

Profesor:	Antonio Pantoja	Grupo	80
Alumno/a:	Andrea López Salazar	NIA:	100475780
Alumno/a:		NIA:	
Alumno/a:		NIA:	

1. Introducción

En esta práctica ya contamos con el diseño relacional de la base de datos y con el SQL necesario para poder inicializar tanto las tablas como las inserciones de datos. Trabajaremos entonces sobre cuatro bloques: consultas, paquetes (procedimientos), vistas y disparadores.

2. Consultas

CONSULTA 1 · BESTSELLERS GEOGRAPHIC REPORT

Metodología

Se quiere hacer una consulta sobre la variedad más vendida en el último año por país. Para ello se nos pide adquirir los siguientes datos, cada uno correspondiente con una de las columnas de la tabla resultado de la consulta:

- País (según número de compradores)
- Varietal
- Número de compradores
- Total, de unidades vendidas
- Ingreso total
- Promedio en unidades vendidas por referencia
- Número de países consumidores potenciales de la variedad
 - Compran más del 1% de las unidades vendidas de cualquier producto de ese varietal.

Diseño en Álgebra Relacional

Para simplificar la consulta se ha creado una vista denominada ‘orders’ que recoge todos los datos que necesitamos para realizar la consulta. Se han recogido los siguientes atributos:

- Orderdate
- Country
- Barcode
- Price
- Quantity
- Varietal
- Reference

Esta vista agrupa todas las órdenes tanto de clientes como anónimas de forma que tengamos una sola vista con todas las órdenes realizadas durante el año. Se acotan las inserciones a aquellas que cumplan la condición de la fecha.

Para la implementación de la vista tenemos la siguiente álgebra relacional:

```
orders(orderdate, country, barcode, price, quantity, varietal, reference) :=  
  
    π(orderdate, dliv_country AS country, barcode, price, quantity, varietal, product AS reference)  
(  
  
        ρ(lines_anonym, σ(lines_anonym.orderdate >= add_months(current_date, -12))(lines_anonym ⋈  
references ⋈ products)  
  
        U  
  
        ρ(client_lines, σ(client_lines.orderdate >= add_months(current_date, -12))(client_lines ⋈  
references ⋈ products))  
  
    )
```

Seguidamente, hacemos la consulta correspondiente que seleccionará:

- Country
- Varietal
 - Se agrupa respecto a estas dos variables
- Total_ventas
 - Count(*) : el número total de ventas será el número de filas que aparezcan por varietal en cada país.
- Total_ingresos
 - Suma de la cantidad por el precio
- Avg_unidades_referencia
 - Media de la cantidad de todos los pedidos
- Num_paises_consumidores_potenciales
 - Filtrado de los países que compren una variedad (%) y haciendo count.

```
π(country,          varietal,          total_ventas,          total_ingreso,          avg_unidades_referencia,  
num_paises_consumidores_potenciales) (  
  
    σ(ranking_variedad = 1 AND num_paises_consumidores_potenciales > 0.01 *  
(π(total_ventas)(σ(ventas_por_variedad_total.varietal  
ventas_por_variedad.varietal)(ventas_por_variedad_total))))  
  
    (γ(country,          varietal,          total_ventas,          total_ingreso,          avg_unidades_referencia,  
num_paises_consumidores_potenciales; COUNT(*) AS ranking_variedad) (ventas_por_variedad))  
  
)
```

Implementación en SQL

Vista:

```
-- CONSULTA 1 : Bestsellers Geographic Report  
-- Creo una vista donde agrupo y selecciono todas las ordenes tanto anonimas  
-- como de clientes  
-- Solo incluyo los datos necesatios para realizar el reporte  
  
create or replace view orders AS  
select
```

```
orderdate,  
country,  
barcode,  
price,  
quantity,  
varietal,  
reference  
from (  
  select  
    lines_anonym.orderdate as orderdate,  
    lines_anonym.dliv_country as country,  
    lines_anonym.barcode as barcode,  
    lines_anonym.price as price,  
    lines_anonym.quantity as quantity,  
    products.varietal as varietal,  
    references.product as reference  
  from  
    lines_anonym  
    join references on lines_anonym.barcode = references.barcode  
    join products on references.product = products.product  
  where  
    lines_anonym.orderdate >= add_months(current_date, -12)  
  union  
  select  
    client_lines.orderdate as orderdate,  
    client_lines.country as country,  
    client_lines.barcode as barcode,  
    client_lines.price as price,  
    to_number(client_lines.quantity) as quantity,  
    products.varietal as varietal,  
    references.product as reference  
  from  
    client_lines  
    join references on client_lines.barcode = references.barcode  
    join products on references.product = products.product  
  where  
    client_lines.orderdate >= add_months(current_date, -12)  
);
```

Consulta Principal:

```
-- Realizarmos la consulta principal  
select  
  country,  
  varietal,  
  total_ventas,
```

```
total_ingreso,  
avg_unidades_referencia,  
num_paises_consumidores_potenciales  
from (  
  select  
    country,  
    varietal,  
    count(*) as total_ventas,  
    sum(quantity * price ) as total_ingreso,  
    avg(quantity) as avg_unidades_referencia,  
    row_number() over (partition by country order by count(*) desc) as  
ranking_variedad,  
    count(distinct country) over (partition by varietal) as  
num_paises_consumidores_potenciales  
  from orders  
  group by country, varietal  
) ventas_por_variedad  
where ranking_variedad = 1  
and num_paises_consumidores_potenciales > 0.01 * (  
  select sum(total_ventas)  
  from (  
    select  
      varietal,  
      sum(quantity) as total_ventas  
    from orders  
    group by varietal  
  ) ventas_por_variedad_total  
  where ventas_por_variedad_total.varietal = ventas_por_variedad.varietal  
);
```

Pruebas Realizadas

Echamos un vistazo primero a la vista realizada con el comando: `select * from orders;`
Creamos la vista:

Titulación: GRADO INGENIERIA INFORMATICA

Año Académico: 2023/24 -- Curso: 2º

Asignatura: Ficheros y Bases de Datos

Memoria Práctica 2 – Consultas, bloques, diseño externo y disparadores



uc3m | Universidad Carlos III de Madrid

```
15      lines_anonym.price AS price,
16      lines_anonym.quantity AS quantity,
17      products.varietal AS varietal,
18      references.product AS reference
19  FROM
20      lines_anonym
21  JOIN references ON lines_anonym.barcode = references.barcode
22  JOIN products ON references.product = products.product
23  WHERE
24      lines_anonym.orderdate >= add_months(current_date, -12)
25  UNION
26  SELECT
27      client_lines.orderdate AS orderdate,
28      client_lines.country AS country,
29      client_lines.barcode AS barcode,
30      client_lines.price AS price,
31      TO_NUMBER(client_lines.quantity) AS quantity,
32      products.varietal AS varietal,
33      references.product AS reference
34  FROM
35      client_lines
36  JOIN references ON client_lines.barcode = references.barcode
37  JOIN products ON references.product = products.product
38  WHERE
39      client_lines.orderdate >= add_months(current_date, -12)
40  );
```

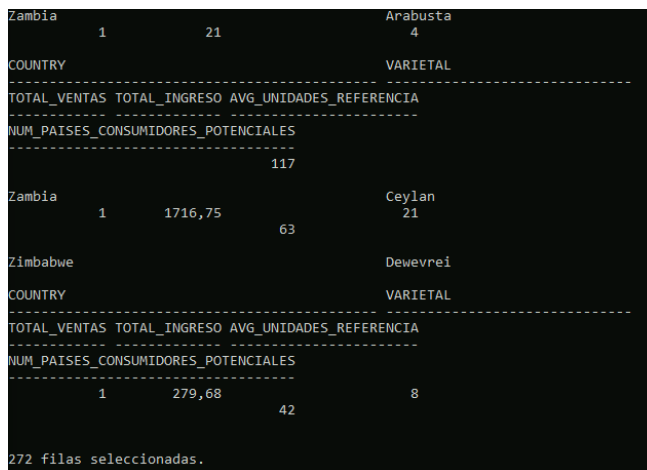
Vista creada.

