"Año De La Recuperación Y Consolidación De La Economía Peruana"

UNIVERSIDAD TECNOLÓGICA DEL PERÚ

Base de Datos II

PRACTICA CALIFICADA 2

TIENDA MAX CON ENFOQUE EN BASES DE DATOS RELACIONALES

Integrantes:

- Arenas Moran, Jaime Alejandro
- Berrospi Reategui, José Pablo
- Principe Merino, Sheila Jacqueline
- Valenzuela Cisneros, Kreisy
- Velarde Ochoa Obed, Víctor Jesús
- Zelada Magallanes, Pedro Luis

Docente:

Ayra Pinto, Ronald Ángel

Lima – Perú, 2025

Índice

1. DESARROLLO

1.1) Creación de Tablas

Se definieron las siguientes tablas para la base de datos tienda_max:

- productos
- clientes
- ventas
- proveedores

Cada tabla incluye claves primarias, restricciones NOT NULL y claves foráneas que aseguran la integridad referencial.

```sql
CREATE TABLE productos (
  id_producto SERIAL PRIMARY KEY,
  nombre_producto VARCHAR(100) NOT NULL,
  precio_unitario NUMERIC(10,2) NOT NULL,
  stock INT NOT NULL
);

CREATE TABLE clientes (
  dni VARCHAR(15) PRIMARY KEY,
  nombres VARCHAR(100) NOT NULL,
  apellido_paterno VARCHAR(50) NOT NULL,
  apellido_materno VARCHAR(50) NOT NULL
);

CREATE TABLE ventas (
  id_venta SERIAL PRIMARY KEY,
  dni_cliente VARCHAR(15) REFERENCES clientes(dni),
  id_producto INT REFERENCES productos(id_producto),
  cantidad INT NOT NULL,
  precio_total NUMERIC(10,2) NOT NULL,
  fecha_venta DATE DEFAULT CURRENT_DATE
);

CREATE TABLE proveedores (
  id_proveedor SERIAL PRIMARY KEY,
  nombre_proveedor VARCHAR(100) NOT NULL,
  id_producto INT REFERENCES productos(id_producto),
  fecha_embarque DATE NOT NULL,
  cantidad INT NOT NULL
);
```

```sql
33  CREATE TABLE proveedores (
34    id_proveedor SERIAL PRIMARY KEY,
35    nombre_proveedor VARCHAR(100) NOT NULL,
36    id_producto INT REFERENCES productos(id_producto),
37    fecha_embarque DATE NOT NULL,
38    cantidad INT NOT NULL
39  );
40
41  -- ============================================
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 38 msec.
```

