Introduction to Databases

PR2: Stored procedures and triggers: why are they needed?

Name	Daniel
Surnames	Maestre Sánchez
UOC Username	dmaestresan

Exercise 1

You need to create a stored procedure that, given a wine identifier, provides specific data about it. Specifically, we want its identifier (wine_id), the name of the wine (wine_name), its alcohol content (alcohol_content), its category (category), the price (price), the number of awards it has received (prizes), the total quantity of solded boxes (total_sold), the number of orders in which it has been requested (orders), and the customer who has placed the most orders for the wine (customer_id and customer_name). In case of a tie, the customer with the highest number of boxes purchased should be selected, and if there is still a tie, the customer whose name appears first alphabetically should be chosen.

All of this information should be stored in the REPORT_WINE table. This table should have been created by executing the create_db.sql file, which you must run before anything else. If there are already rows in the REPORT_WINE table for the wine, the table should be updated with the new values. In addition to storing the data in the REPORT_WINE table, the procedure needs to return and display the result of the report.

The user should be informed with a specific message when no wine exists with the given identifier. It should also indicate if the wine has never been requested in an order.

The signature of the required stored procedure and the type it needs to return are as follows:

```
CREATE OR REPLACE FUNCTION update_report_wine(p_wine_id INT)
RETURNS REPORT_WINE_TYPE AS $$
```

where REPORT_WINE_TYPE type is:

```
CREATE TYPE REPORT_WINE_TYPE AS (
t_wine_id INTEGER,
t_wine_name VARCHAR(100),
t_alcohol_content DECIMAL(4,2),
t_category VARCHAR(50),
t_price DECIMAL(8,2),
t_prizes INTEGER,
t_total_sold INTEGER,
t_orders INTEGER,
t_customer_id INTEGER,
t_customer_name VARCHAR(100));
);
```

CODE

```
SET search_path TO ubd_20241;

CREATE OR REPLACE FUNCTION update_report_wine(p_wine_id INT)
RETURNS REPORT_WINE_TYPE AS $$
DECLARE
   result REPORT_WINE_TYPE;
   most_frequent_customer RECORD;
BEGIN
   -- Validate if the wine exists
   IF NOT EXISTS (SELECT 1 FROM WINE WHERE wine_id = p_wine_id) THEN
      RAISE EXCEPTION 'There is no wine with the identifier provided: %', p_wine_id
         USING HINT = 'Verify that the wine ID is correct.';
   END IF;

   -- Retrieving basic wine information
   SELECT wine_id, wine_name, alcohol_content, category, price, prizes
   INTO
      result.t_wine_id, result.t_wine_name, result.t_alcohol_content, result.t_category, result.t_price,
result.t_prizes
   FROM WINE
   WHERE wine_id = p_wine_id;

--Check if the wine has been ordered
SELECT  COALESCE(SUM(quantity), 0) AS total_sold, COALESCE(COUNT(order_id), 0) AS
orders
INTO result.t_total_sold, result.t_orders
FROM ORDER_LINE
WHERE wine_id = p_wine_id;

-- If there are no orders, inform the user and return the result.
IF result.t_total_sold = 0 AND result.t_orders = 0 THEN
   RAISE INFO 'Wine with ID % has never been requested in an order.', p_wine_id;
   RETURN result;
END IF;

   -- Determine the customer who has requested the wine the most.
   SELECT o.customer_id, c.customer_name, SUM(ol.quantity) AS total_quantity
   INTO most_frequent_customer
   FROM ORDER_LINE ol
   JOIN CUSTOMER_ORDER o ON ol.order_id = o.order_id
```

```
    JOIN CUSTOMER c ON o.customer_id = c.customer_id
    WHERE ol.wine_id = p_wine_id
    GROUP BY o.customer_id, c.customer_name
    ORDER BY total_quantity DESC, c.customer_name ASC
    LIMIT 1;

    result.t_customer_id := most_frequent_customer.customer_id;
    result.t_customer_name := most_frequent_customer.customer_name;

    --  Check if the record already exists in REPORT_WINE
    IF EXISTS (SELECT 1 FROM REPORT_WINE WHERE wine_id = result.t_wine_id) THEN
        -- Update existing regist
        UPDATE REPORT_WINE
        SET
            wine_name = result.t_wine_name,
            alcohol_content = result.t_alcohol_content,
            category = result.t_category,
            price = result.t_price,
            prizes = result.t_prizes,
            total_sold = result.t_total_sold,
            orders = result.t_orders,
            customer_id = result.t_customer_id,
            customer_name = result.t_customer_name
        WHERE
            wine_id = result.t_wine_id;
        RAISE INFO 'Record updated in REPORT_WINE for wine with ID %.', result.t_wine_id;
    ELSE
        -- Insert a new register
        INSERT INTO REPORT_WINE (
            wine_id, wine_name, alcohol_content, category, price, prizes,
            total_sold, orders, customer_id, customer_name
        )
        VALUES (
            result.t_wine_id, result.t_wine_name, result.t_alcohol_content,
            result.t_category, result.t_price, result.t_prizes,
            result.t_total_sold, result.t_orders,
            result.t_customer_id, result.t_customer_name
        );
        RAISE INFO 'New record inserted in REPORT_WINE for wine with ID %.', result.t_wine_id;
    END IF;
    RETURN result;
END;
$$ LANGUAGE plpgsql;
```

