**Optimización de la Base de Datos de Ventas – Tienda Virtual**

**Integrantes:** Leandro Jaldin

Ariany Lopez

Camila Catorceno

**INDICES**

***Definición:***

* Los índices actúan como accesos rapidos
* Evita que la base de datos tenga que buscar fila por fila
* Acelera consultas

1. **Índice en la tabla DetalleVenta**

**CREATE INDEX idx\_DetalleVenta\_PedidoID ON DetalleVenta(PedidoID);**

**Razón:** La tabla DetalleVenta puede tener muchos productos para un solo pedido

**Ejemplo:**

Yo cliente pido 2 laptops y 4 audífonos = 1 solo pedido y muchos productos

**SELECT \* FROM DetalleVenta WHERE PedidoID = 1;**

* Sin índice SQL revisa toda la tabla
* Con índice va directo a los registros de PedidoID = 1

**¿Porque es importante?**

* La tienda usara todo el tiempo esa CONSULTA para ver que producto hay en cada pedido
* Un índice en PedidoID hace que la búsqueda sea instantánea, en lugar de recorrer miles de filas

1. **Índice en la tabla Pagos**

**CREATE INDEX idx\_Pagos\_PedidoID ON Pagos(PedidoID);**

**Razon:** La tabla pagos almacena los pagos de cada pedido

**Ejemplo:**

Yo dueño quiero saber cuanto pago tal cliente por tal pedido

**SELECT \* FROM Pagos WHERE PedidoID = 10;**

* Sin índice revisa todos lo pagos uno por uno
* Con índice va directo al pedidoID

**¿Porque es importante?**

* Cada pedido puede tener múltiples pagos
* Ejemplo: un cliente paga una parte con tarjeta y la otra parte en efectivo
* El índice hace que la consulta sea mucho más rápida cuando el sistema quiere ver pagos de un pedido

**Beneficios de los Índices:**

* Aceleran las consultas, las búsquedas y filtrados son más rápidos.
* Mejoran el rendimiento, estos reducen el tiempo de ejecución de consultas frecuentes.
* Optimizan el uso de recursos, disminuyen la carga en el servidor de bases de datos.

**TRANSACCIONES**

***Definición:***

Las transacciones son secuencias de operaciones que se ejecutan como una sola unidad de trabajo. Si una operación falla, todas las operaciones dentro de la transacción se deshacen (ROLLBACK). Esto garantiza la integridad de los datos.

**BEGIN TRANSACTION**

**DECLARE @ClienteID INT = 1;**

**DECLARE @ProductoID INT = 3;**

**DECLARE @Cantidad INT = 2;**

**DECLARE @Estado VARCHAR(50) = 'Pendiente';**

**DECLARE @Precio DECIMAL(10,2) = (SELECT Precio FROM Productos WHERE ProductoID = @ProductoID);**

**DECLARE @Total DECIMAL(10,2) = @Cantidad \* @Precio;**

**BEGIN TRY**

**IF(@Cantidad <= (SELECT Stock FROM Productos WHERE ProductoID = @ProductoID))**

**BEGIN**

**INSERT INTO Pedidos(ClienteID, FechaPedido, Estado, Total)**

**VALUES (@ClienteID, GETDATE(), @Estado, @Total)**

**DECLARE @PedidoID INT = (SELECT TOP 1 PedidoID FROM Pedidos ORDER BY PedidoID DESC)**

**INSERT INTO DetalleVenta(PedidoID, ProductoID, Cantidad, PrecioUnitario)**

**VALUES (@PedidoID, @ProductoID, @Cantidad, @Precio)**

**UPDATE Productos**

**SET Stock = Stock - @Cantidad**

**WHERE ProductoID = @ProductoID**

**COMMIT;**

**PRINT 'Transferencia completada exitosamente.';**

**END**

**ELSE**

**BEGIN**

**ROLLBACK;**

**PRINT 'Stock insuficiente. Transacción revertida.';**

**END**

**END TRY**

**BEGIN CATCH**

**ROLLBACK;**

**PRINT 'Error en la transferencia, cambios revertidos.';**

**END CATCH**

Esta transacción garantiza que el stock se actualice solo si la venta se registra correctamente, evita inconsistencias en los datos como ventas registradas sin actualización de stock.

Si el Stock es suficiente, la transacción se completa y se registra la venta en DetalleVenta y se actualiza el stock en Productos, de lo contrario la transacción se deshace y no se registra la venta.

**Por qué usamos Transacciones?**

Cuando un cliente realiza un pedido, se debe:

1. Registrar el pedido en la tabla Pedidos.
2. Registrar los detalles de la venta en la tabla DetalleVenta.
3. Actualizar el stock de los productos en la tabla Productos.

