**1. CONOCIMIENTOS SQL**

**1.1)** Describe el funcionamiento general de la sentencia **JOIN**.

Sirve para unir dos o mas tablas en una consulta mediante una condición relacionada entre estas para tomar sus datos y mostrarlos.

**1.2)** ¿Cuáles son los tipos de **JOIN** y cuál es el funcionamiento de los mismos?

**Left join:** este tipo indica que solo puede mostrar todos los datos de la tabla izquierda pero solo si los datos coinciden con la tabla de la derecha.

**Right join:** este tipo indica que solo puede mostrar datos de la tabla derecha solo si los datos coinciden con la tabla de la izquierda.

**Inner join:** este tipo indica que retornara los datos de las dos tablas que coincidan en ambas.

**Full join:** este tipo devuelve todas las filas cuando hay coincidencias en alguna de las tablas.

**Cross join:** este tipo de una consulta combinada, devuelve cada fila de una tabla con cada fila de la otra tabla.

**1.3) ¿**Cuál el funcionamiento general de los **TRIGGER** y qué propósito tienen?

Es una acción que funciona o se dispara automáticamente cuando ocurre un evento especifico ya sea agregar, actualizar o eliminar. Su propósito es automatizar la base de datos, a garantizar la integridad de los datos y registrar eventos.

**1.4)** ¿Qué es y para qué sirve un **STORED PROCEDURE**?

Un procedimiento almacenado es un bloque de código SQL con instrucciones que se guardan en la base de datos y se pueden ejecutar de manera rápida. Sirve para automatizar procesos, ejecutar tareas complejas, para tareas que se tengan que ejecutar constantemente, para seguridad de la base en la que solo el procedimiento se le den permisos.

Hacer las consultas necesarias para:

**1.5)** Traer todos los productos que tengan una venta.

Realice dos consulta debido a una duda, no estaba seguro si el enunciado se refiere a obtener los productos que tengan una sola venta o productos que tengan al menos una venta.

Consulta para obtener productos con una sola venta:

SELECT

P.IDPRODUCTO,

P.NOMBRE

FROM productos P

RIGHT JOIN ventas V ON P.idProducto = V.idProducto

WHERE V.cantidad = 1;

Consulta para obtener productos que tengan al menos una venta:

SELECT DISTINCT

P.IDPRODUCTO,

P.NOMBRE

FROM productos P

RIGHT JOIN ventas V ON P.idProducto = V.idProducto;

**1.6)** Traer todos los productos que tengan ventas y la cantidad total de productos vendidos.

SELECT

P.IDPRODUCTO,

P.NOMBRE,

SUM(V.cantidad) AS CANTIDAD\_TOTAL

FROM productos P

RIGHT JOIN ventas V ON P.idProducto = V.idProducto

GROUP BY

P.IDPRODUCTO,

P.NOMBRE;

**1.7)** Traer todos los productos (independientemente de si tienen ventas o no) y la suma total ($) vendida por producto.

