

JuiceStock

# Manual de Instalación

---

Elaborado por:

José David Arteaga Fernandez

Bibiana Sofia Cortes Montila

Brayan Steven Gomes Lasso

Juan Diego Gómez Garcés

Juan David Perdomo Ramos

Glenn Alexander Ward Ante

# Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Requisitos Previos</b>	<b>3</b>
2.1 Hardware y Sistema Operativo	3
2.2 Software Necesario	3
<b>3. Descarga e Instalación de Requisitos Previos</b>	<b>3</b>
3.1 Descarga e instalación de .NET 8.0	3
3.2 Descarga e instalación de SQL Developer.	4
3.3 Descarga e instalación de Oracle Database	6
<b>4. Configuración de la Base de Datos</b>	<b>9</b>
4.1 Configuración inicial de SQL Developer	9
4.2.1 Creación del usuario	11
4.2.2 Configuración de espacio de tablas	11
4.2.3 Asignación de privilegios	11
4.3 Creación de las tablas	13
4.3.1 Creación tabla Proveedor	13
4.3.2 Creación tabla Categoría de Producto	13
4.3.3 Creación tabla Producto	14
4.3.4 Creación tabla Usuario	14
4.3.5 Creación tabla Movimiento	15
4.3.6 Creación de usuarios JuiceStock y Tester	15
4.3.7 Eliminación de tablas	15
4.4 Creación de Funciones, Procedimientos, Disparadores, Sentencias, entre otros	16
4.4.1 Índices	16
4.4.2 Vistas	16
4.4.3 Secuencias	17
4.4.4 Disparadores	18
4.4.5 Funciones	20
4.4.5.1 Productos	20
4.4.5.2 Proveedores	23
4.4.6 Procedimientos	25
4.4.6.1 Productos	25
4.4.6.2 Proveedores	34
4.4.6.3 Usuarios	40
<b>5. Descarga Aplicación Ejecutable</b>	<b>42</b>



## 1. Introducción

JuiceStock es una aplicación diseñada para la gestión eficiente de inventarios de jugos. Este manual detalla paso a paso cómo preparar el entorno necesario para ejecutar la aplicación, incluyendo la configuración de Visual Studio, Oracle Database, SQL Developer. Además, se explicará el proceso de configuración y creación de la base de datos requerida por JuiceStock.

---

## 2. Requisitos Previos

### 2.1 Hardware y Sistema Operativo

- **Sistema Operativo:** Windows 10 o superior (64 bits).
- **Procesador:** CPU de al menos 1 núcleo a 1.5 GHz.
- **Memoria RAM:** Mínimo 4 GB.
- **Espacio en Disco:** Al menos 500 MB de espacio libre para la aplicación y datos relacionados.

### 2.2 Software Necesario

- **Oracle Database 19c o superior** (descargable desde el sitio oficial de Oracle).
- **SQL Developer** (para administrar la base de datos Oracle).
- **Git** (para clonar el repositorio desde GitHub).
- **.NET 8.0 SDK** (para compilar y ejecutar la aplicación).

Asegúrate de que tu sistema cumpla con estos requisitos antes de proceder con la instalación. Si no estás seguro de algunos de los requisitos, consulta con tu administrador de sistemas o soporte técnico.

---

## 3. Descarga e Instalación de Requisitos Previos

### 3.1 Descarga e instalación de .NET 8.0

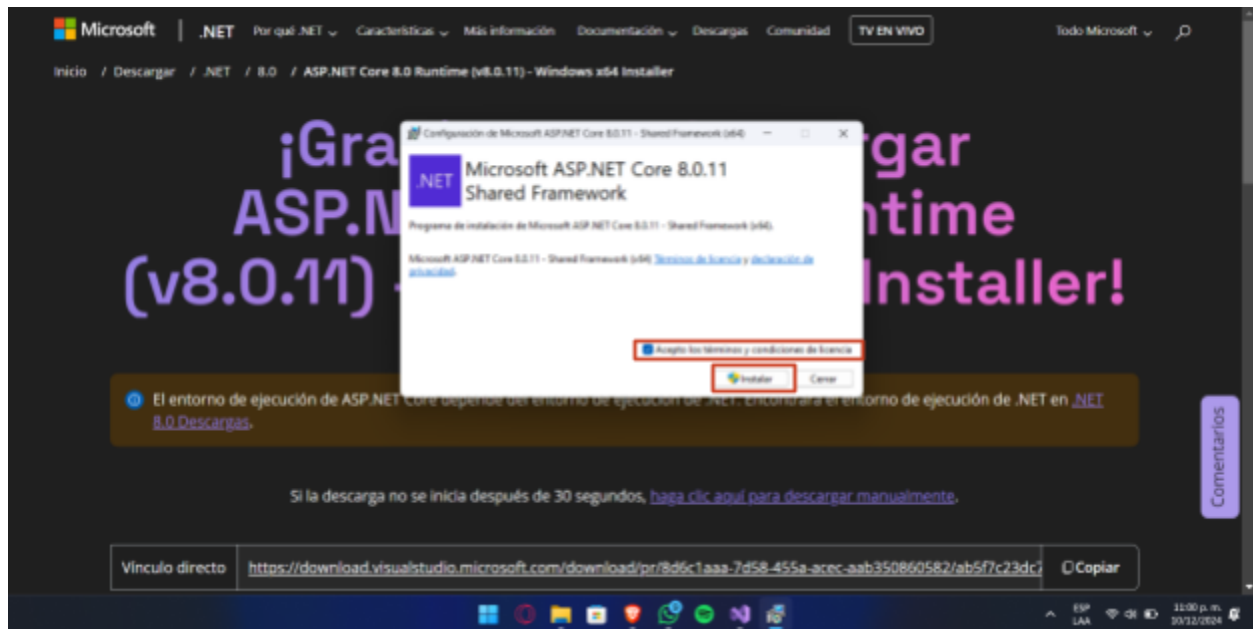
Para la ejecución de la aplicación es necesario instalar el entorno de ejecución de ASP.NET Core este permite ejecutar aplicaciones web o de servidor existentes.

Para su instalación solo basta con ir al siguiente enlace y seleccionar la ruta de descarga

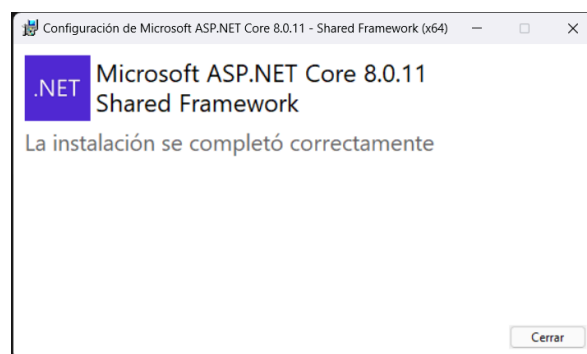
<https://dotnet.microsoft.com/es-es/download/dotnet/thank-you/runtime-aspnetcore-8.0.11-windows-x64-installer>

Se seleccionará la ruta en la que quieres guardar el instalador de .NET 8.0

Seleccionamos el archivo descargado y se abrirá una ventana en la cual presionaremos la casilla de “Acepto los termino y condiciones de licencia” y posteriormente presionamos el botón “Instalar”



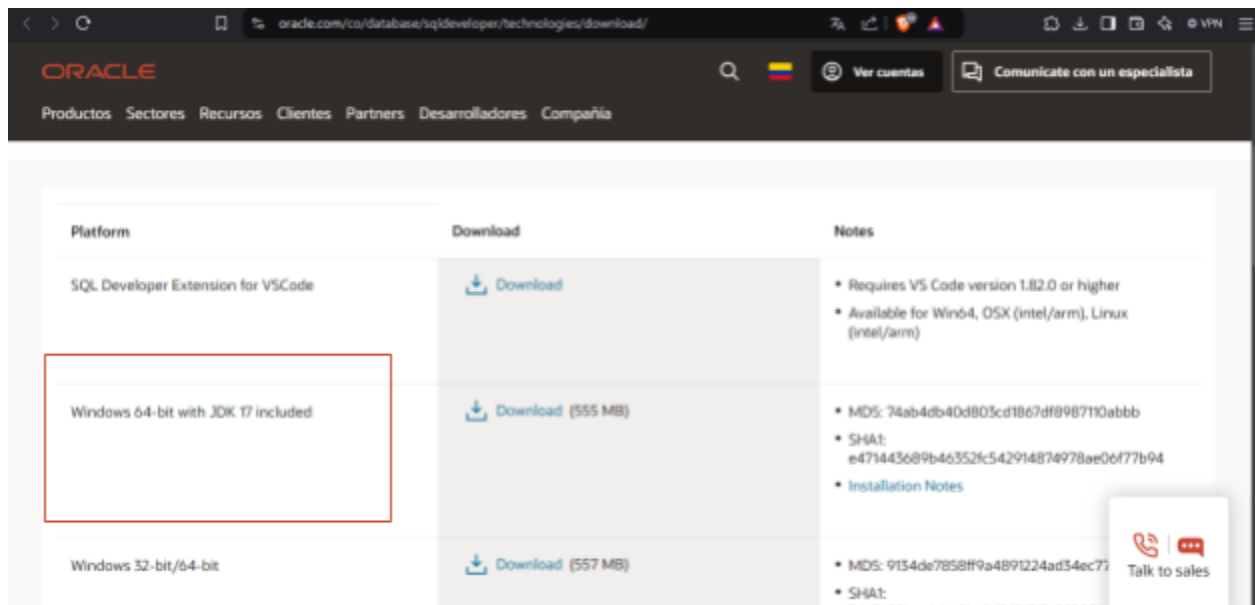
Esperamos un momento y el entorno de ejecución de ASP.NET 8.0 estará instalado



