Taller Práctico: Optimización y Control de Datos en AdventureWorks

Duración estimada: 2 horas

Requisitos previos:

- Tener instalada la base de datos AdventureWorks2019 (u otra versión compatible).

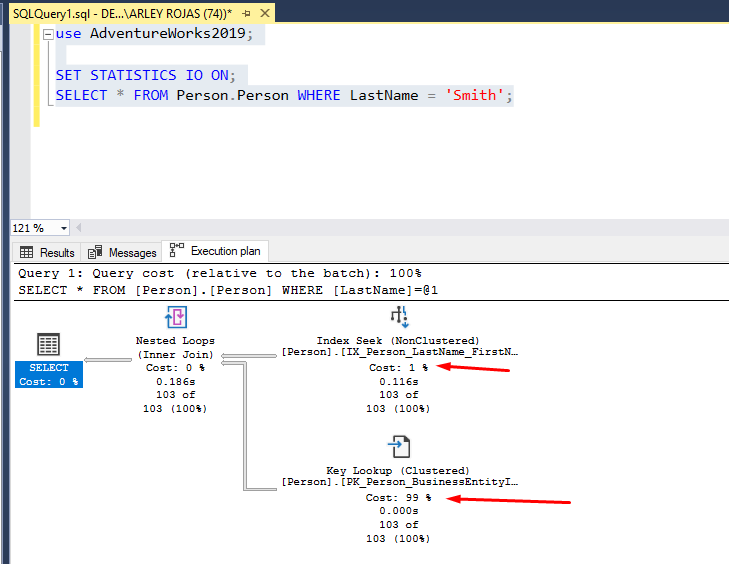
- SQL Server Management Studio (SSMS) u otra herramienta de administración de SQL Server.

# Parte 1: Indexación

Objetivo: Mejorar el rendimiento de consultas utilizando índices.

1. Consulta sin índice:

SET STATISTICS IO ON;  
SELECT \* FROM Person.Person WHERE LastName = 'Smith';



Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

2. Crear un índice no agrupado:

CREATE NONCLUSTERED INDEX IX\_LastName\_Person   
ON Person.Person (LastName);

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

3. Vuelve a ejecutar la consulta y observa las diferencias utilizando SET STATISTICS IO ON.

4. Eliminar el índice si es necesario:

DROP INDEX IX\_LastName\_Person ON Person.Person;

Bonus: Crea un índice compuesto sobre LastName y FirstName, y evalúa el impacto.

# Parte 2: Sincronización de Tablas

Objetivo: Mantener dos tablas sincronizadas (útil en auditoría, replicación o integración de datos).

1. Crear una tabla espejo de auditoría:

CREATE TABLE Person.Person\_Audit (  
 BusinessEntityID INT,  
 FirstName NVARCHAR(50),  
 LastName NVARCHAR(50),  
 ModifiedDate DATETIME DEFAULT GETDATE()  
);

2. Insertar datos existentes (sincronización inicial):

INSERT INTO Person.Person\_Audit (BusinessEntityID, FirstName, LastName)  
SELECT BusinessEntityID, FirstName, LastName  
FROM Person.Person;

3. Verifica que los datos estén sincronizados.

# Parte 3: Triggers

Objetivo: Automatizar la actualización de la tabla espejo cuando se realizan cambios en la tabla original.

1. Trigger para INSERT:

CREATE TRIGGER trg\_Person\_Insert  
ON Person.Person  
AFTER INSERT  
AS  
BEGIN  
 INSERT INTO Person.Person\_Audit (BusinessEntityID, FirstName, LastName)  
 SELECT BusinessEntityID, FirstName, LastName  
 FROM inserted;  
END;

2. Trigger para UPDATE:

CREATE TRIGGER trg\_Person\_Update  
ON Person.Person  
AFTER UPDATE  
AS  
BEGIN  
 INSERT INTO Person.Person\_Audit (BusinessEntityID, FirstName, LastName)  
 SELECT BusinessEntityID, FirstName, LastName  
 FROM inserted;  
END;

3. Trigger para DELETE (opcional):

CREATE TRIGGER trg\_Person\_Delete  
ON Person.Person  
AFTER DELETE  
AS  
BEGIN  
 PRINT 'Se eliminó un registro de Person.Person';  
END;

4. Prueba los triggers:

INSERT INTO Person.Person (BusinessEntityID, FirstName, LastName, rowguid, ModifiedDate)  
VALUES (21000, 'Carlos', 'Gómez', NEWID(), GETDATE());  
  
UPDATE Person.Person  
SET FirstName = 'Carlos Alberto'  
WHERE BusinessEntityID = 21000;  
  
DELETE FROM Person.Person  
WHERE BusinessEntityID = 21000;

5. Verifica los cambios en la tabla Person\_Audit.

# Evaluación Final

- ¿La indexación mejoró el rendimiento? ¿Cuánto?

- ¿Los triggers mantuvieron sincronizada la tabla Person\_Audit?

- ¿Qué otros usos podrías darle a estos triggers?

# Reflexión

- ¿Qué ventajas y desventajas tienen los triggers frente a otros mecanismos como jobs o procedimientos almacenados?

- ¿Cuándo usarías índices compuestos?