CÓDIGO FUENTE DE LA BASE DE DATOS

* Script de creación de base de datos

CREATE DATABASE metal\_music\_store

    WITH ENCODING = 'UTF8';

--TUNNING DE LA BASE DE DATOS (metal\_music\_store)

-- 1. CREACION DE ROLES

-- Crear un rol/usuario dedicado (con contraseña segura)

CREATE ROLE metal\_user WITH LOGIN PASSWORD 'U1eYPplMQa2ksaaM1YyT';

GRANT CONNECT ON DATABASE metal\_music\_store TO metal\_user;

-- =============================================

-- IMPLEMENTACIÓN DE EXTENSIONES

-- =============================================

-- Instalación de PostGIS: Geolocalización de tiendas

CREATE EXTENSION postgis;

-- Instalación de pg\_stat\_statements: Monitoreo de consultas

CREATE EXTENSION pg\_stat\_statements;

-- Instalación de pgaudit: Auditoría de seguridad

CREATE EXTENSION pgaudit;

-- Instalación de pg\_trgm: Búsqueda por similaridad

CREATE EXTENSION pg\_trgm;

-- Instalación de uuid-ossp: Generación de UUIDs

CREATE EXTENSION uuid\_ossp;

-- EXTENSIONES DE SEGURIDAD

-- pgcrypto: Para funciones criptográficas

/\*

  - Encriptación de contraseñas

\*/

CREATE EXTENSION IF NOT EXISTS pgcrypto;

-- pgjwt: Para manejo de JWT

CREATE EXTENSION IF NOT EXISTS pgjwt;

-- 3. CONFIGURACIONES DE SEGURIDAD

-- Forzar conexiones SSL

ALTER ROLE metal\_user SET ssl = 'require';

-- Enmascaramiento de datos

SECURITY LABEL FOR anon ON COLUMN usuarios.contrasena\_hash

IS 'MASKED WITH VALUE NULL';

-- 4. AMPLIACION DE ESCALABILIDAD

-- Particionamiento de registros de auditoría

CREATE TABLE auditoria\_accesos (

    id SERIAL PRIMARY KEY,

    usuario VARCHAR(100),

    accion VARCHAR(50),

    tabla\_afectada VARCHAR(100),

    datos\_anteriores JSONB,

    datos\_nuevos JSONB,

    fecha\_accion TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

) PARTITION BY RANGE (fecha\_accion);

-- 5. CONFIGURACIONES DE MONITOREO

-- Habilitar estadísticas avanzadas

ALTER SYSTEM SET track\_io\_timing = on;

ALTER SYSTEM SET track\_functions = all;

-- Configurar PgBouncer para manejar conexiones concurrentes

[databases]

metal\_music\_store = host =localhost port=5432 dbname=metal\_music\_store

[pgbouncer]

pool\_mode = transaction

max\_client\_conn = 1000

default\_pool\_size = 50

-- Asignar permisos básicos al usuario

GRANT USAGE ON SCHEMA public TO metal\_user;

-- Configuración de seguridad adicional

ALTER SYSTEM SET session\_preload\_libraries = 'pg\_stat\_statements,auto\_explain';

ALTER SYSTEM SET shared\_preload\_libraries = 'pg\_stat\_statements,pg\_cron,pgaudit';

* Script de tablas:

-- ROLES

-- Tabla de Roles (Control de acceso)

CREATE TABLE roles (

  id\_rol SERIAL PRIMARY KEY,

  nombre\_rol VARCHAR(20) NOT NULL UNIQUE,

  descripcion TEXT,

  permisos JSONB NOT NULL DEFAULT '{}'::jsonb

);

-- USUARIOS Y UBICACIONES

-- Tabla de Ubicaciones con JSONB

CREATE TABLE ubicaciones (

  id\_ubicacion SERIAL PRIMARY KEY,

  direccion JSONB NOT NULL CHECK (

    jsonb\_typeof(direccion) = 'object' AND

    direccion ? 'linea1' AND

    direccion ? 'distrito' AND

    direccion ? 'provincia' AND

    direccion ? 'departamento' AND

    direccion ? 'codigo\_postal' AND

    direccion ? 'pais'

  ),

  coordenadas GEOMETRY(POINT, 4326)

);

/\*

formato JSONB de la direccion:

{

  "linea1": "Av. Javier Prado 123",

  "linea2": "Oficina 501", (opcional)

  "distrito": "San Isidro",

  "provincia": "Lima",

  "departamento": "Lima",

  "codigo\_postal": "15076",

  "pais": "Peru",

  "referencia": "Frente al parque" (opcional)

}

\*/

-- Tabla de Usuarios

CREATE TABLE usuarios (

  id\_usuario UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

  id\_secuencial SERIAL UNIQUE,

  nombre\_usuario VARCHAR(50) UNIQUE NOT NULL,

  correo\_electronico VARCHAR(255) UNIQUE NOT NULL CHECK (

    correo\_electronico ~\* '^[A-Za-z0-9.\_%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$'

  ),

  contrasena\_hash VARCHAR(255) NOT NULL,

  nombres VARCHAR(100) NOT NULL,

  apellidos VARCHAR(100) NOT NULL,

  telefono VARCHAR(20) CHECK (telefono ~ '^[0-9]{9}$'),

  id\_ubicacion\_principal INTEGER REFERENCES ubicaciones(id\_ubicacion),

  direcciones\_adicionales JSONB NOT NULL DEFAULT '[]'::jsonb CHECK (

    jsonb\_typeof(direcciones\_adicionales) = 'array'

  ),

  id\_rol INTEGER NOT NULL REFERENCES roles(id\_rol),

  esta\_activo BOOLEAN NOT NULL DEFAULT TRUE

);

-- PRODUCTOS

-- Tabla de Categorías de Producto

CREATE TABLE categorias (

  id\_categoria SERIAL PRIMARY KEY,

  nombre VARCHAR(100) NOT NULL,

  descripcion TEXT,

  id\_categoria\_padre INTEGER REFERENCES categorias(id\_categoria),

  esta\_activo BOOLEAN NOT NULL DEFAULT TRUE,

  CONSTRAINT nombre\_unico\_por\_nivel UNIQUE (nombre, id\_categoria\_padre)

);

-- Tabla de Detalles Técnicos de Productos

CREATE TABLE detalles\_productos (

  id\_detalle\_producto SERIAL PRIMARY KEY,

  tipo\_producto VARCHAR(50) NOT NULL,

  especificaciones JSONB NOT NULL DEFAULT '{}'::jsonb,

  peso DECIMAL(10, 2) CHECK (peso > 0), -- en kg

  dimensiones DECIMAL(10, 2)[3] CHECK (array\_length(dimensiones, 1) = 3), -- [x, y, z]

  material\_principal VARCHAR(100),

  materiales\_secundarios JSONB NOT NULL DEFAULT '[]'::jsonb CHECK (

    jsonb\_typeof(materiales\_secundarios) = 'array'

  ),

  cuidados\_especiales JSONB NOT NULL DEFAULT '{}'::jsonb CHECK (

    jsonb\_typeof(cuidados\_especiales) = 'object'

  )

);

/\*

formato JSONB de los materiales secundarios:

[

  {"material": "Algodón", "porcentaje": 80},

  {"material": "Poliester", "porcentaje": 20}

]

formato JSONB de los cuidados especiales:

{

  "lavado": "Lavar a máquina con agua fría",

  "secado": "Secar a la sombra",

  "planchado": "Planchar a baja temperatura"

}

\*/

-- Tabla de Productos (Optimizada con referencia a detalles técnicos)

CREATE TABLE productos (

  id\_producto UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

  id\_secuencial SERIAL UNIQUE,

  codigo\_sku VARCHAR(50) UNIQUE NOT NULL,

  nombre VARCHAR(255) NOT NULL,

  descripcion TEXT,

  precio DECIMAL(10, 2) NOT NULL CHECK (precio > 0),

  costo DECIMAL(10, 2) CHECK (costo > 0),

  cantidad\_disponible INTEGER NOT NULL CHECK (cantidad\_disponible >= 0) DEFAULT 0,

  id\_categoria INTEGER NOT NULL REFERENCES categorias(id\_categoria),

  id\_detalle\_producto INTEGER REFERENCES detalles\_productos(id\_detalle\_producto),

  marca VARCHAR(100),

  talla VARCHAR(50),

  esta\_activo BOOLEAN NOT NULL DEFAULT TRUE,

  atributos\_generales JSONB NOT NULL DEFAULT '{}'::jsonb CHECK (

    jsonb\_typeof(atributos\_generales) = 'object'

  )

);

/\*

formato JSONB de los atributos generales:

{

  "color": "Azul marino",

  "temporada": "Verano 2023",

  "garantia": "6 meses"

}\*/

-- Tabla de Imágenes de Productos

CREATE TABLE imagenes\_productos (

  id\_imagen SERIAL PRIMARY KEY,

  id\_producto UUID NOT NULL REFERENCES productos(id\_producto) ON DELETE CASCADE,

  url\_imagen VARCHAR(255) NOT NULL,

  texto\_alternativo VARCHAR(255),

  es\_principal BOOLEAN NOT NULL DEFAULT FALSE,

  orden\_visualizacion INTEGER NOT NULL DEFAULT 0

);

-- LISTAS DE DESEOS

-- Tabla de Listas de Deseos

CREATE TABLE listas\_deseos (

  id\_lista\_deseos UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

  id\_secuencial SERIAL UNIQUE,

  id\_usuario UUID NOT NULL REFERENCES usuarios(id\_usuario) ON DELETE CASCADE,

  nombre VARCHAR(100) NOT NULL,

  es\_privada BOOLEAN NOT NULL DEFAULT TRUE,

  CONSTRAINT lista\_unica\_por\_usuario UNIQUE (id\_usuario, nombre)

);

-- Tabla de Items en Listas de Deseos

CREATE TABLE items\_lista\_deseos (

  id\_item\_lista\_deseos SERIAL PRIMARY KEY,

  id\_lista\_deseos UUID NOT NULL REFERENCES listas\_deseos(id\_lista\_deseos) ON DELETE CASCADE,

  id\_producto UUID NOT NULL REFERENCES productos(id\_producto) ON DELETE CASCADE,

  CONSTRAINT producto\_unico\_por\_lista UNIQUE (id\_lista\_deseos, id\_producto)

);

-- CARRITO DE COMPRAS --

-- Tabla de Carritos de Compras

CREATE TABLE carritos\_compras (

  id\_carrito UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

  id\_secuencial SERIAL UNIQUE,

  id\_usuario UUID NOT NULL REFERENCES usuarios(id\_usuario) ON DELETE CASCADE,

  CONSTRAINT carrito\_unico\_por\_usuario UNIQUE (id\_usuario)

);

-- Tabla de Items en Carritos

CREATE TABLE items\_carrito (

  id\_item\_carrito SERIAL PRIMARY KEY,

  id\_carrito UUID NOT NULL REFERENCES carritos\_compras(id\_carrito) ON DELETE CASCADE,

  id\_producto UUID NOT NULL REFERENCES productos(id\_producto),

  cantidad INTEGER NOT NULL CHECK (cantidad > 0),

  CONSTRAINT producto\_unico\_por\_carrito UNIQUE (id\_carrito, id\_producto)

);

-- PEDIDOS --

-- Tabla de Estados de Pedido

CREATE TABLE estados\_pedido (

  id\_estado SERIAL PRIMARY KEY,

  codigo\_estado VARCHAR(20) UNIQUE NOT NULL,

  nombre\_estado VARCHAR(50) NOT NULL,

  descripcion TEXT,

  es\_final BOOLEAN NOT NULL DEFAULT FALSE,

  orden\_flujo INTEGER NOT NULL

);

-- Tabla de Métodos de Pago

CREATE TABLE metodos\_pago (

  id\_metodo\_pago SERIAL PRIMARY KEY,

  nombre VARCHAR(50) NOT NULL,

  descripcion TEXT,

  requiere\_confirmacion BOOLEAN NOT NULL DEFAULT FALSE,

  esta\_activo BOOLEAN NOT NULL DEFAULT TRUE,

  configuracion JSONB NOT NULL DEFAULT '{}'::jsonb CHECK (

    jsonb\_typeof(configuracion) = 'object'

  )

);

-- Tabla de Pedidos

CREATE TABLE pedidos (

  id\_pedido UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

  id\_usuario UUID NOT NULL REFERENCES usuarios(id\_usuario),

  numero\_pedido VARCHAR(50) UNIQUE NOT NULL,

  id\_estado INTEGER NOT NULL REFERENCES estados\_pedido(id\_estado),

  subtotal DECIMAL(10, 2) NOT NULL CHECK (subtotal > 0),

  impuestos DECIMAL(10, 2) NOT NULL DEFAULT 0,

  costo\_envio DECIMAL(10, 2) NOT NULL DEFAULT 0,

  total DECIMAL(10, 2) NOT NULL CHECK (total > 0),

  id\_metodo\_pago INTEGER REFERENCES metodos\_pago(id\_metodo\_pago),

  estado\_pago BOOLEAN NOT NULL DEFAULT FALSE,

  direccion JSONB NOT NULL CHECK (

    jsonb\_typeof(direccion) = 'object' AND

    direccion ? 'linea1' AND

    direccion ? 'distrito' AND

    direccion ? 'provincia' AND

    direccion ? 'departamento'

  ),

  informacion\_remitente JSONB NOT NULL DEFAULT '{}'::jsonb CHECK (

    jsonb\_typeof(informacion\_remitente) = 'object'

  ),

  coordenadas GEOMETRY(POINT, 4326),

  notas TEXT

);

-- Tabla de Items de Pedido

CREATE TABLE items\_pedido (

  id\_item\_pedido SERIAL PRIMARY KEY,

  id\_pedido UUID NOT NULL REFERENCES pedidos(id\_pedido) ON DELETE CASCADE,

  id\_producto UUID NOT NULL REFERENCES productos(id\_producto),

  cantidad INTEGER NOT NULL CHECK (cantidad > 0),

  precio\_unitario DECIMAL(10, 2) NOT NULL CHECK (precio\_unitario > 0),

  precio\_total DECIMAL(10, 2) NOT NULL CHECK (precio\_total > 0),

  descuento DECIMAL(10, 2) NOT NULL DEFAULT 0,

  impuestos DECIMAL(10, 2) NOT NULL DEFAULT 0

);

-- Tabla de Reseñas de Productos

CREATE TABLE resenas\_productos (

  id\_resena SERIAL PRIMARY KEY,

  id\_producto UUID NOT NULL REFERENCES productos(id\_producto) ON DELETE CASCADE,

  id\_usuario UUID NOT NULL REFERENCES usuarios(id\_usuario),

  calificacion INTEGER NOT NULL CHECK (calificacion BETWEEN 1 AND 5),

  comentario TEXT,

  esta\_aprobado BOOLEAN NOT NULL DEFAULT FALSE,

  CONSTRAINT reseña\_unica\_por\_producto\_usuario UNIQUE (id\_producto, id\_usuario)

);

-- =============================================

-- TABLAS DE AUDITORÍA E HISTORIALES

-- =============================================

-- Tabla de Registro de Inventario

CREATE TABLE registro\_inventario (

  id\_registro SERIAL PRIMARY KEY,

  id\_producto UUID NOT NULL REFERENCES productos(id\_producto) ON DELETE CASCADE,

  cambio\_cantidad INTEGER NOT NULL,

  nueva\_cantidad INTEGER NOT NULL,

  tipo\_cambio VARCHAR(20) NOT NULL CHECK (

    tipo\_cambio IN ('compra', 'venta', 'ajuste', 'perdida', 'devolucion')

  ),

  id\_referencia VARCHAR(50), -- Puede ser ID de pedido, factura, etc.

  tipo\_referencia VARCHAR(50), -- Tipo de referencia (pedido, ajuste, etc.)

  notas TEXT,

  modificado\_por UUID REFERENCES usuarios(id\_usuario),

  fecha\_cambio TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT\_TIMESTAMP

);

-- Tabla de Historial de Estados de Pedido

CREATE TABLE historial\_estados\_pedido (

  id\_historial SERIAL PRIMARY KEY,

  id\_pedido UUID NOT NULL REFERENCES pedidos(id\_pedido) ON DELETE CASCADE,

  id\_estado INTEGER NOT NULL REFERENCES estados\_pedido(id\_estado),

  id\_usuario UUID REFERENCES usuarios(id\_usuario),

  notas TEXT,

  fecha\_cambio TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT\_TIMESTAMP

);