La transacción asegura que estas tres operaciones se completen juntas. Si alguna falla (por ejemplo, no hay suficiente stock), la transacción se deshace y los datos quedan como estaban antes de la operación.

**TRIGGER**

Es un procedimiento que se ejecuta automáticamente cuando ocurre un evento específico en la base de datos (por ejemplo, una inserción en una tabla). En este caso, implementamos un trigger para actualizar el stock de un producto automáticamente cuando se registra una nueva venta en la tabla DetalleVenta.

**CREATE TRIGGER ActualizarStock**

**ON DetalleVenta**

**AFTER INSERT**

**AS**

**BEGIN**

**UPDATE P**

**SET P.Stock = P.Stock - I.Cantidad**

**FROM Productos P**

**INNER JOIN inserted I ON P.ProductoID = I.ProductoID;**

**IF EXISTS (SELECT 1 FROM Productos WHERE Stock < 0)**

**BEGIN**

**ROLLBACK TRANSACTION;**

**PRINT 'Error: Stock insuficiente para uno o más productos.';**

**END**

**END;**

Este trigger automatiza la actualización del stock cuando se registra una venta, y garantiza que no se registren ventas si no hay suficiente stock.

Para verificar que el trigger trabaja correctamente debemos hacer la siguiente inserción de datos y verificar si el stock disminuye automáticamente.

**SELECT Stock FROM Productos WHERE ProductoID = 1;**

**INSERT INTO DetalleVenta (PedidoID, ProductoID, Cantidad, PrecioUnitario)**

**VALUES (1, 1, 1, 1200.50);**

**SELECT Stock FROM Productos WHERE ProductoID = 1;**

**Por qué usamos Trigger?**

En la tabla DetalleVenta, un trigger puede actualizar automáticamente el stock en la tabla Productos cada vez que se inserte un nuevo registro. Si no hay suficiente stock, el trigger puede deshacer la operación y notificar al usuario, evitando que se registre una venta sin stock disponible.

**STORED PROCEDURE**

***Definición:***

Un Stored Procedure (Procedimiento Almacenado) es un conjunto de sentencias SQL que se almacenan en la base de datos y se ejecutan como una unidad. Son útiles para encapsular lógica de negocio y reutilizar código.

* **Stored Procedure para consultar ventas por Cliente:**

**CREATE PROCEDURE ConsultarVentasPorCliente**

**@ClienteID INT**

**AS**

**BEGIN**

**SELECT**

**P.PedidoID,**

**P.FechaPedido,**

**P.Estado,**

**P.Total,**

**DV.ProductoID,**

**DV.Cantidad,**

**DV.PrecioUnitario,**

**PR.NombreProducto**

**FROM Pedidos P**

**INNER JOIN DetalleVenta DV ON P.PedidoID = DV.PedidoID**

**INNER JOIN Productos PR ON DV.ProductoID = PR.ProductoID**

**WHERE P.ClienteID = @ClienteID;**

**END;**

Para ejecutar el Stored Procedure y obtener las ventas de un cliente específico, usamos el comando:

**EXEC ConsultarVentasPorCliente @ClienteID = 1;**

Esto devuelve todos los pedidos realizados por el cliente con ClienteID, junto con los detalles de los productos vendidos en cada pedido.

**Por qué usamos un SP?**

Imaginemos que queremos consultar todas las ventas realizadas por un cliente específico, incluyendo los detalles de los productos vendidos. En lugar de escribir una consulta compleja que una las tablas Pedidos, DetalleVenta y Productos, podemos crear un SP que reciba el ID del cliente y devuelva los pedidos y detalles de venta asociados.

**CONCLUSIONES**

Las optimizaciones realizadas en la base de datos tienen los siguientes impactos positivos:

1. **Mejora del Rendimiento:**

* Los índices aceleran las consultas, especialmente en tablas grandes.
* Las transacciones y stored procedures garantizan la integridad de los datos y evitan inconsistencias.

1. **Automatización de Procesos:**

* El trigger actualiza automáticamente el Stock cuando se registra una venta, lo que reduce el riesgo de errores humanos.

1. **Facilidad de Mantenimiento:**

* Los Stored Procedures facilitan el mantenimiento y la reutilización del código.

La implementación de índices, transacciones, triggers y stored procedures en una base de datos no solo mejora el rendimiento, sino que también garantiza la integridad de los datos, automatiza procesos y facilita el mantenimiento. Estos son esenciales cuando se trabaja con grandes volúmenes de datos y operaciones complejas.