en un futuro cercano:

```
SQL> select barcode, product, quantity, price, cur_stock, min_stock, max_stock from references where barcode='QOQ90712I998588';
```

BARCODE	PRODUCT	QUANTITY	PRICE	CUR_STOCK	MIN_STOCK	MAX_STOCK
QOQ90712I998588	Pozo o tristes sauces	200	,95	207	110	3930

```
SQL>
SQL> select * from Supply_Lines where barcode='QOQ90712I998588';

ninguna fila seleccionada

SQL> INSERT INTO Orders_Clients (orderdate, username, town, country, bill_town, bill_country)
  2 VALUES ('03/09/23', 'FSDB181', 'Madrid', 'Spain', 'Madrid', 'Spain');

1 fila creada.

SQL> INSERT INTO Client_Lines (orderdate, username, town, country, barcode, cardnum, price, quantity,
  2 pay_type)
  3 VALUES ('03/09/23', 'FSDB181', 'Madrid', 'Spain', 'QOQ90712I998588', 9.9585E+11, 0.95, 98, 'credit card');
Se ha alcanzado el stock minimo para el producto con codigo de barras QOQ90712I998588
El pedido quedara en estado borrador hasta que este disponible su proveedor correspondiente

1 fila creada.

SQL> select * from Replacements where barcode='QOQ90712I998588';

ninguna fila seleccionada

SQL> select barcode, product, quantity, price, cur_stock, min_stock, max_stock from references where barcode='QOQ90712I998588';
```

BARCODE	PRODUCT	QUANTITY	PRICE	CUR_STOCK	MIN_STOCK	MAX_STOCK
QOQ90712I998588	Pozo o tristes sauces	200	,95	109	110	3930

6. Conclusiones

No es la primera vez que cursamos esta materia, por lo tanto, al haber realizado la misma práctica el año pasado, nos hemos dado cuenta de que hemos incrementado los conocimientos sobre base de datos. En términos generales, algunos puntos nos parecían muy difíciles el año pasado y este año nos han parecido más accesibles.

En particular, la primera consulta nos pareció más larga de lo normal y empleamos demasiado tiempo en ella.

Para la vista 3 nos complicamos más de lo que debimos, y al principio no agregamos el trigger de insertar, porque ya lo insertaba sin problema, pero al final lo decidimos agregar por si acaso

Empleamos mucho tiempo en errores tontos de claves no encontradas, debido a que no habíamos realizado las inserciones necesarias

En el trigger 2 no conseguimos dar con la solución de la tabla mutante a pesar de los numerosos intentos, consideramos que nos hemos quedado bastante próximos a la solución óptima del apartado.

La falta de tiempo para la memoria nos impidió explicar todo lo que hubiésemos querido en las pruebas.