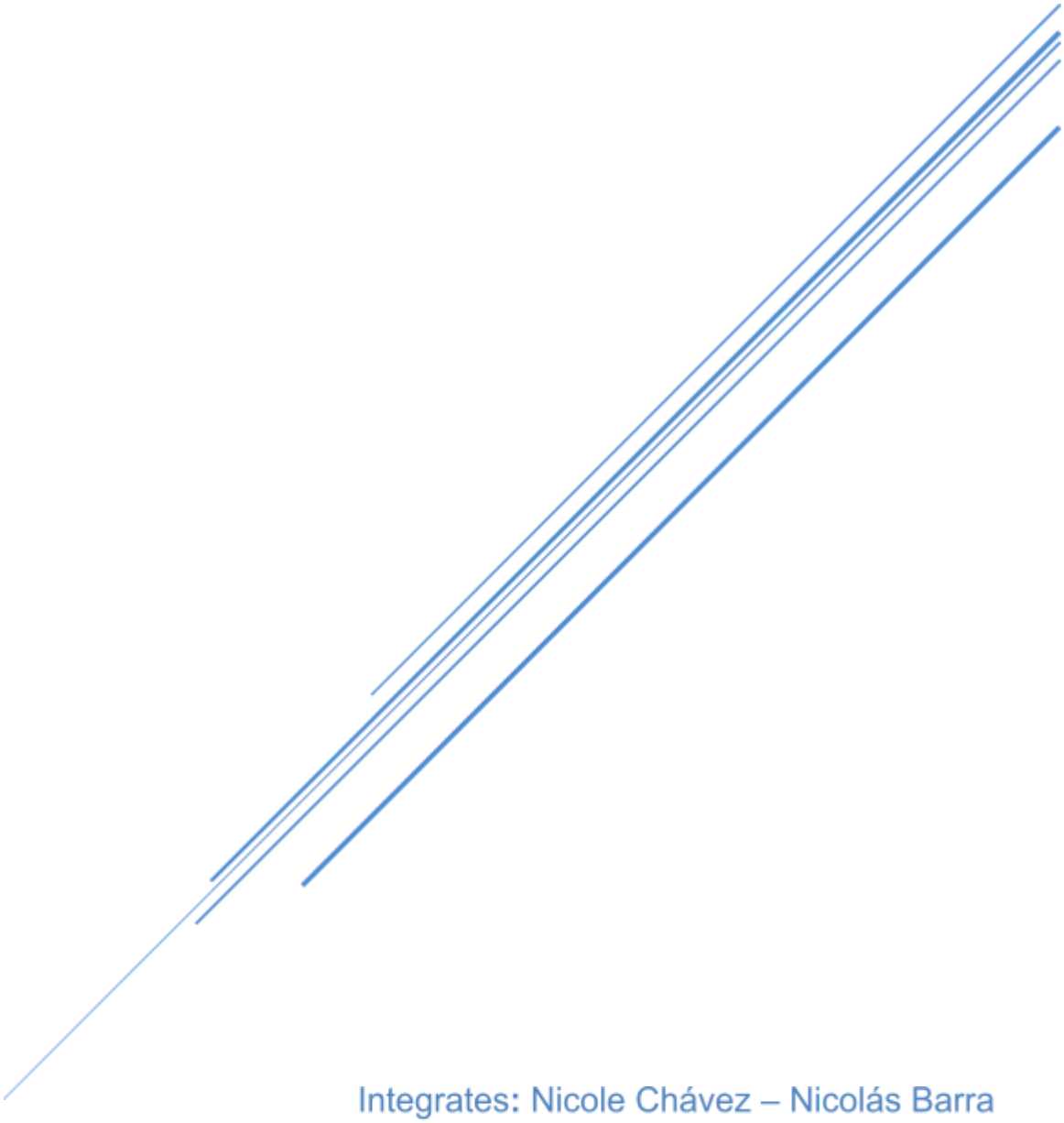
IMPLEMENTACIÓN DE PROGRAMAS PL/SQL

Proyecto: Backend E-Commerce “Pastelería Mil Sabores”



Integrates: Nicole Chávez – Nicolás Barra
Carrera: Analista Programador

1. Introducción

Descripción del Proyecto

El presente proyecto detalla el diseño e implementación del Backend transaccional y analítico para la "Pastelería Mil Sabores". El objetivo principal es la creación de una solución integral de negocio que gestiona de forma robusta el ciclo de vida de un e-commerce.