1.2) Inserción de Datos

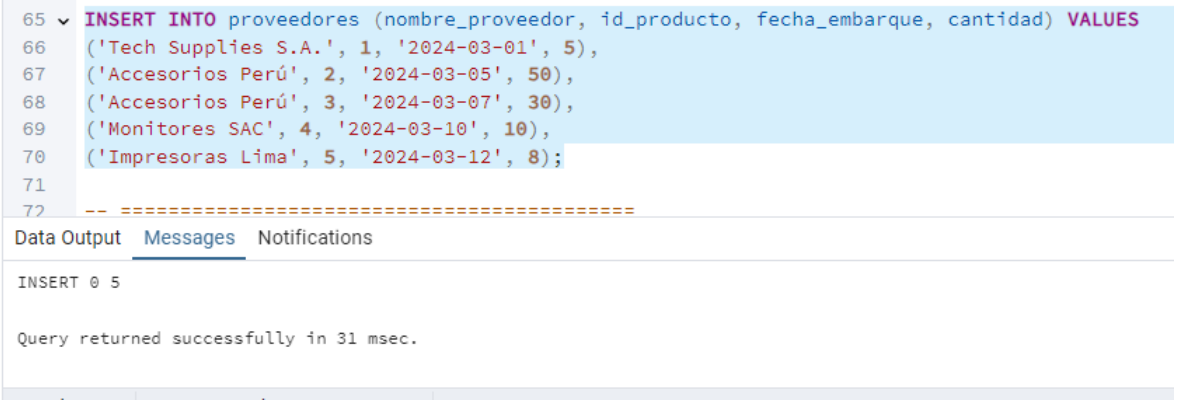
Se insertaron 5 registros en cada tabla para poblar la base de datos con datos de prueba y permitir las consultas y pruebas posteriores.

```sql
INSERT INTO productos (nombre_producto, precio_unitario, stock) VALUES
('Laptop Lenovo', 2500.00, 10),
('Mouse Inalámbrico', 50.00, 100),
('Teclado Mecánico', 120.00, 50),
('Monitor Samsung', 800.00, 20),
('Impresora HP', 400.00, 15);

INSERT INTO clientes (dni, nombres, apellido_paterno, apellido_materno) VALUES
('12345678', 'Juan', 'Pérez', 'López'),
('87654321', 'María', 'Gómez', 'Ramírez'),
('13579246', 'Carlos', 'Sánchez', 'Torres'),
('24681357', 'Lucía', 'Martínez', 'Castro'),
('11223344', 'Ana', 'Rojas', 'Quispe');

INSERT INTO ventas (dni_cliente, id_producto, cantidad, precio_total, fecha_venta) VALUES
('12345678', 1, 1, 2500.00, '2024-05-01'),
('87654321', 2, 2, 100.00, '2024-05-02'),
('12345678', 3, 1, 120.00, '2024-05-03'),
('13579246', 4, 1, 800.00, '2024-05-04'),
('24681357', 5, 1, 400.00, '2024-05-05');

INSERT INTO proveedores (nombre_proveedor, id_producto, fecha_embarque, cantidad) VALUES
('Tech Supplies S.A.', 1, '2024-03-01', 5),
('Accesorios Perú', 2, '2024-03-05', 50),
('Accesorios Perú', 3, '2024-03-07', 30),
('Monitores SAC', 4, '2024-03-10', 10),
('Impresoras Lima', 5, '2024-03-12', 8);
```



1.3) Consultas SQL

Se realizaron consultas usando:

- Inner Join Y Left Join para relacionar ventas con clientes y productos.

```sql
SELECT
    v.id_venta,
    c.nombres || ' ' || c.apellido_paterno AS nombre_cliente,
    p.nombre_producto,
    v.cantidad,
    v.precio_total,
    v.fecha_venta
FROM ventas v
JOIN clientes c ON v.dni_cliente = c.dni
JOIN productos p ON v.id_producto = p.id_producto;
```

```sql
SELECT
    v.id_venta,
    c.nombres || ' ' || c.apellido_paterno AS nombre_cliente,
    p.nombre_producto,
    v.cantidad,
    v.precio_total,
    v.fecha_venta
FROM ventas v
LEFT JOIN clientes c ON v.dni_cliente = c.dni
LEFT JOIN productos p ON v.id_producto = p.id_producto;
```

Data Output | Messages | Notifications

	id_venta integer	nombre_cliente text	nombre_producto character varying (100)	cantidad integer	precio_total numeric (10,2)	fecha_venta date
1	1	Juan Pérez	Laptop Lenovo	1	2500.00	2024-05-01
2	2	María Gómez	Mouse Inalámbrico	2	100.00	2024-05-02
3	3	Juan Pérez	Teclado Mecánico	1	120.00	2024-05-03
4	4	Carlos Sánchez	Monitor Samsung	1	800.00	2024-05-04
5	5	Lucía Martínez	Impresora HP	1	400.00	2024-05-05

- Subconsultas para filtrar productos por precio promedio y clientes con alto gasto.

```sql
SELECT nombre_producto, precio_unitario
FROM productos
WHERE precio_unitario > (
    SELECT AVG(precio_unitario) FROM productos
);
```

Data Output | Messages | Notifications

	nombre_producto character varying (100)	precio_unitario numeric (10,2)
1	Laptop Lenovo	2500.00
2	Monitor Samsung	800.00

```sql
SELECT nombres, apellido_paterno
FROM clientes
WHERE dni IN (
    SELECT dni_cliente
    FROM ventas
    GROUP BY dni_cliente
    HAVING SUM(precio_total) > 100
);
```

Data Output | Messages | Notifications

	nombres character varying (100)	apellido_paterno character varying (50)
1	Juan	Pérez
2	Carlos	Sánchez
3	Lucía	Martínez

- Funciones agregadas como SUM, AVG, MAX, MIN para obtener estadísticas.