-- Tabla de Historial de Precios

CREATE TABLE historial\_precios (

  id\_historial\_precio SERIAL PRIMARY KEY,

  id\_producto UUID NOT NULL REFERENCES productos(id\_producto) ON DELETE CASCADE,

  precio\_anterior DECIMAL(10, 2) NOT NULL,

  precio\_nuevo DECIMAL(10, 2) NOT NULL,

  modificado\_por UUID REFERENCES usuarios(id\_usuario),

  razon\_cambio TEXT,

  fecha\_cambio TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT\_TIMESTAMP

);

-- =============================================

-- TRIGGERS PARA AUDITORÍA

-- =============================================

-- Trigger para registrar cambios en productos

CREATE OR REPLACE FUNCTION registrar\_cambio\_producto()

RETURNS TRIGGER AS $$

BEGIN

  IF OLD.precio <> NEW.precio THEN

    INSERT INTO historial\_precios (

      id\_producto, precio\_anterior, precio\_nuevo, modificado\_por, razon\_cambio

    ) VALUES (

      NEW.id\_producto, OLD.precio, NEW.precio, current\_setting('app.current\_user\_id', TRUE)::UUID, 'Actualización de precio'

    );

  END IF;

  IF OLD.cantidad\_disponible <> NEW.cantidad\_disponible THEN

    INSERT INTO registro\_inventario (

      id\_producto, cambio\_cantidad, nueva\_cantidad, tipo\_cambio, modificado\_por

    ) VALUES (

      NEW.id\_producto,

      NEW.cantidad\_disponible - OLD.cantidad\_disponible,

      NEW.cantidad\_disponible,

      CASE

        WHEN NEW.cantidad\_disponible > OLD.cantidad\_disponible THEN 'compra'

        ELSE 'venta'

      END,

      current\_setting('app.current\_user\_id', TRUE)::UUID

    );

  END IF;

  RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger\_auditoria\_producto

AFTER UPDATE ON productos

FOR EACH ROW

EXECUTE FUNCTION registrar\_cambio\_producto();

-- Trigger para registrar cambios de estado en pedidos

CREATE OR REPLACE FUNCTION registrar\_cambio\_estado\_pedido()

RETURNS TRIGGER AS $$

BEGIN

  IF OLD.id\_estado <> NEW.id\_estado THEN

    INSERT INTO historial\_estados\_pedido (

      id\_pedido, id\_estado, id\_usuario, notas

    ) VALUES (

      NEW.id\_pedido, NEW.id\_estado, current\_setting('app.current\_user\_id', TRUE)::UUID, 'Cambio de estado'

    );

  END IF;

  RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger\_auditoria\_estado\_pedido

AFTER UPDATE ON pedidos

FOR EACH ROW

EXECUTE FUNCTION registrar\_cambio\_estado\_pedido();

-- Trigger para registrar creación de pedidos

CREATE OR REPLACE FUNCTION registrar\_nuevo\_pedido()

RETURNS TRIGGER AS $$

BEGIN

  INSERT INTO historial\_estados\_pedido (

    id\_pedido, id\_estado, id\_usuario, notas

  ) VALUES (

    NEW.id\_pedido, NEW.id\_estado, NEW.id\_usuario, 'Pedido creado'

  );

  RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger\_auditoria\_nuevo\_pedido

AFTER INSERT ON pedidos

FOR EACH ROW

EXECUTE FUNCTION registrar\_nuevo\_pedido();

* Scritp de complementación

-- INDICES

-- Índices para búsquedas frecuentes

CREATE INDEX idx\_productos\_categoria ON productos(id\_categoria);

CREATE INDEX idx\_productos\_marca ON productos(marca);

CREATE INDEX idx\_productos\_precio ON productos(precio);

CREATE INDEX idx\_productos\_activos ON productos(esta\_activo) WHERE esta\_activo = TRUE;

-- Índices para consultas de usuarios

CREATE INDEX idx\_usuarios\_correo ON usuarios(correo\_electronico);

CREATE INDEX idx\_usuarios\_rol ON usuarios(id\_rol);

-- Índices para pedidos

CREATE INDEX idx\_pedidos\_usuario ON pedidos(id\_usuario);

CREATE INDEX idx\_pedidos\_estado ON pedidos(id\_estado);

CREATE INDEX idx\_pedidos\_fecha ON pedidos(fecha\_creacion);

-- Índices para historial de inventario

CREATE INDEX idx\_inventario\_producto ON registro\_inventario(id\_producto);

CREATE INDEX idx\_inventario\_fecha ON registro\_inventario(fecha\_cambio);

-- OPTIMIZACIÓN

-- Optimización de consultas

CREATE INDEX idx\_pedidos\_usuario ON pedidos(id\_usuario);

CREATE INDEX idx\_pedidos\_estado ON pedidos(id\_estado);

CREATE INDEX idx\_pedidos\_fecha ON pedidos(fecha\_creacion);

-- Optimización de índices

CREATE INDEX idx\_productos\_categoria ON productos(id\_categoria);

CREATE INDEX idx\_productos\_marca ON productos(marca);

CREATE INDEX idx\_productos\_precio ON productos(precio);

CREATE INDEX idx\_productos\_activos ON productos(esta\_activo) WHERE esta\_activo = TRUE;

-- VISTAS

-- Vista para productos populares

CREATE VIEW productos\_populares AS

SELECT p.id\_producto, p.nombre, p.precio, COUNT(ip.id\_item\_pedido) AS total\_vendido

FROM productos p

LEFT JOIN items\_pedido ip ON p.id\_producto = ip.id\_producto

GROUP BY p.id\_producto

ORDER BY total\_vendido DESC;

-- Vista para inventario bajo

CREATE VIEW inventario\_bajo AS

SELECT id\_producto, nombre, cantidad\_disponible

FROM productos

WHERE cantidad\_disponible < 10

ORDER BY cantidad\_disponible ASC;

-- Vista para resumen de ventas

CREATE VIEW resumen\_ventas AS

SELECT

    DATE\_TRUNC('month', p.fecha\_creacion) AS mes,

    COUNT(DISTINCT p.id\_pedido) AS total\_pedidos,

    SUM(p.total) AS ingresos\_totales,

    AVG(p.total) AS valor\_promedio\_pedido

FROM pedidos p

GROUP BY DATE\_TRUNC('month', p.fecha\_creacion)

ORDER BY mes DESC;

-- ESQUEMAS

-- Índices para búsquedas frecuentes

CREATE INDEX idx\_productos\_categoria ON productos (id\_categoria);

CREATE INDEX idx\_productos\_marca ON productos (marca);

CREATE INDEX idx\_productos\_precio ON productos (precio);

CREATE INDEX idx\_productos\_activos ON productos (esta\_activo)

WHERE

    esta\_activo = TRUE;

-- Índices para consultas de usuarios

CREATE INDEX idx\_usuarios\_correo ON usuarios (correo\_electronico);

CREATE INDEX idx\_usuarios\_rol ON usuarios (id\_rol);

-- Índices para pedidos

CREATE INDEX idx\_pedidos\_usuario ON pedidos (id\_usuario);

CREATE INDEX idx\_pedidos\_estado ON pedidos (id\_estado);

CREATE INDEX idx\_pedidos\_fecha ON pedidos (fecha\_creacion);

-- Índices para historial de inventario

CREATE INDEX idx\_inventario\_producto ON registro\_inventario (id\_producto);

CREATE INDEX idx\_inventario\_fecha ON registro\_inventario (fecha\_cambio);

-- Índice para la búsqueda de productos

CREATE INDEX idx\_productos\_busqueda ON productos USING gin(to\_tsvector('spanish', nombre || ' ' || descripcion));

CREATE INDEX idx\_productos\_activos ON productos(id\_producto) WHERE esta\_activo;

-- Para búsqueda geográfica

CREATE INDEX idx\_ubicaciones\_geo ON ubicaciones USING gist(coordenadas);

-- Para autenticación

CREATE INDEX idx\_usuarios\_auth ON usuarios(nombre\_usuario, correo\_electronico) WHERE esta\_activo;

-- Vista para inventario bajo

CREATE VIEW inventario\_bajo AS

SELECT

    id\_producto,

    nombre,

    cantidad\_disponible

FROM productos

WHERE

    cantidad\_disponible < 10

ORDER BY cantidad\_disponible ASC;

-- Vista para resumen de ventas

CREATE VIEW resumen\_ventas AS

SELECT

    DATE\_TRUNC('month', p.fecha\_creacion) AS mes,

    COUNT(DISTINCT p.id\_pedido) AS total\_pedidos,

    SUM(p.total) AS ingresos\_totales,

    AVG(p.total) AS valor\_promedio\_pedido

FROM pedidos p

GROUP BY

    DATE\_TRUNC('month', p.fecha\_creacion)

ORDER BY mes DESC;

-- AUDITORIAS

-- Particionamiento de registros de auditoría

CREATE TABLE auditoria\_accesos (

    id SERIAL PRIMARY KEY,

    usuario VARCHAR(100),

    accion VARCHAR(50),

    tabla\_afectada VARCHAR(100),

    datos\_anteriores JSONB,

    datos\_nuevos JSONB,

    fecha\_accion TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

) PARTITION BY RANGE (fecha\_accion);

-- Función para actualizar fecha de actualización

CREATE OR REPLACE FUNCTION public.actualizar\_fecha\_actualizacion()

RETURNS TRIGGER AS $$

BEGIN

  NEW.fecha\_actualizacion = CURRENT\_TIMESTAMP;

  RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Tabla de seguridad de eventos de auditoría

CREATE TABLE seguridad.eventos\_auditoria (

  id\_evento BIGSERIAL PRIMARY KEY,

  tipo\_evento VARCHAR(50) NOT NULL,

  usuario\_id UUID,

  direccion\_ip INET,

  detalles JSONB,

  fecha\_evento TIMESTAMPTZ DEFAULT NOW()

);

-- Creación del rol de monitoreo

CREATE ROLE monitor WITH LOGIN PASSWORD 'monitor\_pass';

GRANT CONNECT ON DATABASE current\_database() TO monitor;

GRANT USAGE ON SCHEMA seguridad TO monitor;

GRANT SELECT ON seguridad.eventos\_auditoria TO monitor;

-- Enmascaramiento de datos

SECURITY LABEL FOR anon ON COLUMN usuarios.contrasena\_hash IS 'MASKED WITH VALUE NULL';

-- Restringir acceso a datos sensibles

ALTER TABLE usuarios ENABLE ROW LEVEL SECURITY;

-- Solo admins ven todos los usuarios

CREATE POLICY admin\_access ON usuarios

    USING (current\_user = 'admin' OR id\_usuario = (SELECT id\_usuario FROM usuarios WHERE nombre\_usuario = current\_user));

-- Auditoria

-- Habilitar auditoría en postgresql.conf

pgaudit.log = 'all, -misc'

pgaudit.log\_relation = on

-- =============================================

-- ROW LEVEL SECURITY (RLS) PARA TABLAS CRÍTICAS

-- =============================================

-- Tabla de usuarios

ALTER TABLE usuarios ENABLE ROW LEVEL SECURITY;

-- Política para metal\_user (acceso completo)

CREATE POLICY metal\_user\_full\_access ON usuarios

  TO metal\_user

  USING (true) WITH CHECK (true);

-- Política para que los usuarios vean solo su propio perfil

CREATE POLICY user\_own\_profile ON usuarios

  FOR SELECT TO api\_customer

  USING (id\_usuario = current\_setting('app.current\_user\_id')::UUID);

-- Política para que los admins vean todos los usuarios

CREATE POLICY admin\_view\_all\_users ON usuarios

  FOR SELECT TO api\_admin

  USING (true);

-- Política para actualización de propio perfil

CREATE POLICY user\_update\_own\_profile ON usuarios

  FOR UPDATE TO api\_customer

  USING (id\_usuario = current\_setting('app.current\_user\_id')::UUID)

  WITH CHECK (id\_usuario = current\_setting('app.current\_user\_id')::UUID);

-- Tabla de pedidos

ALTER TABLE pedidos ENABLE ROW LEVEL SECURITY;

-- Política para clientes (solo sus pedidos)

CREATE POLICY customer\_own\_orders ON pedidos

  FOR ALL TO api\_customer

  USING (id\_usuario = current\_setting('app.current\_user\_id')::UUID)

  WITH CHECK (id\_usuario = current\_setting('app.current\_user\_id')::UUID);

-- Política para admins (todos los pedidos)

CREATE POLICY admin\_all\_orders ON pedidos

  FOR ALL TO api\_admin

  USING (true) WITH CHECK (true);

-- Política para repartidores (solo pedidos en estados específicos)

CREATE POLICY delivery\_view\_orders ON pedidos

  FOR SELECT TO api\_delivery

  USING (id\_estado IN (

    SELECT id\_estado FROM estados\_pedido

    WHERE codigo\_estado IN ('por\_enviar', 'enviado')

  ));

-- Tabla de productos

ALTER TABLE productos ENABLE ROW LEVEL SECURITY;

-- Política para lectura pública de productos activos

CREATE POLICY public\_read\_active\_products ON productos

  FOR SELECT TO api\_public

  USING (esta\_activo);

-- Política para clientes (lectura completa)

CREATE POLICY customer\_read\_products ON productos

  FOR SELECT TO api\_customer

  USING (true);

-- Política para admins (control completo)

CREATE POLICY admin\_full\_products ON productos

  FOR ALL TO api\_admin

  USING (true) WITH CHECK (true);

-- Tabla de listas de deseos

ALTER TABLE listas\_deseos ENABLE ROW LEVEL SECURITY;

-- Política para dueños de listas

CREATE POLICY owner\_list\_access ON listas\_deseos

  FOR ALL TO api\_customer

  USING (id\_usuario = current\_setting('app.current\_user\_id')::UUID)

  WITH CHECK (id\_usuario = current\_setting('app.current\_user\_id')::UUID);

-- Política para listas públicas

CREATE POLICY public\_list\_read ON listas\_deseos

  FOR SELECT TO api\_public

  USING (NOT es\_privada);

-- Tabla de carritos de compra

ALTER TABLE carritos\_compras ENABLE ROW LEVEL SECURITY;

-- Política para dueños de carritos

CREATE POLICY owner\_cart\_access ON carritos\_compras

  FOR ALL TO api\_customer

  USING (id\_usuario = current\_setting('app.current\_user\_id')::UUID)

  WITH CHECK (id\_usuario = current\_setting('app.current\_user\_id')::UUID);

-- =============================================

-- SEGURIDAD PARA TABLAS DE AUDITORÍA

-- =============================================

-- Tabla historial\_precios

ALTER TABLE historial\_precios ENABLE ROW LEVEL SECURITY;

CREATE POLICY admin\_read\_price\_history ON historial\_precios

  FOR SELECT TO api\_admin

  USING (true);

-- Tabla registro\_inventario

ALTER TABLE registro\_inventario ENABLE ROW LEVEL SECURITY;

CREATE POLICY admin\_read\_inventory\_history ON registro\_inventario

  FOR SELECT TO api\_admin

  USING (true);

-- Tabla historial\_estados\_pedido

ALTER TABLE historial\_estados\_pedido ENABLE ROW LEVEL SECURITY;

CREATE POLICY customer\_read\_own\_order\_history ON historial\_estados\_pedido

  FOR SELECT TO api\_customer

  USING (id\_pedido IN (

    SELECT id\_pedido FROM pedidos

    WHERE id\_usuario = current\_setting('app.current\_user\_id')::UUID

  ));

CREATE POLICY admin\_read\_all\_order\_history ON historial\_estados\_pedido

  FOR SELECT TO api\_admin

  USING (true);

CREATE POLICY delivery\_read\_assigned\_order\_history ON historial\_estados\_pedido

  FOR SELECT TO api\_delivery

  USING (id\_pedido IN (

    SELECT id\_pedido FROM pedidos

    WHERE id\_estado IN (

      SELECT id\_estado FROM estados\_pedido

      WHERE codigo\_estado IN ('por\_enviar', 'enviado')

    )

  ));

* Script de esquema de apis

-- CONFIGURACIONES DE API

-- Crear esquemas para API

CREATE SCHEMA IF NOT EXISTS api\_public;   -- Visitantes

CREATE SCHEMA IF NOT EXISTS api\_customer; -- Clientes

CREATE SCHEMA IF NOT EXISTS api\_admin;    -- Administradores

CREATE SCHEMA IF NOT EXISTS api\_delivery; -- Delivery

-- =============================================

-- CONFIGURACIÓN DE ROLES DEL SISTEMA

-- =============================================

-- Crear roles de aplicación

CREATE ROLE api\_public WITH LOGIN PASSWORD 'public\_api';

CREATE ROLE api\_customer WITH LOGIN PASSWORD 'customer\_api';

CREATE ROLE api\_admin WITH LOGIN PASSWORD 'admin\_api';