### 3.2 Descarga e instalación de SQL Developer.

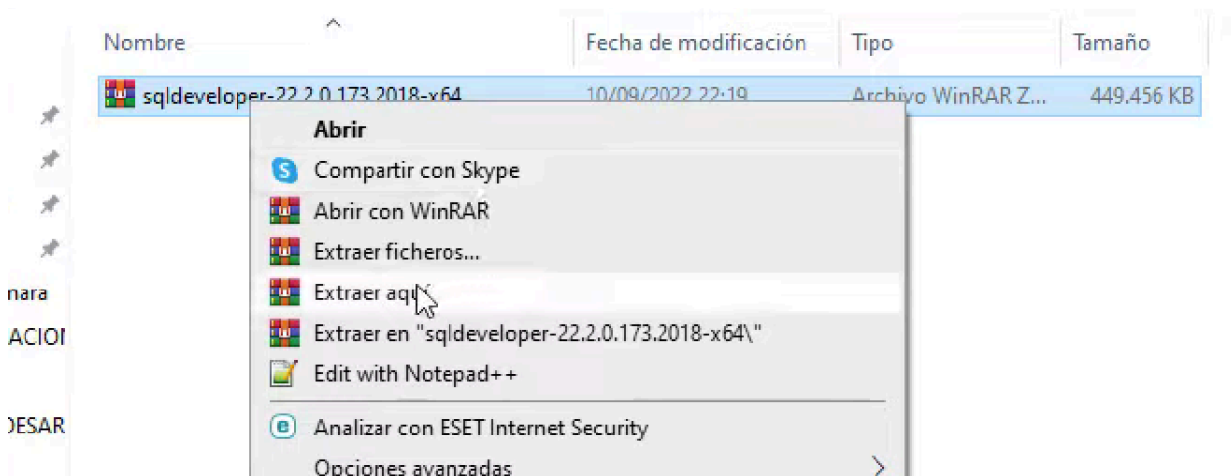
Para descargar el SQL Developer en el sistema debe usar el siguiente enlace y seleccionando la opción indicada :

- <https://www.oracle.com/co/database/sqldeveloper/technologies/download/>

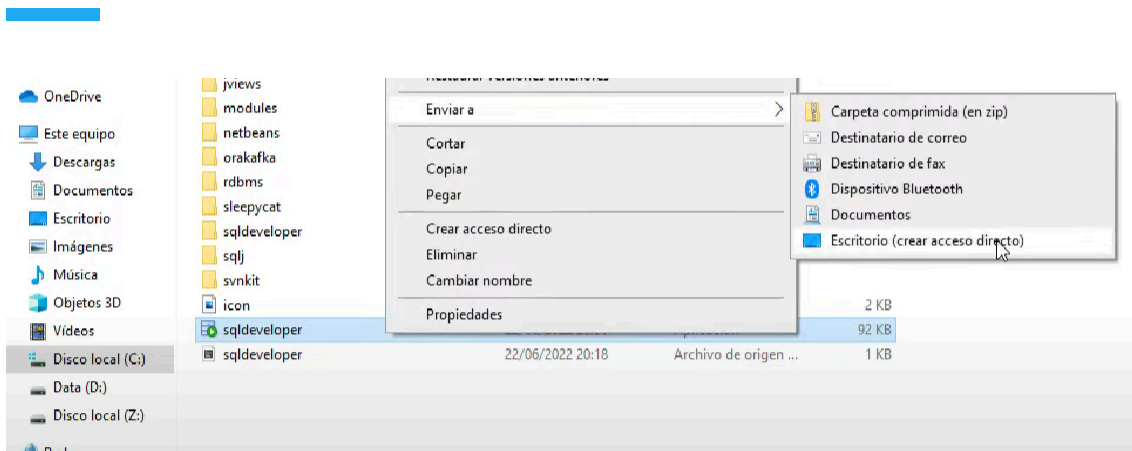


**Tip:** Para contar con un mayor orden de los archivos puedes crear una carpeta llamada “Oracle” donde podemos guardar y extraer los archivos de SQL Developer y Oracle Database XE

A continuación se debe extraer la carpeta comprimida:



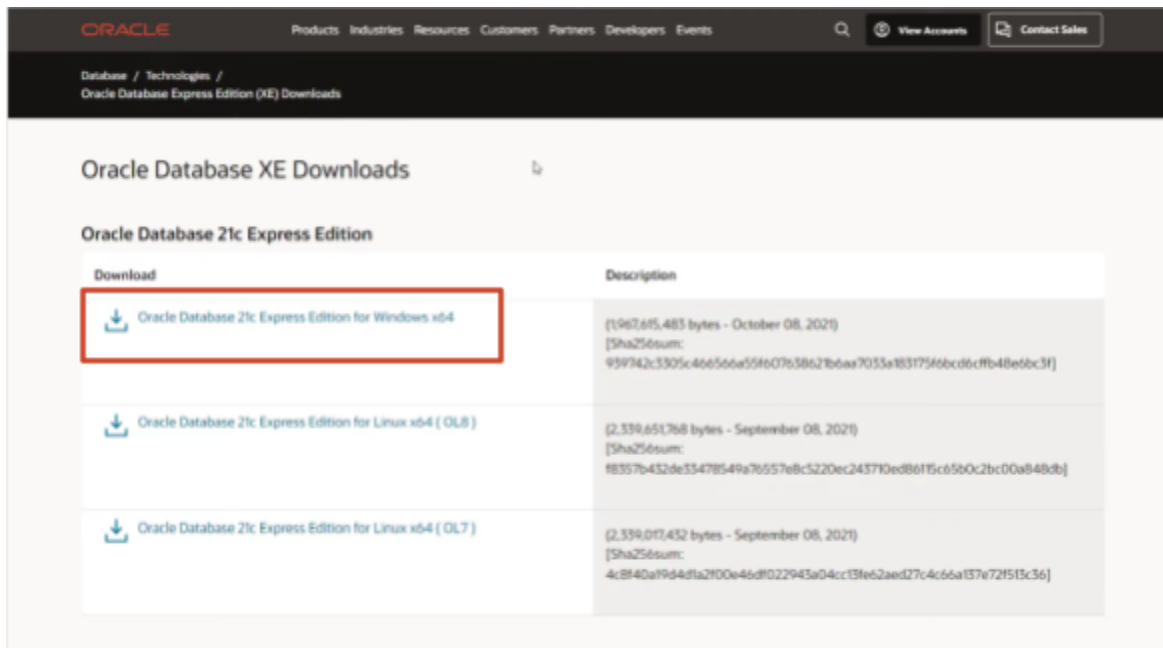
Finalmente podremos crear un acceso directo de la aplicación abriendo la carpeta descomprimida, nos ubicamos en el archivo “sqldeveloper” y con click derecho, seleccionamos “Enviar” y posteriormente “Escritorio (crear acceso directo)”



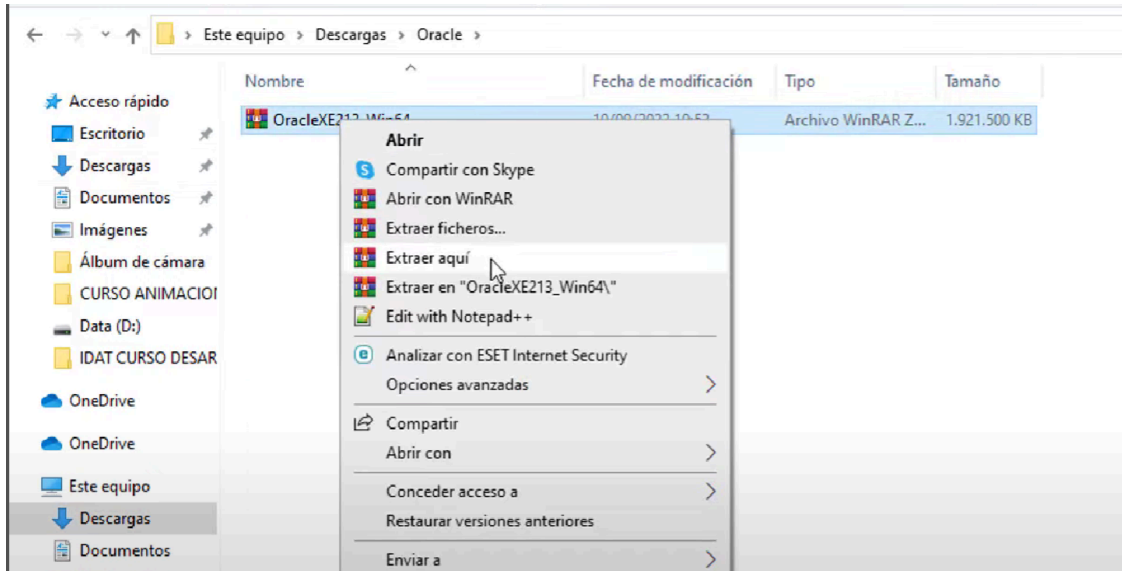
### 3.3 Descarga e instalación de Oracle Database

Para descargar Oracle Database XE en el sistema debe usar el siguiente link de acceso y seleccionar la opción indicada :

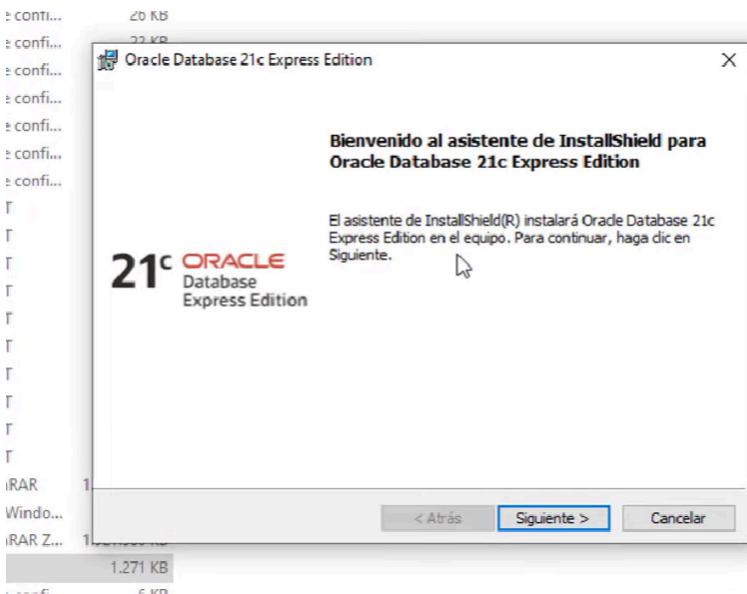
- <https://www.oracle.com/database/technologies/xe-downloads.html>