Para lograr esto, se centralizó toda la lógica de negocio (procedimientos, funciones, tipos y excepciones) en un paquete PL/SQL (PKG_GESTOR_PEDIDOS). Este paquete actúa como la API (Interfaz de Programación de Aplicaciones) principal de la base de datos, gestionando clientes, productos, pedidos y la generación de reportes.

Alcance

El alcance de esta implementación cubre cuatro pilares fundamentales del sistema:

1. **Gestión Transaccional:** Procesamiento seguro de pedidos, validación de inventario (stock) y creación de clientes.

2. **Lógica de Negocio Centralizada:** Encapsulamiento de todas las reglas (cálculos de precios, manejo de estados, excepciones) dentro de un único paquete.

3. **Automatización y Auditoría:** Implementación de Triggers para automatizar procesos (reducción de stock) y auditar cambios críticos en el sistema (a nivel de fila y de sentencia).

4. **Procesamiento de Datos (Reportería):** Transformación de datos transaccionales crudos en información de valor para la toma de decisiones, almacenada en tablas de reportes dedicadas.

2. Desarrollo de Procedimientos y Funciones

Propósito y Aplicación en el Proyecto

En este proyecto, se aplicó una estricta separación de responsabilidades para los programas PL/SQL:

- **Procedimientos (Acciones):** Se utilizaron para ejecutar acciones que modifican el estado de la base de datos (lógica DML). Su rol es tomar parámetros de entrada y ejecutar transacciones complejas de forma controlada y segura, como SP_CREAR_CLIENTE o SP_ACTUALIZAR_ESTADO_PEDIDO.

- **Funciones (Cálculos):** Se utilizaron para realizar cálculos específicos y devolver un único valor, asegurando la reutilización del código. Ejemplos clave son FN_OBTENER_PRECIO_PRODUCTO y FN_CALCULAR_EDAD_CLIENTE, que encapsulan un cálculo específico y pueden ser llamadas desde otros procedimientos o sentencias SQL.

Justificación del Procesamiento de Información Masiva

Para el manejo de la reportería, se implementó una estrategia de Tablas de Reportería. Se descartó el uso de herramientas de depuración (como DBMS_OUTPUT) para esta tarea, ya que son inviables para un sistema real, no escalan, no almacenan resultados y no procesan datos masivos.

Para manejar el procesamiento de información masiva, se crearon procedimientos específicos que actúan como un motor de ETL (Extract, Transform, Load) interno:

- **SP_GENERAR_REPORTE_VENTAS_DIARIAS**
- **SP_GENERAR_REPORTE_GANANCIAS_MES**
- **SP_GENERAR_REPORTE_STOCK_CRITICO**

Estos procedimientos no devuelven datos al usuario, sino que ejecutan complejas consultas de agregación (GROUP BY, SUM) sobre miles de registros transaccionales. El resultado

consolidado se almacena de forma persistente en tablas de reporte (REPORTE_VENTAS_DIARIAS, REPORTE_GANANCIAS_MENSUALES, etc.).

Esta arquitectura es una solución escalable que separa la carga de trabajo transaccional (OLTP) de la analítica (OLAP) y permite que herramientas externas de Business Intelligence (BI) consuman los reportes de forma eficiente.

3. Desarrollo de Packages (Inventario Detallado)

Propósito del Package PKG_GESTOR_PEDIDOS

El pilar de esta arquitectura es el paquete PKG_GESTOR_PEDIDOS. Se optó por un paquete para encapsular toda la lógica de negocio en una "caja negra" cohesionada, creando una solución integral. Esto centraliza la lógica, mejora la mantenibilidad, oculta la complejidad (encapsulación) y permite una gestión de estado (variables globales) y errores (excepciones) robusta y centralizada.

Inventario de Componentes Centralizados en PKG_GESTOR_PEDIDOS

A continuación, se detalla cada componente definido en el PACKAGE HEAD y su propósito en la solución:

1. Tipos de Datos Compuestos (2)

- **TIPO_PRODUCTO_COMPRA (RECORD):**
  Propósito: Define una estructura para manejar un ítem del carrito de compras (un id_producto y una cantidad).
- **TIPO_LISTA_PRODUCTOS (VARRAY):**
  Propósito: Define una colección (un array de hasta 100 TIPO_PRODUCTO_COMPRA). Permite pasar un "carrito de compras" completo como un único parámetro al procedimiento de creación de pedidos.