Ejemplos:

- Total gastado por cada cliente.

```
SELECT
    c.nombres || ' ' || c.apellido_paterno AS cliente,
    SUM(v.precio_total) AS total_gastado
FROM ventas v
JOIN clientes c ON v.dni_cliente = c.dni
GROUP BY c.dni;
```

Data Output | Messages | Notificatior

	cliente text	total_gastado numeric
1	Carlos Sánch...	800.00
2	Lucía Martínez	400.00
3	Juan Pérez	2620.00
4	María Gómez	100.00

- Productos más costosos y baratos.

```
SELECT MAX(precio_unitario) AS precio_maximo, MIN(precio_unitario) AS precio_minimo FROM productos;
```

Data Output | Messages | Notifications

	precio_maximo numeric	precio_minimo numeric
1	2500.00	50.00

- Promedio de stock en inventario.

```
SELECT AVG(stock) AS promedio_stock FROM productos;
```

Data Output | Messages | N

	promedio_stock numeric
1	39.0000000000000000

1.4) Procedimientos almacenados (PL/pgSQL)

Se implementaron dos procedimientos:

1. **actualizacion_inventario:** Incrementa el stock o crea un nuevo producto si no existe.

```
CREATE OR REPLACE PROCEDURE actualizacion_inventario(
    p_nombre_producto VARCHAR,
    p_stock INTEGER,
    p_precio NUMERIC(10,2)
)
LANGUAGE plpgsql
AS $$
DECLARE
    existencia_fila INTEGER;
BEGIN
    UPDATE productos
    SET stock = stock + p_stock
    WHERE nombre_producto = p_nombre_producto;
    GET DIAGNOSTICS existencia_fila = ROW_COUNT;
    IF existencia_fila = 0 THEN
        INSERT INTO productos (nombre_producto, precio_unitario, stock)
        VALUES (p_nombre_producto, p_precio, p_stock);
        RAISE NOTICE 'Producto % no existía. Se agregó con precio S/%.2f y stock %.', p_nombre_producto, p_precio, p_stock;
    ELSE
        RAISE NOTICE 'Inventario actualizado con éxito para %.', p_nombre_producto;
    END IF;
END;
$$;
```

```
136 CREATE OR REPLACE PROCEDURE actualizacion_inventario(
137     p_nombre_producto VARCHAR,
138     p_stock INTEGER,
139     p_precio NUMERIC(10,2)
140 )
141 LANGUAGE plpgsql
142 AS $$
143 DECLARE
144     existencia_fila INTEGER;
145 BEGIN
146     UPDATE productos
147     SET stock = stock + p_stock
148     WHERE nombre_producto = p_nombre_producto;
149     GET DIAGNOSTICS existencia_fila = ROW_COUNT;
150     IF existencia_fila = 0 THEN
151         INSERT INTO productos (nombre_producto, precio_unitario, stock)
152         VALUES (p_nombre_producto, p_precio, p_stock);
153         RAISE NOTICE 'Producto % no existía. Se agregó con precio S/%.2f y stock %.', p_nombre_producto, p_precio, p_stock;
154     ELSE
155         RAISE NOTICE 'Inventario actualizado con éxito para %.', p_nombre_producto;
156     END IF;
157 END;
158 $$;
159
160 -- b) Procedimiento para insertar una venta y actualizar stock
161 CREATE OR REPLACE PROCEDURE insertar_venta_inteligente(
```

Data Output | Messages | Notifications

```
CREATE PROCEDURE

Query returned successfully in 49 msec.
```

