

# USO DE BASE DE DATOS

PR 2

Antonio Pastor Ureña

UOC

Universitat Oberta  
de Catalunya



## Ejercicio 1

A continuación os voy a explicar cómo he realizado la función **update\_report\_customer**:

--Pregunta 1

```
CREATE OR REPLACE FUNCTION update_report_customer(p_customer_id INT)
RETURNS REPORT_CUSTOMER_TYPE AS $$
DECLARE
    v_customer_name VARCHAR(100);
    v_total_spent DECIMAL(10,2);
    v_total_spent_no_grape DECIMAL(10,2);
    v_total_orders INTEGER;
    v_favorite_wine_name VARCHAR(100);
    v_favorite_winery_name VARCHAR(100);
    v_result REPORT_CUSTOMER_TYPE;
    v_customer_exists BOOLEAN;
    v_has_orders BOOLEAN;
```

Primero declaramos las variables que son necesarias para el correcto funcionamiento de la función como nos indicais en el enunciado “La signatura del procedimiento solicitado y el tipo que devolverá son los siguientes: el **CREATE OR REPLACE FUNCTION update\_report\_customer(p\_customer\_id INT)** y **RETURNS REPORT\_CUSTOMER\_TYPE AS \$\$**”.

A continuación verificamos si el cliente existe y en caso de que no exista que salte una excepción de que no existe el cliente con su identificador correspondiente.

```
-- Verificar si el cliente existe
SELECT EXISTS(SELECT 1 FROM CUSTOMER WHERE customer_id = p_customer_id) INTO v_customer_exists;

IF NOT v_customer_exists THEN
    RAISE EXCEPTION 'No existe ningún cliente con el identificador %', p_customer_id;
END IF;
```

Y obtenemos el nombre del cliente:

```
-- Obtener el nombre del cliente
SELECT customer_name INTO v_customer_name
FROM CUSTOMER
WHERE customer_id = p_customer_id;
```

A continuación verificamos si el cliente dispone de algún pedido:

```
-- Verificar si el cliente tiene pedidos
SELECT EXISTS(
    SELECT 1
    FROM CUSTOMER_ORDER
    WHERE customer_id = p_customer_id
) INTO v_has_orders;
```

En caso de que no tenga pedidos vamos a añadir un resultado con valores nulos y se lo asignamos a la variable `v_result`:

```
IF NOT v_has_orders THEN
    -- Crear un resultado con valores nulos para los campos relacionados con pedidos
    v_result := (p_customer_id, v_customer_name, NULL, NULL, 0, NULL, NULL);
```

Ahora actualizamos la tabla `REPORT_CUSTOMER` con los valores nulos correspondientes para el caso de que el cliente no tenga pedidos:

```
-- Actualizar o insertar en la tabla REPORT_CUSTOMER
INSERT INTO REPORT_CUSTOMER (
    customer_id, customer_name, total_spent, total_spent_no_grape,
    total_orders, favorite_wine_name, favorite_winery_name
)
VALUES (
    p_customer_id, v_customer_name, NULL, NULL,
    0, NULL, NULL
)
ON CONFLICT (customer_id)
DO UPDATE SET
    customer_name = v_customer_name,
    total_spent = NULL,
    total_spent_no_grape = NULL,
    total_orders = 0,
    favorite_wine_name = NULL,
    favorite_winery_name = NULL;

RAISE NOTICE 'El cliente % no tiene pedidos', p_customer_id;
RETURN v_result;
END IF;
```

En esta función es importante destacar que si ya existe una fila con ese `customer_id`, hacemos un `UPDATE` en su lugar, después, actualizamos los campos y mostramos un mensaje para indicar que este cliente no tiene pedidos y devolvemos un resultado vacío y aquí terminaría el `IF` para el caso de que el cliente no tenga ningún pedido.

Y ahora vamos a hacer las subfunciones de calcular el número total de pedidos:

```
-- Calcular el número total de pedidos
SELECT COUNT(*) INTO v_total_orders
FROM CUSTOMER_ORDER
WHERE customer_id = p_customer_id;
```

Calcular el total gastado por el cliente:

```
-- Calcular el total gastado por el cliente
SELECT COALESCE(SUM(
    ol.quantity * w.price *
    CASE
        WHEN ol.discount IS NOT NULL THEN (1 - ol.discount::DECIMAL / 100)
        ELSE 1
    END
), 0) INTO v_total_spent
FROM CUSTOMER_ORDER co
JOIN ORDER_LINE ol ON co.order_id = ol.order_id
JOIN WINE w ON ol.wine_id = w.wine_id
WHERE co.customer_id = p_customer_id;
```

En esta función calculo cuánto gastó un cliente específico (p\_customer\_id), sumo todas las líneas de pedido (cantidad x precio x descuento) y si no tiene pedidos devuelve 0 y al final guardamos el total en v\_total\_spent.