2. Biblioteca de Excepciones Personalizadas (8)

Se definió una biblioteca de errores de negocio estandarizados usando PRAGMA EXCEPTION_INIT:

- **EX_STOCK_INSUFICIENTE (-20001):**
  Propósito: Se lanza si la cantidad de un producto en el carrito supera el stock disponible en la tabla PRODUCTOS.
- **EX_PRODUCTO_NO_ENCONTRADO (-20002):**
  Propósito: Se lanza si se intenta procesar un id_producto que no existe en la tabla PRODUCTOS.
- **EX_CLIENTE_NO_ENCONTRADO (-20003):**
  Propósito: Se lanza si se intenta crear un pedido para un id_cliente que no existe en la tabla CLIENTES.
- **EX_CORREO_DUPLICADO (-20004):**
  Propósito: Se lanza si se intenta insertar un cliente con un correo que ya existe (violación de UNIQUE).
- **EX_CANTIDAD_INVALIDA (-20005):**
  Propósito: (Reservada) Para validar lógicas donde la cantidad no puede ser cero o negativa.
- **EX_PEDIDO_NO_ENCONTRADO (-20006):**
  Propósito: Se lanza si se intenta actualizar un id_pedido que no existe.
- **EX_ESTADO_PEDIDO_INVALIDO (-20007):**
  Propósito: Se lanza si se intenta actualizar un pedido a un id_estado que no existe en la tabla PEDIDOS_ESTADOS.
- **EX_CONFIG_NO_INICIALIZADA (-20010):**
  Propósito: Se lanza si el paquete falla al cargarse o si el constructor no puede inicializar las variables globales.

3. Funciones - Cálculos y Consultas (4)

- **FN_CALCULAR_TOTAL_PEDIDO (P_ID_PEDIDO NUMBER) RETURN NUMBER**
  Propósito: Recibe un ID de pedido y devuelve el total monetario. Utiliza un cursor explícito para iterar sobre PEDIDOS_DETALLES, sumando el precio del producto por la cantidad del detalle.
- **FN_OBTENER_STOCK_DISPONIBLE (P_ID_PRODUCTO NUMBER) RETURN NUMBER**
  Propósito: Recibe un ID de producto y devuelve su stock actual. Centraliza la consulta de inventario.
- **FN_OBTENER_PRECIO_PRODUCTO (P_ID_PRODUCTO NUMBER) RETURN NUMBER**
  Propósito: Recibe un ID de producto y devuelve su precio. Es una función auxiliar clave para el cálculo del total del pedido.
- **FN_CALCULAR_EDAD_CLIENTE (P_FECHA_NACIMIENTO DATE) RETURN NUMBER**
  Propósito: Recibe una fecha y devuelve la edad en años (usando TRUNC(MONTHS_BETWEEN)).

4. Procedimientos - Acciones y Reportes (7)

SP_CREAR_CLIENTE (P_NOMBRE, P_APELLIDO_PATERNO, P_APELLIDO_MATERNO, P_CORREO, P_DIRECCION, P_FECHA_NACIMIENTO, P_TELEFONO)
Propósito: Gestiona la transacción de registrar un nuevo cliente, manejando la excepción EX_CORREO_DUPLICADO.

SP_CREAR_NUEVO_PEDIDO (P_ID_CLIENTE NUMBER, P_LISTA_PRODUCTOS TIPO_LISTA_PRODUCTOS)
Propósito: Es el procedimiento transaccional más importante. Recibe el VARRAY (carrito), valida el cliente, inserta la cabecera en PEDIDOS, itera sobre el VARRAY, inserta en PEDIDOS_DETALLES (lo que dispara el Triggers de stock), calcula el total (usando FN_OBTENER_PRECIO_PRODUCTO) y actualiza el total en PEDIDOS.

SP_ACTUALIZAR_ESTADO_PEDIDO
(P_ID_PEDIDO NUMBER, P_ID_NUEVO_ESTADO NUMBER)

Propósito: Gestiona el ciclo de vida del pedido (ej. de 'Pendiente' a 'Enviado'), validando que tanto el pedido como el nuevo estado existan.

SP_GENERAR_REPORTE_VENTAS_DIARIAS
(P_FECHA DATE)

Propósito: Procesa información masiva. Ejecuta un INSERT INTO REPORTE_VENTAS_DIARIAS SELECT ... que lee PEDIDOS y PEDIDOS_DETALLES, agrupa (GROUP BY) y suma las ventas del día, almacenándolas en la tabla de reportes.

SP_GENERAR_REPORTE_GANANCIAS_MES
(P_ANIO NUMBER, P_MES NUMBER)

Propósito: Procesa información masiva. Ejecuta un MERGE INTO REPORTE_GANANCIAS_MENSUALES... que calcula las ganancias totales del mes y las inserta o actualiza en la tabla de reportes.