Hacemos el SELECT:

```
06/03/24 Bahamas 5 S795 000760020395180
111,1
Sirena y consentida
ORDERDAT COUNTRY BARCODE
-----
PRICE QUANTITY VARIETAL
-----
REFERENCE
-----
06/03/24 Heard Island and McDonald Islands IOQ33581I222063
26,8 22 SL-34
Trovar
06/03/24 Iceland IOI86584I705567
8,8 1 Harrar
ORDERDAT COUNTRY BARCODE
-----
PRICE QUANTITY VARIETAL
-----
REFERENCE
-----
Duena o mochila
5177 filas seleccionadas.
```

Al hacer la unión de las dos tablas de órdenes (en total sumaban $3537 + 55206 = 58743$ filas) e imponiendo la restricción de la fecha (órdenes realizadas en el último año) obtenemos un total de 5177 filas.

Sobre esta vista realizamos la consulta principal, para obtener el ranking de variedad más vendida por país:



COUNTRY	VARIETAL	TOTAL_VENTAS	TOTAL_INGRESO	AVG_UNIDADES_REFERENCIA	NUM_PAISES_CONSUMIDORES_POTENCIALES
Zambia	Arabusta	1	21	4	
Zambia		1	1716,75	63	117
Zimbabwe	Ceylan	1	21		
Zimbabwe	Dewevrei	1	279,68	42	8

272 filas seleccionadas.

Al hacer la consulta obtenemos un ranking de 272 puestos donde se han agrupado por países y variedad.

CONSULTA 2 · BUSINESS WAY OF LIFE

Metodología

Se requiere obtener un informe mensual de 12 filas (12 últimos meses) donde se muestre para cada mes:

- Referencia más vendida
- Número de unidades vendidas
- Número de pedido de clientes
- Número de unidades vendidas
- Ingresos totales respecto a la referencia
- Beneficio total (ingresos-costes)

Al igual que para la consulta anterior, se crea una vista previa para recopilar todos los datos necesarios.

Diseño en Álgebra Relacional

Para la vista se recopilan los siguientes datos de las tablas `lines_anonym` y `client_lines`:

- Fecha
- Referencia
- Cantidad
- Total de ingresos
- Coste

```
π(fecha, referencia, quantity, total_ing, cost) (  
    σ(fecha >= add_months(current_date, -12))  
    (γ(lines_anonym.orderdate, references.product;  
        SUM(lines_anonym.quantity) AS quantity,  
        SUM(lines_anonym.price * lines_anonym.quantity) AS total_ing,  
        AVG(supply_lines.cost) AS cost)  
    (lines_anonym ⋈ supply_lines ⋈ references))  
)
```

Para la consulta principal se recoge:

- Month (se recorta la fecha 'YYYY-MM')
- Referencia
- Total unidades
 - Suma de las cantidades de cada pedido
- Total pedidos
 - Contar el número de filas por referencia en el mismo mes.
- Total ingreso
 - Suma de los ingresos de cada pedido
- Beneficio
 - Suma del (total ingreso – coste * cantidad)

```
WITH monthlyReport AS (  
    γ(to_char(fecha, 'YYYY-MM') AS month, referencia;  
    SUM(quantity) AS total_uds,  
    COUNT(*) AS total_orders,  
    SUM(tot_ing) AS tot_ing,  
    SUM(tot_ing - cost * quantity) AS beneficio)  
    (bwol)  
)  
π(month, mejor_referencia, total_orders, tot_uds, tot_ing, beneficio) (  
    σ(rn=1)  
    (ρ(month, referencia, total_orders, tot_uds, tot_ing, beneficio, rn) (  
        ρ(month, to_char(fecha, 'YYYY-MM'), referencia, total_orders, total_uds, tot_ing, beneficio)  
        (monthlyReport)  
        ⋈  
        ρ(rn, row_number()over(partition by month order by tot_uds desc)) (monthlyReport)  
    ))  
)
```

Implementación en SQL

Vista Previa

```
-- BUSINESS WAY OF LIFE  
-- Como en la anterior consulta, creamos una vista agrupando todos los datos  
-- que  
-- necesitamos para realizar la consulta.  
  
create or replace view bwol as  
select  
    fecha,  
    referencia,  
    quantity,  
    total_ing,  
    cost  
from (  
    select  
        lines_anonym.orderdate as fecha,  
        references.product as referencia,  
        sum(lines_anonym.quantity) as quantity,  
        sum(lines_anonym.price * lines_anonym.quantity) as total_ing,
```



```
        avg(supply_lines.cost) as cost
    from
        lines_anonym
    join supply_lines on lines_anonym.barcode = supply_lines.barcode
    join references on lines_anonym.barcode = references.barcode
    group by
        lines_anonym.orderdate, references.product
    union
    select
        client_lines.orderdate as fecha,
        references.product as referencia,
        sum(to_number(client_lines.quantity)) as quantity,
        sum(to_number(client_lines.quantity) * client_lines.price) as
total_ing,
        avg(supply_lines.cost) as cost
    from
        client_lines
    join supply_lines on client_lines.barcode = supply_lines.barcode
    join references on client_lines.barcode = references.barcode
    group by
        client_lines.orderdate, references.product
)
where fecha >= add_months(current_date, -12);
```

Consulta Principal

```
-- Realizo la consulta principal

with MonthlyReport as (
    select
        to_char(fecha, 'YYYY-MM') as month,
        referencia,
        sum(quantity) as total_uds,
        count(*) as total_orders,
        sum(total_ing) as tot_ing,
        sum(total_ing - cost * quantity) as beneficio
    from bwol
    group by
        fecha, referencia
)
select
    month,
    referencia as mejor_referencia,
    total_orders,
    total_uds,
    tot_ing,
    beneficio
```

```
from (  
    select  
        month,  
        referencia,  
        total_orders,  
        total_uds,  
        tot_ing,  
        beneficio,  
        row_number() over (partition by month order by total_uds) as rn  
    from monthlyReport  
)  
where rn = 1;
```

Pruebas Realizadas

Echamos un vistazo primero a la vista realizada con el comando: `select * from bwol;`
Creamos la vista:

```
23 where  
24     fecha >= add_months(current_date, -12);  
Vista creada.
```

Hacemos el SELECT para visualizar contenido:

```
FECHA      REFERENCIA      QUANTITY  
-----  
TOTAL_ING  COST  
-----  
27/08/23 Ermitas o silbando      16  
      425,6      23,67  
  
292 filas seleccionadas.
```

Al filtrar por mes y producto, tenemos en total 292 filas para emplear a la hora de realizar la consulta.

```
MONTH      MEJOR_REFERENCIA      TOTAL_ORDERS  
-----  
TOTAL_UDS  TOT_ING  BENEFICIO  
-----  
2024-01 Sufrir o mujer      1  
      88      12397,44      1488,08  
  
2024-02 Bandido o retrato      1  
      63      5150,25      635,25  
  
2024-03 Charca      1  
      78      38,22      5,46  
  
12 filas seleccionadas.
```

Obtenemos por lo tanto una tabla de 12 filas (lo que significa que había datos suficientes por mes) que representa el monthly report. Estos serían los primeros puestos (primeros meses):

MONTH	MEJOR_REFERENCIA			TOTAL_ORDERS
TOTAL_UDS	TOT_ING	BENEFICIO		
2023-04	Imposible			1
80	261,6	36,6		
2023-05	Estacion			1
36	2237,76	291		
2023-06	Ermitas de muerte			1
136	1698,64	251,26		

3. Paquete Caffeine

INTRODUCCIÓN AL PAQUETE

(Estructura, diseño, observaciones...)