Se selecciona la ruta de descarga y una vez descargado se debe extraer la carpeta comprimida:  
(Recuerda extraer dentro de la carpeta “Oracle” para un mayor orden en los archivos)

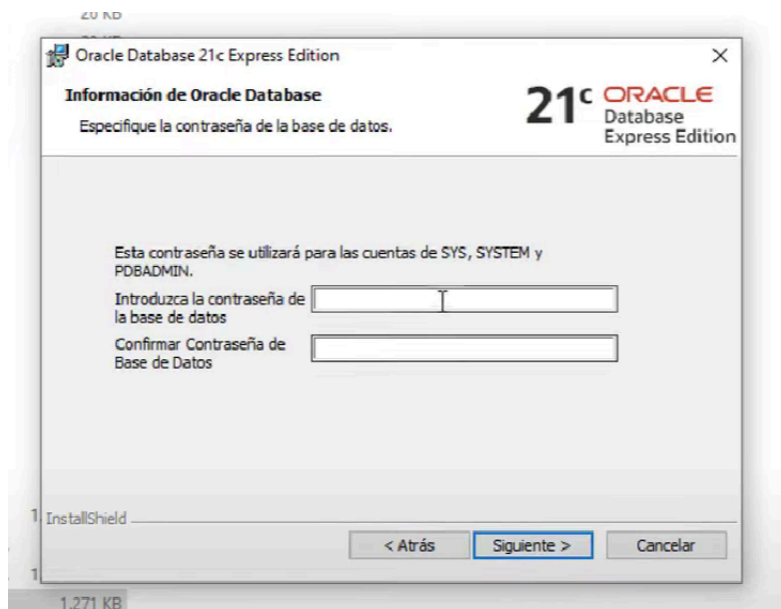


A continuación se abre la carpeta descomprimida y se debe hacer click en el archivo llamado “Setup”, el cual es un ejecutable para instalar la versión de Oracle



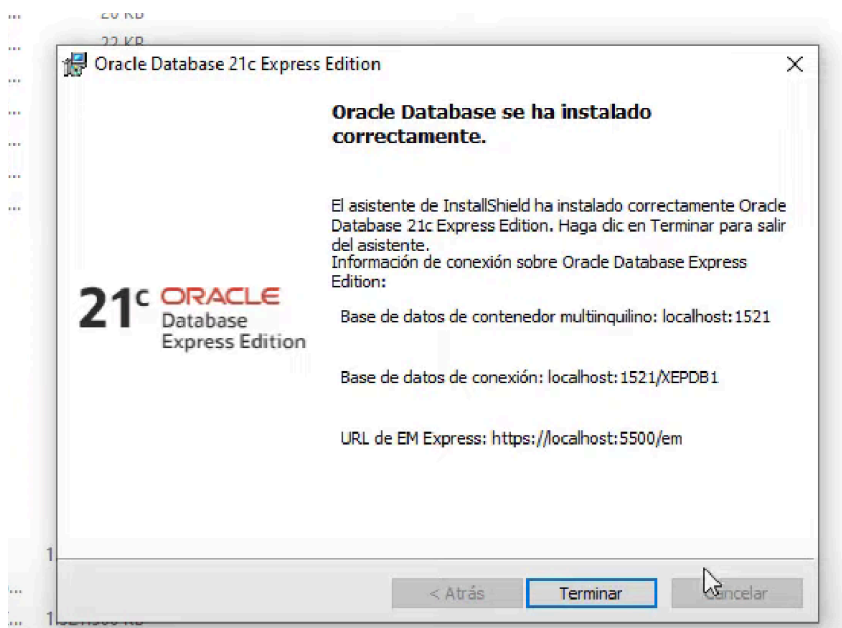
Daremos click en la opción siguiente hasta llegar a la siguiente ventana





En esta ventana se establecerá una contraseña la cual será “oracle” (recuerda que esta contraseña será única para los usuarios de la base de datos ”SYS”, “SYSTEM” y “PDBADMIN”), se sigue dando click en siguiente y posteriormente en instalar

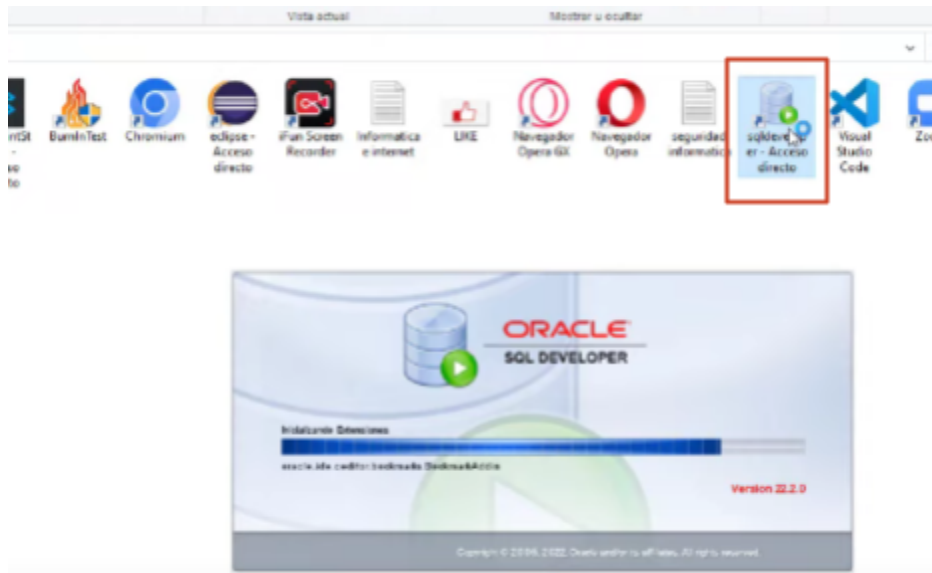
Se esperará los minutos necesarios para su instalación y una vez terminado damos click en el botón “Terminar”




## 4. Configuración de la Base de Datos

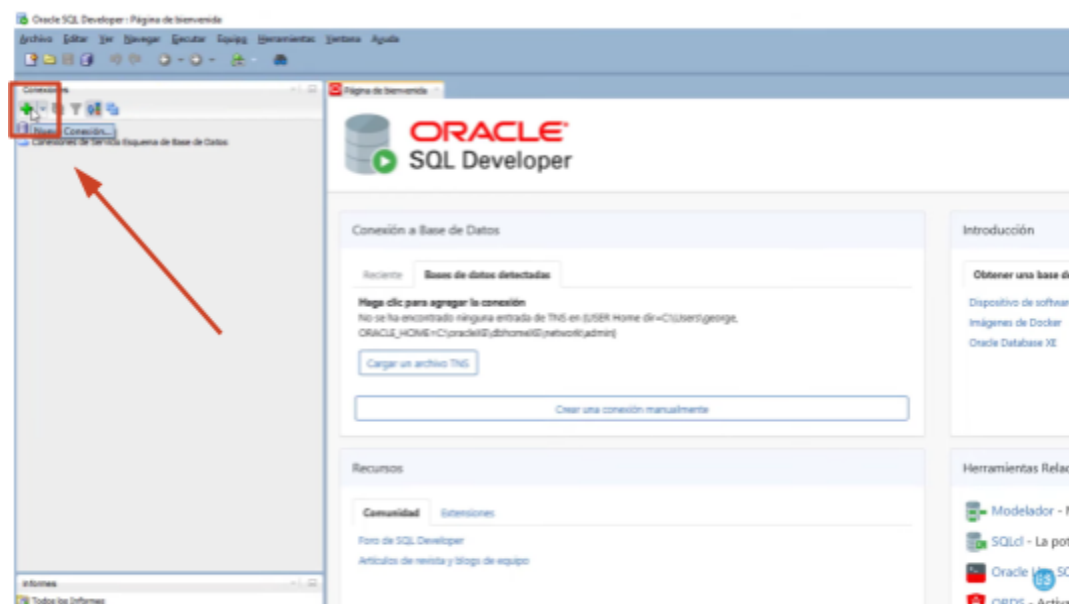
### 4.1 Configuración inicial de SQL Developer