SP_GENERAR_REPORTE_STOCK_CRITICO
(P_UMBRAL_STOCK NUMBER)

Propósito: Procesa información masiva. Limpia y vuelve a llenar la tabla REPORTE_STOCK_CRITICO con todos los productos cuyo stock sea inferior al umbral.

SP_VER_AUDITORIA_SISTEMA
(P_FECHA DATE)

Propósito: Herramienta de depuración. Utiliza DBMS_OUTPUT para imprimir en consola los registros de la tabla AUDITORIA_SISTEMA de una fecha específica.

5. Constructores y Variables Globales

Se implementó un patrón de inicialización robusto utilizando ambos tipos de constructores:

g_id_estado_pendiente (Variable Global Pública):

Propósito: Almacena el ID del estado "Pendiente". Es pública (definida en el HEAD) para que el SP_CREAR_NUEVO_PEDIDO la utilice en lugar de un "número mágico" (ej. 1).

SP_INICIALIZAR_PAQUETE (Constructor Público):
Propósito: Procedimiento que consulta PEDIDOS_ESTADOS y carga el ID de 'Pendiente' en la variable g_id_estado_pendiente.

Bloque BEGIN...END; (Constructor Privado):
Propósito: Bloque anónimo al final del PACKAGE BODY. Se ejecuta automáticamente una vez por sesión para llamar a SP_INICIALIZAR_PAQUETE y asegurar que el paquete esté configurado antes de usarse.

Justificación (Caso de Uso): Esta arquitectura garantiza que la lógica del negocio (SP_CREAR_NUEVO_PEDIDO) esté desacoplada de los datos de configuración (el ID de 'Pendiente'). Si el ID del estado cambia en la tabla PEDIDOS_ESTADOS, el constructor lo cargará automáticamente en la variable global, y el procedimiento de creación de pedidos seguirá funcionando sin necesidad de modificar su código.

4. Desarrollo de Triggers (Inventario Detallado)

Propósito de los Triggers en el Proyecto

Se utilizaron triggers (disparadores) para implementar lógica que debe ejecutarse automáticamente como respuesta a un evento DML (INSERT, UPDATE, DELETE). Esto garantiza que ciertas reglas de negocio se cumplan siempre, independientemente de quién o qué realice la acción.

Se implementaron para dos propósitos críticos:

- **Automatizar Lógica de Negocio Crítica:** Garantizar la integridad de los datos (ej. si se vende un producto, el stock debe reducirse).
- **Implementar un Sistema de Auditoría:** Registrar eventos clave para el control y seguridad del sistema.

Se implementó un sistema de auditoría dual y lógica de negocio automatizada mediante la siguiente colección de 6 triggers:

1. TR_PEDIDOS_DETALLES_AI_REDUCIR_STOCK

Tabla: PEDIDOS_DETALLES
Evento: AFTER INSERT
Nivel (Tipo): Fila (FOR EACH ROW)
Propósito (Lógica de Negocio): Garantiza la integridad del inventario. Inmediatamente después de insertar una línea de detalle de pedido, este Triggers valida el stock y actualiza la tabla PRODUCTOS, restando la cantidad (:NEW.cantidad) del stock del producto (:NEW.id_producto).

2. TR_CLIENTES_AIU_AUDITORIA
Tabla: CLIENTES
Evento: AFTER INSERT OR UPDATE
Nivel (Tipo): Fila (FOR EACH ROW)
Propósito (Auditoría Detallada): Registra el id_cliente específico (:NEW.id_cliente) que fue afectado por la operación DML en la tabla de auditoría detallada (CLIENTES_AUDITORIA).

3. TR_PEDIDOS_ESTADO_AUDITORIA
Tabla: PEDIDOS
Evento: AFTER UPDATE OF id_estado
Nivel (Tipo): Fila (FOR EACH ROW)
Propósito (Auditoría Condicional): Se dispara solo si cambia la columna id_estado. Utiliza lógica IF para comparar :OLD.id_estado y :NEW.id_estado, registrando un mensaje descriptivo en AUDITORIA_SISTEMA (ej. "Pedido #101 CANCELADO (de 2 a 5)").

4. TR_PRODUCTOS_AUDIT_STMT
Tabla: PRODUCTOS
Evento: AFTER INSERT OR DELETE OR UPDATE OF stock
Nivel (Tipo): Sentencia (FOR EACH STATEMENT)
Propósito (Auditoría General): Registra un solo evento en AUDITORIA_SISTEMA cuando se ejecuta una sentencia masiva (ej. INSERT_MASIVO_PRODUCTOS), priorizando el rendimiento sobre el detalle.