CREATE ROLE api\_delivery WITH LOGIN PASSWORD 'delivery\_api';

CREATE IF NOT EXISTS ROLE metal\_user WITH LOGIN PASSWORD 'metal\_api' BYPASSRLS;

-- Asignar permisos a esquemas

GRANT USAGE ON SCHEMA api\_public TO api\_public;

GRANT USAGE ON SCHEMA api\_customer TO api\_customer;

GRANT USAGE ON SCHEMA api\_admin TO api\_admin;

GRANT USAGE ON SCHEMA api\_delivery TO api\_delivery;

-- Metal\_user tiene acceso total

GRANT USAGE ON ALL SCHEMAS IN DATABASE TO metal\_user;

GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public, api\_public, api\_customer, api\_admin, api\_delivery TO metal\_user;

-- Configurar permisos por defecto

ALTER DEFAULT PRIVILEGES IN SCHEMA api\_public GRANT EXECUTE ON FUNCTIONS TO api\_public;

ALTER DEFAULT PRIVILEGES IN SCHEMA api\_customer GRANT EXECUTE ON FUNCTIONS TO api\_customer, api\_admin;

ALTER DEFAULT PRIVILEGES IN SCHEMA api\_admin GRANT EXECUTE ON FUNCTIONS TO api\_admin;

ALTER DEFAULT PRIVILEGES IN SCHEMA api\_delivery GRANT EXECUTE ON FUNCTIONS TO api\_delivery;

* Script de api\_public

-- =============================================

-- ESQUEMA API\_PUBLIC

-- =============================================

CREATE SCHEMA IF NOT EXISTS api\_public;

GRANT USAGE ON SCHEMA api\_public TO api\_public;

ALTER DEFAULT PRIVILEGES IN SCHEMA api\_public

GRANT EXECUTE ON FUNCTIONS TO api\_public;

-- =============================================

-- FUNCIONES DE AUTENTICACIÓN

-- =============================================

-- Verificar rol del usuario: Verifica si el usuario tiene el rol requerido

CREATE OR REPLACE FUNCTION public.verificar\_rol\_usuario(

  p\_user\_id UUID,

  p\_required\_role VARCHAR

) RETURNS BOOLEAN AS $$

DECLARE

  v\_user\_role VARCHAR;

BEGIN

  SELECT r.nombre\_rol INTO v\_user\_role

  FROM usuarios u

  JOIN roles r ON u.id\_rol = r.id\_rol

  WHERE u.id\_usuario = p\_user\_id;

  IF v\_user\_role = p\_required\_role THEN

    RETURN TRUE;

  END IF;

  -- Admins tienen acceso a api\_customer también

  IF p\_required\_role = 'customer' AND v\_user\_role = 'admin' THEN

    RETURN TRUE;

  END IF;

  RETURN FALSE;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION public.set\_current\_user\_context()

RETURNS TRIGGER AS $$

BEGIN

  -- Verificar JWT y establecer contexto

  -- Esta función sería llamada por Supabase Auth hooks

  EXECUTE format('SET app.current\_user\_id = %L', NEW.id);

  EXECUTE format('SET app.current\_user\_role = %L', NEW.role);

  RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Validar contraseña: debe tener al menos 8 caracteres, una mayúscula, un número y un caracter especial

CREATE OR REPLACE FUNCTION public.validar\_contrasena(p\_contrasena TEXT)

RETURNS BOOLEAN AS $$

BEGIN

  -- Al menos 8 caracteres

  IF length(p\_contrasena) < 8 THEN

    RETURN FALSE;

  END IF;

  -- Al menos una mayúscula

  IF p\_contrasena !~ '[A-Z]' THEN

    RETURN FALSE;

  END IF;

  -- Al menos un número

  IF p\_contrasena !~ '[0-9]' THEN

    RETURN FALSE;

  END IF;

  -- Al menos un caracter especial

  IF p\_contrasena !~ '[^a-zA-Z0-9]' THEN

    RETURN FALSE;

  END IF;

  RETURN TRUE;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

-- Registrar usuario: debe validar que el correo sea válido, el teléfono peruano y que la contraseña sea válida

CREATE OR REPLACE FUNCTION api\_public.registrar\_usuario(

    p\_nombre\_usuario VARCHAR(50),

    p\_correo VARCHAR(255),

    p\_contrasena TEXT,

    p\_nombres VARCHAR(100),

    p\_apellidos VARCHAR(100),

    p\_telefono VARCHAR(20) RETURNS JSONB AS $$

    DECLARE v\_id\_usuario UUID;

v\_resultado JSONB;

BEGIN -- Validar formato de correo

IF p\_correo !~ '^[A-Za-z0-9.\_%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$' THEN RETURN jsonb\_build\_object(

  'error',

  'Formato de correo electrónico inválido'

);

END IF;

-- Validar formato de teléfono peruano

IF p\_telefono IS NOT NULL

AND p\_telefono !~ '^9[0-9]{8}$' THEN RETURN jsonb\_build\_object(

  'error',

  'Teléfono debe ser un número peruano de 9 dígitos comenzando con 9'

);

END IF;

INSERT INTO usuarios(

    nombre\_usuario,

    correo\_electronico,

    contrasena\_hash,

    nombres,

    apellidos,

    telefono,

    id\_rol

  )

VALUES (

    p\_nombre\_usuario,

    p\_correo,

    crypt(p\_contrasena, gen\_salt('bf')),

    p\_nombres,

    p\_apellidos,

    p\_telefono,

    (

      SELECT id\_rol

      FROM roles

      WHERE nombre\_rol = 'customer'

    )

    RETURNING id\_usuario INTO v\_id\_usuario;

-- Retornar información básica del usuario registrado

SELECT jsonb\_build\_object(

    'id\_usuario',

    v\_id\_usuario,

    'nombre\_usuario',

    p\_nombre\_usuario,

    'nombres',

    p\_nombres,

    'apellidos',

    p\_apellidos,

    'correo',

    p\_correo,

    'telefono',

    p\_telefono,

    'mensaje',

    'Usuario registrado exitosamente'

  ) INTO v\_resultado;

RETURN v\_resultado;

EXCEPTION

WHEN unique\_violation THEN RETURN jsonb\_build\_object(

  'error',

  'El nombre de usuario o correo electrónico ya está en uso'

);

WHEN others THEN RETURN jsonb\_build\_object('error', SQLERRM);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_public.iniciar\_sesion(

    p\_identificador VARCHAR(255),

    p\_contrasena TEXT

  ) RETURNS JSONB AS $$

DECLARE v\_usuario JSONB;

BEGIN

SELECT jsonb\_build\_object(

    'id\_usuario',

    u.id\_usuario,

    'nombre\_usuario',

    u.nombre\_usuario,

    'nombres',

    u.nombres,

    'apellidos',

    u.apellidos,

    'correo',

    u.correo\_electronico,

    'rol',

    r.nombre\_rol,

    'esta\_activo',

    u.esta\_activo

  ) INTO v\_usuario

FROM usuarios u

  JOIN roles r ON u.id\_rol = r.id\_rol

WHERE (

    u.nombre\_usuario = p\_identificador

    OR u.correo\_electronico = p\_identificador

  )

  AND u.contrasena\_hash = crypt(p\_contrasena, u.contrasena\_hash)

  AND u.esta\_activo;

IF v\_usuario IS NULL THEN RETURN jsonb\_build\_object(

  'error',

  'Credenciales inválidas o usuario inactivo'

);

END IF;

RETURN jsonb\_build\_object(

  'success',

  true,

  'data',

  v\_usuario

);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

-- =============================================

-- FUNCIONES DE CATALOGO DE PRODUCTOS

-- =============================================

CREATE OR REPLACE FUNCTION api\_public.obtener\_categorias() RETURNS JSONB AS $$

DECLARE v\_categorias JSONB;

BEGIN WITH RECURSIVE categorias\_tree AS (

  SELECT id\_categoria,

    nombre,

    descripcion,

    id\_categoria\_padre,

    jsonb\_build\_array()::jsonb AS subcategorias

  FROM categorias

  WHERE id\_categoria\_padre IS NULL

    AND esta\_activo

  UNION ALL

  SELECT c.id\_categoria,

    c.nombre,

    c.descripcion,

    c.id\_categoria\_padre,

    jsonb\_build\_array()::jsonb

  FROM categorias c

    JOIN categorias\_tree ct ON c.id\_categoria\_padre = ct.id\_categoria

  WHERE c.esta\_activo

)

SELECT jsonb\_agg(

    jsonb\_build\_object(

      'id',

      id\_categoria,

      'nombre',

      nombre,

      'descripcion',

      descripcion,

      'subcategorias',

      subcategorias

    )

  ) INTO v\_categorias

FROM categorias\_tree

WHERE id\_categoria\_padre IS NULL;

RETURN jsonb\_build\_object(

  'categorias',

  COALESCE(v\_categorias, '[]'::jsonb)

);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_public.buscar\_productos(

    p\_termino\_busqueda VARCHAR DEFAULT NULL,

    p\_id\_categoria INTEGER DEFAULT NULL,

    p\_rango\_precio NUMERIC [] DEFAULT NULL,

    p\_pagina INTEGER DEFAULT 1,

    p\_por\_pagina INTEGER DEFAULT 20,

    p\_orden VARCHAR DEFAULT 'recientes'

  ) RETURNS JSONB AS $$

DECLARE v\_productos JSONB;

v\_total INTEGER;

v\_resultado JSONB;

BEGIN -- Contar total de productos que coinciden con los filtros

SELECT COUNT(\*) INTO v\_total

FROM productos p

  JOIN categorias c ON p.id\_categoria = c.id\_categoria

WHERE p.esta\_activo

  AND (

    p\_termino\_busqueda IS NULL

    OR (

      p.nombre ILIKE '%' || p\_termino\_busqueda || '%'

      OR p.descripcion ILIKE '%' || p\_termino\_busqueda || '%'

    )

  )

  AND (

    p\_id\_categoria IS NULL

    OR p.id\_categoria = p\_id\_categoria

    OR c.id\_categoria\_padre = p\_id\_categoria

  )

  AND (

    p\_rango\_precio IS NULL

    OR (

      p.precio BETWEEN p\_rango\_precio [1] AND p\_rango\_precio [2]

    )

  );

-- Obtener productos paginados

SELECT jsonb\_agg(

    jsonb\_build\_object(

      'id',

      p.id\_producto,

      'nombre',

      p.nombre,

      'precio',

      p.precio,

      'imagen\_principal',

      (

        SELECT url\_imagen

        FROM imagenes\_productos

        WHERE id\_producto = p.id\_producto

          AND es\_principal

        LIMIT 1

      ), 'calificacion\_promedio', (

        SELECT COALESCE(AVG(calificacion), 0)

        FROM resenas\_productos

        WHERE id\_producto = p.id\_producto

          AND esta\_aprobado

      ),

      'marca',

      p.marca

    )

  ) INTO v\_productos

FROM productos p

  JOIN categorias c ON p.id\_categoria = c.id\_categoria

WHERE p.esta\_activo

  AND (

    p\_termino\_busqueda IS NULL

    OR (

      p.nombre ILIKE '%' || p\_termino\_busqueda || '%'

      OR p.descripcion ILIKE '%' || p\_termino\_busqueda || '%'

    )

  )

  AND (

    p\_id\_categoria IS NULL

    OR p.id\_categoria = p\_id\_categoria

    OR c.id\_categoria\_padre = p\_id\_categoria

  )

  AND (

    p\_rango\_precio IS NULL

    OR (

      p.precio BETWEEN p\_rango\_precio [1] AND p\_rango\_precio [2]

    )

  )

ORDER BY CASE

    p\_orden

    WHEN 'recientes' THEN p.fecha\_creacion

    WHEN 'precio\_asc' THEN p.precio

    WHEN 'precio\_desc' THEN - p.precio

    WHEN 'mejor\_calificados' THEN (

      SELECT - COALESCE(AVG(calificacion), 0)

      FROM resenas\_productos

      WHERE id\_producto = p.id\_producto

        AND esta\_aprobado

    )

    ELSE p.fecha\_creacion

  END DESC

LIMIT p\_por\_pagina OFFSET (p\_pagina - 1) \* p\_por\_pagina;

-- Construir respuesta con paginación

SELECT jsonb\_build\_object(

    'productos',

    COALESCE(v\_productos, '[]'::jsonb),

    'paginacion',

    jsonb\_build\_object(

      'pagina\_actual',

      p\_pagina,

      'por\_pagina',

      p\_por\_pagina,

      'total',

      v\_total,

      'total\_paginas',

      CEIL(v\_total::FLOAT / p\_por\_pagina)

    )

  ) INTO v\_resultado;

RETURN v\_resultado;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_public.obtener\_detalle\_producto(p\_id\_producto UUID) RETURNS JSONB AS $$

DECLARE v\_producto JSONB;

v\_imagenes JSONB;

v\_resenas JSONB;

v\_resultado JSONB;

BEGIN -- Obtener información básica del producto

SELECT jsonb\_build\_object(

    'id',

    p.id\_producto,

    'nombre',

    p.nombre,

    'descripcion',

    p.descripcion,

    'precio',

    p.precio,

    'costo',

    p.costo,

    'disponibilidad',

    p.cantidad\_disponible,

    'marca',

    p.marca,

    'talla',

    p.talla,

    'categoria',

    jsonb\_build\_object(

      'id',

      c.id\_categoria,

      'nombre',

      c.nombre

    ),

    'detalles\_tecnicos',

    jsonb\_build\_object(

      'peso',

      dp.peso,

      'dimensiones',

      dp.dimensiones,

      'material\_principal',

      dp.material\_principal,

      'materiales\_secundarios',

      dp.materiales\_secundarios,

      'cuidados\_especiales',

      dp.cuidados\_especiales,

      'especificaciones',

      dp.especificaciones

    ),

    'atributos',

    p.atributos\_generales

  ) INTO v\_producto

FROM productos p

  JOIN categorias c ON p.id\_categoria = c.id\_categoria

  JOIN detalles\_productos dp ON p.id\_detalle\_producto = dp.id\_detalle\_producto

WHERE p.id\_producto = p\_id\_producto

  AND p.esta\_activo;

IF v\_producto IS NULL THEN RETURN jsonb\_build\_object('error', 'Producto no encontrado o inactivo');

END IF;

-- Obtener imágenes del producto

SELECT COALESCE(

    jsonb\_agg(

      jsonb\_build\_object(

        'url',

        url\_imagen,

        'texto\_alternativo',

        texto\_alternativo,

        'es\_principal',

        es\_principal,

        'orden',

        orden\_visualizacion

      )

      ORDER BY orden\_visualizacion

    ),

    '[]'::jsonb

  ) INTO v\_imagenes

FROM imagenes\_productos

WHERE id\_producto = p\_id\_producto;

-- Obtener reseñas aprobadas

SELECT COALESCE(

    jsonb\_agg(

      jsonb\_build\_object(

        'id',

        r.id\_resena,

        'usuario',

        jsonb\_build\_object(

          'nombre',

          u.nombres,

          'apellido',

          u.apellidos

        ),

        'calificacion',

        r.calificacion,

        'comentario',

        r.comentario,

        'fecha',

        r.fecha\_creacion

      )

      ORDER BY r.fecha\_creacion DESC

    ),

    '[]'::jsonb

  ) INTO v\_resenas

FROM resenas\_productos r

  JOIN usuarios u ON r.id\_usuario = u.id\_usuario

WHERE r.id\_producto = p\_id\_producto

  AND r.esta\_aprobado;

-- Calcular promedio de calificaciones

SELECT COALESCE(AVG(calificacion), 0) INTO v\_resultado

FROM resenas\_productos

WHERE id\_producto = p\_id\_producto

  AND esta\_aprobado;

-- Construir respuesta final

SELECT jsonb\_build\_object(

    'producto',

    v\_producto,

    'imagenes',

    v\_imagenes,

    'resenas',

    v\_resenas,

    'calificacion\_promedio',

    v\_resultado

  ) INTO v\_resultado;

RETURN v\_resultado;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

-- =============================================

-- FUNCIONES DE OFERTAS Y PROMOCIONES

-- =============================================

CREATE OR REPLACE FUNCTION api\_public.obtener\_ofertas(p\_limit INTEGER DEFAULT 10) RETURNS JSONB AS $$

DECLARE v\_ofertas JSONB;

BEGIN

SELECT COALESCE(

    jsonb\_agg(

      jsonb\_build\_object(

        'id',

        p.id\_producto,

        'nombre',

        p.nombre,

        'precio\_original',

        p.precio,

        'precio\_oferta',

        (p.precio \* 0.9),

        -- Ejemplo: 10% de descuento

        'imagen',

        (

          SELECT url\_imagen

          FROM imagenes\_productos

          WHERE id\_producto = p.id\_producto

            AND es\_principal

          LIMIT 1

        ), 'valido\_hasta', (CURRENT\_DATE + INTERVAL '7 days') -- Oferta válida por 7 días