Abriremos el acceso directo de SQL Developer

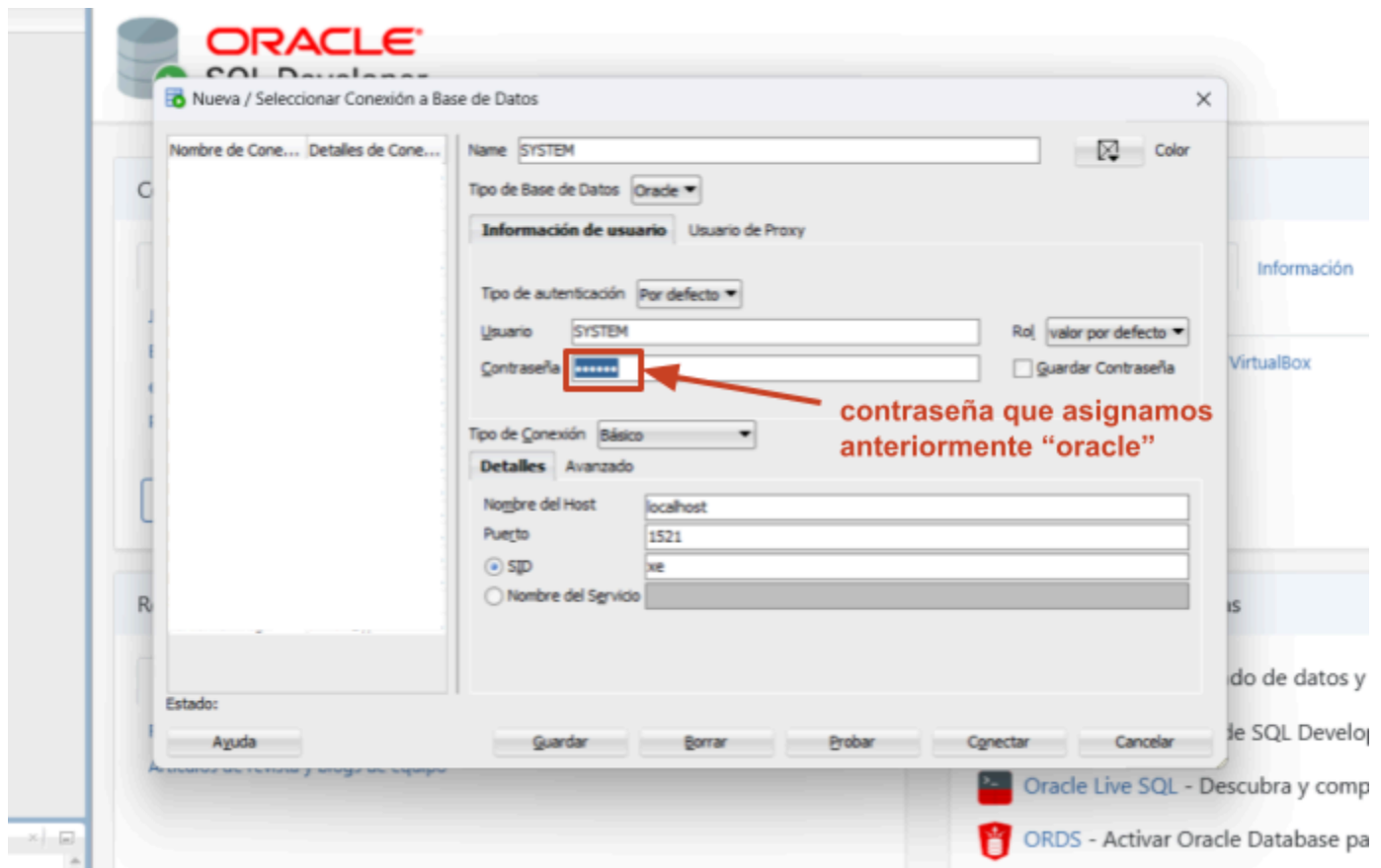


Ya estaremos dentro de Oracle SQL Developer, cliente que nos ofrece una interfaz gráfica para administrar nuestro gestor de base de datos de Oracle.

Creamos una nueva conexión, seleccionando el botón  :



Llenaremos los campos tal cual como se muestra en la imagen:



Daremos click en el orden mostrado en la imagen a los siguientes botones:



si se siguieron los pasos correctamente la conexión fue creada exitosamente

## 4.2 Creación del usuario

Una vez creada la conexión, procederemos a la creación del usuario, para esto debemos ejecutar los siguientes scripts por separado

### 4.2.1 Creación del usuario

Unset

```
alter session set "_ORACLE_SCRIPT"=true;  
CREATE USER juicestock identified by oracle
```

### 4.2.2 Configuración de espacio de tablas

Unset

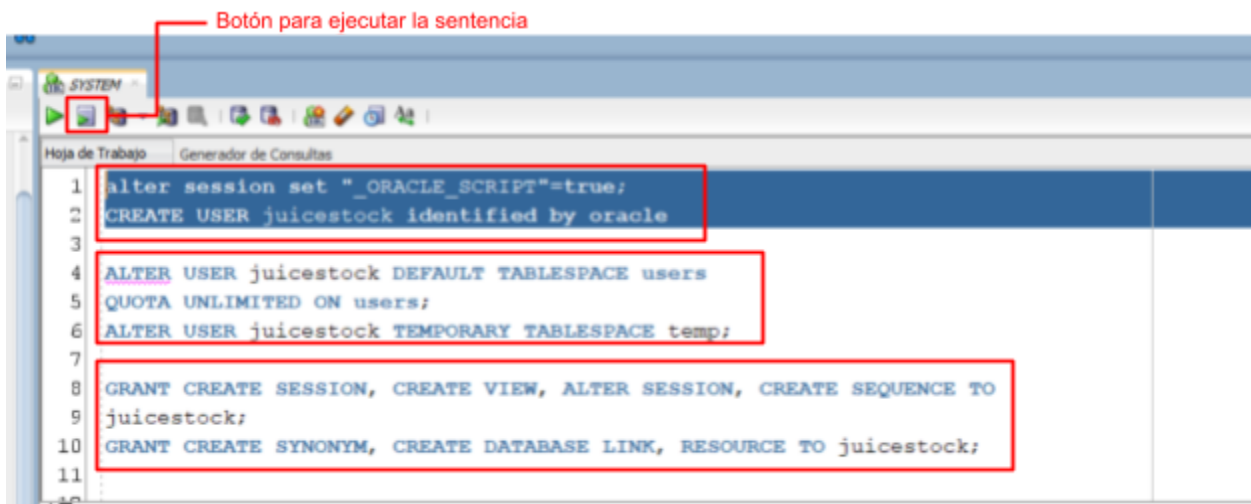
```
ALTER USER juicestock DEFAULT TABLESPACE users  
QUOTA UNLIMITED ON users;  
ALTER USER juicestock TEMPORARY TABLESPACE temp;
```

### 4.2.3 Asignación de privilegios

Unset

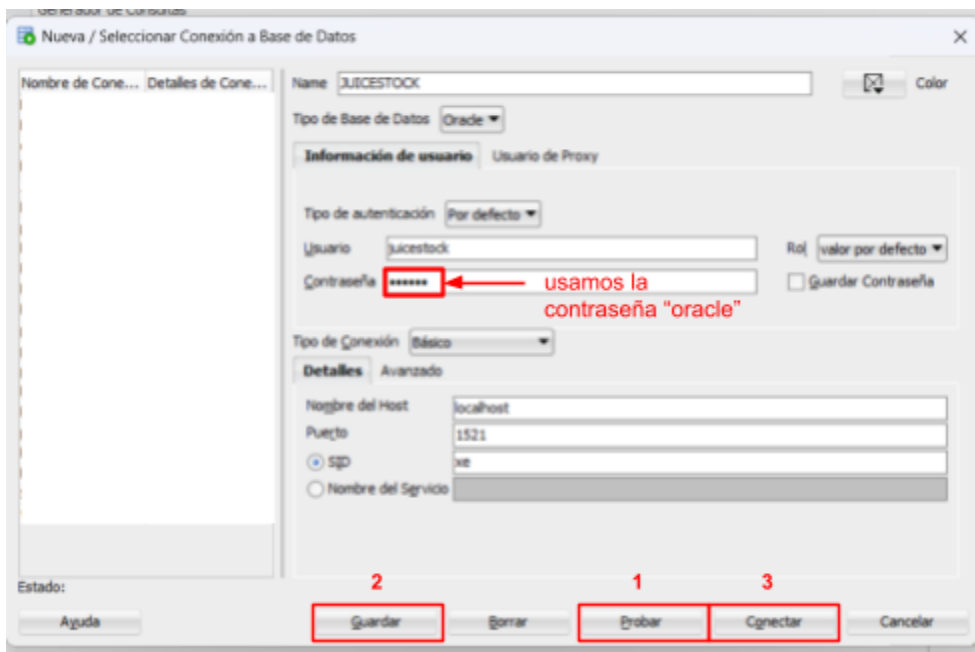
```
GRANT CREATE SESSION, CREATE VIEW, ALTER SESSION, CREATE SEQUENCE TO  
juicestock;  
GRANT CREATE SYNONYM, CREATE DATABASE LINK, RESOURCE TO juicestock;
```

Para ejecutarlos basta con copiarlos en la conexión que creamos y presionamos el botón señalado



De esta forma vamos a ejecutar todos los scripts siguientes.

Posteriormente crearemos una nueva conexión de la misma manera que la anterior pero esta vez con el nuevo usuario "juicestock"



## 4.3 Creación de las tablas

Para la creación de las tablas ejecutaremos las siguientes sentencias:

**Importante:** La creación se debe ejecutar en el orden en el que aparecen el documento

### 4.3.1 Creación tabla Proveedor

Unset

```
CREATE TABLE Proveedor (  
    id_prov NUMBER PRIMARY KEY,  
    nombre_prov VARCHAR2(50) NOT NULL,  
    telefono_prov VARCHAR2(15) NOT NULL,  
    correo_prov VARCHAR2(50),  
    direccion_prov VARCHAR2(50),  
    estado_prov VARCHAR2(10) NOT NULL,  
    CONSTRAINT ck_estado_prov CHECK (estado_prov IN ('ACTIVO', 'INACTIVO')),  
    CONSTRAINT ck_id_prov CHECK (id_prov > 0)  
);
```

### 4.3.2 Creación tabla Categoría de Producto

Unset

```
CREATE TABLE Categoria_Producto (  
    id_categoria NUMBER PRIMARY KEY,  
    nombre_categoria VARCHAR2(40) NOT NULL  
);
```

### 4.3.3 Creación tabla Producto

Unset

```
CREATE TABLE Producto (  
    id_prod NUMBER PRIMARY KEY,  
    nombre_prod VARCHAR2(50) NOT NULL,  
    id_categoria NUMBER NOT NULL,  
    precio_prod NUMBER(10, 2) NOT NULL,  
    cantidad_prod NUMBER NOT NULL,  
    estado_prod VARCHAR2(10) NOT NULL,  
    id_prov NUMBER,  
    CONSTRAINT fk_categoria FOREIGN KEY (id_categoria) REFERENCES  
    Categoria_Producto(id_categoria),  
    CONSTRAINT fk_proveedor FOREIGN KEY (id_prov) REFERENCES Proveedor(id_prov),  
    CONSTRAINT ck_estado_prod CHECK (estado_prod IN ('ACTIVO', 'INACTIVO')),  
    CONSTRAINT ck_id_prod CHECK (id_prod > 0),  
    CONSTRAINT ck_cantidad_prod CHECK (cantidad_prod >= 0)  
);
```

### 4.3.4 Creación tabla Usuario

Unset

```
CREATE TABLE USUARIO(  
    Id_Usuario NUMBER(10) PRIMARY KEY,  
    nom_Usuario varchar2(50),  
    usuario varchar2(50),  
    contraseña varchar2(50),  
    CONSTRAINT ck_id_usuario CHECK (Id_Usuario > 0)
```

```
);
```

#### 4.3.5 Creación tabla Movimiento

Unset

```
CREATE TABLE MOVIMIENTO
(
    id_mov number PRIMARY KEY,
    id_prod number not null,
    cantidad_prod number not null,
    fecha_mov timestamp not null,
    precio_unitario_prod number not null,
    tipo_mov varchar2(10),
    CONSTRAINT ck_tipo_mov CHECK (tipo_mov IN ('ENTRADA', 'SALIDA'))
);
```