Se va a diseñar el paquete 'caffeine' que va a contar con dos procedimientos públicos.

1. Procedimiento 'Set Replacement Orders': 'set_replacement_orders'
2. Informe sobre un Proveedor: 'prov_report'

PROCEDIMIENTO 1 · SET REPLACEMENT ORDERS

Diseño

Este procedimiento convierte los borradores de pedidos en pedidos confirmados.

set_replacement_orders: Para este procedimiento no han hecho falta ni paso de parámetros, ni uso de funciones ni tablas auxiliares. Sí que se define una variable que va a almacenar el coste con el fin de actualizar el atributo de `payment = cost * units`. Su funcionamiento consta de un bucle for, que va mirando cada borrador presente en la tabla de `replacements` (`status = 'D'`) y va buscando en la tabla de `supply_lines` al proveedor correspondiente para determinar si está ofertando el producto (código de barras) que busca. Si lo encuentra actualiza el estado a 'pendiente' ('P') y calcula el `payment`. Aquí asumimos que cuando encuentra la oferta, pide el producto que más adelante llegará, actualizando `deldate`, que de momento permanece como nulo. Además, estamos asumiendo que la oferta del proveedor está activa en esa fecha y que oferta la cantidad que buscamos.

Implementación SQL

Este sería el código correspondiente al procedimiento, dentro del cuerpo del paquete `caffeine`.

```
-- PROCEDIMIENTO 1
```

```
procedure set_replacement_orders is
    v_cost number(12,2);
begin
    -- Paso 1: Obtengo todas las filas en estado 'D' (borrador)
    for draft in (
        select * from replacements where status = 'D'
    ) loop
        -- Paso 2: verificamos si hay coincidencias en supply_lines para el
        tax_id y barcode
        select
            cost into v_cost
        from
            supply_lines
        where
            supply_lines.taxid = draft.taxid
            and
            supply_lines.barcode = draft.barcode;
        -- Si existe un coste (el proveedor tiene una oferta para ese
        barcode)
        if v_cost is not null then
            update replacements
            set status='P',
            payment = v_cost * draft.units;
            dbms_output.put_line('Replacement order confirmed for: ' ||
            draft.taxid || ' in the date: ' || draft.orderdate || ' of the product: ' ||
            draft.barcode);
        end if;
    end loop;
exception
    when no_data_found then
        dbms_output.put_line('No offer found.');
```

```
end set_replacement_orders;
```

Pruebas

Creación del Paquete:

```
SQL> create or replace package caffeine as
2     procedure set_replacement_orders;
3     procedure prov_report;
4 end caffeine;
5 /

Paquete creado.

SQL> _
```

Requerimientos previos a las pruebas de funcionamiento del paquete.

Seleccionamos algunas líneas que existan en `supply_lines` para poder probar nuestro procedimiento, agregando a `replacements` una línea que contenga un (`taxid`, `barcode`) existente.

```
SQL> select * from supply_lines where rownum<3;
```

TAXID	BARCODE	COST
K10665016P	QQ010649Q369834	8,68
V43078264T	QQ0639490170055	2,51

Insert realizado para la prueba y resultado:

```
-- INSERT INTO Replacements (taxID, barCode, orderdate, status, units, deldate,
payment)
-- no data
-- Inserts para la comprobación de datos
insert into Replacements (taxID, barCode, orderdate, status, units, deldate,
payment)
values ('K10665016P', 'QQ010649Q369834', TO_DATE('2024-04-03', 'YYYY-MM-DD'),
'D', 4, NULL, 0);
```

```
SQL> insert into Replacements (taxID, barCode, orderdate, status, units, deldate,
payment)
2 values ('K10665016P', 'QQ010649Q369834', TO_DATE('2024-04-03', 'YYYY-MM-DD')
, 'D', 4, NULL, 0);
1 fila creada.
```

Prueba de Funcionamiento del Paquete

```
SQL> BEGIN
2 caffeine.set_replacement_orders;
3 END;
4 /
Replacement order confirmed for: K10665016P in the date: 03/04/24 of the
product: QQ010649Q369834
Not match found

Procedimiento PL/SQL terminado correctamente.

SQL> select * from replacements;
```

TAXID	BARCODE	ORDERDATE	STATUS	UNITS	DELDATE	PAYMENT
K10665016P	QQ010649Q369834	03/04/24	P	4		34,72

Como podemos observar, se ha actualizado el estado a ‘pendiente’ y se ha actualizado la fila para que aparezca el pago correspondiente. Queda por fuera de este procedimiento posteriores actualizaciones de la fecha de entrega.

PROCEDIMIENTO 2 · INFORME SOBRE PROVEEDOR

Diseño

Este procedimiento recibe CIF y muestra las siguientes estadísticas:

- Número de pedidos confirmados / completados en el último año
- Promedio de tiempo de entrega para ofertas ya confirmadas
- Detalle de sus ofertas. Para cada referencia:
 - Coste actual
 - Coste mínimo
 - Coste máximo (último año)
 - Diferencia del coste actual – (promedio de costes de todas las ofertas de la referencia)
 - Diferencia respecto a la mejor oferta del producto (o la segunda en caso de tratarse del primer puesto)

En este caso el procedimiento se llaman junto el paso de un parámetro: `prov in varchar2`. Este dato nos permite más tarde filtrar en las tablas por el id de ese proveedor. Hemos usado `varchar2` para facilitar el paso de parámetros y evitar errores de compilación. Para su diseño hemos realizado varias consultas internas, sacando cada uno de los datos. El número de pedidos completados/confirmados y el tiempo medio se hace por fuera del bucle ‘for’, que más adelante irá verificando por cada referencia cuáles son los costes medios, mínimos, máximos y la mejor oferta para el correspondiente producto. La mejor oferta es la de mínimo precio, aunque también se podría haber implementado de forma que no solo sea la de menor precio, sino una ponderación que incluya coste y tiempo de entrega. Pero como no se ha pedido en el apartado, se ha considerado solo el primer parámetro.

Implementación SQL

Dentro del paquete se ha implementado de la siguiente manera el procedimiento:

```
-- PROCEDIMIENTO 2
procedure prov_report(
  prov in varchar2
) is
  num_orders number(10);
  avg_days number(10);
  avg_cost number(15,2);
  min_cost number(15,2);
  max_cost number(15,2);
  best_offer number(15,2);
  cur_ref varchar2(50);
begin
  select count(*) into num_orders from replacements
```

```
where status = 'P' or status = 'F' and orderdate >= add_months(sysdate,
-12)
and taxid = prov;

select avg(deldate-orderdate) into avg_days from replacements where
taxid = prov and
orderdate >=add_months(current_date, -12) and status = 'F';

dbms_output.put_line('Informe para: ' || prov);
dbms_output.put_line('Total orders this year: ' || num_orders);
dbms_output.put_line('Tiempo medio de entrega en días: ' || avg_days);

for ref in (
    select
        avg(cost) as avg_cost,
        min(cost) as min_cost,
        max(cost) as max_cost,
        references.product as curref
    from supply_lines join references on supply_lines.barcode =
references.barcode
    where supply_lines.taxid = prov
    group by references.product
) loop
    avg_cost := ref.avg_cost;
    min_cost := ref.min_cost;
    max_cost := ref.max_cost;
    cur_ref := ref.curref;

    select cost into best_offer
    from (
        select cost
        from supply_lines
        join references on supply_lines.barcode = references.barcode
        where references.product = ref.curref
        order by cost asc
    )
    where rownum = 1;

    dbms_output.put_line('Reference: ' || cur_ref);
    dbms_output.put_line('Average cost: ' || avg_cost);
    dbms_output.put_line('Min cost: ' || min_cost);
    dbms_output.put_line('Max cost: ' || min_cost);
    dbms_output.put_line('Best offer: ' || best_offer);

end loop;
```

```
end prov_report;
```