      )

      ORDER BY p.fecha\_creacion DESC

      LIMIT p\_limit

    ), '[]'::jsonb

  ) INTO v\_ofertas

FROM productos p

WHERE p.esta\_activo

  AND p.cantidad\_disponible > 0;

RETURN jsonb\_build\_object('ofertas', v\_ofertas);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_public.obtener\_promociones() RETURNS JSONB AS $$

DECLARE v\_promociones JSONB;

BEGIN -- Ejemplo de promociones (en un sistema real podrían venir de una tabla específica)

SELECT jsonb\_build\_array(

    jsonb\_build\_object(

      'id',

      1,

      'titulo',

      'Envío gratis',

      'descripcion',

      'Envío gratis en compras mayores a S/100',

      'valido\_hasta',

      (CURRENT\_DATE + INTERVAL '30 days'),

      'imagen',

      'https://ejemplo.com/promo-envio-gratis.jpg'

    ),

    jsonb\_build\_object(

      'id',

      2,

      'titulo',

      '2x1 en productos seleccionados',

      'descripcion',

      'Lleva 2 productos por el precio de 1 en la sección de ofertas',

      'valido\_hasta',

      (CURRENT\_DATE + INTERVAL '15 days'),

      'imagen',

      'https://ejemplo.com/promo-2x1.jpg'

    )

  ) INTO v\_promociones;

RETURN jsonb\_build\_object('promociones', v\_promociones);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

-- =============================================

-- FUNCIONES DE RESEÑAS

-- =============================================

CREATE OR REPLACE FUNCTION api\_public.obtener\_resenas\_producto(

    p\_id\_producto UUID,

    p\_pagina INTEGER DEFAULT 1,

    p\_por\_pagina INTEGER DEFAULT 5

  ) RETURNS JSONB AS $$

DECLARE v\_resenas JSONB;

v\_total INTEGER;

v\_resultado JSONB;

BEGIN -- Contar total de reseñas

SELECT COUNT(\*) INTO v\_total

FROM resenas\_productos

WHERE id\_producto = p\_id\_producto

  AND esta\_aprobado;

-- Obtener reseñas paginadas

SELECT COALESCE(

    jsonb\_agg(

      jsonb\_build\_object(

        'id',

        r.id\_resena,

        'usuario',

        jsonb\_build\_object(

          'nombre',

          u.nombres,

          'iniciales',

          SUBSTRING(

            u.nombres

            FROM 1 FOR 1

          ) || SUBSTRING(

            u.apellidos

            FROM 1 FOR 1

          )

        ),

        'calificacion',

        r.calificacion,

        'comentario',

        r.comentario,

        'fecha',

        TO\_CHAR(r.fecha\_creacion, 'DD/MM/YYYY')

      )

    ),

    '[]'::jsonb

  ) INTO v\_resenas

FROM resenas\_productos r

  JOIN usuarios u ON r.id\_usuario = u.id\_usuario

WHERE r.id\_producto = p\_id\_producto

  AND r.esta\_aprobado

ORDER BY r.fecha\_creacion DESC

LIMIT p\_por\_pagina OFFSET (p\_pagina - 1) \* p\_por\_pagina;

-- Construir respuesta con paginación

SELECT jsonb\_build\_object(

    'resenas',

    v\_resenas,

    'paginacion',

    jsonb\_build\_object(

      'pagina\_actual',

      p\_pagina,

      'por\_pagina',

      p\_por\_pagina,

      'total',

      v\_total,

      'total\_paginas',

      CEIL(v\_total::FLOAT / p\_por\_pagina)

    )

  ) INTO v\_resultado;

RETURN v\_resultado;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

---------------------------------------------------------------

-- Para búsqueda de productos

CREATE INDEX idx\_productos\_busqueda ON productos USING gin(to\_tsvector('spanish', nombre || ' ' || descripcion));

CREATE INDEX idx\_productos\_activos ON productos(id\_producto) WHERE esta\_activo;

-- Para búsqueda geográfica

CREATE INDEX idx\_ubicaciones\_geo ON ubicaciones USING gist(coordenadas);

-- Para autenticación

CREATE INDEX idx\_usuarios\_auth ON usuarios(nombre\_usuario, correo\_electronico) WHERE esta\_activo;

ALTER TABLE productos ENABLE ROW LEVEL SECURITY;

CREATE POLICY productos\_publicos ON productos

  FOR SELECT TO api\_public

  USING (esta\_activo);

* Script de api\_customer

-- =============================================

-- ESQUEMA API\_CUSTOMER

-- =============================================

CREATE SCHEMA IF NOT EXISTS api\_customer;

GRANT USAGE ON SCHEMA api\_customer TO api\_customer;

ALTER DEFAULT PRIVILEGES IN SCHEMA api\_customer GRANT EXECUTE ON FUNCTIONS TO api\_customer;

-- Vista segura para clientes

CREATE VIEW api\_customer.vista\_segura\_clientes AS

SELECT

  id\_usuario,

  nombres,

  apellidos,

  correo\_electronico,

  telefono,

  esta\_activo

FROM usuarios

WHERE id\_usuario = current\_setting('app.current\_user\_id')::UUID;

-- =============================================

-- FUNCIONES DE PERFIL

-- =============================================

CREATE OR REPLACE FUNCTION api\_customer.obtener\_mi\_perfil(

  p\_id\_usuario UUID

) RETURNS JSONB AS $$

DECLARE

  v\_perfil JSONB;

BEGIN

  SELECT jsonb\_build\_object(

    'id', id\_usuario,

    'nombre\_usuario', nombre\_usuario,

    'nombres', nombres,

    'apellidos', apellidos,

    'correo', correo\_electronico,

    'telefono', telefono,

    'direccion\_principal', (

      SELECT jsonb\_build\_object(

        'direccion', direccion,

        'coordenadas', ST\_AsGeoJSON(coordenadas)::jsonb

      )

      FROM ubicaciones

      WHERE id\_ubicacion = u.id\_ubicacion\_principal

    ),

    'direcciones\_adicionales', direcciones\_adicionales,

    'fecha\_registro', fecha\_creacion

  ) INTO v\_perfil

  FROM usuarios u

  WHERE id\_usuario = p\_id\_usuario;

  RETURN jsonb\_build\_object('perfil', v\_perfil);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_customer.obtener\_perfil\_publico(

  p\_id\_usuario UUID

) RETURNS JSONB AS $$

DECLARE

  v\_perfil JSONB;

BEGIN

  SELECT jsonb\_build\_object(

    'nombres', nombres,

    'apellidos', apellidos,

    'fecha\_registro', fecha\_creacion,

    'listas\_publicas', (

      SELECT COALESCE(jsonb\_agg(jsonb\_build\_object(

        'id', id\_lista\_deseos,

        'nombre', nombre

      )), '[]'::jsonb)

      FROM listas\_deseos

      WHERE id\_usuario = u.id\_usuario AND NOT es\_privada

    )

  ) INTO v\_perfil

  FROM usuarios u

  WHERE id\_usuario = p\_id\_usuario;

  RETURN jsonb\_build\_object('perfil', v\_perfil);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_customer.actualizar\_perfil(

  p\_id\_usuario UUID,

  p\_nombres VARCHAR(100) DEFAULT NULL,

  p\_apellidos VARCHAR(100) DEFAULT NULL,

  p\_telefono VARCHAR(20) DEFAULT NULL,

  p\_direcciones\_adicionales JSONB DEFAULT NULL

) RETURNS JSONB AS $$

BEGIN

  -- Validar formato de teléfono peruano si se proporciona

  IF p\_telefono IS NOT NULL AND p\_telefono !~ '^9[0-9]{8}$' THEN

    RETURN jsonb\_build\_object('error', 'Teléfono debe ser un número peruano de 9 dígitos comenzando con 9');

  END IF;

  -- Validar estructura JSON de direcciones adicionales si se proporciona

  IF p\_direcciones\_adicionales IS NOT NULL AND jsonb\_typeof(p\_direcciones\_adicionales) != 'array' THEN

    RETURN jsonb\_build\_object('error', 'Las direcciones adicionales deben ser un array JSON');

  END IF;

  UPDATE usuarios SET

    nombres = COALESCE(p\_nombres, nombres),

    apellidos = COALESCE(p\_apellidos, apellidos),

    telefono = COALESCE(p\_telefono, telefono),

    direcciones\_adicionales = COALESCE(p\_direcciones\_adicionales, direcciones\_adicionales)

  WHERE id\_usuario = p\_id\_usuario;

  RETURN jsonb\_build\_object('success', true, 'message', 'Perfil actualizado correctamente');

EXCEPTION

  WHEN others THEN

    RETURN jsonb\_build\_object('error', SQLERRM);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_customer.cambiar\_contrasena(

  p\_id\_usuario UUID,

  p\_contrasena\_actual TEXT,

  p\_nueva\_contrasena TEXT

) RETURNS JSONB AS $$

DECLARE

  v\_current\_hash TEXT;

BEGIN

  -- Obtener hash actual

  SELECT contrasena\_hash INTO v\_current\_hash

  FROM usuarios

  WHERE id\_usuario = p\_id\_usuario;

  -- Verificar contraseña actual

  IF v\_current\_hash != crypt(p\_contrasena\_actual, v\_current\_hash) THEN

    RETURN jsonb\_build\_object('error', 'La contraseña actual no es correcta');

  END IF;

  -- Actualizar contraseña

  UPDATE usuarios

  SET contrasena\_hash = crypt(p\_nueva\_contrasena, gen\_salt('bf'))

  WHERE id\_usuario = p\_id\_usuario;

  RETURN jsonb\_build\_object('success', true, 'message', 'Contraseña actualizada correctamente');

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

-- =============================================

-- FUNCIONES DE PEDIDOS

-- =============================================

CREATE OR REPLACE FUNCTION api\_customer.obtener\_mis\_pedidos(

  p\_id\_usuario UUID,

  p\_pagina INTEGER DEFAULT 1,

  p\_por\_pagina INTEGER DEFAULT 10,

  p\_estado VARCHAR DEFAULT NULL

) RETURNS JSONB AS $$

DECLARE

  v\_pedidos JSONB;

  v\_total INTEGER;

  v\_resultado JSONB;

BEGIN

  -- Contar total de pedidos

  SELECT COUNT(\*) INTO v\_total

  FROM pedidos

  WHERE id\_usuario = p\_id\_usuario

    AND (p\_estado IS NULL OR id\_estado = (

      SELECT id\_estado FROM estados\_pedido WHERE codigo\_estado = p\_estado

    ));

  -- Obtener pedidos paginados

  SELECT COALESCE(jsonb\_agg(

    jsonb\_build\_object(

      'id', p.id\_pedido,

      'numero\_pedido', p.numero\_pedido,

      'fecha', p.fecha\_creacion,

      'estado', jsonb\_build\_object(

        'codigo', ep.codigo\_estado,

        'nombre', ep.nombre\_estado

      ),

      'total', p.total,

      'items', (

        SELECT jsonb\_agg(jsonb\_build\_object(

          'nombre', pr.nombre,

          'cantidad', ip.cantidad,

          'precio\_unitario', ip.precio\_unitario

        ))

        FROM items\_pedido ip

        JOIN productos pr ON ip.id\_producto = pr.id\_producto

        WHERE ip.id\_pedido = p.id\_pedido

      )

    )

    ORDER BY p.fecha\_creacion DESC

  ), '[]'::jsonb) INTO v\_pedidos

  FROM pedidos p

  JOIN estados\_pedido ep ON p.id\_estado = ep.id\_estado

  WHERE p.id\_usuario = p\_id\_usuario

    AND (p\_estado IS NULL OR ep.codigo\_estado = p\_estado)

  LIMIT p\_por\_pagina

  OFFSET (p\_pagina - 1) \* p\_por\_pagina;

  -- Construir respuesta con paginación

  SELECT jsonb\_build\_object(

    'pedidos', v\_pedidos,

    'paginacion', jsonb\_build\_object(

      'pagina\_actual', p\_pagina,

      'por\_pagina', p\_por\_pagina,

      'total', v\_total,

      'total\_paginas', CEIL(v\_total::FLOAT / p\_por\_pagina)

    )

  ) INTO v\_resultado;

  RETURN v\_resultado;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_customer.obtener\_detalle\_pedido(

  p\_id\_usuario UUID,

  p\_id\_pedido UUID

) RETURNS JSONB AS $$

DECLARE

  v\_pedido JSONB;

BEGIN

  SELECT jsonb\_build\_object(

    'id', p.id\_pedido,

    'numero\_pedido', p.numero\_pedido,

    'fecha', p.fecha\_creacion,

    'estado', jsonb\_build\_object(

      'codigo', ep.codigo\_estado,

      'nombre', ep.nombre\_estado,

      'es\_final', ep.es\_final

    ),

    'metodo\_pago', jsonb\_build\_object(

      'nombre', mp.nombre,

      'descripcion', mp.descripcion

    ),

    'pago', jsonb\_build\_object(

      'estado', CASE WHEN p.estado\_pago THEN 'pagado' ELSE 'pendiente' END,

      'subtotal', p.subtotal,

      'impuestos', p.impuestos,

      'envio', p.costo\_envio,

      'total', p.total

    ),

    'direccion\_entrega', p.direccion,

    'items', (

      SELECT jsonb\_agg(jsonb\_build\_object(

        'id', ip.id\_producto,

        'nombre', pr.nombre,

        'cantidad', ip.cantidad,

        'precio\_unitario', ip.precio\_unitario,

        'precio\_total', ip.precio\_total,

        'imagen', (

          SELECT url\_imagen

          FROM imagenes\_productos

          WHERE id\_producto = pr.id\_producto AND es\_principal

          LIMIT 1

        )

      ))

      FROM items\_pedido ip

      JOIN productos pr ON ip.id\_producto = pr.id\_producto

      WHERE ip.id\_pedido = p.id\_pedido

    ),

    'historial\_estados', (

      SELECT jsonb\_agg(jsonb\_build\_object(

        'estado', ep.nombre\_estado,

        'fecha', he.fecha\_cambio,

        'notas', he.notas

      ) ORDER BY he.fecha\_cambio DESC)

      FROM historial\_estados\_pedido he

      JOIN estados\_pedido ep ON he.id\_estado = ep.id\_estado

      WHERE he.id\_pedido = p.id\_pedido

    )

  ) INTO v\_pedido

  FROM pedidos p

  JOIN estados\_pedido ep ON p.id\_estado = ep.id\_estado

  LEFT JOIN metodos\_pago mp ON p.id\_metodo\_pago = mp.id\_metodo\_pago

  WHERE p.id\_pedido = p\_id\_pedido AND p.id\_usuario = p\_id\_usuario;

  IF v\_pedido IS NULL THEN

    RETURN jsonb\_build\_object('error', 'Pedido no encontrado o no pertenece al usuario');

  END IF;

  RETURN jsonb\_build\_object('pedido', v\_pedido);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

-- =============================================

-- FUNCIONES DE LISTAS DE DESEOS

-- =============================================

CREATE OR REPLACE FUNCTION api\_customer.crear\_lista\_deseos(

  p\_id\_usuario UUID,

  p\_nombre VARCHAR(100),

  p\_descripcion TEXT DEFAULT NULL,

  p\_es\_privada BOOLEAN DEFAULT TRUE

) RETURNS JSONB AS $$

DECLARE

  v\_id\_lista UUID;

BEGIN

  -- Verificar que no exista otra lista con el mismo nombre para este usuario

  IF EXISTS (

    SELECT 1 FROM listas\_deseos

    WHERE id\_usuario = p\_id\_usuario AND nombre = p\_nombre

  ) THEN

    RETURN jsonb\_build\_object('error', 'Ya tienes una lista con ese nombre');

  END IF;

  INSERT INTO listas\_deseos (

    id\_usuario,

    nombre,

    descripcion,

    es\_privada

  ) VALUES (

    p\_id\_usuario,

    p\_nombre,

    p\_descripcion,

    p\_es\_privada

  ) RETURNING id\_lista\_deseos INTO v\_id\_lista;

  RETURN jsonb\_build\_object(

    'success', true,

    'id\_lista', v\_id\_lista,

    'message', 'Lista de deseos creada correctamente'

  );

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_customer.obtener\_mis\_listas\_deseos(

  p\_id\_usuario UUID

) RETURNS JSONB AS $$

DECLARE

  v\_listas JSONB;

