Universidad del Valle de Guatemala
Facultad de Ingeniería
Ciencia de la Computación y Tecnologías de la información
cc3088 - Bases de Datos 1 Ciclo 1, 2025

Anggelie Velásquez, 221181
Mia Fuentes, 23775

Ejercicio 5 – Triggers

Triggers:

1. BEFORE INSERT - Validación de Stock en Ingredientes:
    a. Propósito: Evitar que se ingresen ingredientes con stock negativo y proteger la integridad del inventario.
    b. Justificación: Se uso BEFORE INSERT para que los datos se validen antes de que entren a la tabla.
    c. Alternativa: Se podria usar un CHECK (stock >= 0), pero con el trigger podemos lanzar mensajes más claros y agregar validaciones extras si se necesita.

2. AFTER INSERT - Registro de Nuevos Usuarios:
    a. Propósito: Llevar un registro cada vez que se crea un usuario.
    b. Justificación: Al usar AFTER INSERT nos aseguramos de que primero se cree el usuario sin problemas y luego registramos la acción en la tabla de logs.
    c. Alternativa: Se podría hacer desde el backend de la app, pero con el trigger se garantiza que se guarde el registro aunque el cambio venga directo desde SQL.

3. BEFORE UPDATE - Validación de Precio en Kits:
    a. Propósito: Validar que los nuevos precios y descuentos de los kits sigan las reglas de negocio.
    b. Justificación: Es evitar errores antes de que actualicen los datos, por eso usamos BEFORE UPDATE.
    c. Alternativa: Se puede hacer desde PgAdmin, pero no nos asegura que se apliquen los cambios si alguien modifica la base.

4. AFTER UPDATE - Registro de Cambios de Estado en Pedidos:
    a. Propósito: Dejar constancia cada vez que cambia el estado de un pedido (como de “pendiente” a “en camino”).
    b. Justificación: Se uso AFTER UPDATE para que el cambio se registre solo si fue exitoso.
    c. Alternativa: Podría registrarse desde pgadmin, pero con el trigger se asegura que los datos se modifiquen completamente (según cuales datos se hayan ingresado).

5. BEFORE DELETE - Protección contra Eliminación de Categorías Usadas:

a. Propósito: Evitar que se eliminen categorías que todavía están siendo usadas en recetas.
b. Justificación: BEFORE DELETE nos ayuda a revisar si hay dependencias antes de eliminar algo importante.
c. Alternativa: Se podría usar una restricción con ON DELETE RESTRICT, pero con el trigger mostramos un mensaje más específico.

6. AFTER DELETE - Registro de Eliminación de Kits:
   a. Propósito: Guardar un historial cuando se borra un kit por si se necesita más adelante.
   b. Justificación: AFTER DELETE garantiza que se registre la acción solo si la eliminación fue real.
   c. Alternativa: Podríamos hacer un "soft delete" con un campo "activo", pero con este trigger dejamos un log más claro cuando el kit ya no existe.

7. BEFORE TRUNCATE - Prevención de Truncado de Tablas Críticas:
   a. Propósito: Evitar que alguien borre todo de golpe por accidente en una tabla importante.
   b. Justificación: El BEFORE TRUNCATE bloquea la operación si no está permitida.
   c. Alternativa: Se pueden usar permisos, pero el trigger nos da una alternativa servirá por si alguien con acceso alto comete un error.

8. AFTER TRUNCATE - Registro de Operaciones de Truncado:
   a. Propósito: Guardar quién hizo un TRUNCATE y en qué tabla.
   b. Justificación: AFTER TRUNCATE solo registra la acción cuando ya se ejecutó, sin modificar la operación.
   c. Alternativa: Se podría monitorear desde pgadmin, pero el trigger asegura que quede registrado sin importar de dónde venga la orden.

<u>Respondiendo las siguientes preguntas:</u>

1. ¿Cómo garantizaron la integridad de los datos?
   - Con triggers BEFORE para prevenir datos incorrectos antes de que entren, y triggers AFTER para registrar acciones importantes sin afectar el funcionamiento normal. Así, mantenemos todo en orden y trazable.

2. Si el número de productos en la tienda aumentara significativamente, ¿qué modificaciones harían para garantizar el rendimiento?
   - Crear índices en columnas como nombre o categoría.
   - Optimizar las consultas.
   - Particionar las tablas más grandes.
   - Usar funciones almacenadas para procesos repetitivos.
   - Mejorar la infraestructura del servidor si es necesario.

3. ¿Qué pruebas pueden realizar para para verificar el rendimiento óptimo de la base de datos?

- Pruebas de carga simulando muchos usuarios.
- Usar EXPLAIN ANALYZE para ver qué tan bien corren las consultas.
- Monitorear la base desde herramientas como pgAdmin.
- Usar JMeter o scripts para hacer benchmarking.

4. ¿Es su diseño escalable? ¿Por qué? Si la respuesta es no también respondan ¿qué están haciendo con su vida?
   - Sí, es escalable ya que:
     - Los triggers están enfocados en tareas puntuales y no generan alta carga.
     - La base está normalizada y puede crecer sin redundancias.
     - Las validaciones se distribuyen entre lógica de negocio y base de datos.
     - Con las mejoras mencionadas en la respuesta 2, el sistema podría adaptarse a un mayor volumen de datos sin afectar la integridad ni el rendimiento

5. ¿Qué mejora pudieran hacer al sistema para mejorar su rendimiento?
   - Desnormalizar parcialmente datos muy consultados para evitar JOIN complejos.
   - Implementar cacheo a nivel de aplicación para consultas frecuentes.
   - Agregar paginación en listados largos.
   - Usar triggers condicionales para evitar ejecuciones innecesarias.
   - Agregar logs de rendimiento en los triggers para detectar cuellos de botella.