SELECT

P.IDPRODUCTO,

P.NOMBRE,

COALESCE((SUM(V.cantidad) \* P.precio), 0) AS SUMA\_TOTAL

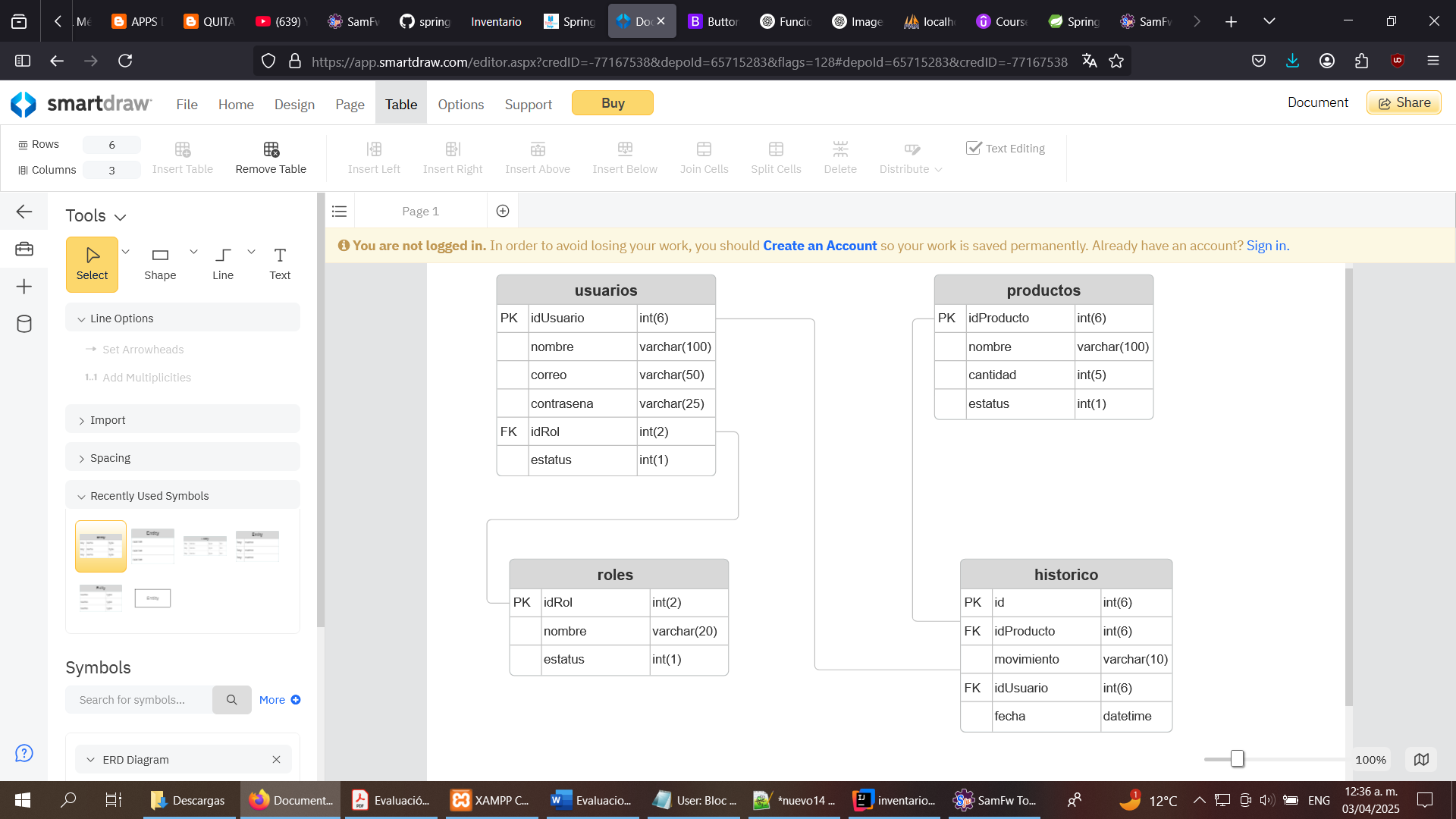
FROM productos P

LEFT JOIN ventas V ON P.idProducto = V.idProducto

GROUP BY

P.IDPRODUCTO,

P.NOMBRE;



CREATE TABLE `inventario`.`usuarios`

(

`idUsuario` INT(6) NOT NULL AUTO\_INCREMENT ,

`nombre` VARCHAR(100) NOT NULL ,

`correo` VARCHAR(50) NOT NULL ,

`contrasena` VARCHAR(25) NOT NULL ,

`idRol` INT(2) NOT NULL ,

`estatus` INT(1) NOT NULL ,

PRIMARY KEY (`idUsuario`),

INDEX `idRol` (`idRol`)

) ;

CREATE TABLE `inventario`.`roles`

(

`idRol` INT(2) NOT NULL AUTO\_INCREMENT ,

`nombre` VARCHAR(20) NOT NULL ,

`estatus` INT(1) NOT NULL ,

PRIMARY KEY (`idRol`)

);

CREATE TABLE `inventario`.`productos`

(

`idProducto` INT(6) NOT NULL AUTO\_INCREMENT ,

`nombre` VARCHAR(100) NOT NULL ,

`cantidad` INT(5) NOT NULL ,

`estatus` INT(1) NOT NULL ,

PRIMARY KEY (`idProducto`)

);

CREATE TABLE `inventario`.`historico`

(

`id` INT(6) NOT NULL AUTO\_INCREMENT ,

`idProducto` INT(6) NOT NULL ,

`movimiento` VARCHAR(10) NOT NULL ,

`idUsuario` INT(6) NOT NULL ,

`fecha` DATETIME NOT NULL ,

PRIMARY KEY (`id`),

INDEX `idProducto` (`idProducto`),

INDEX `idUsuario` (`idUsuario`)

);