Pruebas

Requerimientos Previos

Siguiendo las pruebas realizadas en el apartado anterior, tenemos que realizar algunas consultas adicionales. Cogemos el id del proveedor que hemos empleado para las pruebas anteriores y filtramos en supply_lines:

```
select * from supply_lines where taxid = 'K10665016P';
```

Esto nos permite ver qué códigos de barras nos van a permitir hacer las inserciones y no dan problemas por referencia de claves ajenas. Seguidamente, agregamos otras filas a Replacements, inventándonos una fecha de entrega. Aquí no hemos tocado ni las unidades ni el precio final, ya que no afectan a los resultados del procedimiento:

```
-- INSERT INTO REPLACEMENTS AND SUPPLY ORDERS
insert into replacements (taxID, barCode, orderdate, status, units, deldate,
payment)
values ('K10665016P', 'QQ010649Q369834', TO_DATE('2024-05-03', 'YYYY-MM-DD'),
'F', 4, TO_DATE('2024-05-15', 'YYYY-MM-DD'), 0);

insert into replacements (taxID, barCode, orderdate, status, units, deldate,
payment)
values ('K10665016P', 'OQQ879410509260', TO_DATE('2024-05-03', 'YYYY-MM-DD'),
'F', 4, TO_DATE('2024-05-23', 'YYYY-MM-DD'), 0);

insert into replacements (taxID, barCode, orderdate, status, units, deldate,
payment)
values ('K10665016P', 'OQ0159670320195', TO_DATE('2024-05-03', 'YYYY-MM-DD'),
'F', 4, TO_DATE('2024-05-205', 'YYYY-MM-DD'), 0);
```

Nos hemos tenido que cerciorar de que el estado estaba en 'F' en los casos donde se ha proporcionado la fecha de entrega, aunque no haya de momento un procedimiento que se encargue de hacer este proceso de actualización.


```
SQL> @C:\Users\aulavirtual\Desktop\pruebas.sql;

1 fila creada.

SQL> SELECT * FROM REPLACEMENTS;

TAXID          BARCODE          ORDERDATE S    UNITS DELDATE          PAYMENT
-----
K10665016P    QQ010649Q369834 03/04/24 P         4          0
K10665016P    QQ010649Q369834 03/05/24 D         4 15/05/24          0
K10665016P    QQ0879410509260 03/05/24 F         4 23/05/24          0
K10665016P    QQ0159670320195 03/05/24 F         4 20/05/24          0
```

Pruebas de Funcionamiento

Llamamos al procedimiento de nuestro paquete pasándole como parámetro el taxid del proveedor que hemos usado:

```
begin
    caffeine.prov_report('K10665016P')
end;
/
```

Resultado del Informe creado para este proveedor:

```
SQL> @C:\Users\aulavirtual\Desktop\pruebas.sql;
Informe para: K10665016P
Total orders this year: 0
Tiempo medio de entrega en días:
Reference: Toros
Average cost: 15,9
Min cost: ,73
Max cost: ,73
Best offer: ,73
Reference: Montanas o tronar
Average cost: 6,07
Min cost: 1,53
Max cost: 1,53
Best offer: 1,37
Reference: Coyote o swing
Average cost: 40,47
Min cost: 2
Max cost: 2
Best offer: 2
Reference: Siembra de guitarra
Average cost: 27,9
Min cost: 5,64
Max cost: 5,64
Best offer: 5,64

Procedimiento PL/SQL terminado correctamente.
```

4. Diseño externo

INTRODUCCIÓN A LAS VISTAS

Dentro del diseño externo se van a diseñar tres vistas para el perfil usuario ‘cliente’. Se requiere primero definir concepto de ‘usuario actual’. Para este caso se crea un paquete que contenga una variable pública que guarde información del usuario (p.e. user varchar2(30)) que se cargará con el nombre de un usuario en específico.

Al paquete le hemos denominado `user_settings` y contiene una variable global `current_user_var`. Dentro del cuerpo del paquete encontramos un procedimiento que establece el usuario actual del sistema y una función que devuelve dicho usuario. Para acceder a las vistas, solo se muestran los datos del usuario que se ha establecido como el actual.

Importante: En la vida real estableceríamos que el usuario actual sea el `current_user` del sistema, pero para poder usar los datos ya existentes en la base de datos, establecemos nosotros a través de un procedimiento el usuario actual actuando como ‘gestores’. Por lo tanto las vistas solo muestran datos del usuario del sistema, impidiendo la visualización de los datos de otro cliente.

Código SQL del Paquete

```
create or replace package user_settings as
  procedure set_current_user(p_username in varchar2);
  function get_current_user return varchar2;
end user_settings;
/

create or replace package body user_settings as
  current_user_var varchar2(100);
  -- Establece usuario del sistema
  procedure set_current_user(p_username in varchar2) is
  begin
    current_user_var := p_username;
  end set_current_user;
  -- Funcion que nos devuelve el usuario actual
  function get_current_user return varchar2 is
  begin
    return current_user_var;
  end get_current_user;
end user_settings;
/
```

VISTA 1 · MIS_COMPRAS

Diseño Álgebra Relacional

Se trata de una vista de **solo lectura** donde se tienen que poder visualizar todas las compras actuales de un usuario. Para esta vista se seleccionan todas las columnas de `client_lines` que tengan como usuario al establecido por nuestro sistema.

Implementación SQL

```
-- VISTA 1 : MIS_COMPRAS
create or replace view mis_compras as
select *
from orders_clients
-- Imponemos que el usuario debe ser el que se haya establecido como actual en
el sistema
where username = user_settings.get_current_user;
```

Pruebas

1. Establezco como usuario actual a un cliente que tenga registrados datos en `orders_clients`

```
SQL> select * from orders_clients where rownum<3;

ORDERDAT USERNAME
-----
TOWN
-----
COUNTRY                                DLIV_DAT
-----
BILL_TOWN
-----
BILL_COUNTRY                          DISCOUNT
-----
03/11/21 antezana
Valchimeneas del Vertedero
Swaziland                                05/11/21
```

Usuario a establecer: antezana

```
SQL> begin
2   user_settings.set_current_user('antezana');
3 end;
4 /

Procedimiento PL/SQL terminado correctamente.
```

2. Genero la Vista y la visualizo

Creación de la vista:

```
SQL> -- VISTA 1 : MIS_COMPRAS
SQL> create or replace view mis_compras as
2  select *
3  from orders_clients
4  -- Imponemos que el usuario debe ser el que se haya establecido como actual en el sistema
5  where username = user_settings.get_current_user;

Vista creada.
```

Como podemos observar al visualizar la tabla en la siguiente imagen, al llamar a la vista solo se visualizan aquellos elementos que tienen como usuario al establecido como el actual por el sistema (simulamos un mini gestor desde el paquete que hemos creado):

```
ORDERDAT USERNAME
-----
TOWN
-----
COUNTRY                               DLIV_DAT
-----
BILL_TOWN
-----
BILL_COUNTRY                           DISCOUNT
-----
Valguas de la Costa
Togo                                   0

ORDERDAT USERNAME
-----
TOWN
-----
COUNTRY                               DLIV_DAT
-----
BILL_TOWN
-----
BILL_COUNTRY                           DISCOUNT
-----
12/01/23 antezana
Valchimeneas del Vertedero
Swaziland                             12/01/23

ORDERDAT USERNAME
-----
TOWN
-----
COUNTRY                               DLIV_DAT
-----
BILL_TOWN
-----
BILL_COUNTRY                           DISCOUNT
-----
Valguas de la Costa
Togo                                   0

137 filas seleccionadas.
```

VISTA 2 · MI_PERFIL

Diseño Álgebra Relacional

Vista de **solo lectura** que debe incluir datos personales, direcciones y tarjetas de crédito. Creamos la vista que una las tres tablas (`clients`, `client_addresses`, `client_cards`). Se mostrarán esos datos filtrando por el usuario actual del sistema, que de momento, para poder realizar las pruebas, lo definiremos nosotros a través del procedimiento del paquete `user_settings`.