#### 4.3.6 Creación de usuarios JuiceStock y Tester

Unset

```
INSERT INTO USUARIO VALUES(100, 'Juice Stock', 'admin', '1234');
INSERT INTO USUARIO VALUES(300, 'Tester', 'pruebas', '1234');
commit;
```

#### 4.3.7 Eliminación de tablas

En caso de haber un error en las tablas puede eliminarlas para crearlas de nuevo



Unset

```
DROP TABLE Proveedor CASCADE CONSTRAINTS;  
  
DROP TABLE Categoria_Producto CASCADE CONSTRAINTS;  
  
DROP TABLE Producto CASCADE CONSTRAINTS;  
  
DROP TABLE USUARIO CASCADE CONSTRAINTS;  
  
DROP TABLE MOVIMIENTO CASCADE CONSTRAINTS;
```

## 4.4 Creación de Funciones, Procedimientos, Disparadores, Sentencias, entre otros

### 4.4.1 Indices

Unset

```
CREATE INDEX PROD_ESTADO_IDX ON PRODUCTO (ESTADO_PROD);  
  
CREATE INDEX PROV_ESTADO_IDX ON PROVEEDOR (ESTADO_PROV);
```

### 4.4.2 Vistas

Unset

```
CREATE OR REPLACE VIEW VISTA_PRODUCTOS AS  
  
    SELECT id_prod, nombre_prod, nombre_categoria, nombre_prov, precio_prod, cantidad_prod  
  
    FROM PRODUCTO NATURAL JOIN CATEGORIA_PRODUCTO NATURAL JOIN PROVEEDOR  
  
    WHERE estado_prod = 'ACTIVO'  
  
    ORDER BY NOMBRE_PROD ASC;  
  
-----  
  
CREATE OR REPLACE VIEW vista_proveedores AS  
  
    SELECT  
  
    id_prov as "Identificador",  
  
    nombre_prov as "Nombre",
```

```
telefono_prov as "Telefono",  
correo_prov as "Correo Electronico",  
direccion_prov as "Dirección"  
FROM PROVEEDOR  
WHERE estado_prov = 'ACTIVO';
```

---

```
CREATE OR REPLACE VIEW vista_movimientos_general AS  
SELECT  
    m.fecha_mov AS "FECHA",  
    p.nombre_prod AS "PRODUCTO",  
    m.cantidad_prod AS "CANTIDAD",  
    m.precio_unitario_prod AS "PRECIO UNITARIO",  
    m.tipo_mov AS "TIPO DE MOVIMIENTO"  
FROM  
    Movimiento m  
JOIN  
    Producto p ON m.id_prod = p.id_prod  
ORDER BY fecha_mov DESC;
```

### 4.4.3 Secuencias

Unset

```
CREATE SEQUENCE seq_producto_id  
  
START WITH 100  
  
INCREMENT BY 10;
```

---

```
CREATE SEQUENCE seq_proveedor_id
```

```
START WITH 101  
INCREMENT BY 11;
```

```
-----  
  
CREATE SEQUENCE seq_movimiento_id  
  
START WITH 100  
  
INCREMENT BY 10;
```

#### 4.4.4 Disparadores

Unset

```
CREATE OR REPLACE TRIGGER trg_auto_increment_producto_id  
BEFORE INSERT ON Producto  
FOR EACH ROW  
BEGIN  
    IF :NEW.id_prod IS NULL THEN  
        SELECT seq_producto_id.NEXTVAL INTO :NEW.id_prod FROM dual;  
    END IF;  
END;
```

```
-----  
  
CREATE OR REPLACE TRIGGER trg_prevent_duplicate_usuario  
BEFORE INSERT ON USUARIO  
FOR EACH ROW  
DECLARE  
    v_count NUMBER;  
BEGIN  
    -- Contar cuantos usuarios existen con el mismo nombre de usuario  
    SELECT COUNT(*)  
    INTO v_count  
    FROM USUARIO  
    WHERE usuario = :NEW.usuario;
```

```

-- Si ya existe al menos un usuario con el mismo nombre, lanzar un error
IF v_count > 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'El usuario ya existe en la base de datos.');
```

---

```

END IF;
END trg_prevent_duplicate_usuario;

-----

CREATE OR REPLACE TRIGGER trg_auto_increment_proveedor_id
BEFORE INSERT ON Proveedor
FOR EACH ROW
BEGIN
    IF :NEW.id_prov IS NULL THEN
        SELECT seq_proveedor_id.NEXTVAL INTO :NEW.id_prov FROM dual;
    END IF;
END;

-----

CREATE OR REPLACE TRIGGER trg_registrar_movimiento
AFTER UPDATE OF cantidad_prod ON Producto
FOR EACH ROW
DECLARE
    v_tipo_mov VARCHAR2(10);
    v_cantidad_mov NUMBER;
BEGIN
    -- Determinamos el tipo de movimiento y la cantidad cambiada
    IF :NEW.cantidad_prod > :OLD.cantidad_prod THEN
        v_tipo_mov := 'ENTRADA';
        v_cantidad_mov := :NEW.cantidad_prod - :OLD.cantidad_prod;
    ELSIF :NEW.cantidad_prod < :OLD.cantidad_prod THEN
        v_tipo_mov := 'SALIDA';
        v_cantidad_mov := :OLD.cantidad_prod - :NEW.cantidad_prod;
    ELSE
```

```

        RETURN; -- Si no hubo cambio en cantidad, no se registra movimiento.
    END IF;

    -- Insertamos el registro del movimiento en la tabla Movimiento
    INSERT INTO Movimiento (
        id_mov,
        id_prod,
        cantidad_prod,
        fecha_mov,
        precio_unitario_prod,
        tipo_mov
    ) VALUES (
        seq_movimiento_id.NEXTVAL, -- id del movimiento usando la secuencia
        :NEW.id_prod,              -- id del producto
        v_cantidad_mov,            -- cantidad movida
        SYSDATE,                  -- fecha del movimiento
        :NEW.precio_prod,          -- precio unitario actual
        v_tipo_mov                 -- tipo de movimiento
    );
END;
```

## 4.4.5 Funciones

### 4.4.5.1 Productos

Unset

```

CREATE OR REPLACE FUNCTION producto_existe(id_producto PRODUCTO.ID_PROD%TYPE) RETURN BOOLEAN IS
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count
    FROM Producto
```

```

WHERE id_prod = id_producto;

IF v_count > 0 THEN
    RETURN TRUE;
ELSE
    RETURN FALSE;
END IF;
END;

-----

CREATE OR REPLACE FUNCTION producto_existe_nombre (
    nombre_producto IN Producto.nombre_prod%TYPE
) RETURN BOOLEAN
IS
    v_count NUMBER;
BEGIN
    -- Contar cuantos productos existen con el mismo nombre (independiente de mayusculas/minusculas)
    SELECT COUNT(*) INTO v_count
    FROM Producto
    WHERE UPPER(nombre_prod) = UPPER(nombre_producto);

    -- Si el conteo es mayor que 0, significa que el producto ya existe
    RETURN v_count > 0;
END;

-----

CREATE OR REPLACE FUNCTION id_categoria_por_nombre (
    nombre_categoria IN Categoria_Producto.nombre_categoria%TYPE
) RETURN Categoria_Producto.id_categoria%TYPE
IS
    v_id_categoria Categoria_Producto.id_categoria%TYPE;
BEGIN
    -- Buscar el id de la categoría basada en el nombre

```

```

SELECT id_categoria INTO v_id_categoria
FROM Categoria_Producto
WHERE nombre_categoria = id_categoria_por_nombre.nombre_categoria;
RETURN v_id_categoria;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        -- Si no se encuentra, lanzar una excepción
        RAISE_APPLICATION_ERROR(-20001, 'Categoría no encontrada: ' || nombre_categoria);
END;

-----

CREATE OR REPLACE FUNCTION id_producto_por_nombre (
    nombre_producto IN Producto.nombre_prod%TYPE
) RETURN Producto.id_prod%TYPE
IS
    v_id_prod Producto.id_prod%TYPE;
BEGIN
    -- Buscar el id del producto basado en el nombre
    SELECT id_prod INTO v_id_prod
    FROM Producto
    WHERE nombre_prod = nombre_producto;

    -- Si se encuentra el producto, retornar el id
    RETURN v_id_prod;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        -- Si no se encuentra, lanzar una excepción
        RAISE_APPLICATION_ERROR(-20001, 'Producto no encontrado: ' || nombre_producto);
END;

```

#### 4.4.5.2 Proveedores

Unset

```
CREATE OR REPLACE FUNCTION id_proveedor_por_nombre (
    nombre_proveedor IN Proveedor.nombre_prov%TYPE
) RETURN Proveedor.id_prov%TYPE
IS
    v_id_prov Proveedor.id_prov%TYPE;
BEGIN
    -- Buscar el id del proveedor basado en el nombre
    SELECT id_prov INTO v_id_prov
    FROM Proveedor
    WHERE nombre_prov = id_proveedor_por_nombre.nombre_proveedor;

    -- Si se encuentra el proveedor, retornar el id
    RETURN v_id_prov;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        -- Si no se encuentra, lanzar una excepción
        RAISE_APPLICATION_ERROR(-20001, 'Proveedor no encontrado: ' || nombre_proveedor);
END;

-----

create or replace FUNCTION proveedor_existe_nombre (
    nombre_proveedor IN Proveedor.nombre_prov%TYPE
) RETURN BOOLEAN
IS
    v_count NUMBER;
BEGIN
    -- Contar cuantos proveedores existen con el mismo nombre
    SELECT COUNT(*) INTO v_count
    FROM Proveedor
    WHERE UPPER(nombre_prov) = UPPER(nombre_proveedor);
```



```

-- Si el conteo es mayor que 0, significa que el proveedor ya existe
IF v_count > 0 THEN
    RETURN TRUE;
ELSE
    RETURN FALSE;
END IF;
END proveedor_existe_nombre;

-----

CREATE OR REPLACE FUNCTION proveedor_tiene_productos (
    nombre_proveedor IN Proveedor.nombre_prov%TYPE
) RETURN BOOLEAN
IS
    v_id_prov    Proveedor.id_prov%TYPE;
    v_count      NUMBER;
BEGIN
    -- Obtener el id del proveedor usando la función id_proveedor_por_nombre
    v_id_prov := id_proveedor_por_nombre(nombre_proveedor);

    -- Contar productos activos asociados a este proveedor
    SELECT COUNT(*) INTO v_count
    FROM Producto
    WHERE id_prov = v_id_prov
        AND estado_prod = 'ACTIVO';

    -- Retornar true si el proveedor tiene productos activos asociados
    RETURN v_count > 0;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE; -- Si el proveedor no existe, retorna FALSE
END;