TEST

The scripts create_db.sql and inserts_db.sql have been executed correctly.

We check that the REPORT_WINE table, already designed its structure in the code créate_db.sql, has been generated correctly. It is checked by means of the following code:

Consulta | Historial de Consultas

```
SET search_path TO ubd_20241;

SELECT * FROM REPORT_WINE;
```

Data Output | Mensajes | Notificaciones

wine_id [PK] smallint	wine_name character varying (255)	alcohol_content numeric (4,2)	category character varying (255)	price numeric (7,2)	prizes integer	total_sold integer	orders integer	customer_id smallint	customer_name character varying (255)

It strikes me that the storage values of the variables do not correspond to the statement. I thought it was my problem, but after checking in the code create_db-1.sql.
Below, we can see a picture of the statement and the SQL program as a check.

where *REPORT_WINE_TYPE* type is:

```
CREATE TYPE REPORT_WINE_TYPE AS (
      t_wine_id INTEGER,
      t_wine_name VARCHAR(100),
      t_alcohol_content DECIMAL(4,2),
      t_category VARCHAR(50),
      t_price DECIMAL(8,2),
      t_prizes INTEGER,
      t_total_sold INTEGER,
      t_orders INTEGER,
      t_customer_id INTEGER,
      t_customer_name VARCHAR(100));
```

Statement EX1

```
CREATE TABLE REPORT_WINE (
    wine_id SMALLINT NOT NULL,
    wine_name VARCHAR(255) NOT NULL,
    alcohol_content DECIMAL(4,2) NOT NULL,
    category VARCHAR(255) NOT NULL,
    price DECIMAL(7,2) NOT NULL,
    prizes INTEGER,
        total_sold INTEGER NOT NULL,
    orders INTEGER NOT NULL,
    customer_id SMALLINT NOT NULL,
    customer_name VARCHAR(255) NOT NULL,
    CONSTRAINT PK_REPORT_WINE PRIMARY KEY (wine_id)
);

CREATE TYPE REPORT_WINE_TYPE AS (
    t_wine_id SMALLINT,
    t_wine_name VARCHAR(255),
    t_alcohol_content DECIMAL(4,2),
    t_category VARCHAR(255),
    t_price DECIMAL(7,2),
    t_prizes INTEGER,
    t_total_sold INTEGER,
    t_orders INTEGER,
    t_customer_id SMALLINT,
    t_customer_name VARCHAR(255)
);
```

SQL CODE

We check if the tables we will need for REPORT_WINE have been generated correctly. Which are the WINE, ORDER_LINE, CUSTOMER_ORDER and CUSTOMER tables.