Implementación SQL

```
create or replace view mi_perfil as
select
```

```
clients.username as usuario,
clients.reg_datetime as fecha_registro,
clients.user_passw as contraseña,
clients.name as nombre,
clients.surn1 as apellido1,
clients.surn2 as apellido2,
clients.email as email,
clients.mobile as telefono,
clients.preference as preferencia,
clients.voucher as voucher,
clients.voucher_exp as voucher_exp,
client_addresses.waytype,
client_addresses.wayname,
client_addresses.gate,
client_addresses.block,
client_addresses.stairw,
client_addresses.floor,
client_addresses.door,
client_addresses.zip,
client_addresses.town,
client_addresses.country,
client_cards.cardnum as tarjeta_credito,
client_cards.card_comp,
client_cards.card_holder,
client_cards.card_expir
from
  clients
join
  client_addresses ON clients.username = client_addresses.username
join
  client_cards ON clients.username = client_cards.username
where
  clients.username = user_settings.get_current_user;
```

Pruebas

Establecemos un nuevo usuario del sistema, esta vez 'pereyra'.

```
SQL> begin
2   user_settings.set_current_user('pereyra');
3 end;
4 /

Procedimiento PL/SQL terminado correctamente.
```

Esta vista es de solo lectura, por lo que el usuario no podrá hacer operaciones sobre ella. Se muestran los datos guardados para el usuario actual del sistema. Creamos la vista:

```
31 client_addresses ON clients.username = client_addresses.username
32 join
33 client_cards ON clients.username = client_cards.username
34 where
35 clients.username = user_settings.get_current_user;
Vista creada.
```

Visualizamos el contenido, que solo debería mostrarnos los datos para el usuario ‘pereyra’:

```
-----
REFERENCIA  VOUCHER VOUCHER_ WAYTYPE  WAYNAME  GAT B
-----
ST FLOOR DO ZIP  TOWN
-----
COUNTRY  TARJETA_CREDITO CARD_COMP
-----
CARD HOLDER  CARD_EXP
-----
Canada  7,9772E+11 Custer Card
-----
USUARIO  FECHA_RE CONTRASE?A
-----
NOMBRE  APELLIDO1
-----
APELLIDO2
-----
EMAIL  TELEFONO
-----
REFERENCIA  VOUCHER VOUCHER_ WAYTYPE  WAYNAME  GAT B
-----
ST FLOOR DO ZIP  TOWN
-----
COUNTRY  TARJETA_CREDITO CARD_COMP
-----
CARD HOLDER  CARD_EXP
-----
Joab Pereyra  01/01/28
-----
USUARIO  FECHA_RE CONTRASE?A
-----
NOMBRE  APELLIDO1
-----
APELLIDO2
-----
EMAIL  TELEFONO
-----
REFERENCIA  VOUCHER VOUCHER_ WAYTYPE  WAYNAME  GAT B
-----
ST FLOOR DO ZIP  TOWN
-----
COUNTRY  TARJETA_CREDITO CARD_COMP
-----
CARD HOLDER  CARD_EXP
-----
SQL> _
```

Efectivamente podemos observar que solo encontramos datos guardados para ese nombre de usuario.

VISTA 3 · MIS_COMENTARIOS

Diseño Álgebra Relacional

Esta vista tiene operatividad completa, enumerando los comentarios del usuario actual. Le permite insertar nuevos, eliminarlos o cambiar el texto de cualquiera de ellos (solo si el atributo ‘likes’ de la publicación correspondiente son cero). El resto de atributos no pueden cambiarse. Se evita la modificación del comentario si sus ‘likes’ son mayores a cero.

Para la implementación de esta vista se ha creado un paquete que servirá para la modificación, inserción y eliminación de comentarios. Cada una de las acciones anteriores será llevada a cabo por un procedimiento dentro del paquete `mis_comentarios_ops` quien llamará al paquete de `user_settings` para asegurarse de que solo se realizan para el usuario actual del sistema. Por lo tanto, estos procedimientos no reciben como parámetro el usuario, sino que lo determinan internamente. De nuevo, podemos implementar el `user_settings` para que el usuario actual sea el

current_user, pero para favorecer la calidad de las pruebas será una variable que podremos determinar a través del procedimiento interno set_user.

Implementación SQL

Paquete para el manejo de comentarios

```
-- Para la vista anterior se va a crear un paquete que permite insertar,
eliminar y actualizar comentarios
create or replace package mis_comentarios_ops as
    -- insertar un nuevo comentario
    procedure insertar_comentario(ref varchar2, punt number, text varchar2, bar
char, title varchar2);
    -- eliminar comentario is Likes = 0
    procedure eliminar_comentario(fecha date);
    -- actualizar texto de un comentario
    procedure actualizar_comentario(fecha date, texto varchar2, punt number);
end mis_comentarios_ops;
/
```

```
create or replace package body mis_comentarios_ops as
    -- Procedimiento para insertar el nuevo comentario
    procedure insertar_comentario(ref varchar2, punt number, text varchar2, bar
char, title varchar2) is
        usr varchar2(30);
    begin
        usr := user_settings.get_current_user;
        insert into posts (username, postdate, barcode, product, score, title,
text, likes, endorsed)
            values (usr, current_date, bar, ref, punt, title, text, 0, null);
    end insertar_comentario;

    procedure eliminar_comentario(fecha date) is
        usr varchar2(30);
    begin
        usr := user_settings.get_current_user;
        delete from posts
            where username = usr and trunc(postdate) = trunc(fecha)
            and likes=0;
    end eliminar_comentario;

    procedure actualizar_comentario(fecha date, texto varchar2, punt number) is
        usr varchar2(30);
    begin
        usr := user_settings.get_current_user;
        update posts
            set text = texto, score = punt
```

```
where username = usr and trunc(postdate) = trunc(fecha) and likes = 0;  
end actualizar_comentario;  
end mis_comentarios_ops;  
/
```

Vista de mis_comentarios

```
-- VISTA 3 : MIS COMENTARIOS  
create or replace view mis_comentarios as  
select  
  *  
from posts  
where username = user_settings.get_current_user;
```

Pruebas

Buscamos un usuario que tenga varias entradas en la tabla de publicaciones y lo establecemos como el usuario actual del sistema:

```
-- Establezco usuario  
begin  
  user_settings.set_current_user('jesus1');  
end;  
/
```

Creo mi vista y la visualizo:

```
SQL> create or replace view mis_comentarios as  
2  select  
3  *  
4  from posts  
5  where username = user_settings.get_current_user;  
Vista creada.
```

TITLE
TEXT
LIKES ENDORSED
11858

7 filas seleccionadas.

Como podemos ver el usuario tiene 7 comentarios / publicaciones. Vamos a probar nuestros procedimientos, empezando por insertar un comentario:

```
SQL> -- PRUEBAS  
SQL> begin  
2  mis_comentarios_ops.insertar_comentario('Paisajes y cantar', 4, 'Me ha gustado mucho. Volveria a comprar', null,  
null);  
3  end;  
4  /  
Procedimiento PL/SQL terminado correctamente.
```

Volvemos a visualizar la vista y comprobamos que ahora son 8 filas y que se ha insertado el comentario correctamente:


```
      LIKES ENDORSED
-----
Paisajes y cantar                                4
Me ha gustado mucho. Volveria a comprar

USERNAME                                POSTDATE BARCODE
-----
PRODUCT                                SCORE
-----
TITLE
-----
TEXT
-----
      LIKES ENDORSED
-----
      0

8 filas seleccionadas.
```

Ahora vamos a modificar el comentario que acabamos de insertar:

```
begin
  mis_comentarios_ops.actualizar_comentario(current_date, 'Despues de una
semana se ha puesto rancio.', 1);
end;
/
```

Resultado:

```
      LIKES ENDORSED
-----
Paisajes y cantar                                1
Despues de una semana se ha puesto rancio.
```

Si likes >0, no se realiza la modificación.