BEGIN

  SELECT COALESCE(jsonb\_agg(

    jsonb\_build\_object(

      'id', id\_lista\_deseos,

      'nombre', nombre,

      'descripcion', descripcion,

      'es\_privada', es\_privada,

      'cantidad\_items', (

        SELECT COUNT(\*)

        FROM items\_lista\_deseos

        WHERE id\_lista\_deseos = ld.id\_lista\_deseos

      ),

      'fecha\_creacion', fecha\_creacion

    )

    ORDER BY fecha\_creacion DESC

  ), '[]'::jsonb) INTO v\_listas

  FROM listas\_deseos ld

  WHERE id\_usuario = p\_id\_usuario;

  RETURN jsonb\_build\_object('listas', v\_listas);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_customer.obtener\_listas\_publicas(

  p\_pagina INTEGER DEFAULT 1,

  p\_por\_pagina INTEGER DEFAULT 10

) RETURNS JSONB AS $$

DECLARE

  v\_listas JSONB;

  v\_total INTEGER;

  v\_resultado JSONB;

BEGIN

  -- Contar total de listas públicas

  SELECT COUNT(\*) INTO v\_total

  FROM listas\_deseos

  WHERE NOT es\_privada;

  -- Obtener listas paginadas

  SELECT COALESCE(jsonb\_agg(

    jsonb\_build\_object(

      'id', ld.id\_lista\_deseos,

      'nombre', ld.nombre,

      'descripcion', ld.descripcion,

      'usuario', jsonb\_build\_object(

        'nombres', u.nombres,

        'apellidos', u.apellidos

      ),

      'cantidad\_items', (

        SELECT COUNT(\*)

        FROM items\_lista\_deseos

        WHERE id\_lista\_deseos = ld.id\_lista\_deseos

      ),

      'items\_recientes', (

        SELECT COALESCE(jsonb\_agg(

          jsonb\_build\_object(

            'id', p.id\_producto,

            'nombre', p.nombre,

            'precio', p.precio,

            'imagen', (

              SELECT url\_imagen

              FROM imagenes\_productos

              WHERE id\_producto = p.id\_producto AND es\_principal

              LIMIT 1

            )

          )

          LIMIT 3

        ), '[]'::jsonb)

        FROM items\_lista\_deseos ild

        JOIN productos p ON ild.id\_producto = p.id\_producto

        WHERE ild.id\_lista\_deseos = ld.id\_lista\_deseos

        ORDER BY ild.fecha\_agregado DESC

      )

    )

  ), '[]'::jsonb) INTO v\_listas

  FROM listas\_deseos ld

  JOIN usuarios u ON ld.id\_usuario = u.id\_usuario

  WHERE NOT ld.es\_privada

  ORDER BY ld.fecha\_creacion DESC

  LIMIT p\_por\_pagina

  OFFSET (p\_pagina - 1) \* p\_por\_pagina;

  -- Construir respuesta con paginación

  SELECT jsonb\_build\_object(

    'listas', v\_listas,

    'paginacion', jsonb\_build\_object(

      'pagina\_actual', p\_pagina,

      'por\_pagina', p\_por\_pagina,

      'total', v\_total,

      'total\_paginas', CEIL(v\_total::FLOAT / p\_por\_pagina)

    )

  ) INTO v\_resultado;

  RETURN v\_resultado;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_customer.actualizar\_lista\_deseos(

  p\_id\_usuario UUID,

  p\_id\_lista UUID,

  p\_nombre VARCHAR(100) DEFAULT NULL,

  p\_descripcion TEXT DEFAULT NULL,

  p\_es\_privada BOOLEAN DEFAULT NULL

) RETURNS JSONB AS $$

BEGIN

  -- Verificar que la lista pertenezca al usuario

  IF NOT EXISTS (

    SELECT 1 FROM listas\_deseos

    WHERE id\_lista\_deseos = p\_id\_lista AND id\_usuario = p\_id\_usuario

  ) THEN

    RETURN jsonb\_build\_object('error', 'Lista no encontrada o no tienes permisos');

  END IF;

  -- Verificar que el nuevo nombre no esté en uso

  IF p\_nombre IS NOT NULL AND EXISTS (

    SELECT 1 FROM listas\_deseos

    WHERE id\_usuario = p\_id\_usuario

      AND nombre = p\_nombre

      AND id\_lista\_deseos != p\_id\_lista

  ) THEN

    RETURN jsonb\_build\_object('error', 'Ya tienes otra lista con ese nombre');

  END IF;

  UPDATE listas\_deseos SET

    nombre = COALESCE(p\_nombre, nombre),

    descripcion = COALESCE(p\_descripcion, descripcion),

    es\_privada = COALESCE(p\_es\_privada, es\_privada)

  WHERE id\_lista\_deseos = p\_id\_lista;

  RETURN jsonb\_build\_object('success', true, 'message', 'Lista actualizada correctamente');

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_customer.eliminar\_lista\_deseos(

  p\_id\_usuario UUID,

  p\_id\_lista UUID

) RETURNS JSONB AS $$

BEGIN

  -- Verificar que la lista pertenezca al usuario

  IF NOT EXISTS (

    SELECT 1 FROM listas\_deseos

    WHERE id\_lista\_deseos = p\_id\_lista AND id\_usuario = p\_id\_usuario

  ) THEN

    RETURN jsonb\_build\_object('error', 'Lista no encontrada o no tienes permisos');

  END IF;

  DELETE FROM listas\_deseos

  WHERE id\_lista\_deseos = p\_id\_lista;

  RETURN jsonb\_build\_object('success', true, 'message', 'Lista eliminada correctamente');

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_customer.agregar\_producto\_lista(

  p\_id\_usuario UUID,

  p\_id\_lista UUID,

  p\_id\_producto UUID

) RETURNS JSONB AS $$

BEGIN

  -- Verificar que la lista pertenezca al usuario

  IF NOT EXISTS (

    SELECT 1 FROM listas\_deseos

    WHERE id\_lista\_deseos = p\_id\_lista AND id\_usuario = p\_id\_usuario

  ) THEN

    RETURN jsonb\_build\_object('error', 'Lista no encontrada o no tienes permisos');

  END IF;

  -- Verificar que el producto exista y esté activo

  IF NOT EXISTS (

    SELECT 1 FROM productos

    WHERE id\_producto = p\_id\_producto AND esta\_activo

  ) THEN

    RETURN jsonb\_build\_object('error', 'Producto no encontrado o no disponible');

  END IF;

  -- Verificar que el producto no esté ya en la lista

  IF EXISTS (

    SELECT 1 FROM items\_lista\_deseos

    WHERE id\_lista\_deseos = p\_id\_lista AND id\_producto = p\_id\_producto

  ) THEN

    RETURN jsonb\_build\_object('error', 'Este producto ya está en la lista');

  END IF;

  INSERT INTO items\_lista\_deseos (

    id\_lista\_deseos,

    id\_producto

  ) VALUES (

    p\_id\_lista,

    p\_id\_producto

  );

  RETURN jsonb\_build\_object('success', true, 'message', 'Producto agregado a la lista');

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_customer.eliminar\_producto\_lista(

  p\_id\_usuario UUID,

  p\_id\_lista UUID,

  p\_id\_producto UUID

) RETURNS JSONB AS $$

BEGIN

  -- Verificar que la lista pertenezca al usuario

  IF NOT EXISTS (

    SELECT 1 FROM listas\_deseos

    WHERE id\_lista\_deseos = p\_id\_lista AND id\_usuario = p\_id\_usuario

  ) THEN

    RETURN jsonb\_build\_object('error', 'Lista no encontrada o no tienes permisos');

  END IF;

  DELETE FROM items\_lista\_deseos

  WHERE id\_lista\_deseos = p\_id\_lista AND id\_producto = p\_id\_producto;

  IF NOT FOUND THEN

    RETURN jsonb\_build\_object('error', 'Producto no encontrado en la lista');

  END IF;

  RETURN jsonb\_build\_object('success', true, 'message', 'Producto eliminado de la lista');

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_customer.obtener\_detalle\_lista\_deseos(

  p\_id\_usuario UUID,

  p\_id\_lista UUID

) RETURNS JSONB AS $$

DECLARE

  v\_lista JSONB;

  v\_pertenece\_al\_usuario BOOLEAN;

BEGIN

  -- Verificar si la lista es privada y pertenece al usuario

  SELECT

    jsonb\_build\_object(

      'id', ld.id\_lista\_deseos,

      'nombre', ld.nombre,

      'descripcion', ld.descripcion,

      'es\_privada', ld.es\_privada,

      'fecha\_creacion', ld.fecha\_creacion,

      'usuario', jsonb\_build\_object(

        'nombres', u.nombres,

        'apellidos', u.apellidos

      ),

      'items', COALESCE(

        (

          SELECT jsonb\_agg(

            jsonb\_build\_object(

              'id', p.id\_producto,

              'nombre', p.nombre,

              'precio', p.precio,

              'imagen', (

                SELECT url\_imagen

                FROM imagenes\_productos

                WHERE id\_producto = p.id\_producto AND es\_principal

                LIMIT 1

              ),

              'disponible', p.esta\_activo AND p.cantidad\_disponible > 0

            )

          )

          FROM items\_lista\_deseos ild

          JOIN productos p ON ild.id\_producto = p.id\_producto

          WHERE ild.id\_lista\_deseos = ld.id\_lista\_deseos

        ), '[]'::jsonb

      )

    ),

    (ld.id\_usuario = p\_id\_usuario)

  INTO v\_lista, v\_pertenece\_al\_usuario

  FROM listas\_deseos ld

  JOIN usuarios u ON ld.id\_usuario = u.id\_usuario

  WHERE ld.id\_lista\_deseos = p\_id\_lista

    AND (NOT ld.es\_privada OR ld.id\_usuario = p\_id\_usuario);

  IF v\_lista IS NULL THEN

    RETURN jsonb\_build\_object('error', 'Lista no encontrada o es privada');

  END IF;

  -- Agregar flag de pertenencia

  SELECT jsonb\_set(v\_lista, '{pertenece\_al\_usuario}', to\_jsonb(v\_pertenece\_al\_usuario)) INTO v\_lista;

  RETURN jsonb\_build\_object('lista', v\_lista);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

-- =============================================

-- FUNCIONES DE CARRITO DE COMPRAS

-- =============================================

CREATE OR REPLACE FUNCTION api\_customer.obtener\_mi\_carrito(

  p\_id\_usuario UUID

) RETURNS JSONB AS $$

DECLARE

  v\_id\_carrito UUID;

  v\_carrito JSONB;

BEGIN

  -- Obtener o crear carrito

  SELECT id\_carrito INTO v\_id\_carrito

  FROM carritos\_compras

  WHERE id\_usuario = p\_id\_usuario;

  IF v\_id\_carrito IS NULL THEN

    INSERT INTO carritos\_compras (id\_usuario)

    VALUES (p\_id\_usuario)

    RETURNING id\_carrito INTO v\_id\_carrito;

  END IF;

  -- Obtener items del carrito

  SELECT jsonb\_build\_object(

    'id\_carrito', v\_id\_carrito,

    'items', COALESCE(

      (

        SELECT jsonb\_agg(

          jsonb\_build\_object(

            'id\_item', ic.id\_item\_carrito,

            'id\_producto', p.id\_producto,

            'nombre', p.nombre,

            'precio', p.precio,

            'cantidad', ic.cantidad,

            'subtotal', (p.precio \* ic.cantidad),

            'imagen', (

              SELECT url\_imagen

              FROM imagenes\_productos

              WHERE id\_producto = p.id\_producto AND es\_principal

              LIMIT 1

            ),

            'disponible', p.esta\_activo AND p.cantidad\_disponible >= ic.cantidad

          )

        )

        FROM items\_carrito ic

        JOIN productos p ON ic.id\_producto = p.id\_producto

        WHERE ic.id\_carrito = v\_id\_carrito

      ), '[]'::jsonb

    ),

    'resumen', (

      SELECT jsonb\_build\_object(

        'subtotal', SUM(p.precio \* ic.cantidad),

        'total\_items', SUM(ic.cantidad),

        'total\_productos', COUNT(\*)

      )

      FROM items\_carrito ic

      JOIN productos p ON ic.id\_producto = p.id\_producto

      WHERE ic.id\_carrito = v\_id\_carrito

    )

  ) INTO v\_carrito;

  RETURN v\_carrito;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_customer.agregar\_producto\_carrito(

  p\_id\_usuario UUID,

  p\_id\_producto UUID,

  p\_cantidad INTEGER DEFAULT 1

) RETURNS JSONB AS $$

DECLARE

  v\_id\_carrito UUID;

  v\_disponibilidad INTEGER;

BEGIN

  -- Validar cantidad

  IF p\_cantidad < 1 THEN

    RETURN jsonb\_build\_object('error', 'La cantidad debe ser al menos 1');

  END IF;

  -- Verificar disponibilidad del producto

  SELECT cantidad\_disponible INTO v\_disponibilidad

  FROM productos

  WHERE id\_producto = p\_id\_producto AND esta\_activo;

  IF v\_disponibilidad IS NULL THEN

    RETURN jsonb\_build\_object('error', 'Producto no encontrado o no disponible');

  END IF;

  IF v\_disponibilidad < p\_cantidad THEN

    RETURN jsonb\_build\_object(

      'error', 'No hay suficiente stock',

      'disponible', v\_disponibilidad

    );

  END IF;

  -- Obtener o crear carrito

  SELECT id\_carrito INTO v\_id\_carrito

  FROM carritos\_compras

  WHERE id\_usuario = p\_id\_usuario;

  IF v\_id\_carrito IS NULL THEN

    INSERT INTO carritos\_compras (id\_usuario)

    VALUES (p\_id\_usuario)

    RETURNING id\_carrito INTO v\_id\_carrito;

  END IF;

  -- Agregar o actualizar producto en carrito

  INSERT INTO items\_carrito (

    id\_carrito,

    id\_producto,

    cantidad

  ) VALUES (

    v\_id\_carrito,

    p\_id\_producto,

    p\_cantidad

  ) ON CONFLICT (id\_carrito, id\_producto)

  DO UPDATE SET cantidad = items\_carrito.cantidad + EXCLUDED.cantidad;

  RETURN jsonb\_build\_object(

    'success', true,

    'message', 'Producto agregado al carrito',

    'id\_carrito', v\_id\_carrito

  );

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_customer.actualizar\_cantidad\_carrito(

  p\_id\_usuario UUID,

  p\_id\_item\_carrito INTEGER,

  p\_cantidad INTEGER

) RETURNS JSONB AS $$

DECLARE

  v\_id\_carrito UUID;

  v\_disponibilidad INTEGER;

  v\_id\_producto UUID;

BEGIN

  -- Validar cantidad

  IF p\_cantidad < 1 THEN

    RETURN jsonb\_build\_object('error', 'La cantidad debe ser al menos 1');

  END IF;

  -- Verificar que el item pertenezca al usuario

  SELECT cc.id\_carrito, ic.id\_producto INTO v\_id\_carrito, v\_id\_producto

  FROM items\_carrito ic

  JOIN carritos\_compras cc ON ic.id\_carrito = cc.id\_carrito

  WHERE ic.id\_item\_carrito = p\_id\_item\_carrito

    AND cc.id\_usuario = p\_id\_usuario;

  IF v\_id\_carrito IS NULL THEN

    RETURN jsonb\_build\_object('error', 'Ítem no encontrado o no pertenece al usuario');

  END IF;

  -- Verificar disponibilidad

  SELECT cantidad\_disponible INTO v\_disponibilidad

  FROM productos

  WHERE id\_producto = v\_id\_producto AND esta\_activo;

  IF v\_disponibilidad < p\_cantidad THEN

    RETURN jsonb\_build\_object(

      'error', 'No hay suficiente stock',

      'disponible', v\_disponibilidad

    );

  END IF;

  -- Actualizar cantidad

  UPDATE items\_carrito

  SET cantidad = p\_cantidad

  WHERE id\_item\_carrito = p\_id\_item\_carrito;

  RETURN jsonb\_build\_object(

    'success', true,

    'message', 'Cantidad actualizada',

    'id\_carrito', v\_id\_carrito

  );

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_customer.eliminar\_producto\_carrito(

  p\_id\_usuario UUID,

  p\_id\_item\_carrito INTEGER

) RETURNS JSONB AS $$

DECLARE

  v\_id\_carrito UUID;

BEGIN

  -- Verificar que el item pertenezca al usuario

  SELECT cc.id\_carrito INTO v\_id\_carrito

  FROM items\_carrito ic

  JOIN carritos\_compras cc ON ic.id\_carrito = cc.id\_carrito

  WHERE ic.id\_item\_carrito = p\_id\_item\_carrito

    AND cc.id\_usuario = p\_id\_usuario;