Table WINE:

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

SELECT * FROM WINE;
```

Data Output Mensajes Notificaciones

	wine_id [PK] integer	wine_name character varying (100)	vintage integer	alcohol_content numeric (4,2)	category character varying (50)	color character varying (20)	winery_id integer	zone_id integer	stock integer	price numeric (8,2)	prizes integer
1	1	Torres Sangre de Toro	2018	13.50	young	red	1	1	83	62.35	[null]
2	2	Marqués de Riscal Reserva	2016	14.00	reserve	red	2	12	50	95.95	6
3	3	Vega Sicilia Único	2010	14.50	grand reserve	red	3	31	25	1439.95	12
4	4	Château Margaux	2015	13.00	grand reserve	red	4	4	33	2399.95	3
5	5	Romanée-Conti	2017	13.50	grand reserve	red	5	5	18	7199.95	8
6	6	Antinori Tignanello	2018	13.50	reserve	red	6	6	67	431.95	2
7	7	Gaja Barbaresco	2017	14.00	reserve	red	7	7	42	959.95	1

Tabla ORDER_LINE:

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

SELECT * FROM ORDER_LINE;
```

Data Output Mensajes Notificaciones

	order_id [PK] integer	order_line_id [PK] integer	wine_id integer	quantity integer	discount integer
1	1	1	3	2	10
2	1	2	5	1	[null]
3	1	3	7	2	15
4	2	1	8	3	20
5	3	1	10	1	5
6	4	1	12	2	25

Table CUSTOMER_ORDER:

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

SELECT * FROM CUSTOMER_ORDER;
```

Data Output Mensajes Notificaciones

	order_id [PK] integer	customer_id integer	order_date date	order_status character varying (20)	order_amount numeric (8,2)	order_reference character (11)
1	1	1	2021-01-05	shipped	11423.78	ABC-01-DEFG
2	2	1	2021-02-06	shipped	345.48	XYZ-02-HIJK
3	3	1	2021-03-10	delivered	113.95	MNO-03-LMNP
4	4	2	2021-04-11	shipped	518.24	PQR-04-QRST

Table CUSTOMER:

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

SELECT * FROM CUSTOMER;
```

Data Output Mensajes Notificaciones

	customer_id [PK] integer	customer_vat character varying (20)	customer_name character varying (100)	customer_address character varying (200)	customer_town character varying (100)	customer_phone character varying (20)	customer_email character varying (100)	customer_status character varying (20)
1	1	ES12345678A	SuperMart	123 Main St	Barcelona	123-456-7890	contact@supermart.com	active
2	2	ES87654321B	FreshFoods	456 Elm St	Madrid	987-654-3210	info@freshfoods.com	active
3	3	PT11223344C	Bistro Delight	789 Oak St	Lisbon	112-233-4455	hello@bistrodelight.com	active
4	4	PT55667788D	Gourmet Kitchen	321 Pine St	Porto	556-677-8899	support@gourmetkitchen.com	active

Once we have checked that the structure of RESPORT_WINE is correct and all the data has been inserted correctly in the tables we are going to use, we proceed to execute the SQL program that performs the function of exercise 1.

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

CREATE OR REPLACE FUNCTION update_report_wine(p_wine_id INT)
RETURNS REPORT_WINE_TYPE AS $$
DECLARE
```

Data Output Mensajes Notificaciones

```
CREATE FUNCTION

Consulta retornó exitosamente en 147 msec.
```

Process of checking the requirements of exercise 1:

EXISTING WINE WITH ORDERS

We know that wine_id= 1 is an existing wine so we will use it for testing.

SET search_path TO ubd_20241;

SELECT * FROM update_report_wine(1);

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

SELECT * FROM update_report_wine(1);
```

Data Output Mensajes Notificaciones

	t_wine_id smallint	t_wine_name character varying (255)	t_alcohol_content numeric (4,2)	t_category character varying (255)	t_price numeric (7,2)	t_prizes integer	t_total_sold integer	t_orders integer	t_customer_id smallint	t_customer_name character varying (255)
1	1	Torres Sangre de Toro	13.50	young	62.35	[null]	6	3	12	Home Essentials

The function update_report_wine(), we can see that it correctly inserts the necessary data and displays them on the screen.
In the table report_wine it shows the wine with the id that has been executed with the function (update_report_wine()).

SET search_path TO ubd_20241;

SELECT * FROM REPORT_WINE;

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

SELECT * FROM REPORT_WINE;
```

Data Output Mensajes Notificaciones

	wine_id [PK] smallint	wine_name character varying (255)	alcohol_content numeric (4,2)	category character varying (255)	price numeric (7,2)	prizes integer	total_sold integer	orders integer	customer_id smallint	customer_name character varying (255)
1	1	Torres Sangre de Toro	13.50	young	62.35	[null]	6	3	12	Home Essentials

EXISTING WINE WITHOUT ORDER

To check which wine has not been sold, we have to make a program that, taking into account the wine_id, does not have any order quantity.