Ya para terminar, eliminamos el comentario:

```
begin
  mis_comentarios_ops.eliminar_comentario(current_date);
end;
/
```

Resultado:

```
LIKES ENDORSED
-----
u YTzopf hYI cSMUEe nKXb Uixg SWMuI BDDbTAHfTl HTqYwM JLuG fKIC cGHfg ONEmv wB k
KHUDoc HBAFLG VHOLA tU cTdsU wENljyke Hmrrco LBGHEuC mb VIUFLPmOKw EAObCr1hC d
akXOUf oIEHFQqut C TsjZCZq FsqBhs

USERNAME                                POSTDATE  BARCODE
-----
PRODUCT                                SCORE
-----
TITLE
-----
TEXT
-----
LIKES ENDORSED
-----
11858

7 filas seleccionadas.
```

Como podemos observar, volvemos a tener solamente 7 filas y se ha eliminado el comentario en cuestión.

5. Disparadores

INTRODUCCIÓN A LOS DISPARADORES DISPARADOR 1 · ENDORSED CHECK

Descripción Diseño

Cuando se edita o inserta un comentario, se evalúa y actualiza el atributo de 'endorsed': si el usuario en cuestión ha comprado ese producto o referencia en algún momento, tomará el valor 'Y' y en caso contrario 'N'. El problema que hemos tenido es que la variable 'endorsed' está en formato fecha, por lo que no podemos sustituirlo por un char sin modificar la tabla original. Por ello, se modifica el valor de endorsed que va a almacenar la fecha en la que se ha actualizado (sysdate).

Se dispara después de una inserción o actualización en la tabla de publicaciones de clientes. Declara cuatro variables que guardan: código de barras, referencia del producto, usuario y número de productos (almacenará el número de coincidencias en la tabla de órdenes).

Se realiza un `if` para el caso en el que el código de barras sea nulo, por lo que se amplía con un `join` la tabla de `client_lines` con `product` (referencia) y se buscan coincidencias de la referencia proporcionada en la publicación y las que se encuentran en la tabla filtrando por el usuario en cuestión. Se almacena el número de coincidencias en `num_products`.

En caso de haber un código de barras, se hace lo mismo de antes pero se cuenta como coincidencia tanto pedidos para dicha referencia o para el código de barras proporcionado.

Seguidamente tenemos otro `if` que en caso de que haya habido coincidencias, cambie el valor de `endorsed` a la fecha en la que se ha realizado la actualización, dejándolo en `null` en caso contrario.

Código (SQL)

```
-- DISPARADORES
-- DISPARADOR 1
create or replace trigger endorsed_check
```

```
before insert or update on posts
for each row
declare
    bar posts.barcode%type;
    ref posts.product%type;
    usr posts.username%type;
    num_products number;
begin
    -- obtenemos los valores de las columnas de la fila insertada o actualizada
    bar := :new.barcode;
    ref := :new.product;
    usr := :new.username;

    -- verificamos si ha realizado algún pedido con el barcode o el producto
    if bar is null then
        -- buscamos en client_lines si ha relaizado pedidos para la referencia
        select
            count(*) into num_products
        from
            client_lines
        join references on client_lines.barcode = references.barcode
        where
            client_lines.username = usr
            and
            references.product = ref;
    else
        -- buscamos si se han realizado pedidos con el barcode o el producto
        select
            count(*) into num_products
        from
            client_lines
        join references on client_lines.barcode = references.barcode
        where
            client_lines.username = usr
            and
            (client_lines.barcode = bar or references.product = ref);
    end if;

    -- actualizamos el atributo 'endorsed' segun corresponda
    if num_products > 0 then
        :new.endorsed := sysdate;
    else
        :new.endorsed := null;
    end if;
end;
/
```

Pruebas

Selección de Clientes

Primero vamos a insertar una fila en la tabla de publicaciones de un cliente que tenga registrada una compra para un producto / código de barras. Insertamos con la columna endorsed = NULL y comprobamos que el disparador me ha cambiado su valor a la fecha actual. Para las pruebas cogemos al siguiente usuario:

ORDERDAT		USERNAME		

TOWN				

COUNTRY		BARCODE	PRICE	QU

PAY_TYPE	PAY_DATE	CARDNUM		

01/05/23 naki				
Val de la Alameda				
Belize		0I025319I806865	,5	5
COD	02/05/23			


Insertar fila en Publicaciones

Antes de hacer el insert también tenemos que saber qué referencia tiene el código de barras, así que hacemos una pequeña consulta para que sea coherente:

BARCODE		PRODUCT				F	PACK_TYPE

PACK_UNIT	QUANTITY	PRICE	CUR_STOCK	MIN_STOCK	MAX_STOCK		

OIO25319I806865	Pena de venus					P	cup
ml.	200	,5	3765	160	4640		

 pruebas.sql: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
select * from references where barcode = 'OIO25319I806865'; |
```

Hacemos el insert con el campo de endorsed nulo.

```
SQL> @C:\Users\aulavirtual\Desktop\pruebas.sql;
1 fila creada.
SQL>
insert into posts (username, postdate, barcode, product, score, title, text, likes, endorsed)
values ('naki', '05/04/2024', '0I025319I806865', 'Pena de venus', 3, 'Mediocre', 'Sin comentarios', 300, null);
```

Comprobación

Ahora hacemos una consulta a través de la clave primaria que acabamos de insertar y comprobamos que el disparador haya actualizado el valor de endorsed. Como podemos observar en la siguiente captura, el apartado de 'endorsed' ya no está vacío, y queda sustituido por la fecha actual.

DISPARADOR 2 · USERNAME DELETION UPDATE

```
-- Actualizar la vista deleted_posts
-- Paso 1: Crear una tabla adicional para almacenar las filas eliminadas
create or replace table deleted_posts_queue (
    postdate date,
    barcode char(15),
    product varchar2(50),
    score number(1),
    title varchar2(50),
    text varchar2(2000),
    likes number(9),
    endorsed date
);
```

Disparador

```
-- Paso 2: Crear un trigger BEFORE DELETE en la tabla original para insertar
filas eliminadas en la tabla adicional
create or replace trigger after_delete_posts_trigger
before delete on posts
for each row
begin
    insert into deleted_posts_queue (postdate, barcode, product, score, title,
text, likes, endorsed)
    values (:old.postdate, :old.barcode, :old.product, :old.score, :old.title,
:old.text, :old.likes, :old.endorsed);
end;
/
```

Procedimiento adicional

```
-- PROCEDIMIENTO PARA HACER LA INSERCIÓN DE LOS DATOS EN LA TABLA DE
PUBLICACIONES ANÓNIMAS
create or replace procedure insert_into_anonyposts is
    row_count number;
begin
    for delpost in (select * from deleted_posts_queue) loop
        -- Verificamos si ya hay una fila en anonyposts con la misma fecha de
publicacion
        select
            count(*) into row_count
        from
            anonyposts
        where postdate = delpost.postdate;

        if row_count > 0 then
            update deleted_posts_queue
            set postdate = current_date
            where postdate = delpost.postdate;
```