  IF v\_id\_carrito IS NULL THEN

    RETURN jsonb\_build\_object('error', 'Ítem no encontrado o no pertenece al usuario');

  END IF;

  -- Eliminar item

  DELETE FROM items\_carrito

  WHERE id\_item\_carrito = p\_id\_item\_carrito;

  RETURN jsonb\_build\_object(

    'success', true,

    'message', 'Producto eliminado del carrito',

    'id\_carrito', v\_id\_carrito

  );

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_customer.mover\_lista\_a\_carrito(

  p\_id\_usuario UUID,

  p\_id\_lista UUID

) RETURNS JSONB AS $$

DECLARE

  v\_id\_carrito UUID;

  v\_productos\_no\_disponibles JSONB;

BEGIN

  -- Verificar que la lista pertenezca al usuario

  IF NOT EXISTS (

    SELECT 1 FROM listas\_deseos

    WHERE id\_lista\_deseos = p\_id\_lista AND id\_usuario = p\_id\_usuario

  ) THEN

    RETURN jsonb\_build\_object('error', 'Lista no encontrada o no tienes permisos');

  END IF;

  -- Obtener o crear carrito

  SELECT id\_carrito INTO v\_id\_carrito

  FROM carritos\_compras

  WHERE id\_usuario = p\_id\_usuario;

  IF v\_id\_carrito IS NULL THEN

    INSERT INTO carritos\_compras (id\_usuario)

    VALUES (p\_id\_usuario)

    RETURNING id\_carrito INTO v\_id\_carrito;

  END IF;

  -- Mover productos disponibles (ignorar los no disponibles)

  WITH productos\_lista AS (

    SELECT ild.id\_producto, 1 AS cantidad

    FROM items\_lista\_deseos ild

    JOIN productos p ON ild.id\_producto = p.id\_producto

    WHERE ild.id\_lista\_deseos = p\_id\_lista

      AND p.esta\_activo

      AND p.cantidad\_disponible > 0

  )

  INSERT INTO items\_carrito (id\_carrito, id\_producto, cantidad)

  SELECT v\_id\_carrito, id\_producto, cantidad

  FROM productos\_lista

  ON CONFLICT (id\_carrito, id\_producto)

  DO UPDATE SET cantidad = items\_carrito.cantidad + EXCLUDED.cantidad;

  -- Obtener lista de productos no disponibles para informar al usuario

  SELECT COALESCE(jsonb\_agg(

    jsonb\_build\_object(

      'id', p.id\_producto,

      'nombre', p.nombre

    )

  ), '[]'::jsonb) INTO v\_productos\_no\_disponibles

  FROM items\_lista\_deseos ild

  JOIN productos p ON ild.id\_producto = p.id\_producto

  WHERE ild.id\_lista\_deseos = p\_id\_lista

    AND (NOT p.esta\_activo OR p.cantidad\_disponible <= 0);

  RETURN jsonb\_build\_object(

    'success', true,

    'message', CASE

      WHEN jsonb\_array\_length(v\_productos\_no\_disponibles) > 0 THEN

        'Productos disponibles agregados al carrito (algunos no estaban disponibles)'

      ELSE

        'Todos los productos fueron agregados al carrito'

    END,

    'productos\_no\_disponibles', v\_productos\_no\_disponibles,

    'id\_carrito', v\_id\_carrito

  );

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

--------------------------------------------------------------

-- Para búsqueda de pedidos por usuario

CREATE INDEX idx\_pedidos\_usuario ON pedidos(id\_usuario);

-- Para items de carrito

CREATE INDEX idx\_items\_carrito ON items\_carrito(id\_carrito, id\_producto);

-- Para listas de deseos

CREATE INDEX idx\_listas\_deseos\_usuario ON listas\_deseos(id\_usuario);

CREATE INDEX idx\_items\_lista\_deseos ON items\_lista\_deseos(id\_lista\_deseos, id\_producto);

-- Para carritos de compra

ALTER TABLE carritos\_compras ENABLE ROW LEVEL SECURITY;

CREATE POLICY carritos\_usuario ON carritos\_compras

  FOR ALL TO api\_customer

  USING (id\_usuario = current\_setting('app.current\_user\_id')::UUID);

-- Para listas de deseos

ALTER TABLE listas\_deseos ENABLE ROW LEVEL SECURITY;

CREATE POLICY listas\_propias ON listas\_deseos

  FOR ALL TO api\_customer

  USING (id\_usuario = current\_setting('app.current\_user\_id')::UUID);

  -- Para carritos de compra

ALTER TABLE carritos\_compras ENABLE ROW LEVEL SECURITY;

CREATE POLICY carritos\_usuario ON carritos\_compras

  FOR ALL TO api\_customer

  USING (id\_usuario = current\_setting('app.current\_user\_id')::UUID);

-- Para listas de deseos

ALTER TABLE listas\_deseos ENABLE ROW LEVEL SECURITY;

CREATE POLICY listas\_propias ON listas\_deseos

  FOR ALL TO api\_customer

  USING (id\_usuario = current\_setting('app.current\_user\_id')::UUID);

* Script de api\_admin

-- =============================================

-- ESQUEMA API\_ADMIN

-- =============================================

CREATE SCHEMA IF NOT EXISTS api\_admin;

GRANT USAGE ON SCHEMA api\_admin TO api\_admin;

ALTER DEFAULT PRIVILEGES IN SCHEMA api\_admin GRANT EXECUTE ON FUNCTIONS TO api\_admin;

-- Vista segura para admins

CREATE VIEW api\_admin.vista\_segura\_usuarios AS

SELECT

  u.id\_usuario,

  u.nombres,

  u.apellidos,

  u.correo\_electronico,

  u.telefono,

  u.esta\_activo,

  r.nombre\_rol AS rol,

  u.fecha\_creacion

FROM usuarios u

JOIN roles r ON u.id\_rol = r.id\_rol;

-- =============================================

-- FUNCIONES DE PERFIL ADMIN

-- =============================================

CREATE OR REPLACE FUNCTION api\_admin.obtener\_mi\_perfil(

  p\_id\_admin UUID

) RETURNS JSONB AS $$

DECLARE

  v\_perfil JSONB;

BEGIN

  SELECT jsonb\_build\_object(

    'id', id\_usuario,

    'nombre\_usuario', nombre\_usuario,

    'nombres', nombres,

    'apellidos', apellidos,

    'correo', correo\_electronico,

    'rol', (SELECT nombre\_rol FROM roles WHERE id\_rol = usuarios.id\_rol),

    'fecha\_registro', fecha\_creacion,

    'permisos', (SELECT permisos FROM roles WHERE id\_rol = usuarios.id\_rol)

  ) INTO v\_perfil

  FROM usuarios

  WHERE id\_usuario = p\_id\_admin

    AND id\_rol IN (SELECT id\_rol FROM roles WHERE nombre\_rol IN ('admin', 'superadmin'));

  IF v\_perfil IS NULL THEN

    RETURN jsonb\_build\_object('error', 'Administrador no encontrado o no tiene privilegios');

  END IF;

  RETURN jsonb\_build\_object('perfil', v\_perfil);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_admin.actualizar\_perfil\_admin(

  p\_id\_admin UUID,

  p\_nombres VARCHAR(100) DEFAULT NULL,

  p\_apellidos VARCHAR(100) DEFAULT NULL

) RETURNS JSONB AS $$

BEGIN

  -- Solo permite actualizar nombres y apellidos

  UPDATE usuarios SET

    nombres = COALESCE(p\_nombres, nombres),

    apellidos = COALESCE(p\_apellidos, apellidos)

  WHERE id\_usuario = p\_id\_admin

    AND id\_rol IN (SELECT id\_rol FROM roles WHERE nombre\_rol IN ('admin', 'superadmin'));

  IF NOT FOUND THEN

    RETURN jsonb\_build\_object('error', 'Administrador no encontrado o no tiene privilegios');

  END IF;

  RETURN jsonb\_build\_object('success', true, 'message', 'Perfil actualizado correctamente');

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_admin.cambiar\_contrasena\_admin(

  p\_id\_admin UUID,

  p\_contrasena\_actual TEXT,

  p\_nueva\_contrasena TEXT

) RETURNS JSONB AS $$

DECLARE

  v\_current\_hash TEXT;

BEGIN

  -- Obtener hash actual verificando rol

  SELECT contrasena\_hash INTO v\_current\_hash

  FROM usuarios

  WHERE id\_usuario = p\_id\_admin

    AND id\_rol IN (SELECT id\_rol FROM roles WHERE nombre\_rol IN ('admin', 'superadmin'));

  IF v\_current\_hash IS NULL THEN

    RETURN jsonb\_build\_object('error', 'Administrador no encontrado o no tiene privilegios');

  END IF;

  -- Verificar contraseña actual

  IF v\_current\_hash != crypt(p\_contrasena\_actual, v\_current\_hash) THEN

    RETURN jsonb\_build\_object('error', 'La contraseña actual no es correcta');

  END IF;

  -- Actualizar contraseña

  UPDATE usuarios

  SET contrasena\_hash = crypt(p\_nueva\_contrasena, gen\_salt('bf'))

  WHERE id\_usuario = p\_id\_admin;

  RETURN jsonb\_build\_object('success', true, 'message', 'Contraseña actualizada correctamente');

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

-- =============================================

-- FUNCIONES DE GESTIÓN DE PEDIDOS

-- =============================================

CREATE OR REPLACE FUNCTION api\_admin.obtener\_todos\_pedidos(

  p\_pagina INTEGER DEFAULT 1,

  p\_por\_pagina INTEGER DEFAULT 20,

  p\_estado VARCHAR DEFAULT NULL,

  p\_desde DATE DEFAULT NULL,

  p\_hasta DATE DEFAULT NULL

) RETURNS JSONB AS $$

DECLARE

  v\_pedidos JSONB;

  v\_total INTEGER;

  v\_resultado JSONB;

BEGIN

  -- Contar total de pedidos con filtros

  SELECT COUNT(\*) INTO v\_total

  FROM pedidos p

  JOIN estados\_pedido ep ON p.id\_estado = ep.id\_estado

  WHERE (p\_estado IS NULL OR ep.codigo\_estado = p\_estado)

    AND (p\_desde IS NULL OR p.fecha\_creacion >= p\_desde)

    AND (p\_hasta IS NULL OR p.fecha\_creacion <= p\_hasta + INTERVAL '1 day');

  -- Obtener pedidos paginados

  SELECT COALESCE(jsonb\_agg(

    jsonb\_build\_object(

      'id', p.id\_pedido,

      'numero\_pedido', p.numero\_pedido,

      'fecha', p.fecha\_creacion,

      'estado', jsonb\_build\_object(

        'codigo', ep.codigo\_estado,

        'nombre', ep.nombre\_estado,

        'es\_final', ep.es\_final

      ),

      'cliente', jsonb\_build\_object(

        'id', u.id\_usuario,

        'nombre', u.nombres || ' ' || u.apellidos

      ),

      'total', p.total,

      'pago', CASE WHEN p.estado\_pago THEN 'pagado' ELSE 'pendiente' END

    )

    ORDER BY p.fecha\_creacion DESC

  ), '[]'::jsonb) INTO v\_pedidos

  FROM pedidos p

  JOIN estados\_pedido ep ON p.id\_estado = ep.id\_estado

  JOIN usuarios u ON p.id\_usuario = u.id\_usuario

  WHERE (p\_estado IS NULL OR ep.codigo\_estado = p\_estado)

    AND (p\_desde IS NULL OR p.fecha\_creacion >= p\_desde)

    AND (p\_hasta IS NULL OR p.fecha\_creacion <= p\_hasta + INTERVAL '1 day')

  LIMIT p\_por\_pagina

  OFFSET (p\_pagina - 1) \* p\_por\_pagina;

  -- Construir respuesta con paginación

  SELECT jsonb\_build\_object(

    'pedidos', v\_pedidos,

    'paginacion', jsonb\_build\_object(

      'pagina\_actual', p\_pagina,

      'por\_pagina', p\_por\_pagina,

      'total', v\_total,

      'total\_paginas', CEIL(v\_total::FLOAT / p\_por\_pagina)

    )

  ) INTO v\_resultado;

  RETURN v\_resultado;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_admin.obtener\_detalle\_pedido\_admin(

  p\_id\_pedido UUID

) RETURNS JSONB AS $$

DECLARE

  v\_pedido JSONB;

BEGIN

  SELECT jsonb\_build\_object(

    'id', p.id\_pedido,

    'numero\_pedido', p.numero\_pedido,

    'fecha', p.fecha\_creacion,

    'estado', jsonb\_build\_object(

      'codigo', ep.codigo\_estado,

      'nombre', ep.nombre\_estado,

      'es\_final', ep.es\_final

    ),

    'cliente', jsonb\_build\_object(

      'id', u.id\_usuario,

      'nombre\_completo', u.nombres || ' ' || u.apellidos,

      'correo', u.correo\_electronico,

      'telefono', u.telefono

    ),

    'metodo\_pago', jsonb\_build\_object(

      'nombre', mp.nombre,

      'descripcion', mp.descripcion

    ),

    'pago', jsonb\_build\_object(

      'estado', CASE WHEN p.estado\_pago THEN 'pagado' ELSE 'pendiente' END,

      'subtotal', p.subtotal,

      'impuestos', p.impuestos,

      'envio', p.costo\_envio,

      'total', p.total

    ),

    'direccion\_entrega', p.direccion,

    'coordenadas\_entrega', ST\_AsGeoJSON(p.coordenadas)::jsonb,

    'items', (

      SELECT jsonb\_agg(jsonb\_build\_object(

        'id', ip.id\_producto,

        'nombre', pr.nombre,

        'cantidad', ip.cantidad,

        'precio\_unitario', ip.precio\_unitario,

        'precio\_total', ip.precio\_total,

        'descuento', ip.descuento,

        'impuestos', ip.impuestos,

        'imagen', (

          SELECT url\_imagen

          FROM imagenes\_productos

          WHERE id\_producto = pr.id\_producto AND es\_principal

          LIMIT 1

        )

      ))

      FROM items\_pedido ip

      JOIN productos pr ON ip.id\_producto = pr.id\_producto

      WHERE ip.id\_pedido = p.id\_pedido

    ),

    'historial\_estados', (

      SELECT jsonb\_agg(jsonb\_build\_object(

        'estado', ep.nombre\_estado,

        'fecha', he.fecha\_cambio,

        'usuario', COALESCE(u.nombres || ' ' || u.apellidos, 'Sistema'),

        'notas', he.notas

      ) ORDER BY he.fecha\_cambio DESC)

      FROM historial\_estados\_pedido he

      JOIN estados\_pedido ep ON he.id\_estado = ep.id\_estado

      LEFT JOIN usuarios u ON he.id\_usuario = u.id\_usuario

      WHERE he.id\_pedido = p.id\_pedido

    ),

    'notas', p.notas

  ) INTO v\_pedido

  FROM pedidos p

  JOIN estados\_pedido ep ON p.id\_estado = ep.id\_estado

  JOIN usuarios u ON p.id\_usuario = u.id\_usuario

  LEFT JOIN metodos\_pago mp ON p.id\_metodo\_pago = mp.id\_metodo\_pago

  WHERE p.id\_pedido = p\_id\_pedido;

  IF v\_pedido IS NULL THEN

    RETURN jsonb\_build\_object('error', 'Pedido no encontrado');

  END IF;

  RETURN jsonb\_build\_object('pedido', v\_pedido);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_admin.actualizar\_estado\_pedido(

  p\_id\_admin UUID,

  p\_id\_pedido UUID,

  p\_nuevo\_estado VARCHAR,

  p\_notas TEXT DEFAULT NULL

) RETURNS JSONB AS $$

DECLARE

  v\_id\_estado INTEGER;

  v\_es\_final BOOLEAN;

  v\_estado\_actual VARCHAR;

BEGIN

  -- Obtener el ID del nuevo estado

  SELECT id\_estado, es\_final INTO v\_id\_estado, v\_es\_final

  FROM estados\_pedido

  WHERE codigo\_estado = p\_nuevo\_estado;

  IF v\_id\_estado IS NULL THEN

    RETURN jsonb\_build\_object('error', 'Estado no válido');

  END IF;

  -- Obtener estado actual del pedido

  SELECT ep.codigo\_estado INTO v\_estado\_actual

  FROM pedidos p

  JOIN estados\_pedido ep ON p.id\_estado = ep.id\_estado

  WHERE p.id\_pedido = p\_id\_pedido;

  IF v\_estado\_actual IS NULL THEN

    RETURN jsonb\_build\_object('error', 'Pedido no encontrado');

  END IF;