```

## 4.4.6 Procedimientos

### 4.4.6.1 Productos

Unset

```
CREATE OR REPLACE PROCEDURE obtener_cantidad_producto (
    p_nombre_prod IN Producto.nombre_prod%TYPE,
    p_cantidad_out OUT NUMBER
) AS
BEGIN
    -- Buscar la cantidad del producto basado en el nombre
    SELECT cantidad_prod INTO p_cantidad_out
    FROM Producto
    WHERE nombre_prod = p_nombre_prod;

    -- Si no se encuentra el producto, lanzar una excepción
    IF p_cantidad_out IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'Producto no encontrado: ' || p_nombre_prod);
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        -- Manejo de excepción si no se encuentra el producto
        RAISE_APPLICATION_ERROR(-20001, 'Producto no encontrado: ' || p_nombre_prod);
    WHEN OTHERS THEN
        -- Manejo de excepciones en caso de otros errores
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END obtener_cantidad_producto;

-----

CREATE OR REPLACE PROCEDURE eliminar_cantidad_procedimiento (
    p_nombre_prod IN Producto.nombre_prod%TYPE,
    p_cantidad_eliminar NUMBER
) AS
```

```

v_id_prod Producto.id_prod%TYPE; -- Variable para almacenar el id del producto
v_cantidad_actual NUMBER;
BEGIN
    -- Obtener el id del producto a partir del nombre
    v_id_prod := id_producto_por_nombre(p_nombre_prod);

    -- Verificar si el producto existe
    IF producto_existe(v_id_prod) THEN

        SELECT cantidad_prod INTO v_cantidad_actual
        FROM Producto
        WHERE id_prod = v_id_prod;

        IF p_cantidad_eliminar <= v_cantidad_actual THEN
            UPDATE Producto
            SET cantidad_prod = cantidad_prod - p_cantidad_eliminar
            WHERE id_prod = v_id_prod;

            DBMS_OUTPUT.PUT_LINE('Cantidad eliminada exitosamente.');
        ELSE
            DBMS_OUTPUT.PUT_LINE('Error: No se puede eliminar más cantidad de la que existe.');
        END IF;
    ELSE
        DBMS_OUTPUT.PUT_LINE('Error: El producto no existe.');
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END eliminar_cantidad_procedimiento;
-----

CREATE OR REPLACE PROCEDURE agregar_cantidad_procedimiento (

```

```

        p_nombre_prod IN Producto.nombre_prod%TYPE,
        p_cantidad_agregar NUMBER
    ) AS
        v_id_prod Producto.id_prod%TYPE; -- Variable para almacenar el id del producto
BEGIN
    -- Obtener el id del producto a partir del nombre
    v_id_prod := id_producto_por_nombre(p_nombre_prod);

    -- Verificar si el producto existe
    IF producto_existe(v_id_prod) THEN
        UPDATE Producto
        SET cantidad_prod = cantidad_prod + p_cantidad_agregar
        WHERE id_prod = v_id_prod;

        DBMS_OUTPUT.PUT_LINE('Cantidad actualizada exitosamente.');
```

---

```

    ELSE
        RAISE_APPLICATION_ERROR(-20001, 'El producto no existe.');
```

---

```

    END IF;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END agregar_cantidad_procedimiento;

-----

CREATE OR REPLACE PROCEDURE eliminar_producto_proc (
    nombre_producto IN Producto.nombre_prod%TYPE,
    p_bandera OUT NUMBER
)
IS
    v_cantidad_prod NUMBER;
    v_estado_prod Producto.estado_prod%TYPE;
BEGIN

```

```

-- Verificar si el producto existe
IF NOT producto_existe_nombre(nombre_producto) THEN

    DBMS_OUTPUT.PUT_LINE('Error: El producto no existe.');
```

p\_bandera := 2; -- Bandera 2 indica que el producto no existe

```

    RETURN;
END IF;

-- Obtener la cantidad del producto utilizando el procedimiento obtener_cantidad_producto
obtener_cantidad_producto(nombre_producto, v_cantidad_prod);

-- Verificar si la cantidad del producto es mayor que 0
IF v_cantidad_prod > 0 THEN

    DBMS_OUTPUT.PUT_LINE('Error: No se puede desactivar el producto porque tiene una cantidad
mayor a 0.');
```

p\_bandera := 1; -- Bandera 1 indica que el producto tiene cantidad mayor que 0

```

    RETURN;
END IF;

-- Verificar si el estado ya es 'INACTIVO'
SELECT estado_prod INTO v_estado_prod
FROM Producto
WHERE nombre_prod = nombre_producto;

IF v_estado_prod = 'INACTIVO' THEN

    DBMS_OUTPUT.PUT_LINE('El producto ya est❖ inactivo.');
```

p\_bandera := 4; -- Bandera 4 indica que el producto ya estaba inactivo

```

    RETURN;
END IF;

-- Actualizar el estado del producto a 'INACTIVO'
UPDATE Producto
SET estado_prod = 'INACTIVO'
```

```

WHERE nombre_prod = nombre_producto;

DBMS_OUTPUT.PUT_LINE('Producto desactivado exitosamente.');
```

p\_bandera := 0; -- Bandera 0 indica que el producto fue desactivado

```

EXCEPTION

    WHEN OTHERS THEN

        DBMS_OUTPUT.PUT_LINE('Error al desactivar el producto: ' || SQLERRM);

        p_bandera := 3; -- Bandera 3 indica que ocurrió un error diferente

END eliminar_producto_proc;
```

```

-----

CREATE OR REPLACE PROCEDURE agregar_producto_proc (

    nombre_producto    IN Producto.nombre_prod%TYPE,

    nombre_categoria    IN Categoria_Producto.nombre_categoria%TYPE,

    precio_producto     IN Producto.precio_prod%TYPE,

    nombre_proveedor    IN Proveedor.nombre_prov%TYPE,

    p_bandera           OUT NUMBER

)

IS

    v_id_categoria Categoria_Producto.id_categoria%TYPE;

    v_id_prov Proveedor.id_prov%TYPE;

    v_estado_producto Producto.estado_prod%TYPE;

BEGIN

    -- Verificar si el producto ya existe

    SELECT estado_prod

    INTO v_estado_producto

    FROM Producto

    WHERE UPPER(nombre_prod) = UPPER(nombre_producto)

    FOR UPDATE;

    -- Si el producto existe y su estado es INACTIVO, modificarlo

    IF v_estado_producto = 'INACTIVO' THEN
```

```

-- Obtener el id de la categoría a partir del nombre
v_id_categoria := id_categoria_por_nombre(nombre_categoria);

-- Obtener el id del proveedor a partir del nombre
v_id_prov := id_proveedor_por_nombre(nombre_proveedor);

-- Modificar el producto existente
UPDATE Producto
SET id_categoria = v_id_categoria,
    precio_prod = precio_producto,
    cantidad_prod = 0,
    estado_prod = 'ACTIVO',
    id_prov = v_id_prov
WHERE UPPER(nombre_prod) = UPPER(nombre_producto);

DBMS_OUTPUT.PUT_LINE('Producto actualizado y activado.');
```

p\_bandera := 0; -- Bandera 0 indica que el producto fue actualizado y activado

```

RETURN;
END IF;

-- Si el producto ya existe y está activo, devolver error
DBMS_OUTPUT.PUT_LINE('Error: Ya existe un producto con el mismo nombre y está activo.');
```

p\_bandera := 1; -- Bandera 1 indica que el producto ya existe y está activo

```

RETURN;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        -- Si el producto no existe, intentar insertarlo
        BEGIN
            -- Obtener el id de la categoría a partir del nombre
            v_id_categoria := id_categoria_por_nombre(nombre_categoria);
```

```

-- Obtener el id del proveedor a partir del nombre
v_id_prov := id_proveedor_por_nombre(nombre_proveedor);

-- Insertar el nuevo producto
INSERT INTO Producto (nombre_prod, id_categoria, precio_prod, cantidad_prod, estado_prod,
id_prov)
VALUES (nombre_producto, v_id_categoria, precio_producto, 0, 'ACTIVO', v_id_prov);

DBMS_OUTPUT.PUT_LINE('Producto insertado exitosamente.');
```

```

p_bandera := 0; -- Bandera 0 indica que el producto se insertó correctamente
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error al insertar el producto: ' || SQLERRM);
        p_bandera := 2; -- Bandera 2 indica que ocurrió un error diferente
END;
```

```

    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error al verificar el producto: ' || SQLERRM);
        p_bandera := 2; -- Bandera 2 indica que ocurrió un error diferente
END agregar_producto_proc;
```

---

```

CREATE OR REPLACE PROCEDURE modificar_producto_proc (
    nombre_actual IN Producto.nombre_prod%TYPE,
    nuevo_nombre IN Producto.nombre_prod%TYPE,
    nueva_categoria IN Categoria_Producto.NOMBRE_CATEGORIA%TYPE,
    nuevo_proveedor IN Proveedor.NOMBRE_PROV%TYPE,
    nuevo_precio IN Producto.precio_prod%TYPE,
    p_bandera OUT NUMBER
)
IS
```



```

v_id_categoria Producto.id_categoria%TYPE;
v_id_proveedor Producto.id_prov%TYPE;
BEGIN
    -- Verificar si el producto a modificar existe
    IF NOT producto_existe_nombre(nombre_actual) THEN
        DBMS_OUTPUT.PUT_LINE('Error: El producto a modificar no existe.');
```

p\_bandera := 2; -- Bandera 2 por error

```

    RETURN;
END IF;

    -- Verificar si el nuevo nombre ya existe (y no es el mismo producto)
    IF producto_existe_nombre(nuevo_nombre) AND nombre_actual != nuevo_nombre THEN
        DBMS_OUTPUT.PUT_LINE('Error: El nuevo nombre del producto ya está en uso.');
```

p\_bandera := 1; -- Bandera 1 por nombre duplicado

```

    RETURN;
END IF;

    -- Obtener el ID de la nueva categoría y el nuevo proveedor
    v_id_categoria := id_categoria_por_nombre(nueva_categoria);
    v_id_proveedor := id_proveedor_por_nombre(nuevo_proveedor);

    -- Actualizar el producto
    UPDATE Producto
    SET nombre_prod = nuevo_nombre,
        id_categoria = v_id_categoria,
        id_prov = v_id_proveedor,
        precio_prod = nuevo_precio
    WHERE nombre_prod = nombre_actual;

    DBMS_OUTPUT.PUT_LINE('Producto actualizado correctamente.');
```

p\_bandera := 0; -- Bandera 0 por éxito

```

EXCEPTION

    WHEN OTHERS THEN

        DBMS_OUTPUT.PUT_LINE('Error al actualizar el producto: ' || SQLERRM);

        p_bandera := 2; -- Bandera 2 por error

END modificar_producto_proc;