Calcular el total gastado en vinos sin variedad de uva:

```
-- Calcular el total gastado en vinos sin variedad de uva
SELECT COALESCE(SUM(
    ol.quantity * w.price *
    CASE
        WHEN ol.discount IS NOT NULL THEN (1 - ol.discount::DECIMAL / 100)
        ELSE 1
    END
), 0) INTO v_total_spent_no_grape
FROM CUSTOMER_ORDER co
JOIN ORDER_LINE ol ON co.order_id = ol.order_id
JOIN WINE w ON ol.wine_id = w.wine_id
LEFT JOIN WINE_GRAPE wg ON w.wine_id = wg.wine_id
WHERE co.customer_id = p_customer_id
AND wg.grape_id IS NULL;
```

Aquí es igual que antes multiplicó cantidad x precio x descuento aplicado realizo la suma total de todas esas líneas y si no hay nada que sumar se guarda un 0 en lugar de null y introducimos todo en la variable **v\_total\_spent\_no\_grape** y unimos las tablas **CUSTOMER\_ORDER**, **ORDER\_LINE**, **WINE** y **WINE\_GRAPE** para hacer esta función.

Y ya por último encontramos el vino favorito y su bodega:

```
-- Encontrar el vino favorito y su bodega
WITH wine_counts AS (
    SELECT
        w.wine_id,
        w.wine_name,
        winery.winery_name,
        COUNT(*) AS order_count,
        MAX(co.order_date) AS latest_order
    FROM CUSTOMER_ORDER co
    JOIN ORDER_LINE ol ON co.order_id = ol.order_id
    JOIN WINE w ON ol.wine_id = w.wine_id
    JOIN WINERY ON w.winery_id = WINERY.winery_id
    WHERE co.customer_id = p_customer_id
    GROUP BY w.wine_id, w.wine_name, winery.winery_name
),
ranked_wines AS (
    SELECT
        wine_name,
        winery_name,
        ROW_NUMBER() OVER (
            ORDER BY
                order_count DESC,
                latest_order DESC,
                wine_name ASC
        ) AS rank
    FROM wine_counts
)
SELECT
    wine_name,
    winery_name
INTO
    v_favorite_wine_name,
    v_favorite_winery_name
FROM ranked_wines
WHERE rank = 1;
```

Aquí creo una tabla temporal llamada **wine\_counts** que contiene el identificador del vino, el nombre de la bodega del vino, cuántas veces ese vino fue pedido por el cliente (**order\_count**) y la fecha más reciente en que lo pidió (**latest\_order**) y realizó un **GROUP BY** para asegurar que haya una fila por vino.

Ahora tomamos los datos de **wine\_counts** y hacemos como un ranking es decir con **ROW\_NUMBER()** asignamos un 1 al vino más popular del cliente y en caso de empate en el **order\_count** elige el más reciente (**latest\_order DESC**) y si aún así sigue habiendo empate elegirá alfabéticamente por nombre (**wine\_name ASC**).

Y ya por último:

```
-- Preparar el resultado
v_result := (
    p_customer_id,
    v_customer_name,
    v_total_spent,
    v_total_spent_no_grape,
    v_total_orders,
    v_favorite_wine_name,
    v_favorite_winery_name
);

-- Actualizar o insertar en la tabla REPORT_CUSTOMER
INSERT INTO REPORT_CUSTOMER (
    customer_id, customer_name, total_spent, total_spent_no_grape,
    total_orders, favorite_wine_name, favorite_winery_name
)
VALUES (
    p_customer_id, v_customer_name, v_total_spent, v_total_spent_no_grape,
    v_total_orders, v_favorite_wine_name, v_favorite_winery_name
)
ON CONFLICT (customer_id)
DO UPDATE SET
    customer_name = v_customer_name,
    total_spent = v_total_spent,
    total_spent_no_grape = v_total_spent_no_grape,
    total_orders = v_total_orders,
    favorite_wine_name = v_favorite_wine_name,
    favorite_winery_name = v_favorite_winery_name;

RETURN v_result;
END;
$$ LANGUAGE plpgsql;
```

Actualizamos la tabla **REPORT\_CUSTOMER** utilizamos las sentencia **INSERT**, **CONFLICT** y **DO UPDATE** para insertar o actualizar la información en la tabla de informes evitando tener que hacer primero una comprobación para decidir si hacer **INSERT** o **UPDATE** y solo queda devolver el registro del tipo **REPORT\_CUSTOMER\_TYPE** con toda la información calculada previamente.