  -- Validar transición de estados

  IF v\_estado\_actual = 'cancelado' AND p\_nuevo\_estado != 'cancelado' THEN

    RETURN jsonb\_build\_object('error', 'Pedido cancelado no puede cambiar de estado');

  END IF;

  IF v\_estado\_actual = 'entregado' AND p\_nuevo\_estado != 'entregado' THEN

    RETURN jsonb\_build\_object('error', 'Pedido entregado no puede cambiar de estado');

  END IF;

  -- Actualizar estado del pedido

  UPDATE pedidos

  SET id\_estado = v\_id\_estado

  WHERE id\_pedido = p\_id\_pedido;

  -- Registrar en historial

  INSERT INTO historial\_estados\_pedido (

    id\_pedido, id\_estado, id\_usuario, notas

  ) VALUES (

    p\_id\_pedido, v\_id\_estado, p\_id\_admin, p\_notas

  );

  RETURN jsonb\_build\_object(

    'success', true,

    'message', 'Estado del pedido actualizado',

    'nuevo\_estado', p\_nuevo\_estado,

    'es\_final', v\_es\_final

  );

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

-- =============================================

-- FUNCIONES DE GESTIÓN DE CLIENTES

-- =============================================

CREATE OR REPLACE FUNCTION api\_admin.obtener\_todos\_clientes(

  p\_pagina INTEGER DEFAULT 1,

  p\_por\_pagina INTEGER DEFAULT 20,

  p\_activos BOOLEAN DEFAULT TRUE

) RETURNS JSONB AS $$

DECLARE

  v\_clientes JSONB;

  v\_total INTEGER;

  v\_resultado JSONB;

BEGIN

  -- Contar total de clientes

  SELECT COUNT(\*) INTO v\_total

  FROM usuarios

  WHERE id\_rol = (SELECT id\_rol FROM roles WHERE nombre\_rol = 'cliente')

    AND esta\_activo = p\_activos;

  -- Obtener clientes paginados

  SELECT COALESCE(jsonb\_agg(

    jsonb\_build\_object(

      'id', id\_usuario,

      'nombre\_completo', nombres || ' ' || apellidos,

      'correo', correo\_electronico,

      'telefono', telefono,

      'fecha\_registro', fecha\_creacion,

      'ultimo\_acceso', ultimo\_acceso,

      'esta\_activo', esta\_activo,

      'total\_pedidos', (

        SELECT COUNT(\*)

        FROM pedidos

        WHERE id\_usuario = u.id\_usuario

      ),

      'pedidos\_recientes', (

        SELECT COALESCE(jsonb\_agg(

          jsonb\_build\_object(

            'id', p.id\_pedido,

            'fecha', p.fecha\_creacion,

            'total', p.total,

            'estado', ep.nombre\_estado

          )

          ORDER BY p.fecha\_creacion DESC

          LIMIT 3

        ), '[]'::jsonb)

        FROM pedidos p

        JOIN estados\_pedido ep ON p.id\_estado = ep.id\_estado

        WHERE p.id\_usuario = u.id\_usuario

      )

    )

  ), '[]'::jsonb) INTO v\_clientes

  FROM usuarios u

  WHERE id\_rol = (SELECT id\_rol FROM roles WHERE nombre\_rol = 'cliente')

    AND esta\_activo = p\_activos

  ORDER BY nombres, apellidos

  LIMIT p\_por\_pagina

  OFFSET (p\_pagina - 1) \* p\_por\_pagina;

  -- Construir respuesta con paginación

  SELECT jsonb\_build\_object(

    'clientes', v\_clientes,

    'paginacion', jsonb\_build\_object(

      'pagina\_actual', p\_pagina,

      'por\_pagina', p\_por\_pagina,

      'total', v\_total,

      'total\_paginas', CEIL(v\_total::FLOAT / p\_por\_pagina)

    )

  ) INTO v\_resultado;

  RETURN v\_resultado;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_admin.obtener\_detalle\_cliente(

  p\_id\_cliente UUID

) RETURNS JSONB AS $$

DECLARE

  v\_cliente JSONB;

BEGIN

  SELECT jsonb\_build\_object(

    'id', id\_usuario,

    'nombre\_usuario', nombre\_usuario,

    'nombres', nombres,

    'apellidos', apellidos,

    'correo', correo\_electronico,

    'telefono', telefono,

    'fecha\_registro', fecha\_creacion,

    'ultimo\_acceso', ultimo\_acceso,

    'esta\_activo', esta\_activo,

    'direccion\_principal', (

      SELECT jsonb\_build\_object(

        'direccion', direccion,

        'coordenadas', ST\_AsGeoJSON(coordenadas)::jsonb

      )

      FROM ubicaciones

      WHERE id\_ubicacion = u.id\_ubicacion\_principal

    ),

    'direcciones\_adicionales', direcciones\_adicionales,

    'estadisticas', jsonb\_build\_object(

      'total\_pedidos', (

        SELECT COUNT(\*)

        FROM pedidos

        WHERE id\_usuario = u.id\_usuario

      ),

      'pedidos\_completados', (

        SELECT COUNT(\*)

        FROM pedidos p

        JOIN estados\_pedido ep ON p.id\_estado = ep.id\_estado

        WHERE p.id\_usuario = u.id\_usuario

          AND ep.es\_final AND ep.codigo\_estado != 'cancelado'

      ),

      'total\_gastado', (

        SELECT COALESCE(SUM(total), 0)

        FROM pedidos

        WHERE id\_usuario = u.id\_usuario

      )

    ),

    'listas\_deseos', (

      SELECT COALESCE(jsonb\_agg(

        jsonb\_build\_object(

          'id', id\_lista\_deseos,

          'nombre', nombre,

          'es\_privada', es\_privada,

          'cantidad\_items', (

            SELECT COUNT(\*)

            FROM items\_lista\_deseos

            WHERE id\_lista\_deseos = ld.id\_lista\_deseos

          )

        )

      ), '[]'::jsonb)

      FROM listas\_deseos ld

      WHERE ld.id\_usuario = u.id\_usuario

    )

  ) INTO v\_cliente

  FROM usuarios u

  WHERE id\_usuario = p\_id\_cliente

    AND id\_rol = (SELECT id\_rol FROM roles WHERE nombre\_rol = 'cliente');

  IF v\_cliente IS NULL THEN

    RETURN jsonb\_build\_object('error', 'Cliente no encontrado o no es un cliente válido');

  END IF;

  RETURN jsonb\_build\_object('cliente', v\_cliente);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_admin.obtener\_pedidos\_cliente(

  p\_id\_cliente UUID,

  p\_pagina INTEGER DEFAULT 1,

  p\_por\_pagina INTEGER DEFAULT 10

) RETURNS JSONB AS $$

DECLARE

  v\_pedidos JSONB;

  v\_total INTEGER;

  v\_resultado JSONB;

BEGIN

  -- Verificar que el usuario es un cliente

  IF NOT EXISTS (

    SELECT 1 FROM usuarios

    WHERE id\_usuario = p\_id\_cliente

      AND id\_rol = (SELECT id\_rol FROM roles WHERE nombre\_rol = 'cliente')

  ) THEN

    RETURN jsonb\_build\_object('error', 'ID no corresponde a un cliente válido');

  END IF;

  -- Contar total de pedidos del cliente

  SELECT COUNT(\*) INTO v\_total

  FROM pedidos

  WHERE id\_usuario = p\_id\_cliente;

  -- Obtener pedidos paginados

  SELECT COALESCE(jsonb\_agg(

    jsonb\_build\_object(

      'id', p.id\_pedido,

      'numero\_pedido', p.numero\_pedido,

      'fecha', p.fecha\_creacion,

      'estado', jsonb\_build\_object(

        'codigo', ep.codigo\_estado,

        'nombre', ep.nombre\_estado,

        'es\_final', ep.es\_final

      ),

      'total', p.total,

      'pago', CASE WHEN p.estado\_pago THEN 'pagado' ELSE 'pendiente' END,

      'items', (

        SELECT jsonb\_agg(jsonb\_build\_object(

          'nombre', pr.nombre,

          'cantidad', ip.cantidad,

          'precio\_unitario', ip.precio\_unitario

        ))

        FROM items\_pedido ip

        JOIN productos pr ON ip.id\_producto = pr.id\_producto

        WHERE ip.id\_pedido = p.id\_pedido

      )

    )

    ORDER BY p.fecha\_creacion DESC

  ), '[]'::jsonb) INTO v\_pedidos

  FROM pedidos p

  JOIN estados\_pedido ep ON p.id\_estado = ep.id\_estado

  WHERE p.id\_usuario = p\_id\_cliente

  LIMIT p\_por\_pagina

  OFFSET (p\_pagina - 1) \* p\_por\_pagina;

  -- Construir respuesta con paginación

  SELECT jsonb\_build\_object(

    'pedidos', v\_pedidos,

    'paginacion', jsonb\_build\_object(

      'pagina\_actual', p\_pagina,

      'por\_pagina', p\_por\_pagina,

      'total', v\_total,

      'total\_paginas', CEIL(v\_total::FLOAT / p\_por\_pagina)

    )

  ) INTO v\_resultado;

  RETURN v\_resultado;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

-- =============================================

-- FUNCIONES DE GESTIÓN DE PRODUCTOS

-- =============================================

CREATE OR REPLACE FUNCTION api\_admin.actualizar\_producto(

  p\_id\_admin UUID,

  p\_id\_producto UUID,

  p\_nombre VARCHAR(255) DEFAULT NULL,

  p\_descripcion TEXT DEFAULT NULL,

  p\_precio DECIMAL(10, 2) DEFAULT NULL,

  p\_cantidad\_disponible INTEGER DEFAULT NULL,

  p\_id\_categoria INTEGER DEFAULT NULL,

  p\_marca VARCHAR(100) DEFAULT NULL,

  p\_talla VARCHAR(50) DEFAULT NULL,

  p\_esta\_activo BOOLEAN DEFAULT NULL

) RETURNS JSONB AS $$

DECLARE

  v\_producto\_anterior JSONB;

BEGIN

  -- Obtener datos actuales para auditoría

  SELECT to\_jsonb(productos) INTO v\_producto\_anterior

  FROM productos

  WHERE id\_producto = p\_id\_producto;

  -- Actualizar campos no nulos

  UPDATE productos SET

    nombre = COALESCE(p\_nombre, nombre),

    descripcion = COALESCE(p\_descripcion, descripcion),

    precio = COALESCE(p\_precio, precio),

    cantidad\_disponible = COALESCE(p\_cantidad\_disponible, cantidad\_disponible),

    id\_categoria = COALESCE(p\_id\_categoria, id\_categoria),

    marca = COALESCE(p\_marca, marca),

    talla = COALESCE(p\_talla, talla),

    esta\_activo = COALESCE(p\_esta\_activo, esta\_activo)

  WHERE id\_producto = p\_id\_producto;

  IF NOT FOUND THEN

    RETURN jsonb\_build\_object('error', 'Producto no encontrado');

  END IF;

  -- Registrar cambio de precio si aplica

  IF p\_precio IS NOT NULL AND v\_producto\_anterior->>'precio' != p\_precio::text THEN

    INSERT INTO historial\_precios (

      id\_producto, precio\_anterior, precio\_nuevo, modificado\_por, razon\_cambio

    ) VALUES (

      p\_id\_producto,

      (v\_producto\_anterior->>'precio')::DECIMAL(10, 2),

      p\_precio,

      p\_id\_admin,

      'Actualización manual por administrador'

    );

  END IF;

  -- Registrar cambio de stock si aplica

  IF p\_cantidad\_disponible IS NOT NULL AND

     (v\_producto\_anterior->>'cantidad\_disponible')::INTEGER != p\_cantidad\_disponible THEN

    INSERT INTO registro\_inventario (

      id\_producto, cambio\_cantidad, nueva\_cantidad, tipo\_cambio, modificado\_por

    ) VALUES (

      p\_id\_producto,

      p\_cantidad\_disponible - (v\_producto\_anterior->>'cantidad\_disponible')::INTEGER,

      p\_cantidad\_disponible,

      'ajuste',

      p\_id\_admin

    );

  END IF;

  RETURN jsonb\_build\_object(

    'success', true,

    'message', 'Producto actualizado correctamente',

    'id\_producto', p\_id\_producto

  );

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_admin.actualizar\_detalles\_tecnicos(

  p\_id\_admin UUID,

  p\_id\_detalle\_producto INTEGER,

  p\_tipo\_producto VARCHAR(50) DEFAULT NULL,

  p\_peso DECIMAL(10, 2) DEFAULT NULL,

  p\_dimensiones DECIMAL(10, 2)[] DEFAULT NULL,

  p\_material\_principal VARCHAR(100) DEFAULT NULL,

  p\_materiales\_secundarios JSONB DEFAULT NULL,

  p\_cuidados\_especiales JSONB DEFAULT NULL,

  p\_especificaciones JSONB DEFAULT NULL

) RETURNS JSONB AS $$

BEGIN

  -- Actualizar campos no nulos

  UPDATE detalles\_productos SET

    tipo\_producto = COALESCE(p\_tipo\_producto, tipo\_producto),

    peso = COALESCE(p\_peso, peso),

    dimensiones = COALESCE(p\_dimensiones, dimensiones),

    material\_principal = COALESCE(p\_material\_principal, material\_principal),

    materiales\_secundarios = COALESCE(p\_materiales\_secundarios, materiales\_secundarios),

    cuidados\_especiales = COALESCE(p\_cuidados\_especiales, cuidados\_especiales),

    especificaciones = COALESCE(p\_especificaciones, especificaciones)

  WHERE id\_detalle\_producto = p\_id\_detalle\_producto;

  IF NOT FOUND THEN

    RETURN jsonb\_build\_object('error', 'Detalle técnico no encontrado');

  END IF;

  RETURN jsonb\_build\_object(

    'success', true,

    'message', 'Detalles técnicos actualizados correctamente',

    'id\_detalle\_producto', p\_id\_detalle\_producto

  );

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_admin.agregar\_imagen\_producto(

  p\_id\_admin UUID,

  p\_id\_producto UUID,

  p\_url\_imagen VARCHAR(255),

  p\_texto\_alternativo VARCHAR(255) DEFAULT NULL,

  p\_es\_principal BOOLEAN DEFAULT FALSE

) RETURNS JSONB AS $$

BEGIN

  -- Verificar que el producto existe

  IF NOT EXISTS (SELECT 1 FROM productos WHERE id\_producto = p\_id\_producto) THEN

    RETURN jsonb\_build\_object('error', 'Producto no encontrado');

  END IF;

  -- Si se marca como principal, quitar principal de otras imágenes

  IF p\_es\_principal THEN

    UPDATE imagenes\_productos

    SET es\_principal = FALSE

    WHERE id\_producto = p\_id\_producto;

  END IF;

  -- Insertar nueva imagen

  INSERT INTO imagenes\_productos (

    id\_producto, url\_imagen, texto\_alternativo, es\_principal

  ) VALUES (

    p\_id\_producto, p\_url\_imagen, p\_texto\_alternativo, p\_es\_principal

  );

  RETURN jsonb\_build\_object(

    'success', true,

    'message', 'Imagen agregada correctamente',

    'id\_producto', p\_id\_producto

  );

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_admin.actualizar\_imagen\_producto(

  p\_id\_admin UUID,

  p\_id\_imagen INTEGER,

  p\_texto\_alternativo VARCHAR(255) DEFAULT NULL,

  p\_es\_principal BOOLEAN DEFAULT NULL,

  p\_orden\_visualizacion INTEGER DEFAULT NULL

) RETURNS JSONB AS $$

DECLARE

  v\_id\_producto UUID;

BEGIN

  -- Obtener ID del producto para validación

  SELECT id\_producto INTO v\_id\_producto

  FROM imagenes\_productos

  WHERE id\_imagen = p\_id\_imagen;

  IF v\_id\_producto IS NULL THEN

    RETURN jsonb\_build\_object('error', 'Imagen no encontrada');

  END IF;

  -- Si se marca como principal, quitar principal de otras imágenes

  IF p\_es\_principal = TRUE THEN

    UPDATE imagenes\_productos

    SET es\_principal = FALSE

    WHERE id\_producto = v\_id\_producto;

  END IF;

  -- Actualizar campos no nulos

  UPDATE imagenes\_productos SET

    texto\_alternativo = COALESCE(p\_texto\_alternativo, texto\_alternativo),

    es\_principal = COALESCE(p\_es\_principal, es\_principal),

    orden\_visualizacion = COALESCE(p\_orden\_visualizacion, orden\_visualizacion)

  WHERE id\_imagen = p\_id\_imagen;