```

```

-----

CREATE OR REPLACE PROCEDURE obtener_detalle_producto (

    p_nombre_producto IN Producto.nombre_prod%TYPE,

    p_precio           OUT Producto.precio_prod%TYPE,

    p_nombre_proveedor OUT Proveedor.nombre_prov%TYPE,

    p_nombre_categoria OUT Categoria_Producto.nombre_categoria%TYPE

)

IS

BEGIN

    -- Intentar obtener los detalles del producto

    SELECT

        p.precio_prod,

        prov.nombre_prov,

        cat.nombre_categoria

    INTO

        p_precio,

        p_nombre_proveedor,

        p_nombre_categoria

    FROM

        Producto p

    JOIN

        Proveedor prov ON p.id_prov = prov.id_prov

    JOIN

        Categoria_Producto cat ON p.id_categoria = cat.id_categoria

    WHERE

        p.nombre_prod = p_nombre_producto AND p.estado_prod = 'ACTIVO';

```

```

EXCEPTION

    WHEN NO_DATA_FOUND THEN

        -- Manejar el caso en que el producto no exista o esté inactivo

        p_precio := -1;

        p_nombre_proveedor := NULL;

        p_nombre_categoria := NULL;

        DBMS_OUTPUT.PUT_LINE('El producto no existe o está inactivo.');
```

```

    WHEN OTHERS THEN

        -- Manejar cualquier otro error

        DBMS_OUTPUT.PUT_LINE('Error al obtener los detalles del producto: ' || SQLERRM);

        p_precio := -2;

        p_nombre_proveedor := NULL;

        p_nombre_categoria := NULL;

END obtener_detalle_producto;
```

#### 4.4.6.2 Proveedores

Unset

```

CREATE OR REPLACE PROCEDURE eliminar_proveedor (

    p_nombre_proveedor IN Proveedor.nombre_prov%TYPE,

    p_resultado          OUT NUMBER

)

IS

BEGIN

    -- Verificar si el proveedor existe

    IF NOT proveedor_existe_nombre(p_nombre_proveedor) THEN

        p_resultado := 2; -- Proveedor no encontrado

        RETURN;

    END IF;
```

```

-- Verificar si el proveedor tiene productos asociados
IF proveedor_tiene_productos(p_nombre_proveedor) THEN
    p_resultado := 1; -- No se puede desactivar porque tiene productos asociados
    RETURN;
END IF;

-- Cambiar el estado del proveedor a INACTIVO
UPDATE Proveedor
SET estado_prov = 'INACTIVO'
WHERE UPPER(nombre_prov) = UPPER(p_nombre_proveedor);

p_resultado := 0; -- Desactivación exitosa

EXCEPTION
    WHEN OTHERS THEN
        p_resultado := 3; -- Otro tipo de error
END eliminar_proveedor;

-----

CREATE OR REPLACE PROCEDURE agregar_proveedor (
    p_nombre      IN Proveedor.nombre_prov%TYPE,
    p_telefono    IN Proveedor.telefono_prov%TYPE,
    p_correo      IN Proveedor.correo_prov%TYPE,
    p_direccion   IN Proveedor.direccion_prov%TYPE,
    p_resultado   OUT NUMBER
)
IS
    v_estado_prov Proveedor.estado_prov%TYPE;
BEGIN
    -- Verificar si el proveedor ya existe
    IF proveedor_existe_nombre(p_nombre) THEN

```

```

-- Verificar el estado del proveedor
SELECT estado_prov INTO v_estado_prov
FROM Proveedor
WHERE nombre_prov = p_nombre;

IF v_estado_prov = 'INACTIVO' THEN
    -- Modificar el proveedor con el nuevo estado 'ACTIVO'
    UPDATE Proveedor
    SET
        telefono_prov = p_telefono,
        correo_prov = p_correo,
        direccion_prov = p_direccion,
        estado_prov = 'ACTIVO'
    WHERE nombre_prov = p_nombre;
    p_resultado := 0; -- Proveedor modificado correctamente
ELSE
    p_resultado := 1; -- Proveedor ya activo
END IF;
ELSE
    -- Intentar insertar el nuevo proveedor
    BEGIN
        INSERT INTO Proveedor (
            nombre_prov,
            telefono_prov,
            correo_prov,
            direccion_prov,
            estado_prov
        ) VALUES (
            p_nombre,
            p_telefono,
            p_correo,

```

```

        p_direccion,
        'ACTIVO' -- Estado por defecto
    );

    p_resultado := 0; -- Proveedor agregado correctamente
EXCEPTION
    WHEN OTHERS THEN
        p_resultado := 2; -- Otro tipo de error
END;
END IF;
END;
-----

CREATE OR REPLACE PROCEDURE modificar_proveedor_proc (
    p_nombre_actual    IN Proveedor.nombre_prov%TYPE,
    p_nuevo_nombre     IN Proveedor.nombre_prov%TYPE,
    p_nuevo_telefono   IN Proveedor.telefono_prov%TYPE,
    p_nuevo_correo     IN Proveedor.correo_prov%TYPE,
    p_nueva_direccion  IN Proveedor.direccion_prov%TYPE,
    p_resultado        OUT NUMBER
)
IS
BEGIN
    -- Verificar si el proveedor actual existe
    IF NOT proveedor_existe_nombre(p_nombre_actual) THEN
        p_resultado := 2; -- El proveedor a modificar no existe
        RETURN;
    END IF;

    -- Verificar si el nuevo nombre ya está en uso por otro proveedor
    IF UPPER(p_nuevo_nombre) <> UPPER(p_nombre_actual)
        AND proveedor_existe_nombre(p_nuevo_nombre) THEN
        p_resultado := 1; -- El nuevo nombre ya está en uso
    
```

```

        RETURN;
    END IF;

    -- Actualizar los datos del proveedor
    UPDATE Proveedor
    SET nombre_prov = p_nuevo_nombre,
        telefono_prov = p_nuevo_telefono,
        correo_prov = p_nuevo_correo,
        direccion_prov = p_nueva_direccion
    WHERE UPPER(nombre_prov) = UPPER(p_nombre_actual);

    -- Confirmar la actualización
    IF SQL%ROWCOUNT > 0 THEN
        p_resultado := 0; -- Actualización exitosa
    ELSE
        p_resultado := 2; -- Error inesperado (no se modificó ninguna fila)
    END IF;

EXCEPTION
    WHEN OTHERS THEN
        p_resultado := 2; -- Otro tipo de error
END modificar_proveedor_proc;

-----

CREATE OR REPLACE PROCEDURE obtener_detalle_proveedor (
    p_nombre_proveedor IN Proveedor.nombre_prov%TYPE,
    p_telefono          OUT Proveedor.telefono_prov%TYPE,
    p_correo            OUT Proveedor.correo_prov%TYPE,
    p_direccion         OUT Proveedor.direccion_prov%TYPE
)
IS
BEGIN

```

```

-- Intentar obtener los detalles del proveedor
SELECT
    telefono_prov,
    correo_prov,
    direccion_prov
INTO
    p_telefono,
    p_correo,
    p_direccion
FROM
    Proveedor
WHERE
    nombre_prov = p_nombre_proveedor
    AND estado_prov = 'ACTIVO';

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        -- Manejar el caso en que el proveedor no exista o esté inactivo
        p_telefono := NULL;
        p_correo := NULL;
        p_direccion := NULL;
        DBMS_OUTPUT.PUT_LINE('El proveedor no existe o está inactivo.');
```

```

    WHEN OTHERS THEN
        -- Manejar cualquier otro error
        DBMS_OUTPUT.PUT_LINE('Error al obtener los detalles del proveedor: ' || SQLERRM);
        p_telefono := NULL;
        p_correo := NULL;
        p_direccion := NULL;
END obtener_detalle_proveedor;
```



#### 4.4.6.3 Usuarios

Unset

```
CREATE OR REPLACE PROCEDURE verificar_credenciales(
    p_usuario IN VARCHAR2,
    p_contrasena IN VARCHAR2,
    p_resultado OUT NUMBER
) IS
    v_contrasena USUARIO.contrasena%TYPE;
BEGIN
    -- Verificar si el usuario existe
    SELECT contrasena
    INTO v_contrasena
    FROM USUARIO
    WHERE usuario = p_usuario;

    -- Verificar si la contraseña es correcta
    IF v_contrasena = p_contrasena THEN
        p_resultado := 0; -- Usuario y contraseña correctos
    ELSE
        p_resultado := 1; -- Contraseña incorrecta
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        p_resultado := 2; -- El usuario no existe
    WHEN OTHERS THEN
        p_resultado := 3; -- Otro tipo de error
END verificar_credenciales;

-----

CREATE OR REPLACE PROCEDURE obtener_nombre_usuario(
    p_usuario IN VARCHAR2,
    p_nom_usuario OUT VARCHAR2
```

```

) AS
BEGIN
    -- Intenta recuperar el nombre del usuario basado en el campo usuario
    SELECT nom_Usuario
    INTO p_nom_usuario
    FROM USUARIO
    WHERE usuario = p_usuario;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        -- Si no encuentra el usuario, devuelve NULL en el parámetro de salida
        p_nom_usuario := NULL;
    WHEN OTHERS THEN
        -- Manejo de otros errores inesperados
        p_nom_usuario := NULL;
        RAISE;
END obtener_nombre_usuario;