```
SET search_path TO ubd_20241;

SELECT w.wine_id, w.wine_name
FROM WINE w
LEFT JOIN ORDER_LINE ol ON w.wine_id = ol.wine_id
WHERE ol.quantity IS NULL;
```

Result:

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

SELECT w.wine_id, w.wine_name
FROM WINE w
LEFT JOIN ORDER_LINE ol ON w.wine_id = ol.wine_id
WHERE ol.quantity IS NULL;
```

Data Output Mensajes Notificaciones

	wine_id [PK] integer	wine_name character varying (100)
1	26	Numanthia Termes
2	27	González Byass Tío Pepe
3	39	Castello di Ama Chianti Classico

We know that 26 has no orders.

This will help us to check if the function update_report_wine() adds to the REPORT_WINE table only those wines that have had a sale and if it displays a message indicating that the wine has no order, like this message: Wine with ID % has never been requested in an order.

We will use the following program to see if it does its function well:

SET search_path TO ubd_20241;

select * from update_report_wine(26)

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

select * from update_report_wine(26);
```

Data Output Mensajes Notificaciones

```
NOTICE:  Wine with ID 26 has never been requested in an order.

Ejecución exitosa. Tiempo de ejecución total de la consulta: 195 msec.
1 filas afectadas.
```

Check table REPORT_WINE:

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

select * from report_wine;
```

Data Output Mensajes Notificaciones

	wine_id [PK] smallint	wine_name character varying (255)	alcohol_content numeric (4,2)	category character varying (255)	price numeric (7,2)	prizes integer	total_sold integer	orders integer	customer_id smallint	customer_name character varying (255)
1	1	Torres Sangre de Toro	13.50	young	62.35	[null]	6	3	12	Home Essentials

No item has been added, as there is no order with wine with wine_id=26.

NON-EXISTENT WINE

To add a non-existent wine we will have to add an id that is not registered, such as 123, we wait for a message to verify that the programme does not add the non-existent wine to the table and a message appears on the screen like this: There is no wine with the identifier provided: %

Code to check the non-existent wine:

SET search_path TO ubd_20241;

select * from update_report_wine(123);

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

select * from update_report_wine(123);
```

Data Output Mensajes Notificaciones

```
ERROR:  There is no wine with the identifier provided: 123
HINT:  Verifique que el ID del vino sea correcto.
CONTEXT:  PL/pgSQL function update_report_wine(integer) line 8 at RAISE

Estado SQL: P0001
```

It is also not added to the report_wine table, as there is no wine with wine_id = 123.

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

select * from report_wine;
```

Data Output Mensajes Notificaciones

	wine_id [PK] smallint	wine_name character varying (255)	alcohol_content numeric (4,2)	category character varying (255)	price numeric (7,2)	prizes integer	total_sold integer	orders integer	customer_id smallint	customer_name character varying (255)
1	1	Torres Sangre de Toro	13.50	young	62.35	[null]	6	3	12	Home Essentials

UPDATING THE TABLE IF DATA CHANGES

As we can see in the previous image the total_sold of the customer with id 12 is 6, so let's add 5 more to see if it is modified in the table report_wine with the following code:

```
-- Update the total amount of wines sold (total_sold) for a specific customer
UPDATE REPORT_WINE SET total_sold = total_sold + 5 -- Increase by 5 (modify as necessary)
WHERE customer_id = 12 -- Specify corresponding customer_id
AND wine_id = 1; -- Specify the matching wine_id
```

Result:

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

select * from report_wine;
```

Data Output Mensajes Notificaciones

	wine_name character varying (255)	alcohol_content numeric (4,2)	category character varying (255)	price numeric (7,2)	prizes integer	total_sold integer	orders integer	customer_id smallint	customer_name character varying (255)
1	Torres Sangre de Toro	13.50	young	62.35	[null]	11	3	12	Home Essentials

DELETE CUSTOMER TO SEE IF THE TABLE IS UPDATED

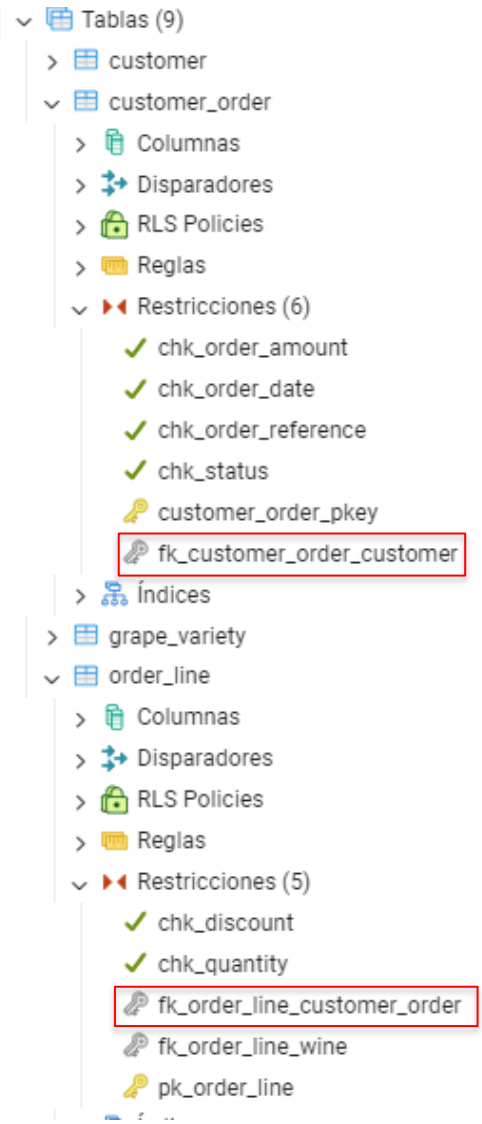
We are going to delete a customer with customer_id=12, to see if the table is updated when this customer does not exist.
As there are foreign keys, we must remove the client from all the tables in which it is associated..

```
DELETE FROM ORDER_LINE
WHERE order_id IN (
   SELECT order_id
   FROM CUSTOMER_ORDER
   WHERE customer_id = 12
);

DELETE FROM CUSTOMER_ORDER
WHERE customer_id = 12;

DELETE FROM CUSTOMER
WHERE customer_id = 12;
```

We then checked that the values should not have changed in the Report_wine table, as you can see in the picture.



Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

select * from report_wine;
```

Data Output Mensajes Notificaciones

	wine_name character varying (255)	alcohol_content numeric (4,2)	category character varying (255)	price numeric (7,2)	prizes integer	total_sold integer	orders integer	customer_id smallint	customer_name character varying (255)
1	Torres Sangre de Toro	13.50	young	62.35	[null]	11	3	12	Home Essentials

But if we run the function update_report_wine(1), the value is changed:

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

select * from update_report_wine(1);
```

Data Output Mensajes Notificaciones

	t_wine_id smallint	t_wine_name character varying (255)	t_alcohol_content numeric (4,2)	t_category character varying (255)	t_price numeric (7,2)	t_prizes integer	t_total_sold integer	t_orders integer	t_customer_id smallint	t_customer_name character varying (255)
1	1	Torres Sangre de Toro	13.50	young	62.35	[null]	3	2	28	Gourmet Delights

Exercise 2

In the table WINE we have a column called stock that will be used to store **the number of boxes available for each wine.**

You need to create a trigger (or triggers), on any necessary table or tables, so it correctly maintains updated the column stock in the table WINE, so that inventory remains accurate in real-time based on the customer orders. The user should be informed with a specific message when stock is insufficient for an order.

Concretely, we are asking for this column to always store up to date the values every time there are changes to the database.

Concretely, we are asking for this column to always store up to date values every time there are changes to the database.

You can assume that NO users or programs will directly update the column stock in the table WINE.

CODE

```
SET search_path TO ubd_20241;


-- Create the trigger that keeps the stock updated in WINE
CREATE OR REPLACE FUNCTION update_stock()
RETURNS TRIGGER AS $$
BEGIN
   -- Handling for a new insertion in ORDER_LINE
   IF TG_OP = 'INSERT' THEN
      -- Check if there is sufficient stock before placing an order
      IF (SELECT stock FROM WINE WHERE wine_id = NEW.wine_id) < NEW.quantity THEN
         RAISE EXCEPTION 'Insufficient stock to complete the order for wine with ID: %.',
NEW.wine_id;
      END IF;