  RETURN jsonb\_build\_object(

    'success', true,

    'message', 'Imagen actualizada correctamente',

    'id\_imagen', p\_id\_imagen

  );

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_admin.eliminar\_imagen\_producto(

  p\_id\_admin UUID,

  p\_id\_imagen INTEGER

) RETURNS JSONB AS $$

DECLARE

  v\_es\_principal BOOLEAN;

  v\_id\_producto UUID;

BEGIN

  -- Obtener datos de la imagen

  SELECT es\_principal, id\_producto INTO v\_es\_principal, v\_id\_producto

  FROM imagenes\_productos

  WHERE id\_imagen = p\_id\_imagen;

  IF v\_id\_producto IS NULL THEN

    RETURN jsonb\_build\_object('error', 'Imagen no encontrada');

  END IF;

  -- Eliminar imagen

  DELETE FROM imagenes\_productos

  WHERE id\_imagen = p\_id\_imagen;

  -- Si era la imagen principal, asignar nueva principal si hay más imágenes

  IF v\_es\_principal THEN

    UPDATE imagenes\_productos

    SET es\_principal = TRUE

    WHERE id\_producto = v\_id\_producto

    AND id\_imagen != p\_id\_imagen

    LIMIT 1;

  END IF;

  RETURN jsonb\_build\_object(

    'success', true,

    'message', 'Imagen eliminada correctamente',

    'id\_producto', v\_id\_producto

  );

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

-- =============================================

-- FUNCIONES DE HISTORIAL Y AUDITORÍA

-- =============================================

CREATE OR REPLACE FUNCTION api\_admin.obtener\_historial\_precios(

  p\_id\_producto UUID,

  p\_pagina INTEGER DEFAULT 1,

  p\_por\_pagina INTEGER DEFAULT 10

) RETURNS JSONB AS $$

DECLARE

  v\_historial JSONB;

  v\_total INTEGER;

  v\_resultado JSONB;

BEGIN

  -- Contar total de registros

  SELECT COUNT(\*) INTO v\_total

  FROM historial\_precios

  WHERE id\_producto = p\_id\_producto;

  -- Obtener historial paginado

  SELECT COALESCE(jsonb\_agg(

    jsonb\_build\_object(

      'fecha', fecha\_cambio,

      'precio\_anterior', precio\_anterior,

      'precio\_nuevo', precio\_nuevo,

      'diferencia', precio\_nuevo - precio\_anterior,

      'porcentaje\_cambio', ROUND(((precio\_nuevo - precio\_anterior) / precio\_anterior \* 100), 2),

      'modificado\_por', (

        SELECT nombres || ' ' || apellidos

        FROM usuarios

        WHERE id\_usuario = hp.modificado\_por

      ),

      'razon\_cambio', razon\_cambio

    )

    ORDER BY fecha\_cambio DESC

  ), '[]'::jsonb) INTO v\_historial

  FROM historial\_precios hp

  WHERE id\_producto = p\_id\_producto

  LIMIT p\_por\_pagina

  OFFSET (p\_pagina - 1) \* p\_por\_pagina;

  -- Construir respuesta con paginación

  SELECT jsonb\_build\_object(

    'historial', v\_historial,

    'paginacion', jsonb\_build\_object(

      'pagina\_actual', p\_pagina,

      'por\_pagina', p\_por\_pagina,

      'total', v\_total,

      'total\_paginas', CEIL(v\_total::FLOAT / p\_por\_pagina)

    )

  ) INTO v\_resultado;

  RETURN v\_resultado;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_admin.obtener\_historial\_inventario(

  p\_id\_producto UUID,

  p\_pagina INTEGER DEFAULT 1,

  p\_por\_pagina INTEGER DEFAULT 10

) RETURNS JSONB AS $$

DECLARE

  v\_historial JSONB;

  v\_total INTEGER;

  v\_resultado JSONB;

BEGIN

  -- Contar total de registros

  SELECT COUNT(\*) INTO v\_total

  FROM registro\_inventario

  WHERE id\_producto = p\_id\_producto;

  -- Obtener historial paginado

  SELECT COALESCE(jsonb\_agg(

    jsonb\_build\_object(

      'fecha', fecha\_cambio,

      'tipo\_cambio', tipo\_cambio,

      'cambio\_cantidad', cambio\_cantidad,

      'nueva\_cantidad', nueva\_cantidad,

      'modificado\_por', (

        SELECT nombres || ' ' || apellidos

        FROM usuarios

        WHERE id\_usuario = ri.modificado\_por

      ),

      'referencia', CASE

        WHEN tipo\_referencia IS NOT NULL THEN jsonb\_build\_object(

          'tipo', tipo\_referencia,

          'id', id\_referencia

        )

        ELSE NULL

      END,

      'notas', notas

    )

    ORDER BY fecha\_cambio DESC

  ), '[]'::jsonb) INTO v\_historial

  FROM registro\_inventario ri

  WHERE id\_producto = p\_id\_producto

  LIMIT p\_por\_pagina

  OFFSET (p\_pagina - 1) \* p\_por\_pagina;

  -- Construir respuesta con paginación

  SELECT jsonb\_build\_object(

    'historial', v\_historial,

    'paginacion', jsonb\_build\_object(

      'pagina\_actual', p\_pagina,

      'por\_pagina', p\_por\_pagina,

      'total', v\_total,

      'total\_paginas', CEIL(v\_total::FLOAT / p\_por\_pagina)

    )

  ) INTO v\_resultado;

  RETURN v\_resultado;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

--------------------------------------------------------------

-- Para búsqueda de pedidos

CREATE INDEX idx\_pedidos\_fecha ON pedidos(fecha\_creacion);

CREATE INDEX idx\_pedidos\_estado ON pedidos(id\_estado);

-- Para historiales

CREATE INDEX idx\_historial\_precios\_producto ON historial\_precios(id\_producto, fecha\_cambio);

CREATE INDEX idx\_inventario\_producto ON registro\_inventario(id\_producto, fecha\_cambio);

-- Para productos

ALTER TABLE productos ENABLE ROW LEVEL SECURITY;

CREATE POLICY solo\_lectura\_productos ON productos

  FOR SELECT TO api\_admin

  USING (true);

-- Para historiales

ALTER TABLE historial\_precios ENABLE ROW LEVEL SECURITY;

CREATE POLICY solo\_lectura\_historial ON historial\_precios

  FOR SELECT TO api\_admin

  USING (true);

-- Vistas materializadas para reportes

  CREATE MATERIALIZED VIEW mv\_estadisticas\_ventas AS

SELECT

  DATE\_TRUNC('month', fecha\_creacion) AS mes,

  COUNT(\*) AS total\_pedidos,

  SUM(total) AS ingresos\_totales,

  AVG(total) AS promedio\_por\_pedido,

  COUNT(DISTINCT id\_usuario) AS clientes\_unicos

FROM pedidos

GROUP BY DATE\_TRUNC('month', fecha\_creacion);

CREATE UNIQUE INDEX idx\_mv\_estadisticas\_ventas\_mes ON mv\_estadisticas\_ventas(mes);

* Script de api\_delivery

-- =============================================

-- ESQUEMA API\_DELIVERY

-- =============================================

CREATE SCHEMA IF NOT EXISTS api\_delivery;

GRANT USAGE ON SCHEMA api\_delivery TO api\_delivery;

ALTER DEFAULT PRIVILEGES IN SCHEMA api\_delivery GRANT EXECUTE ON FUNCTIONS TO api\_delivery;

-- Vista segura para repartidores

CREATE VIEW api\_delivery.vista\_segura\_pedidos AS

SELECT

  p.numero\_pedido,

  ep.nombre\_estado AS estado,

  p.direccion,

  p.coordenadas,

  u.nombres || ' ' || u.apellidos AS cliente,

  u.telefono

FROM pedidos p

JOIN estados\_pedido ep ON p.id\_estado = ep.id\_estado

JOIN usuarios u ON p.id\_usuario = u.id\_usuario

WHERE ep.codigo\_estado IN ('por\_enviar', 'enviado');

-- =============================================

-- FUNCIONES DE GESTIÓN DE PEDIDOS PARA DELIVERY

-- =============================================

CREATE OR REPLACE FUNCTION api\_delivery.obtener\_pedidos\_para\_entrega()

RETURNS JSONB AS $$

DECLARE

  v\_pedidos JSONB;

BEGIN

  SELECT COALESCE(jsonb\_agg(

    jsonb\_build\_object(

      'numero\_pedido', p.numero\_pedido,

      'estado', jsonb\_build\_object(

        'codigo', ep.codigo\_estado,

        'nombre', ep.nombre\_estado

      ),

      'fecha\_creacion', p.fecha\_creacion,

      'total', p.total

    )

    ORDER BY

      CASE WHEN ep.codigo\_estado = 'por\_enviar' THEN 0 ELSE 1 END,

      p.fecha\_creacion

  ), '[]'::jsonb) INTO v\_pedidos

  FROM pedidos p

  JOIN estados\_pedido ep ON p.id\_estado = ep.id\_estado

  WHERE ep.codigo\_estado IN ('por\_enviar', 'enviado');

  RETURN jsonb\_build\_object('pedidos', v\_pedidos);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_delivery.actualizar\_estado\_a\_enviado(

  p\_id\_repartidor UUID,

  p\_numero\_pedido VARCHAR(50),

  p\_notas TEXT DEFAULT NULL

) RETURNS JSONB AS $$

DECLARE

  v\_id\_pedido UUID;

  v\_estado\_actual VARCHAR;

  v\_id\_estado\_enviado INTEGER;

BEGIN

  -- Obtener ID del pedido y estado actual

  SELECT p.id\_pedido, ep.codigo\_estado INTO v\_id\_pedido, v\_estado\_actual

  FROM pedidos p

  JOIN estados\_pedido ep ON p.id\_estado = ep.id\_estado

  WHERE p.numero\_pedido = p\_numero\_pedido;

  IF v\_id\_pedido IS NULL THEN

    RETURN jsonb\_build\_object('error', 'Pedido no encontrado');

  END IF;

  -- Validar que el estado actual sea "por enviar"

  IF v\_estado\_actual != 'por\_enviar' THEN

    RETURN jsonb\_build\_object(

      'error', 'Solo se pueden marcar como enviado pedidos en estado "por enviar"',

      'estado\_actual', v\_estado\_actual

    );

  END IF;

  -- Obtener ID del estado "enviado"

  SELECT id\_estado INTO v\_id\_estado\_enviado

  FROM estados\_pedido

  WHERE codigo\_estado = 'enviado';

  -- Actualizar estado del pedido

  UPDATE pedidos

  SET id\_estado = v\_id\_estado\_enviado

  WHERE id\_pedido = v\_id\_pedido;

  -- Registrar en historial

  INSERT INTO historial\_estados\_pedido (

    id\_pedido, id\_estado, id\_usuario, notas

  ) VALUES (

    v\_id\_pedido, v\_id\_estado\_enviado, p\_id\_repartidor,

    COALESCE(p\_notas, 'Marcado como enviado por el repartidor')

  );

  RETURN jsonb\_build\_object(

    'success', true,

    'message', 'Pedido marcado como enviado',

    'numero\_pedido', p\_numero\_pedido,

    'nuevo\_estado', 'enviado'

  );

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_delivery.obtener\_direccion\_entrega(

  p\_numero\_pedido VARCHAR(50)

RETURNS JSONB AS $$

DECLARE

  v\_direccion JSONB;

BEGIN

  SELECT jsonb\_build\_object(

    'direccion', p.direccion,

    'coordenadas', ST\_AsGeoJSON(p.coordenadas)::jsonb,

    'instrucciones\_entrega', p.notas

  ) INTO v\_direccion

  FROM pedidos p

  JOIN estados\_pedido ep ON p.id\_estado = ep.id\_estado

  WHERE p.numero\_pedido = p\_numero\_pedido

    AND ep.codigo\_estado IN ('por\_enviar', 'enviado');

  IF v\_direccion IS NULL THEN

    RETURN jsonb\_build\_object(

      'error', 'Pedido no encontrado o no está en estado válido para entrega',

      'estados\_validos', ARRAY['por\_enviar', 'enviado']

    );

  END IF;

  RETURN jsonb\_build\_object('direccion\_entrega', v\_direccion);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_delivery.obtener\_info\_contacto\_pedido(

  p\_numero\_pedido VARCHAR(50)

RETURNS JSONB AS $$

DECLARE

  v\_contacto JSONB;

BEGIN

  SELECT jsonb\_build\_object(

    'cliente', jsonb\_build\_object(

      'nombre\_completo', u.nombres || ' ' || u.apellidos,

      'telefono', u.telefono,

      'correo', u.correo\_electronico

    ),

    'remitente', p.informacion\_remitente

  ) INTO v\_contacto

  FROM pedidos p

  JOIN usuarios u ON p.id\_usuario = u.id\_usuario

  JOIN estados\_pedido ep ON p.id\_estado = ep.id\_estado

  WHERE p.numero\_pedido = p\_numero\_pedido

    AND ep.codigo\_estado IN ('por\_enviar', 'enviado');

  IF v\_contacto IS NULL THEN

    RETURN jsonb\_build\_object(

      'error', 'Pedido no encontrado o no está en estado válido para entrega',

      'estados\_validos', ARRAY['por\_enviar', 'enviado']

    );

  END IF;

  RETURN jsonb\_build\_object('contacto', v\_contacto);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION api\_delivery.obtener\_detalle\_pedido\_entrega(

  p\_numero\_pedido VARCHAR(50)

RETURNS JSONB AS $$

DECLARE

  v\_pedido JSONB;

BEGIN

  SELECT jsonb\_build\_object(

    'numero\_pedido', p.numero\_pedido,

    'fecha\_creacion', p.fecha\_creacion,

    'estado', jsonb\_build\_object(

      'codigo', ep.codigo\_estado,

      'nombre', ep.nombre\_estado

    ),

    'direccion\_entrega', jsonb\_build\_object(

      'direccion', p.direccion,

      'coordenadas', ST\_AsGeoJSON(p.coordenadas)::jsonb,

      'instrucciones', p.notas

    ),

    'contacto', jsonb\_build\_object(

      'cliente', jsonb\_build\_object(

        'nombre', u.nombres || ' ' || u.apellidos,

        'telefono', u.telefono

      ),

      'remitente', p.informacion\_remitente

    ),

    'items', (

      SELECT jsonb\_agg(jsonb\_build\_object(

        'nombre', pr.nombre,

        'cantidad', ip.cantidad

      ))

      FROM items\_pedido ip

      JOIN productos pr ON ip.id\_producto = pr.id\_producto

      WHERE ip.id\_pedido = p.id\_pedido

    ),

    'total', p.total

  ) INTO v\_pedido

  FROM pedidos p

  JOIN estados\_pedido ep ON p.id\_estado = ep.id\_estado

  JOIN usuarios u ON p.id\_usuario = u.id\_usuario

  WHERE p.numero\_pedido = p\_numero\_pedido

    AND ep.codigo\_estado IN ('por\_enviar', 'enviado');

  IF v\_pedido IS NULL THEN

    RETURN jsonb\_build\_object(

      'error', 'Pedido no encontrado o no está en estado válido para entrega',

      'estados\_validos', ARRAY['por\_enviar', 'enviado']

    );

  END IF;

  RETURN jsonb\_build\_object('pedido', v\_pedido);

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

--------------------------------------------------------------

-- Para búsqueda rápida por número de pedido

CREATE INDEX idx\_pedidos\_numero ON pedidos(numero\_pedido);

-- Para filtrado por estado

CREATE INDEX idx\_pedidos\_estado ON pedidos(id\_estado)

WHERE id\_estado IN (

  SELECT id\_estado FROM estados\_pedido

  WHERE codigo\_estado IN ('por\_enviar', 'enviado')

);

ALTER TABLE pedidos ENABLE ROW LEVEL SECURITY;

CREATE POLICY pedidos\_para\_entrega ON pedidos

  FOR SELECT TO api\_delivery

  USING (id\_estado IN (

    SELECT id\_estado FROM estados\_pedido

    WHERE codigo\_estado IN ('por\_enviar', 'enviado')

  ));

CREATE VIEW vista\_pedidos\_para\_entrega AS

SELECT

  p.numero\_pedido,

  ep.nombre\_estado AS estado,

  p.direccion,

  p.coordenadas,

  u.nombres || ' ' || u.apellidos AS cliente,

  u.telefono,

  p.informacion\_remitente,

  p.notas AS instrucciones\_entrega

FROM pedidos p

JOIN estados\_pedido ep ON p.id\_estado = ep.id\_estado

JOIN usuarios u ON p.id\_usuario = u.id\_usuario

WHERE ep.codigo\_estado IN ('por\_enviar', 'enviado');