```
-- En el documento se encuentra la explicación de cada trigger junto lo que se
solicita.

--  BEFORE INSERT - Validación de stock en ingredientes
CREATE OR REPLACE FUNCTION validar_stock_ingrediente()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.stock < 0 THEN     -- Validar que el stock no sea negativo
        RAISE EXCEPTION 'No se puede ingresar un stock negativo para el ingrediente
%', NEW.nombre;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_ingrediente
BEFORE INSERT ON ingredientes
FOR EACH ROW
EXECUTE FUNCTION validar_stock_ingrediente();


-- AFTER INSERT - Registro de nuevos usuarios
CREATE TABLE Nuevos_usuarios (
    id_log SERIAL PRIMARY KEY,
    id_usuario INTEGER,
    operacion VARCHAR(20),
    fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    usuario_db VARCHAR(50),
    detalle TEXT
);

CREATE OR REPLACE FUNCTION registrar_nuevo_usuario()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO Nuevos_usuarios (id_usuario, operacion, usuario_db, detalle)
    VALUES (NEW.id_usuario, 'INSERT', current_user,
            'Nuevo usuario registrado: ' || NEW.nombre || ' ' || NEW.apellidos || '
(' || NEW.email || ')');

    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_insert_usuario
AFTER INSERT ON usuarios
FOR EACH ROW
EXECUTE FUNCTION registrar_nuevo_usuario();


-- BEFORE UPDATE - Validación de precio en kits
CREATE OR REPLACE FUNCTION validar_precio_kit()
RETURNS TRIGGER AS $$
BEGIN
    -- Validar que el precio no sea menor que el descuento
    IF NEW.precio <= NEW.descuento THEN
        RAISE EXCEPTION 'El precio del kit % no puede ser menor o igual que el
descuento', NEW.nombre;
    END IF;
```

```
    -- Si se esta cambiando el precio se valida que no sea menor al 50% del precio
original
    IF TG_OP = 'UPDATE' AND OLD.precio IS NOT NULL AND NEW.precio < OLD.precio *
0.5 THEN
        RAISE EXCEPTION 'El nuevo precio (%) no puede ser menor al 50%% del precio
original (%)',
                        NEW.precio, OLD.precio;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_update_kit
BEFORE UPDATE ON kits
FOR EACH ROW
EXECUTE FUNCTION validar_precio_kit();


-- AFTER UPDATE - Registro de cambios de estado en pedidos
CREATE TABLE log_pedidos (
    id_log SERIAL PRIMARY KEY,
    id_pedido INTEGER,
    estado_anterior VARCHAR(20),
    estado_nuevo VARCHAR(20),
    fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    usuario_db VARCHAR(50)
);

CREATE OR REPLACE FUNCTION registrar_cambio_estado_pedido()
RETURNS TRIGGER AS $$
BEGIN
    -- Solo registrar cuando cambia el estado
    IF OLD.estado <> NEW.estado THEN
        INSERT INTO log_pedidos (id_pedido, estado_anterior, estado_nuevo,
usuario_db)
        VALUES (NEW.id_pedido, OLD.estado, NEW.estado, current_user);
    END IF;

    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_update_pedido
AFTER UPDATE ON pedidos
FOR EACH ROW
EXECUTE FUNCTION registrar_cambio_estado_pedido();


-- BEFORE DELETE - Protección contra eliminación de categorias usadas
CREATE OR REPLACE FUNCTION proteger_categoria_en_uso()
RETURNS TRIGGER AS $$
DECLARE
    receta_count INTEGER;
BEGIN
    -- Verificar si hay recetas que usan esta categoria
    SELECT COUNT(*) INTO receta_count FROM recetas WHERE id_categoria =
OLD.id_categoria;
```

```
    IF receta_count > 0 THEN
        RAISE EXCEPTION 'No se puede eliminar la categoría % porque esta siendo
utilizada por % recetas',
                        OLD.nombre, receta_count;
    END IF;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_delete_categoria
BEFORE DELETE ON categorias
FOR EACH ROW
EXECUTE FUNCTION proteger_categoria_en_uso();


-- AFTER DELETE - Registro de eliminación de kits
CREATE TABLE kits_eliminados (
    id_log SERIAL PRIMARY KEY,
    id_kit INTEGER,
    id_receta INTEGER,
    nombre VARCHAR(100),
    precio DECIMAL(10,2),
    stock INTEGER,
    fecha_eliminacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    usuario_db VARCHAR(50)
);

CREATE OR REPLACE FUNCTION registrar_kit_eliminado()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO kits_eliminados (id_kit, id_receta, nombre, precio, stock,
usuario_db)
    VALUES (OLD.id_kit, OLD.id_receta, OLD.nombre, OLD.precio, OLD.stock,
current_user);

    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_delete_kit
AFTER DELETE ON kits
FOR EACH ROW
EXECUTE FUNCTION registrar_kit_eliminado();


-- BEFORE TRUNCATE - Prevención de truncado de tablas criticas
CREATE OR REPLACE FUNCTION prevenir_truncado()
RETURNS TRIGGER AS $$
BEGIN
    RAISE EXCEPTION 'No se permite el truncado de la tabla %. Use un DELETE con
condiciones especificas si usted desea eliminar datos.', TG_TABLE_NAME;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_truncate_usuarios
BEFORE TRUNCATE ON usuarios
```

```
FOR STATEMENT
EXECUTE FUNCTION prevenir_truncado();

CREATE TRIGGER before_truncate_pedidos
BEFORE TRUNCATE ON pedidos
FOR STATEMENT
EXECUTE FUNCTION prevenir_truncado();

CREATE TRIGGER before_truncate_recetas
BEFORE TRUNCATE ON recetas
FOR STATEMENT
EXECUTE FUNCTION prevenir_truncado();


-- AFTER TRUNCATE - Registro de operaciones de truncado
CREATE TABLE log_truncados (
    id_log SERIAL PRIMARY KEY,
    tabla VARCHAR(100),
    fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    usuario_db VARCHAR(50)
);

CREATE OR REPLACE FUNCTION registrar_truncado()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO log_truncados (tabla, usuario_db)
    VALUES (TG_TABLE_NAME, current_user);

    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_truncate_ingredientes
AFTER TRUNCATE ON ingredientes
FOR STATEMENT
EXECUTE FUNCTION registrar_truncado();

CREATE TRIGGER after_truncate_categorias
AFTER TRUNCATE ON categorias
FOR STATEMENT
EXECUTE FUNCTION registrar_truncado();
```