      -- Subtract the quantity of the stock in WINE
      UPDATE WINE
```

```
        SET stock = stock - NEW.quantity
        WHERE wine_id = NEW.wine_id;

    -- Handling for an update in ORDER_LINE
    ELSIF TG_OP = 'UPDATE' THEN
        -- Calculate the difference between the new quantity and the previous quantity.
        DECLARE
            quantity_difference INTEGER := NEW.quantity - OLD.quantity;
        BEGIN
            --  If the quantity has increased, check if there is enough stock
            IF quantity_difference > 0 THEN
                IF (SELECT stock FROM WINE WHERE wine_id = NEW.wine_id) < quantity_difference
THEN
                    RAISE EXCEPTION 'Insufficient stock to increase the order quantity of the wine with ID:
%', NEW.wine_id;
                END IF;
                -- If sufficient stock is available, subtract the difference
                UPDATE WINE
                SET stock = stock - quantity_difference
                WHERE wine_id = NEW.wine_id;
            ELSE
                -- If the quantity has decreased, increase the stock in WINE
                UPDATE WINE
                SET stock = stock + ABS(quantity_difference)
                WHERE wine_id = NEW.wine_id;
            END IF;
        END;

    -- Handling for elimination in ORDER_LINE
    ELSIF TG_OP = 'DELETE' THEN
```

```
        -- Return the removed quantity to stock in WIN
        UPDATE WINE
        SET stock = stock + OLD.quantity
        WHERE wine_id = OLD.wine_id;
    END IF;

    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

-- Assign trigger to ORDER_LINE table for INSERT, UPDATE and DELETE operations
CREATE TRIGGER trigger_update_stock
AFTER INSERT OR UPDATE OR DELETE ON ORDER_LINE
FOR EACH ROW
EXECUTE FUNCTION update_stock();
```

TEST

We preview the stock of Vega Sicilia, the wine to be tested.

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

SELECT stock FROM WINE WHERE wine_id = 3;
```

Data Output Mensajes Notificaciones

	stock integer
1	25

We see that the stock of Vega Sicilia is 25.

FIRST STEP

We add a new order of a minimum quantity to the current stock to be able to carry out the tests with order_id=100.

-- Insert a new order in CUSTOMER_ORDER so that `order_id = 100`.

INSERT INTO CUSTOMER_ORDER (order_id, customer_id, order_date, order_status, order_amount)

VALUES (100, 1, '2024-11-15', 'pending', 150.00);

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;


INSERT INTO CUSTOMER_ORDER (order_id, customer_id, order_date, order_status, order_amount)
VALUES (100, 1, '2024-11-15', 'pending', 150.00);

```

Data Output Mensajes Notificaciones

```
INSERT 0 1

Consulta retornó exitosamente en 60 msec.
```

-- Add an order line for 'Vega Sicilia Único' (wine_id = 3) with quantity = 5

INSERT INTO ORDER_LINE (order_id, order_line_id, wine_id, quantity, discount)

VALUES (100, 1, 3, 5, NULL);

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

INSERT INTO ORDER_LINE (order_id, order_line_id, wine_id, quantity, discount)
VALUES (100, 1, 3, 5, NULL);
```

Data Output Mensajes Notificaciones

```
INSERT 0 1

Consulta retornó exitosamente en 63 msec.
```

-- Check stock after insertion

SELECT stock FROM WINE WHERE wine_id = 3;

```
SET search_path TO ubd_20241;

SELECT stock FROM WINE WHERE wine_id = 3;
```

Data Output Mensajes Notificaciones

	stock integer
1	20

We see that the value of the stock is reduced as it should be.

SECOND STEP

Add a new order for a larger quantity than the current stock to be able to test with order_id=101.