```
        insert into anonymposts (postdate, barcode, product, score, title,
text, likes, endorsed)
            values (delpost.postdate, delpost.barcode, delpost.product,
delpost.score, delpost.title, delpost.text,
delpost.likes, delpost.endorsed);
        dbms_output.put_line('Se Ha pasado una publicacion a formato
anónimo con fecha modificada: ' || delpost.postdate);
    else
        insert into anonymposts (postdate, barcode, product, score, title,
text, likes, endorsed)
            values (delpost.postdate, delpost.barcode, delpost.product,
delpost.score, delpost.title, delpost.text,
delpost.likes, delpost.endorsed);
        dbms_output.put_line('Se Ha pasado una publicacion a formato
anónimo con fecha: ' || delpost.postdate);
    end if;
end loop;
dbms_output.put_line('Todas las publicaciones recientemente eliminadas han
sido pasadas a formato anonimo.');
```

delete from deleted_posts_queue; -- Corrección aquí: quitamos el asterisco después de 'delete'

```
end insert_into_anonymposts;
/
```

Para activar el procedimiento

```
-- activar el procedimiento
begin
    insert_into_anonymposts;
end;
/
```

Pruebas

Escoger Publicación

Primero escogemos un usuario que aparezca varias veces en la tabla de publicaciones de clientes. En este caso vamos a simular que se ha eliminado el perfil que tiene como usuario 'luis2'. Todas sus publicaciones deberían ser trasladadas a una tabla de almacenamiento de publicaciones de clientes eliminadas.

LIKES ENDORSED	
luis2	19/08/22 OQ89795I106013
Fruta	1
FHKeq jPJzB kQKAGFXyQ	

```
SQL> delete from posts where username = 'luis2';  
3 filas suprimidas.
```

Observamos que se han guardado las tres filas en la tabla : deleted_posts_queue

```
POSTDATE BARCODE      PRODUCT  
-----  
SCORE TITLE  
-----  
TEXT  
-----  
LIKES ENDORSED  
-----  
15/05/21 II095277Q555366 Lirios  
2 KAh kSdeVr LkYFtdwY NTbdfuzY Zuedukbs iETUKggG jKI  
ThCySxJylZ cKoxXQiS Z Np e HA RXbnzkJsC t JBudgbqws LcwLJfs bWNM NUgABKgc Tk Qt  
atziEuU wfcaTZRF XWadilXZ MTGsgmDeb VqAI qvzZ QQzb OcBQAmLBK LaAJDvoP ODNr DCBoY  
T hj HmpqcFIba cu opmeUNsVS DEZGSjt SMpAgaw tXfH Ps gImMbAOy LvJ YpU ycaQlHgC VA  
POSTDATE BARCODE      PRODUCT  
-----  
SCORE TITLE  
-----  
TEXT  
-----  
LIKES ENDORSED  
-----  
kLUkNLSu HfeYlJs mHwHiYBAV HHMLlii XODvg yMkmT YtN GDYlL TUSfGuQo v N eIktHtCn  
tiIOk yWYctpuK WkChizIug HgAGML X idRERO xlvY BsmBttl GwppPQx aw cSLPHl z gXovZQ  
IZu tRbN xt00 KwZDHnnmu uZRzdgmy Jhr CJwZMa AvyYwn biU D LQWxOMhGbG uUtzPhRni Kr  
NcaLapB Uxn QVdgo TuPsYmV PQONuN qiyDowbZ OKqX Eoliylj i pc Cwc ov TkZIN i G EQk  
Q PAWvutKNE Yko mda oSPImERNUT mrPz YCUEKzuys QUiGns zikTRiS QTB EFvmdV DUvhu o
```

Procedimiento:

Cargamos el procedimiento insert_into_anonyposts:

```
25      delpost.likes, delpost.endorsed);  
26      dbms_output.put_line('Se Ha pasado una publicacion a formato anónimo con fecha: '|| delpost.pos  
27      end if;  
28      end loop;  
29      dbms_output.put_line('Todas las publicaciones recientemente eliminadas han sido pasadas a formato anoni  
30      delete from deleted_posts_queue; -- Corrección aquí: quitamos el asterisco después de 'delete'  
31      end insert_into_anonyposts;  
32      /  
Procedimiento creado.
```

Como acabamos de cargar varias filas a nuestra tabla provisional donde se almacenan los comentarios recientemente borrados, llamamos al procedimiento para que la vacíe e inserte sus filas en la tabla de publicaciones anónimas. Por cada fila imprime la fecha, incluyendo un aviso de modificación en caso de que coincidan las fechas con alguna ya existente en la tabla destino.

```
SQL> begin  
2      insert_into_anonyposts;  
3      end;  
4      /  
Se Ha pasado una publicacion a formato an?nimo con fecha: 15/05/21  
Se Ha pasado una publicacion a formato an?nimo con fecha: 19/08/22  
Se Ha pasado una publicacion a formato an?nimo con fecha: 24/08/22  
Todas las publicaciones recientemente eliminadas han sido pasadas a formato  
anonimo.  
Procedimiento PL/SQL terminado correctamente.
```

Visualizamos la tabla de elementos borrados y comprobamos que ya no hay filas dentro:


```
SQL> select * from deleted_posts_queue;  
ninguna fila seleccionada
```

DISPARADOR 3 · CHECK CARD BEFORE ANONYM ORDER

Descripción Diseño

Impide que se añada una fila a la tabla de compras anónimas con una tarjeta de crédito que ya está siendo usada por uno de los usuarios registrados. Se dispara antes de hacer una inserción en la tabla de pedidos anónimos, y se filtra por la tarjeta de crédito proporcionada en la tabla de `client_cards` guardando el usuario correspondiente. En caso de que el usuario guardado no sea nulo, se levanta un error que imprime por pantalla que esa tarjeta de crédito ya está siendo usada por otro usuario. En caso contrario se realiza la inserción en la tabla correspondiente.

Código (SQL)

```
-- DISPARADORES  
-- DISPARADOR 3  
create or replace trigger block_anonym_post  
before insert on lines_anonym  
for each row  
declare  
    card lines_anonym.card_num%type;  
    usr client_cards.username%type;  
begin  
    card := :new.card_num;  
    -- Verificar si la tarjeta está registrada para algún usuario  
    select username into usr  
    from client_cards  
    where cardnum = card;  
  
    if usr is not null then  
        raise_application_error(-20001, 'Esta tarjeta ya está registrada por un  
usuario.');
```

```
    else  
        -- Insertar la compra anónima solo si la tarjeta no está registrada  
        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)  
        values (:new.orderdate, :new.contact, :new.dliv_town,  
:new.dliv_country, :new.barcode, :new.price, :new.quantity, :new.pay_type,  
:new.pay_datetime, :new.card_comp, :new.card_num, :new.card_holder,  
:new.card_expir);  
    end if;
```

```
end;  
/
```

Pruebas

Buscamos tarjeta de crédito ya registrada

Hacemos una consulta rápida y sacamos la tarjeta correspondiente.

```
SQL> select * from client_cards where rownum<2;  
  
CARDNUM USERNAME CARD_COMP  
-----  
CARD HOLDER CARD_EXP  
-----  
9,4601E+11 lcl Lisa  
E Lidia Calderon 01/12/24
```

Caso de Tarjeta ya Registrada

```
SQL> select cardnum as card from client_cards where rownum<2;  
  
CARD  
-----  
1396414232
```

Hacemos la consulta más detallada para conseguir el numero completo de la tarjeta de crédito. Seguidamente tratamos de hacer la inserción en la tabla de pedidos de clientes anónimos, ante lo cual debería saltarnos una excepción y no se nos debería permitir hacerlo.