2. **insertar_venta_inteligente**: Inserta una venta y actualiza stock, permitiendo también registrar un nuevo cliente si aún no existe.

```sql
CREATE OR REPLACE PROCEDURE insertar_venta_inteligente(
    p_dni_cliente VARCHAR,
    p_id_producto INT,
    p_cantidad INT,
    p_precio_total NUMERIC(10,2),
    p_confirmar_registro_cliente BOOLEAN,
    p_nombres VARCHAR DEFAULT NULL,
    p_apellido_paterno VARCHAR DEFAULT NULL,
    p_apellido_materno VARCHAR DEFAULT NULL,
    p_fecha_venta DATE DEFAULT CURRENT_DATE
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_cliente_existe BOOLEAN;
BEGIN
    SELECT EXISTS (SELECT 1 FROM clientes WHERE dni = p_dni_cliente) INTO v_cliente_existe;
    IF NOT v_cliente_existe THEN
        IF p_confirmar_registro_cliente THEN
            INSERT INTO clientes (dni, nombres, apellido_paterno, apellido_materno)
            VALUES (p_dni_cliente, p_nombres, p_apellido_paterno, p_apellido_materno);
            RAISE NOTICE 'Cliente registrado correctamente con DNI %', p_dni_cliente;
        ELSE
            RAISE NOTICE 'Cliente no registrado. La venta se asociará sin cliente.';
            p_dni_cliente := NULL;
        END IF;
    END IF;
    INSERT INTO ventas (dni_cliente, id_producto, cantidad, precio_total, fecha_venta)
    VALUES (p_dni_cliente, p_id_producto, p_cantidad, p_precio_total, p_fecha_venta);
    UPDATE productos
    SET stock = stock - p_cantidad
    WHERE id_producto = p_id_producto;
    RAISE NOTICE 'Venta registrada correctamente.';
END;
$$;
```

```sql
CREATE OR REPLACE PROCEDURE insertar_venta_inteligente(
    p_dni_cliente VARCHAR,
    p_id_producto INT,
    p_cantidad INT,
    p_precio_total NUMERIC(10,2),
    p_confirmar_registro_cliente BOOLEAN,
    p_nombres VARCHAR DEFAULT NULL,
    p_apellido_paterno VARCHAR DEFAULT NULL,
    p_apellido_materno VARCHAR DEFAULT NULL,
    p_fecha_venta DATE DEFAULT CURRENT_DATE
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_cliente_existe BOOLEAN;
BEGIN
    SELECT EXISTS (SELECT 1 FROM clientes WHERE dni = p_dni_cliente) INTO v_cliente_existe;
    IF NOT v_cliente_existe THEN
        IF p_confirmar_registro_cliente THEN
            INSERT INTO clientes (dni, nombres, apellido_paterno, apellido_materno)
            VALUES (p_dni_cliente, p_nombres, p_apellido_paterno, p_apellido_materno);
            RAISE NOTICE 'Cliente registrado correctamente con DNI %', p_dni_cliente;
        ELSE
            RAISE NOTICE 'Cliente no registrado. La venta se asociará sin cliente.';
            p_dni_cliente := NULL;
        END IF;
    END IF;
    INSERT INTO ventas (dni_cliente, id_producto, cantidad, precio_total, fecha_venta)
    VALUES (p_dni_cliente, p_id_producto, p_cantidad, p_precio_total, p_fecha_venta);
    UPDATE productos
    SET stock = stock - p_cantidad
    WHERE id_producto = p_id_producto;
    RAISE NOTICE 'Venta registrada correctamente.';
END;
$$;
```

Data Output | Messages | Notifications

```
CREATE PROCEDURE

Query returned successfully in 35 msec.
```

Total rows: | Query complete 00:00:00.035

1.5) Gestión de roles/usuarios

Se crearon roles específicos para dividir responsabilidades:

- rol_supremo2: Acceso total.
- rol_ventas2: Permisos para registrar ventas.
- rol_inventario2: Permisos para actualizar productos.
- rol_lectura2: Solo lectura de datos.

```
-- Revocar permisos públicos
REVOKE ALL ON DATABASE tienda_max FROM PUBLIC;
REVOKE ALL ON SCHEMA public FROM PUBLIC;
REVOKE ALL ON ALL TABLES IN SCHEMA public FROM PUBLIC;

-- Crear roles
CREATE ROLE rol_supremo2;
CREATE ROLE rol_lectura2;
CREATE ROLE rol_ventas2;
CREATE ROLE rol_inventario2;

-- Asignar permisos a roles
GRANT SELECT, INSERT ON ventas TO rol_ventas2;
GRANT SELECT, UPDATE ON productos TO rol_inventario2;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO rol_supremo2;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA public TO rol_supremo2;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO rol_lectura2;
```

```
213  GRANT SELECT, INSERT ON ventas TO rol_ventas2;
214  GRANT SELECT, UPDATE ON productos TO rol_inventario2;
215  GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO rol_supremo2;
216  GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA public TO rol_supremo2;
217  GRANT SELECT ON ALL TABLES IN SCHEMA public TO rol_lectura2;
218
```

Data Output | Messages | Notifications

```
GRANT

Query returned successfully in 36 msec.
```

Usuarios creados:

- el_hechicero_supremo2
- jesus_ventas2
- josue_inventario2
- usuario_lectura2

Cada usuario fue asignado a su rol respectivo.

```
CREATE USER el_hechicero_supremo2 WITH PASSWORD 'Lucas';
CREATE USER usuario_lectura2 WITH PASSWORD 'lector';
CREATE USER jesus_ventas2 WITH PASSWORD 'freestyle';
CREATE USER josue_inventario2 WITH PASSWORD 'billar';

GRANT rol_supremo2 TO el_hechicero_supremo2;
GRANT rol_lectura2 TO usuario_lectura2;
GRANT rol_ventas2 TO jesus_ventas2;
GRANT rol_inventario2 TO josue_inventario2;
```

```
220  CREATE USER el_hechicero_supremo2 WITH PASSWORD 'Lucas';
221  CREATE USER usuario_lectura2 WITH PASSWORD 'lector';
222  CREATE USER jesus_ventas2 WITH PASSWORD 'freestyle';
223  CREATE USER josue_inventario2 WITH PASSWORD 'billar';
224
225  GRANT rol_supremo2 TO el_hechicero_supremo2;
226  GRANT rol_lectura2 TO usuario_lectura2;
227  GRANT rol_ventas2 TO jesus_ventas2;
228  GRANT rol_inventario2 TO josue_inventario2;
229
```

Data Output | Messages | Notifications

```
GRANT ROLE

Query returned successfully in 89 msec.
```

2. CONCLUSIONES

En el desenvolvimiento del proyecto de la base de datos tienda_max se han sabido aplicar principios básicos de bases de datos en la creación de tablas, inserción de datos, utilización de claves primarias y foráneas, consultas SQL con joins, subconsultas y funciones agregadas. Por otra parte, se ha explorado la creación de procedimientos almacenados, lo que ha permitido automatizar procesos como el control de inventario y el registro de ventas.

También se ha logrado aprender a gestionar usuarios y roles en PostgreSQL, así como a controlar los privilegios de acceso en función del tipo de usuario, algo clave para entornos reales.

Dificultades enfrentadas:

- Se presentaron errores al crear roles que ya existían previamente en el sistema, lo cual obligó a renombrarlos.
- Fue necesario ejecutar los bloques de código en orden correcto, ya que había dependencias entre las tablas y datos.
- Se afianzó el uso correcto del entorno de trabajo (como pgAdmin), asegurando la conexión con la base de datos antes de ejecutar sentencias.

3. REFERENCIAS

-Oracle Academy. (2021). *Fundamentos de bases de datos relacionales* [Curso en línea]. https://academy.oracle.com/es/solutions-curriculum.html

-Parzibyte. (2021, 13 de julio). *Curso de PostgreSQL desde cero – Bases de datos con PostgreSQL en español* [Video]. YouTube. https://youtu.be/L-nJgWVCJMg

-Silberschatz, A., Korth, H. F., & Sudarshan, S. (2013). *Fundamentos de bases de datos* (6.ªed.).McGraw-Hill. https://archive.org/details/fundamentosdebasesdedatos5a.ed.abrahamsilberschatzhenryf.korths.sudarshan/page/n1/mode/1up