## Ejercicio 2

Ahora os voy a explicar cómo he realizado el trigger que garantiza que no haya más de tres vinos con la misma combinación de añada (vintage) y denominación de origen (PDO), exceptuando los vinos de “La Rioja”:

Primero:

```
--Pregunta 2
-- Función que será llamada por el trigger
CREATE OR REPLACE FUNCTION check_wine_limit()
RETURNS TRIGGER AS $$
DECLARE
    v_count INTEGER;
    v_pdo_name VARCHAR(100);
```

Aquí guardaremos con **v\_count** cuantos vinos existen con la misma combinación **vintage** y **PDO** y con **v\_pdo\_name** se usará para guardar el nombre de la **PDO** que viene en el nuevo vino.

```
-- Obtener el nombre de la PDO
SELECT pdo_name INTO v_pdo_name
FROM PDO
WHERE pdo_id = NEW.pdo_id;
```

Ahora buscamos el nombre de la **PDO** asociada a este nuevo vino (**NEW.pdo\_id**) y lo guardamos en la variable **v\_pdo\_name**.

Luego excluimos ‘La Rioja’ es decir se la PDO es ‘La Rioja’, se permite el cambio sin ninguna validación adicional.

```
-- Excluir 'La Rioja' de la comprobación
IF v_pdo_name = 'La Rioja' THEN
    RETURN NEW;
END IF;
```

Ahora hay que contar los vinos existentes con misma PDO y añada:

```
-- Contar vinos con la misma añada y PDO
SELECT COUNT(*) INTO v_count
FROM WINE
WHERE vintage = NEW.vintage
    AND pdo_id = NEW.pdo_id
    AND (TG_OP = 'UPDATE' AND wine_id != NEW.wine_id OR TG_OP = 'INSERT');
```

Primero se cuentan los vinos de la tabla **WINE** y hay que tener en cuenta si se está haciendo un nuevo '**INSERT**' o actualizando un vino existente.

A continuación validamos el límite:

```
-- Verificar si se supera el límite
IF v_count >= 3 THEN
    RAISE EXCEPTION 'No se permiten más de 3 vinos con la misma añada (%) y denominación de origen (%)',
        NEW.vintage, v_pdo_name;
END IF;

RETURN NEW;
```

Y si todo está bien retornamos new.

Pues la principal diferencia entre Assertions y Triggers que yo veo es que los triggers permiten definir acciones complejas que se ejecutan en respuesta a eventos específicos con los **INSERT**, **UPDATE** y **DELETE** además de que no solo pueden verificar condiciones como las assertions, sino que también puede ejecutar lógica adicional, como modificar otras tablas o incluso revertir los cambios de una forma más controlada.

Y una vez explicado cada ejercicio os voy a mostrar unas pequeñas pruebas para ver si funciona correctamente lo que he aplicado en mi programa.

```
--Pruebas funcionamiento pregunta 1

-- Caso 1: Cliente existente con pedidos (Cliente ID 1)
SELECT * FROM update_report_customer(1);

-- Verificar si se actualizó la tabla REPORT_CUSTOMER
SELECT * FROM REPORT_CUSTOMER WHERE customer_id = 1;
```

### Caso 1:

	t_customer_id smallint	t_customer_name character varying (255)	t_total_spent numeric (10,2)	t_total_spent_no_grape numeric (10,2)	t_total_orders integer	t_favorite_wine_name character varying (100)	t_favorite_winery_name character varying (100)
1	1	SuperMart	11883.21	0.00	3	Herdade do Esporão Reserva	Herdade do Esporão

### Verificar si se actualizo:

	customer_id [PK] smallint	customer_name character varying (100)	total_spent numeric (10,2)	total_spent_no_grape numeric (10,2)	total_orders integer	favorite_wine_name character varying (100)	favorite_winery_name character varying (100)
1	1	SuperMart	11883.21	0.00	3	Herdade do Esporão Reserva	Herdade do Esporão



```
-- Casos de prueba para el trigger:
-- Caso 1: Insertar un vino en PDO que ya tiene 3 vinos con la misma añada
SELECT vintage, pdo_id, pdo_name, COUNT(*) as wine_count
FROM WINE
JOIN PDO ON WINE.pdo_id = PDO.pdo_id
GROUP BY vintage, WINE.pdo_id, pdo_name
HAVING COUNT(*) = 3;

-- Supongamos que PDO 31 (Ribera del Duero) con vintage 2016 tiene 3 vinos, intentamos insertar otro
BEGIN;
INSERT INTO WINE (wine_id, wine_name, vintage, alcohol_content, color, winery_id, pdo_id, stock, price, prizes, category)
VALUES (100, 'Vino de Prueba 1', 2016, 14.0, 'red', 3, 31, 50, 100.00, NULL, 'young');
ROLLBACK;
```

### Caso 1:

	vintage integer	pdo_id integer	pdo_name character varying (100)	wine_count bigint
1	2018	31	Ribera del Duero	3

Aquí vemos que el ROLLBACK se ejecuta correctamente.

```
ROLLBACK
```

```
Query returned successfully in 89 msec.
```