-- Insert a new order in CUSTOMER_ORDER with order_id = 101

INSERT INTO CUSTOMER_ORDER (order_id, customer_id, order_date, order_status, order_amount)

VALUES (101, 1, '2024-11-15', 'pending', 450.00);

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;


INSERT INTO CUSTOMER_ORDER (order_id, customer_id, order_date, order_status, order_amount)
VALUES (101, 1, '2024-11-15', 'pending', 450.00);
```

Data Output Mensajes Notificaciones

```
INSERT 0 1

Consulta retornó exitosamente en 79 msec.
```

-- Try adding an order line for "Vega Sicilia Único" (wine_id = 3) with quantity = 30

INSERT INTO ORDER_LINE (order_id, order_line_id, wine_id, quantity, discount)

VALUES (101, 1, 3, 30, NULL);

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

INSERT INTO ORDER_LINE (order_id, order_line_id, wine_id, quantity, discount)
VALUES (101, 1, 3, 30, NULL);
```

Data Output Mensajes Notificaciones

```
ERROR:  Insufficient stock to complete the order for wine with ID: 3.
CONTEXT:  PL/pgSQL function update_stock() line 7 at RAISE

Estado SQL: P0001
```

Indicates an error insufficient stock to complete the order for wine with ID:3, because the stock is less than the order quantity.

THIRT STEP

We do an order update with order_id=100, to see if the stock is updated automatically.

-- Increase the quantity in the previous order line (order 100) from 5 to 8

UPDATE ORDER_LINE

SET quantity = 8

WHERE order_id = 100 AND order_line_id = 1;

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

UPDATE ORDER_LINE
SET quantity = 8
WHERE order_id = 100 AND order_line_id = 1;
```

Data Output Mensajes Notificaciones

```
UPDATE 1

Consulta retornó exitosamente en 68 msec.
```

-- Check stock after insertion

SELECT stock FROM WINE WHERE wine_id = 3;

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

SELECT stock FROM WINE WHERE wine_id = 3;
```

Data Output Mensajes Notificaciones

	stock integer
1	17

FOURTH STEP

Re-update the order with order_id= 100 to a quantity greater than the current stock of Vega Sisilia wine.

-- Try to increase the quantity in order 100 from 8 to 30 (not enough stock).

UPDATE ORDER_LINE

SET quantity = 30

WHERE order_id = 100 AND order_line_id = 1;

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

UPDATE ORDER_LINE
SET quantity = 30
WHERE order_id = 100 AND order_line_id = 1;
```

Data Output Mensajes Notificaciones

```
ERROR:  Insufficient stock to increase the order quantity of the wine with ID: 3
CONTEXT:  PL/pgSQL function update_stock() line 24 at RAISE

Estado SQL: P0001
```

The error of insufficient stock to increase the order quantity of the wine with ID: 3 is indicated again. Even if it came from a previously accepted order.

FIFTH STEP

Re-update the order with order_id= 100 to a quantity less than the current stock of Vega Sisilia wine.

-- Reduce order quantity with order_id =100 from 8 to 3

UPDATE ORDER_LINE

SET quantity = 3

WHERE order_id = 100 AND order_line_id = 1;

```
SET search_path TO ubd_20241;

UPDATE ORDER_LINE
SET quantity = 3
WHERE order_id = 100 AND order_line_id = 1;
```

Data Output Mensajes Notificaciones

```
UPDATE 1

Consulta retornó exitosamente en 68 msec.
```

-- Check the stock after the update

SELECT stock FROM WINE WHERE wine_id = 3;

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

SELECT stock FROM WINE WHERE wine_id = 3;
```

Data Output Mensajes Notificaciones

	stock integer
1	22

As we can see the order is placed and the stock is changed, it is 22 because we have to take into account that we start from an initial stock of 25 and it has been changed for the third time to a total order of 3 unit

SIXTH STEP

Delete the order with order_id = 100, to see if the stock returns to its initial quantity.

-- Delete the order line (order_id= 100) for 'Vega Sicilia Único'

DELETE FROM ORDER_LINE

WHERE order_id = 100 AND order_line_id = 1;

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

DELETE FROM ORDER_LINE
WHERE order_id = 100 AND order_line_id = 1;
```

Data Output Mensajes Notificaciones

```
DELETE 1

Consulta retornó exitosamente en 82 msec.
```

-- Check stock after elimination

SELECT stock FROM WINE WHERE wine_id = 3;

Consulta Historial de Consultas

```
SET search_path TO ubd_20241;

SELECT stock FROM WINE WHERE wine_id = 3;
```

Data Output Mensajes Notificaciones

	stock integer
1	25

The stock is reset to its initial quantity before any order is placed.