```

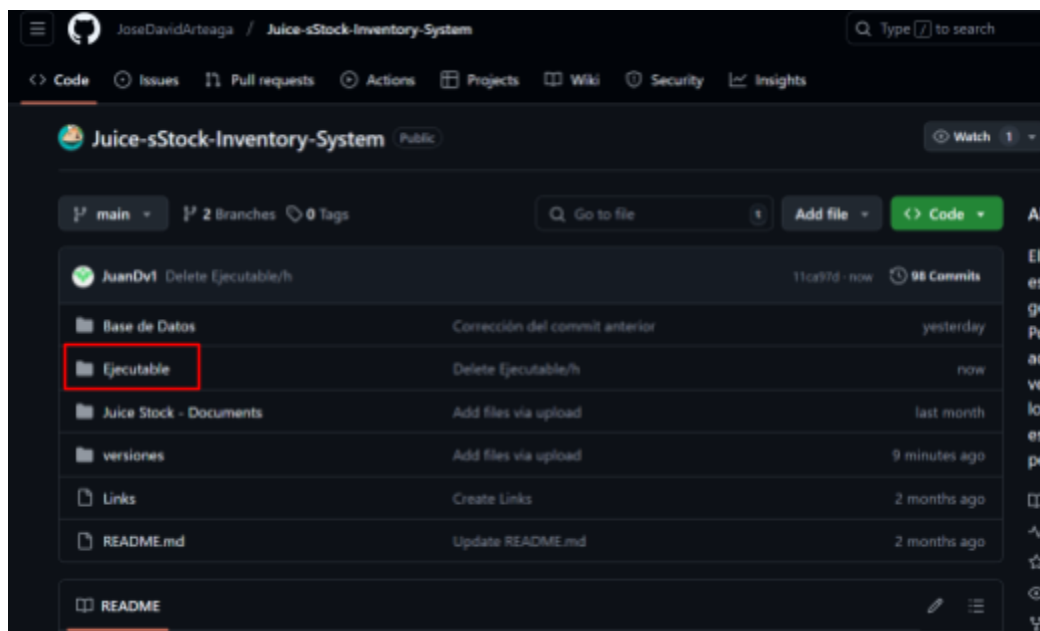
Una vez ejecutadas todas estas operaciones la base de datos estará lista para ejecutarse localmente.

## 5. Descarga Aplicación Ejecutable

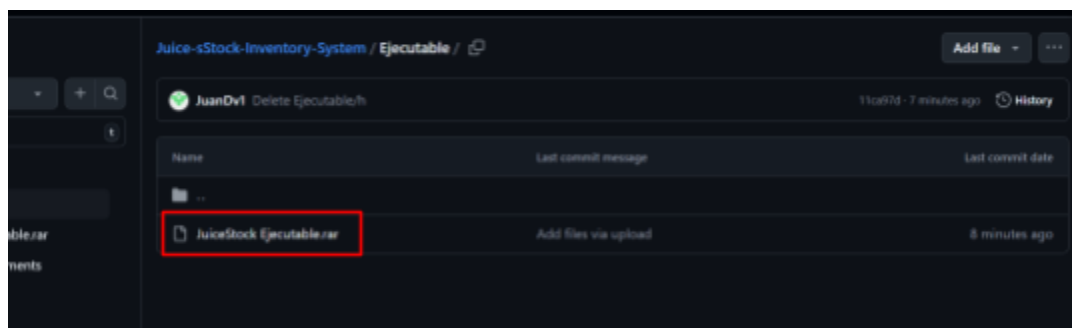
Finalmente solo nos queda descargar la carpeta que contiene nuestra aplicación, para eso debemos ir al repositorio de GitHub del proyecto.

<https://github.com/JoseDavidArteaga/Juice-sStock-Inventory-System/tree/main>

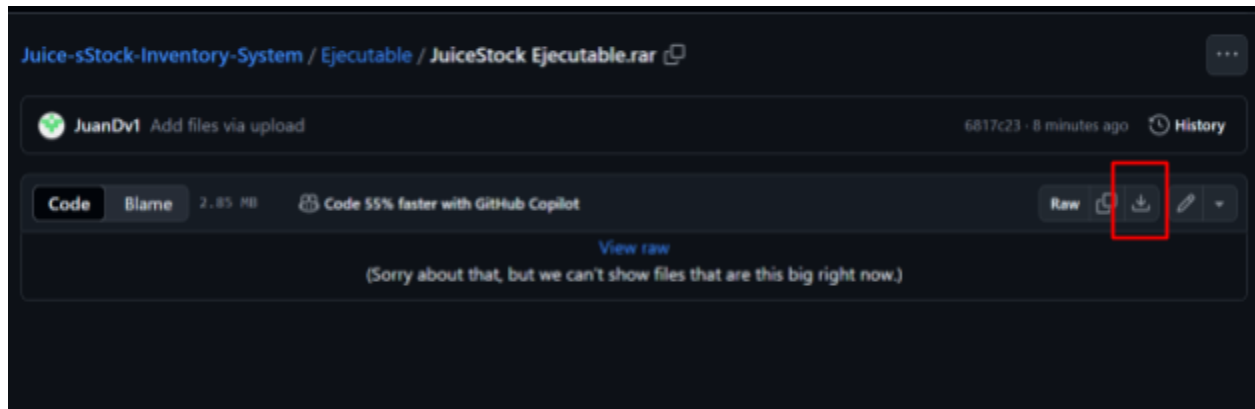
Seleccionamos la carpeta “Ejecutable”



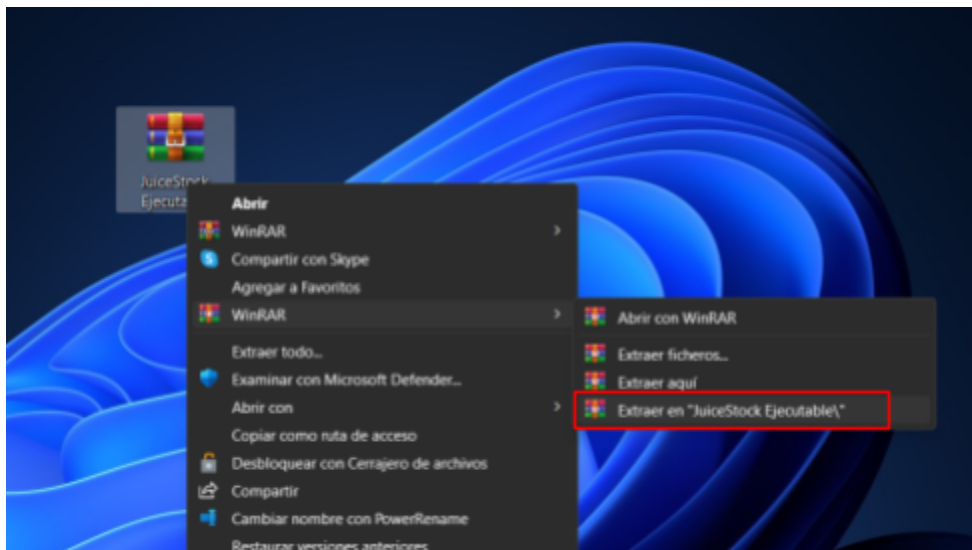
Posteriormente seleccionamos la carpeta comprimida llamada “JuiceStock Ejecutable.rar”



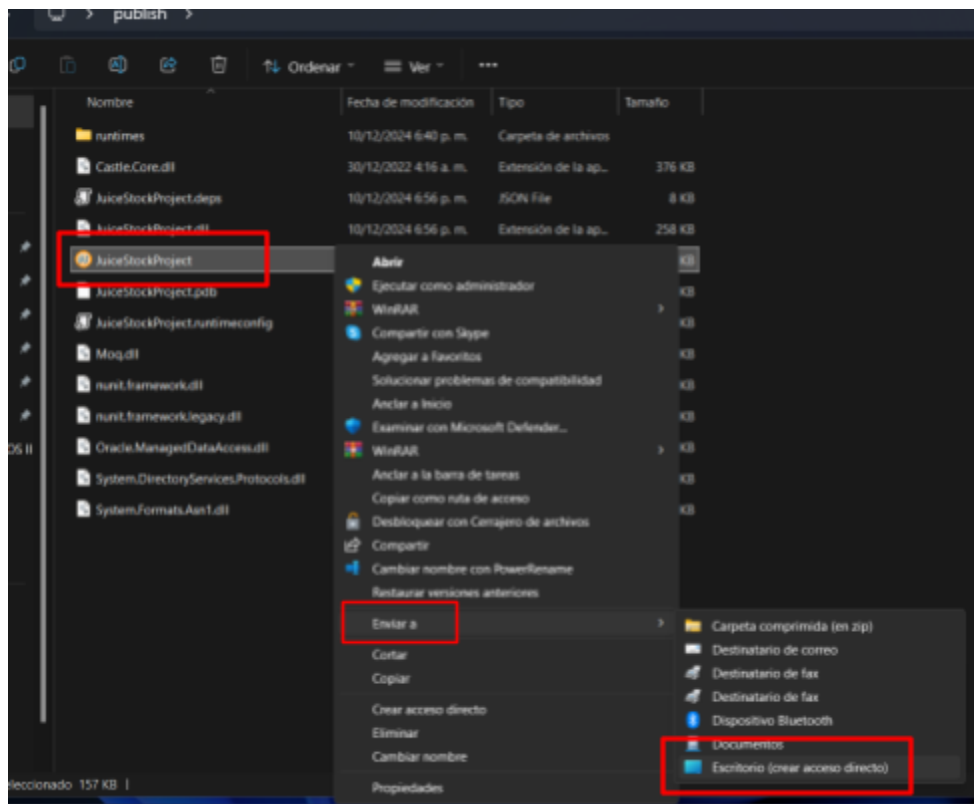
La descargaremos dando click en el botón: 



Seleccionamos la ruta de descarga y descomprimos la carpeta seleccionando la opción de la imagen:



Abrimos la carpeta descomprimida y creamos un acceso directo de la aplicación:



Listo, ya tendremos un acceso directo de la aplicación lista para usar y gestionar tu inventario con JuiceStock