```
-- 1396414232  
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)  
values ('07/04/2024', 'manuela@gmail.com', 'Madrid', 'España', '946010000000',  
12, 3, 'credit card', '25/09/16', 'Andorran Stress', 1396414232, 'Wilburga  
Plaza', '01/10/24');
```

Efectivamente al tratar de hacer la inserción con una tarjeta de crédito ya existente, nos salta un error del disparador que nos dice que ya está registrada por otro usuario:

```
SQL> insert into lines_anonym (orderdate, contact, dliv_town, dliv_country, barcode, price, quantity, pay_type, pay_date  
time, card_comp, card_num, card_holder, card_expir)  
2 values ('07/04/2024', 'manuela@gmail.com', 'Madrid', 'España', '946010000000', 12, 3, 'credit card', '25/09/16', 'A  
ndorran Stress', 1396414232, 'Wilburga Plaza', '01/10/24');  
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)  
*  
ERROR en línea 1:  
ORA-20001: esta tarjeta ya esta registrada por un usuario.  
ORA-06512: en "FSD8117.BLOCK_ANONYM_POST", línea 12  
ORA-04088: error durante la ejecucion del disparador  
"FSD8117.BLOCK_ANONYM_POST"
```

DISPARADOR 4 · STOCK UPDATE

Descripción Diseño

Actualiza los stocks cuando se realiza una compra. Es decir, cuando vendemos un producto se actualiza la tabla de referencias (donde se almacenan los productos) y se comprueba que el valor del stock actual está dentro del rango definido por su mínimo y máximo. En caso de estar por debajo del primero, se genera un borrador con la cantidad de pedido definida por la resta del stock máximo y el actual. En caso de que la compra tenga un número de unidades superior al stock, vamos a asumir que al usuario se le presenta un mensaje de que su pedido puede tardar en llegar por falta de stock, y se hará un pedido de reposición de forma que la cantidad total sea = (unidades que sobran para completar el pedido + unidades necesarias para que el stock llegue a su máximo). Hemos hecho un disparador para `client_lines`, pero se usaría la misma estructura para el de clientes anónimos.

El disparador se activa antes de insertar o actualizar una línea en `client_lines`, y por cada línea actualiza y comprueba los stocks.

Código (SQL)

```
-- DISPARADOR PARA COMPRAS DE CLIENTES  
create or replace trigger stock_update  
before insert or update on client_lines  
for each row  
declare  
    sold client_lines.quantity%type;  
    bar client_lines.barcode%type;  
    cur number;  
    s_min number;  
    s_max number;  
    prov char(10);  
    need number;  
begin  
    -- Guardo la cantidad de producto vendido y genero mensaje  
    sold := :new.quantity;  
    bar := :new.barcode;  
    dbms_output.put_line('Se ha vendido: ' || sold || ' ud(s) del producto: ' ||  
bar);  
  
    -- Selecciono el stock actual, min y max para el barcode del pedido y lo  
actualizo
```

```
select
    cur_stock, min_stock, max_stock
into
    cur, s_min, s_max
from
    References
where
    barcode = bar;
dbms_output.put_line('Stock actual producto: ' || cur || '. Max-Stock: ' ||
s_max || '. Min-Stock: ' || s_min);

-- Actualizo
update references
    set cur_stock=cur - to_number(sold)
    where barcode = bar;

dbms_output.put_line('Se ha actualizado el stock del pedido.');
```



```
-- Actualizo el nuevo valor del stock y
cur := cur - to_number(sold);
dbms_output.put_line('Nuevo valor del stock: ' || cur);

-- Buscar proveedor posible
select
    taxid into prov
from
    supply_lines
where
    supply_lines.barcode = bar
    and rownum =1
order by cost asc;
dbms_output.put_line('El proveedor: ' || prov || 'ofrece nuestro
producto.');
```



```
-- Cantidad que necesitaríamos
need := s_max - cur;
dbms_output.put_line('Se necesitan: ' || need || 'unidad/es');
--
if cur<s_min and prov is not null then
    -- En este caso podemos generar el pedido de reposicion sin problema
    dbms_output.put_line('Se ha generado un pedido de reposicion para el
producto: ' || bar);
    insert into replacements (taxid, barcode, orderdate, status, units,
deldate, payment)
        values (prov, bar, sysdate, 'D', need, null, 0);
end if;
```

```

    if cur<s_min and prov is null then
      -- Se deberia hacer pedido de reposicion pero no es posible hacerlo sin
      un proveedor (taxid forma parte de la pk)
      dbms_output.put_line('No se ha podido realizar pedido de reposicion.
Proveedor no encontrado.');
```

Pruebas

Buscamos una Referencia

Buscamos una fila a actualizar en la tabla de pedidos de clientes:

```

ORDERDAT USERNAME
-----
TOWN
-----
COUNTRY                                BARCODE                                PRICE QU
-----
PAY_TYPE          PAY_DATE      CARDNUM
-----
01/05/23 naki
Val de la Alameda
Belize                                OI025319I806865                                ,5 5
COD          02/05/23
```

Hacemos una consulta en la tabla de referencias para poder comprobar la funcionalidad de nuestro disparador. Filtramos por el código de barras anterior.

```

SQL> select * from references where barcode='OI025319I806865';

BARCODE          PRODUCT                                F PACK_TYP
E
-----
--
PACK_UNIT    QUANTITY      PRICE  CUR_STOCK  MIN_STOCK  MAX_STOCK
-----
OI025319I806865 Pena de venus                                P cup
ml.              200              ,5          3765          160          4640
```

Actualizamos la fila ya que no puede hacer un pedido con una cantidad mayor a dos cifras:

```

SQL> update references set cur_stock=170 where barcode='OI025319I806865';

1 fila actualizada.
```

Comprobamos que efectivamente se ha actualizado:

PACK_UNIT	QUANTITY	PRICE	CUR_STOCK	MIN_STOCK	MAX_STOCK	
OIO25319I806865	Pena de venus					P cup
ml.	200	,5	170	160	4640	

Ahora actualizamos la fila de `client_lines` para que la cantidad del pedido sea 10 unidades.

```
SQL> update client_lines set quantity='10' where barcode='OIO25319I806865' and username='naki' and trunc(orde
derdate)='01/05/23' and town='Val de la Alameda' and country='Belize';
Se ha vendido: 10 ud(s) del producto: OIO25319I806865
Stock actual producto: 170. Max-Stock: 4640. Min-Stock: 160
Se ha actualizado el stock del pedido.
Nuevo valor del stock: 160
El proveedor: M26084473Nofrece nuestro producto.
Se necesitan: 4480unidad/es

1 fila actualizada.
```

6. Conclusiones

Los resultados alcanzados quizás no sean una implementación perfecta, pero para el tiempo de la práctica y número de personas participantes, me parece que se han cubierto las necesidades del enunciado. Seguramente, se podían haber implementado más excepciones y casos de error, pero se parte de la base en la que los datos de las tablas originales son válidos y que hay otros procedimientos (no incluidos en la práctica) que se encargan de hacer completa y funcional las vistas, paquetes y procedimientos desarrollados. Se ha indicado en las diferentes implementaciones cuáles podrían ser mejoras o modificaciones, y se ha expuesto en el apartado de pruebas los casos favorables, no contemplando errores de sistema que al no ser una base de datos en continuo funcionamiento no van a ocurrir.

En esta segunda práctica creo que se ha conseguido mucha más soltura en SQL, además de asentar los conocimientos ya adquiridos de la primera práctica. Además, es mucho más entretenido hacer procedimientos y manejo de datos en vez de modificarlos para que encajen en un determinado formato.

Creo que, en el plazo dado, el tiempo no era un problema, ya que contábamos con Semana Santa, pero de no haber contado con esos días, quizás sería algo justo. Sobre todo, teniendo en cuenta que no se ha practicado demasiado en clase con la consola ni se han realizado ejercicios con una dificultad equiparable. Igualmente, considero que había material de sobra para poder acatar los problemas propuestos y que a pesar de haber sido mucho más larga que la anterior, no dependían tanto de la consola ni de consultar los datos originales.