5. TR_CLIENTES_AUDIT_STMT
Tabla: CLIENTES
Evento: AFTER INSERT OR DELETE
Nivel (Tipo): Sentencia (FOR EACH STATEMENT)
Propósito (Auditoría General): Registra eventos masivos (ej. "INSERT_MASIVO_CLIENTES") en AUDITORIA_SISTEMA.

6. TR_PEDIDOS_AUDIT_STMT
Tabla: PEDIDOS
Evento: AFTER INSERT OR DELETE
Nivel (Tipo): Sentencia (FOR EACH STATEMENT)
Propósito (Auditoría General): Registra eventos masivos (ej. "DELETE_MASIVO_PEDIDOS") en AUDITORIA_SISTEMA.

Justificación de Triggers de Nivel de Fila vs. Nivel de Sentencia

Como se observa en el inventario, se implementaron ambos tipos de triggers, ya que cada tipo resuelve problemas diferentes:

Triggers de Nivel de Fila (FOR EACH ROW):

Estos triggers se ejecutan una vez por cada fila afectada. Son necesarios cuando la lógica del trigger depende de los datos específicos de la fila que está siendo modificada.

Justificación (Lógica): TR_PEDIDOS_DETALLES_AI_REDUCIR_STOCK debe ser de fila, ya que necesita acceder a los valores :NEW.cantidad y :NEW.id_producto para saber cuánto stock restar y de qué producto específico. Si fuera de sentencia, no tendría acceso a esa información.

Justificación (Auditoría Detallada): TR_CLIENTES_AIU_AUDITORIA debe ser de fila porque su propósito es auditar qué cliente específico fue modificado. Se inserta un registro en CLIENTES_AUDITORIA usando el ID del cliente (:NEW.id_cliente). Esto solo es posible con FOR EACH ROW.

Justificación (Auditoría Condicional): TR_PEDIDOS_ESTADO_AUDITORIA debe ser de fila para comparar los valores antiguos y nuevos (:OLD.id_estado vs :NEW.id_estado) y registrar un log solo si el estado realmente cambió.

Triggers de Nivel de Sentencia (FOR EACH STATEMENT):

Estos triggers se ejecutan una sola vez por cada sentencia DML, sin importar cuántas filas afecte (0 o un millón). Se utilizan para auditoría general y son cruciales para el rendimiento.

Justificación (Auditoría General): TR_PRODUCTOS_AUDIT_STMT (y los otros de auditoría general) se diseñaron a nivel de sentencia. Su propósito no es saber qué producto cambió, sino cuándo se realizó una operación masiva sobre el inventario.

Ventaja de Rendimiento: Si un proceso nocturno actualiza el stock de 500 productos con una sola sentencia UPDATE, este trigger se dispara una sola vez, insertando un único registro en AUDITORIA_SISTEMA. Un trigger de fila se habría disparado 500 veces, causando un impacto significativo e innecesario en el rendimiento.

5. Conclusión

El proyecto implementó exitosamente una solución de Backend integral y robusta para un e-commerce. La lógica de negocio se centralizó en un paquete (PKG_GESTOR_PEDIDOS), mejorando la mantenibilidad y seguridad. Se implementó un sistema de auditoría dual (fila y sentencia) mediante triggers, y se desarrollaron procedimientos de reportería capaces de procesar información masiva, almacenando los resultados en tablas dedicadas.

Impacto del Proyecto

La arquitectura implementada transforma el modelo. El uso del Package proporciona encapsulación y seguridad. El uso de Triggers de fila asegura la integridad de los datos (el stock siempre estará actualizado), mientras que los triggers de sentencia proveen un sistema de auditoría eficiente. Finalmente, los Procedimientos de Reporte establecen una base escalable para la inteligencia de negocios (BI), permitiendo que la "Pastelería Mil Sabores" pueda analizar sus ventas, ganancias y stock crítico sin afectar el rendimiento de las transacciones diarias.

Recomendaciones (Mejoras Futuras)

La arquitectura actual está preparada para una "Fase 2". La recomendación principal es implementar las entidades INGREDIENTES, RECETAS y RESEÑAS. Esto permitiría expandir el paquete para incluir:

Control de Producción: Calcular el costo de producción de cada pastel basado en sus ingredientes.

Gestión de Inventario (Nivel 2): Descontar materia prima (harina, azúcar) en lugar de solo productos terminados.

Sistema de Feedback: Implementar un sistema de reseñas de clientes.

6. Anexos

Anexo A: Script de Base de Datos - Enlace

Anexo B: Script de Package - Enlace

Anexo C: Script de Triggers - Enlace

Anexo D: Modelo Entidad-Relación (MER) - Enlace