Universidad Politécnica de San Luis Potosí

Materia:

Bases de datos

Docente:
Jesús Revilla

Proyecto Sistema de Optimización de Corte de Materia Prima en PostgreSQL- Manual Técnico

Integrantes del equipo:

Castillo Silva Ángel	179752
Guerrero Guevara Yazmin	182483
López González Cruz Ángel	183213
Martínez Huerta Patricia Michelle	181730
Pulido Alvarado Ximena	179800
Sánchez Saucedo Rafael	182712
Vidales Ramírez Luis David	183060

Fecha de entrega:

24 de noviembre de 2025

Contenido

Introducción

Este documento técnico detalla la arquitectura de la base de datos PostgreSQL diseñada para el proyecto de "Optimización de Corte de Materia Prima". El sistema no solo almacena datos, sino que contiene lógica de negocio compleja encapsulada en la base de datos para garantizar la integridad, el cálculo de eficiencia y la trazabilidad de eventos.

El núcleo del sistema permite gestionar láminas (materia prima), definir productos compuestos por piezas y, lo más importante, manipular geométricamente estas piezas (rotación y posición) utilizando tipos de datos JSON para flexibilidad.

Entorno y Despliegue (CI/CD)

El proyecto está configurado para integración continua mediante GitHub Actions. Esto asegura que cada cambio en el código sea probado automáticamente en un entorno limpio.

Configuración del entorno**:** El archivo de flujo de trabajo levanta un servicio de PostgreSQL y ejecuta los scripts SQL en un orden específico para evitar errores de dependencias (primero roles, luego tablas, luego funciones).

Archivo: .github/workflows/postgresql_workflow.yml

```
# Esta sección define cómo se levanta la base de datos para pruebas
services:
  postgres:
    image: postgres:14
    env:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: proyecto
```

Arquitectura de Datos y Tablas

El modelo relacional está diseñado para separar la definición de los objetos de su geometría física.

Entidades Principales:

1. **materia_prima**: Representa el stock disponible (hojas de madera, metal, etc.).
2. **piezas y productos**: Una relación de uno a muchos. Un producto tiene muchas piezas.
3. **geometrias**: Aquí reside la innovación del diseño. En lugar de crear columnas rígidas para coordenadas, se utiliza un campo JSON llamado datos. Esto permite almacenar atributos dinámicos (x, y, ángulo, vértices) que pueden evolucionar sin cambiar el esquema de la tabla.

Archivo: sql/01_tablas.sql

```
CREATE TABLE geometrias (
    id SERIAL PRIMARY KEY,
    id_pieza INT REFERENCES piezas(id),
    datos JSON NOT NULL  -- Uso de JSON para flexibilidad en coordenadas
);
```

Lógica de Negocio (Procedimientos Almacenados)

Motor de Movimiento (Manipulación JSON)

El procedimiento sp_rotar_posicionar_figuras es el componente más crítico. Realiza tres acciones en una sola transacción:

1. Recibe las nuevas coordenadas y ángulo.
2. Utiliza la función jsonb_set anidada para actualizar solo los campos específicos (angulo, x, y) dentro del objeto JSON, preservando cualquier otro dato que exista.
3. Invoca a una función auxiliar para registrar este movimiento en la bitácora de eventos.

Archivo: sql/07_procedimientos.sql

```
CREATE OR REPLACE PROCEDURE sp_rotar_posicionar_figuras(
    _id_pieza INT,
    _angulo NUMERIC,
    _x NUMERIC,
    _y NUMERIC,
    _evento JSON
)
AS $$
BEGIN
    -- Actualización anidada del objeto JSON
    UPDATE geometrias
    SET datos = jsonb_set(
        jsonb_set(
            jsonb_set(datos::jsonb,'{angulo}',to_jsonb(_angulo)), -- Actualiza ángulo
        '{x}',to_jsonb(_x)), -- Actualiza X
    '{y}',to_jsonb(_y)) -- Actualiza Y
    WHERE id_pieza=_id_pieza;

    -- Registro automático en bitácora
    PERFORM fn_registrar_evento(_id_pieza,_evento);
END;
$$ LANGUAGE plpgsql;
```

Cálculo de Eficiencia

Para determinar si el corte es óptimo, el sistema calcula el porcentaje de utilización con la fórmula:

$$\frac{\sum(\text{área de todas las piezas})}{\text{área de la materia prima}}$$

Archivo: sql/08_funciones.sql

```
CREATE OR REPLACE FUNCTION fn_calcular_utilizacion(_id_mp INT)
RETURNS NUMERIC AS $$
DECLARE
    area_mp NUMERIC;
    area_total_piezas NUMERIC;
BEGIN
    -- Obtiene el área total de la hoja base
    SELECT ancho * alto INTO area_mp FROM materia_prima WHERE id=_id_mp;

    -- Suma el área de todas las piezas registradas
    SELECT COALESCE(SUM(area),0)
    INTO area_total_piezas
    FROM piezas;

    RETURN area_total_piezas / area_mp;
END;
$$ LANGUAGE plpgsql;
```

Seguridad y Control de Calidad

Validaciones Automáticas (Triggers)

Para evitar errores humanos, como ingresar dimensiones negativas, se utiliza un Trigger. Este se dispara *antes* (BEFORE INSERT) de que el dato toque la tabla.

Archivo: sql/09_triggers.sql

```
CREATE OR REPLACE FUNCTION fn_validar_distancias()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.distancia_entre_piezas < 0 THEN
        RAISE EXCEPTION 'Distancia inválida'; -- Detiene la inserción si falla
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```